AIX เวอร์ชัน 7.2

แนวคิดการเขียนโปรแกรมทั่วไป

AIX เวอร์ชัน 7.2

แนวคิดการเขียนโปรแกรมทั่วไป

หมายเหตุ

ก่อนใช้ข้อมูลนี้ รวมถึงผลิตภัณฑ์ที่สนับสนุน โปรดอ่าน ข้อมูลใน “คำประกาศ” ในหน้า 919

This edition applies to AIX Version 7.2 and to all subsequent releases and modifications until otherwise indicated in new editions.

สารบัญ

เกี่ยวกับเอกสารนี้

คอลเล็กชันหัวข้อนี้ให้ผู้พัฒนาแอ็พพลิเคชันมีข้อมูล ทั้งหมดเกี่ยวกับการเขียนแอ็พพลิเคชันสำหรับระบบปฏิบัติการ AIX® โปรแกรมเมอร์สามารถใช้คอลเล็กชันหัวข้อนี้เพื่อหาความรู้เกี่ยวกับคำแนะนำ การโปรแกรมมิง และรีซอร์ส หัวข้อประกอบด้วยการจัดการ อินพุตและเอาต์พุต, curses, ระบบไฟล์ และไดเรกทอรี, lex และ yacc, logical volume programming, shared libraries, การสนับสนุนโปรแกรมขนาดใหญ่ และ System Management Interface Tool (SMIT)

การเน้น

หลักการเน้นความสำคัญต่อไปนี้ถูกใช้ในเอกสารนี้:

ตัวหนา	ระบุคำสั่ง รูทีนย่อย คีย์เวิร์ด ไฟล์ โครงสร้าง ไดเร็กทอรี และรายการอื่นๆ ที่มีชื่อ ถูกกำหนดไว้แล้วโดยระบบ รวมทั้งระบุอ็อบเจ็กต์กราฟิก เช่น ปุ่ม เลเบล และไอคอนที่ผู้ใช้เลือก
ตัวเอียง	ระบุพารามิเตอร์ที่ชื่อแท้จริง หรือค่าจะถูกกำหนดโดยผู้ใช้
โมโนสเปซ	ระบุตัวอย่างค่าข้อมูลที่ระบุ ตัวอย่างข้อความที่คล้ายกับที่คุณจะเห็นเมื่อถูกแสดง ตัวอย่าง ของส่วนของโค้ดโปรแกรมที่คล้ายกับที่คุณอาจเขียนในฐานะที่เป็นโปรแกรมเมอร์ ข้อความจากระบบ หรือข้อมูลที่คุณควรพิมพ์

การคำนึงถึงขนาดตัวพิมพ์ใน AIX

ทุกอย่างในระบบปฏิบัติการ AIX® นั้นต้องคำนึงถึงขนาดตัวพิมพ์ หมายความว่าระบบจะถือว่าอักษรตัวพิมพ์ใหญ่ และตัวพิมพ์เล็กแตกต่างกัน ตัวอย่าง คุณสามารถใช้คำสั่ง **ls** เพื่อแสดงรายการไฟล์ ถ้าคุณพิมพ์ `LS` ระบบจะตอบกลับคำสั่งนั้นว่า `is not found` เช่นเดียวกับ **FILEA**, **FiLea** และ **filea** ถือเป็นชื่อไฟล์ที่ต่างกัน แม้ว่าจะอยู่ในไดเร็กทอรีเดียวกัน เพื่อหลีกเลี่ยงสาเหตุการเกิดการดำเนินการที่ต้องการ ให้กระทำ ทำให้แน่ใจเสมอว่าคุณใช้ขนาดตัวพิมพ์ถูกต้อง

ISO 9000

ISO 9000 registered quality systems were used in the development and manufacturing of this product.

แนวคิดโปรแกรมมิงทั่วไป

คอลเล็กชันหัวข้อนี้ให้ผู้พัฒนาแอ็พพลิเคชันมีข้อมูล ทั้งหมดเกี่ยวกับการเขียนแอ็พพลิเคชันสำหรับระบบปฏิบัติการ AIX® โปรแกรมเมอร์สามารถใช้ข้อมูลนี้เพื่อหาความรู้เกี่ยวกับคำแนะนำ การโปรแกรมมิง และรีซอร์ส หัวข้อประกอบด้วยการจัดการ อินพุตและเอาต์พุต curses ระบบไฟล์ และไดเร็กทอรี lex และ yacc, logical volume programming, shared libraries, การสนับสนุนโปรแกรมขนาดใหญ่ และ System Management Interface Tool (SMIT)

The AIX operating system is designed to support The Open Group's Single UNIX Specification Version 3 (UNIX 03) for portability of operating systems based on the UNIX operating system. Many new interfaces, and some current ones, have been added or enhanced to meet this specification. To determine the correct way to develop a UNIX 03 portable application, see The Open Group's UNIX 03 specification on The UNIX System website (http://www.unix.org).

สิ่งใหม่ในแนวคิดโปรแกรมมิงทั่วไป

อ่านเกี่ยวกับข้อมูลใหม่ หรือที่เปลี่ยนแปลงอย่างมากสำหรับ คอลเล็กชันหัวข้อโปรแกรมมิงทั่วไป

วิธีการดู มีอะไรใหม่หรือมีอะไรที่เปลี่ยนแปลง

ในไฟล์ PDF นี้ คุณอาจเห็นแถบการแก้ไข (|) ในขอบด้านซ้าย ที่ระบุข้อมูลใหม่ และที่เปลี่ยนแปลง

October 2016

ข้อมูลต่อไปนี้ เป็นข้อสรุปของอัพเดตที่ทำกับชุดหัวข้อ แนวคิดทั่วไปเกี่ยวกับการเขียน โปรแกรม

อัพเดตหัวข้อต่อไปนี้ด้วยข้อมูลเกี่ยวกับ probe manager ใหม่และตัวแปรที่มีในตัว ที่เกี่ยวข้อง:

- อิลิเมนต์ภาษา
- "การรัน ProbeVue" ในหน้า 340

December 2015

ข้อมูลต่อไปนี้ เป็นข้อสรุปของอัพเดตที่ทำกับชุดหัวข้อ แนวคิดทั่วไปเกี่ยวกับการเขียน โปรแกรม

- เพิ่มโปรแกรมจัดการโพรบต่อไปนี้ในหัวข้อ "การรัน ProbeVue" ในหน้า 340:
  - โปรแกรมจัดการโพรบ I/O
  - โปรแกรมจัดการเครือข่ายโพรบ
  - โปรแกรมจัดการโพรบ Sysproc
- เพิ่มสองฟังก์ชันต่อไปนี้ในหัวข้อ Vue Functions:
  - convert_ip4_addr
  - convert_ip6_addr
- อัพเดตหัวข้อต่อไปนี้ด้วยการเปลี่ยนแปลงเพื่อให้มีผลกับโปรแกรมจัดการโพรบใหม่และตัวแปรบิวด์อิน ที่เกี่ยวข้องที่ตำแหน่งที่ต่างกัน:

- อิลิเมนต์ภาษา
- สคริปต์ภาษา
- ฟังก์ชัน Vue

- อัพเดตข้อมูลเกี่ยวกับแบบจำลองเธรด 1:1 ใน“โมเดลเธรดและตัวประมวลผลเสมือน” ในหน้า 460หัวข้อ โมเดลนี้เป็นโมเดลดีฟอลต์

เครื่องมือและยูทิลีตี

ส่วนนี้ให้ภาพรวมของเครื่องมือและยูทิลิตีที่คุณสามารถใช้เพื่อพัฒนาโปรแกรมภาษาที่คอมไพล์ด้วย C

เครื่องมือจำนวนมากจะถูกแสดงไว้ เพื่อช่วยคุณพัฒนาโปรแกรมที่คอมไพล์ด้วย C ทูลจะแสดงวิธีใช้ด้วยภารกิจโปรแกรมมิงต่อไปนี้ :

รูทีนย่อยและคำสั่ง shell จะถูกแสดงเพื่อใช้ในโปรแกรมที่คอมไพล์ด้วย C

- การป้อนโปรแกรมลงใน ระบบ
- การตรวจสอบโปรแกรม
- การคอมไพล์และการลิงก์โปรแกรม
- รูทีนย่อย
- คำสั่ง Shell

การป้อนโปรแกรมลงในระบบ

ระบบมีเอดิเตอร์บรรทัดที่เรียกว่า **ed** สำหรับใช้ในการป้อนโปรแกรมลงในไฟล์ ระบบยังมีเอดิเตอร์เต็มจอที่เรียกว่า **vi** ซึ่งแสดงข้อมูลเต็มหน้าจอในแต่ละครั้ง และอนุญาตให้แก้ไขไฟล์แบบโต้ตอบ

การตรวจสอบโปรแกรม

ใช้คำสั่งต่อไปนี้เพื่อตรวจสอบรูปแบบของโปรแกรมสำหรับความสอดคล้องกัน และความถูกต้อง:

คำสั่ง	คำอธิบาย
cb	จัดรูปแบบซอร์สโปรแกรมภาษา C ใหม่ให้เป็นรูปแบบที่สอดคล้องกัน ซึ่งใช้ระดับการย่อหน้าเพื่อแสดงโครงสร้างของโปรแกรม
cflow	สร้างไดอะแกรมของตรรกะการไหลของซอร์สโปรแกรมในภาษา C
cxref	สร้างรายการของการอ้างอิงภายนอกสำหรับแต่ละโมดูลของซอร์สโปรแกรมในภาษา C ซึ่งรวมถึงตำแหน่งที่การอ้างอิงถูกแก้ปัญหา (ถ้าเป็นการแก้ปัญหาในโปรแกรม)
lint	ตรวจสอบข้อผิดพลาดทางไวยากรณ์และชนิดข้อมูลในซอร์สโปรแกรมภาษา C คำสั่ง **lint** อาจตรวจสอบพื้นที่ของโปรแกรมเหล่านี้ มากกว่าที่คอมไพลเลอร์ในภาษา C ทำ และแสดงข้อความจำนวนมากที่ชี้ถึงปัญหาที่เป็นไปได้

การคอมไพล์และการลิงก์โปรแกรม

หากต้องการแปลงซอร์สโค้ดไปเป็นโปรแกรมที่ระบบสามารถรันได้ คุณต้องประมวลผลไฟล์ต้นฉบับด้วยคอมไพลเลอร์และ linkage editor

คอมไพลเลอร์ คือโปรแกรมที่อ่านข้อความจากไฟล์ และเปลี่ยนภาษาโปรแกรมในไฟล์ให้เป็นรูปแบบ ที่ระบบสามารถเข้าใจได้ *linkage editor* จะเชื่อมต่อโมดูลโปรแกรม และกำหนดวิธีวางโปรแกรมที่เสร็จสิ้นแล้วลงในหน่วยความจำ หากต้องการสร้างรูปแบบสุดท้ายของโปรแกรม ระบบต้องดำเนินการดังต่อไปนี้:

1. ถ้าไฟล์มีคอมไพลเลอร์ซอร์สโค้ด คอมไพลเลอร์จะแปลงซอร์สโค้ดไปเป็นอ็อบเจ็กต์โค้ด
2. ถ้าไฟล์มีภาษาแอสเซมเบลอร์ แอสเซมเบลอร์จะแปลไฟล์ไปเป็นอ็อบเจ็กต์โค้ด
3. linkage editor จะลิงก์อ็อบเจ็กต์ไฟล์ที่สร้างในขั้นตอนก่อนหน้านี้ ด้วยอ็อบเจ็กต์ไฟล์อื่นที่ระบุในคำสั่งคอมไพลเลอร์

ภาษาโปรแกรมอื่นๆ ที่มีสำหรับใช้บน ระบบปฏิบัติการได้แก่ C++, FORTRAN, COBOL และ Assembler และ ภาษาคอมไพลเลอร์อื่นๆ

คุณสามารถเขียนส่วนต่างๆ ของโปรแกรมในภาษาที่แตกต่างกัน และเรียกรูทีนหลักเพียงหนึ่งส่วน และเริ่มต้นรูทีนที่แยกจากกันเพื่อเรียกใช้งาน คุณยังสามารถใช้โปรแกรมคอมไพลเลอร์ เพื่อสร้างอ็อบเจ็กต์โค้ดและลิงก์โปรแกรม

การแก้ไขข้อผิดพลาดในโปรแกรม

คุณสามารถใช้ทูลการดีบักต่อไปนี้ ซึ่งได้จัดเตรียมไว้พร้อมกับระบบปฏิบัติการพื้นฐาน:

- โปรแกรมดีบักภาษาสัญลักษณ์ **dbx** อนุญาตให้คุณดีบักโปรแกรมที่เขียนด้วยภาษา C, C++, FORTRAN, COBOL และภาษาแอสเซมเบลอร์
- โปรแกรมดีบัก **adb** มีคำสั่งย่อยที่คุณสามารถใช้เพื่อตรวจสอบ ดีบัก และซ่อมแซมไบนารีไฟล์ที่สามารถเรียกทำงานได้ และเพื่อตรวจสอบไฟล์ข้อมูลที่ไม่ใช่ ASCII
- KDB Kernel Debugger และคำสั่ง kdb สามารถช่วยคุณพิจารณาข้อผิดพลาดในโค้ดที่ทำงานอยู่ในเคอร์เนล คุณสามารถใช้โปรแกรมดีบักนี้ เพื่อดีบักไดรเวอร์อุปกรณ์และส่วนขยายเคอร์เนล
- ตัวช่วย trace จะช่วยแยกปัญหาของระบบ โดยมอนิเตอร์เหตุการณ์ของระบบที่เลือกไว้

เมื่อข้อผิดพลาดทางไวยากรณ์หรือความขัดแย้งของการตั้งชื่อพารามิเตอร์ถูกค้นพบในไฟล์โปรแกรม คุณสามารถใช้เท็กซ์เอดิเตอร์หรือโปรแกรมค้นหาสตริงและแก้ไขสตริง เพื่อหาตำแหน่งและเปลี่ยนสตริงในไฟล์ โปรแกรมค้นหาสตริงหรือแก้ไขสตริงจะสอดแทรกคำสั่ง **grep**, **sed** และ **awk** หากต้องการทำการเปลี่ยนแปลงจำนวนมากในไฟล์โปรแกรมตั้งแต่หนึ่งไฟล์ขึ้นไป และคุณสามารถสอดแทรกคำสั่งในโปรแกรมเชลล์ จากนั้นรันโปรแกรมเชลล์ เพื่อหาตำแหน่งและเปลี่ยนโค้ดในไฟล์

การสร้างและการรักษาโปรแกรม

ตัวช่วยต่อไปนี้ช่วยคุณควบคุมการเปลี่ยนแปลงโปรแกรม และสร้างโปรแกรมจากซอร์สโมดูลจำนวนมาก ตัวช่วยเหล่านี้สามารถเป็นประโยชน์ในสภาวะแวดล้อมของการพัฒนาโปรแกรม ซึ่งซอร์สโมดูลจำนวนมากจะถูกสร้างขึ้น

- คำสั่ง **make** จะ build โปรแกรมจากซอร์สโมดูล เนื่องจากคำสั่ง **make** จะคอมไพล์เฉพาะโมดูลที่เปลี่ยนแปลงเหล่านั้น เนื่องจากการ build ครั้งล่าสุดที่ใช้ สามารถลดเวลาการคอมไพล์ เมื่อซอร์สโมดูลจำนวนมากต้องถูกประมวลผล
- Source Code Control System (SCCS) อนุญาตให้คุณคงเวอร์ชันของโปรแกรมที่แยกออกจากกัน โดยไม่เก็บแยกสำเนาที่สมบูรณ์ของแต่ละเวอร์ชัน การใช้ SCCS สามารถลดข้อกำหนดของหน่วยเก็บ และช่วยในการติดตามการพัฒนาของโปรเจ็กต์ ที่ต้องการก็บเวอร์ชันของโปรแกรมขนาดใหญ่หลายๆ เวอร์ชัน

รูทีนย่อย

รูทีนย่อยของไลบรารีระบบจะจัดการกับสถานการณ์โปรแกรมมิงที่ซับซ้อนหรือที่ทำซ้ำ ดังนั้นคุณจึงสามารถให้ความสนใจกับสถานการณ์โปรแกรมมิงโดยเฉพาะ

คำสั่ง Shell

คุณสามารถสอดแทรกฟังก์ชันของคำสั่ง shell ในโปรแกรมภาษา C คำสั่ง shell ใดๆ ที่ใช้ในโปรแกรมต้องพร้อมใช้งานบนระบบทั้งหมด ที่ใช้โปรแกรม

คุณยังสามารถใช้รูทีนย่อย **fork** และ **exec** ในโปรแกรม เพื่อรันคำสั่งตามกระบวนที่อยู่ในส่วนหนึ่งของระบบที่แยกออกจากโปรแกรม รูทีนย่อย **system** ยังรันคำสั่ง shell ในโปรแกรม และรูทีนย่อย **popen** จะใช้ตัวกรอง shell

หลักการที่เกี่ยวข้อง:

"การจัดการกับสตริงด้วย sed" ในหน้า 656
โปรแกรม **sed** ทำการแก้ไขโดยไม่มีการโต้ตอบ กับผู้ที่ร้องขอการแก้ไข
"การสร้างตัววิเคราะห์คำด้วยคำสั่ง lex" ในหน้า 560
คำสั่ง **lex** ช่วยในการเขียน โปรแกรมภาษา C ที่สามารถรับและแปลอินพุตสตรีมอักขระ เป็นการดำเนินการของโปรแกรม
"คำสั่ง make" ในหน้า 593
หัวข้อนี้ให้ข้อมูลเกี่ยวกับการททำการคอมไพล์ซ้ำ และการลิงก์กระบวนการใหม่ให้ง่ายขึ้นโดยใช้คำสั่ง **make**
"รูทีนย่อย ตัวอย่างโปรแกรม และไลบรารี" ในหน้า 788
หัวข้อนี้ให้ข้อมูลเกี่ยวกับสิ่งที่รูทีนย่อย เป็น วิธีใช้รูทีนย่อย และที่เก็บรูทีนย่อย

ไลบรารี Curses

ไลบรารี curses จัดเตรียมชุดของฟังก์ชันที่ช่วยให้คุณจัดการ การแสดงผลเทอร์มินัลไม่ว่าจะเป็นชนิดเทอร์มินัลใด ไลบรารี curses สนับสนุน สี อย่างไรก็ตาม ไม่สนับสนุนอักขระมัลติไบต์ การอ้างอิงทั้งหมดกับ อักขระในเอกสาร curses อ้างอิงกับอักขระไบต์เดี่ยว

ตลอด เอกสารนี้ ไลบรารี curses เรียกว่า *curses*

พื้นฐานของการโปรแกรมมิง curses คือโครงสร้างข้อมูลหน้าต่าง การใช้โครงสร้าง นี้ คุณสามารถจัดการข้อมูลบนจอแสดงผลของเทอร์มินัล คุณสามารถสั่ง curses ให้ปฏิบัติกับจอแสดงผลเทอร์มินัลเป็นหนึ่งหน้าต่างขนาดใหญ่หรือคุณสามารถ สร้างหลายหน้าต่างบนจอแสดงผล หน้าต่างมีได้หลายขนาด และซ้อนทับกันได้ แอ็พพลิเคชัน curses ปกติมีหนึ่งหน้าต่างใหญ่ และหนึ่งหน้าต่างย่อยอยู่ภายใน

แต่ละหน้าต่างบนจอแสดงผลเทอร์มินัลมีโครงสร้างข้อมูลหน้าต่างเป็นของตัวเอง โครงสร้างนี้เก็บข้อมูลภาวะเกี่ยวกับหน้าต่าง เช่นขนาด และตำแหน่งบนจอแสดงผล Curses ใช้โครงสร้างข้อมูลหน้าต่าง เพื่อรับข้อมูลที่เกี่ยวข้องจำเป็นต้องดำเนินการคำสั่งของคุณให้สำเร็จ

คำศัพท์

เมื่อ โปรแกรมมิงกับ curses คุณควรคุ้นเคยกับประโยคดังต่อไปนี้:

คำศัพท์	นิยาม
อักขระปัจจุบัน	อักขระที่เคอร์เซอร์โลจิคัลอยู่
บรรทัดปัจจุบัน	บรรทัดที่เคอร์เซอร์โลจิคัลอยู่
curscr	หน้าต่างค่าเริ่มต้นเสมือนที่จัดเตรียมโดย curses curscr (หน้าจอ ปัจจุบัน) คือการแสดงภายในของข้อมูลที่แสดงอยู่บนจอแสดงผลภายนอกของ เทอร์มินัลในขณะนี้ อย่าแก้ไข curscr
จอแสดงผล	จอแสดงผลฟิสิคัลที่เชื่อมต่อกับเวิร์กสเตชัน
โลจิคัลเคอร์เซอร์	ตำแหน่งเคอร์เซอร์ภายในแต่ละหน้าต่าง โครงสร้างข้อมูลหน้าต่าง ติดตามตำแหน่งของเคอร์เซอร์โลจิคัลของตัวเอง
pad	pad คือหน้าต่างที่ไม่ถูกจำกัดโดยขนาดของหน้าจอ
ฟิสิคัลเคอร์เซอร์	เคอร์เซอร์ที่แสดงบนจอแสดงผล เวิร์กสเตชันใช้เคอร์เซอร์นี้เพื่อเขียนไปที่ จอแสดงผล มีหนึ่งฟิสิคัลเคอร์เซอร์เท่านั้นต่อจอแสดงผล
หน้าจอ	หน้าต่างที่ใช้พื้นที่จอแสดงผลทั้งหมด หน้าจอมีความหมายตรงกับ stdscr
stdscr	หน้าต่างเริ่มต้นเสมือน (หน้าจอมาตรฐาน) ที่จัดเตรียมโดย curses ที่ แสดงจอแสดงผลทั้งหมด
หน้าต่าง	ตัวชี้ไปที่โครงสร้างข้อมูล C และการแสดงกราฟิกของ โครงสร้างข้อมูลนั้นบนจอแสดงผล หน้าต่างในแบบจินตนาการคืออาร์เรย์สองมิติ แสดงถึงลักษณะของจอแสดงผลบางส่วนหรือทั้งหมดจะเป็นอย่างไรในเวลาหนึ่ง

ระเบียบการตั้งชื่อ

รูทีนย่อย curses หนึ่งรูทีนมีได้มากกว่าหนึ่งเวอร์ชัน รูทีนย่อย Curses ที่มีหลายเวอร์ชันเป็นไปตามระเบียบการตั้งชื่อที่ไม่ซ้ำกัน ซึ่งระบุ แยกเวอร์ชัน ระเบียบเหล่านี้เพิ่มส่วนเติมหน้าให้กับรูทีนย่อย curses มาตรฐาน และระบุอาร์กิวเมนต์ที่รูทีนย่อยต้องการหรือการดำเนินการที่มี เมื่อรูทีนย่อยถูกเรียก เวอร์ชันต่างกันของชื่อรูทีนย่อย curses ใช้ส่วนเติมหน้าดังต่อไปนี้:

ส่วนเติมหน้า	Description
w	ระบุรูทีนย่อยที่ต้องการอาร์กิวเมนต์หน้าต่าง
p	ระบุรูทีนย่อยที่ต้องการอาร์กิวเมนต์ pad
mv	ระบุรูทีนย่อยที่ก่อนอื่นทำการย้ายไปที่พิกัดที่โปรแกรม ระบุ

ถ้ารูทีนย่อย curses มีหลายเวอร์ชันและไม่ได้รวมหนึ่งใน ส่วนเติมหน้าก่อนหน้านี้ stdscr หน้าต่างเริ่มต้นของ curses (หน้าจอมาตรฐาน) จะถูกใช้ รูทีนย่อยหลักที่ใช้ stdscr เป็นแมโครที่สร้าง ในไฟล์ **/usr/include/curses.h** โดยใช้คำสั่ง **#define** ตัวประมวลผลก่อนแทนที่คำสั่งเหล่านี้เวลาทำการ คอมไพล์ ดังนั้นแมโครเหล่านี้ไม่แสดงใน โค้ดแอสเซมเบลอร์ที่คอมไพล์แล้ว การติดตาม โปรแกรมดีบัก หรือซอร์สโค้ด curses

ถ้ารูทีนย่อย curses มีเพียงเวอร์ชันเดียว ไม่จำเป็นต้องใช้ stdscr ตัวอย่างเช่นรูทีนย่อย **printw** พิมพ์ สตริงไปที่ stdscr รูทีนย่อย **wprintw** พิมพ์ สตริงไปที่หน้าต่างโดยระบุอาร์กิวเมนต์ *window* รูทีนย่อย **mvprintw** ย้ายพิกัดที่ระบุ ไปที่ stdscr แล้วดำเนินการฟังก์ชันเดียวกับรูทีนย่อย **printw** เช่นเดียวกันรูทีนย่อย **mvwprintw** ย้ายพิกัดที่ระบุไปที่หน้าต่างที่ระบุแล้วดำเนินการฟังก์ชันเดียวกับรูทีนย่อย **wprintw**

โครงสร้างของโปรแกรม curses

โดยทั่วไปโปรแกรม curses มีความคืบหน้าในการทำงานดังนี้:

1. เริ่ม curses
2. ตรวจสอบการสนับสนุนสี (ทางเลือก)
3. เริ่มการทำงานของสี (ทางเลือก)
4. สร้างหน้าต่าง
5. จัดการหน้าต่าง
6. ทำลายหน้าต่าง
7. หยุดการทำงาน curses

บางขั้นตอนเป็นทางเลือก ดังนั้นโปรแกรมของคุณไม่จำเป็นต้องปฏิบัติตามการดำเนินการนี้ ทุกอย่าง

ค่าส่งคืน

มีข้อยกเว้นบางประการ รูทีนย่อย curses ทั้งหมดส่งกลับ ค่าจำนวนเต็ม ERR หรือค่าจำนวนเต็ม OK รูทีนย่อยที่ไม่ดำเนิน ตามระเบียบนี้จะถูกบันทึกอย่างเหมาะสม รูทีนย่อยที่ส่งกลับ ตัวชี้จะส่งกลับตัวชี้ null หรือข้อผิดพลาดเสมอ

การกำหนดค่าเริ่มต้น curses

ส่วนนี้อธิบายคำสั่งที่ใช้สำหรับการกำหนดค่าเริ่มต้น curses

ใช้คำสั่งต่อไปนี้เพื่อกำหนดค่าเริ่มต้น curses:

คำสั่ง	คำอธิบาย
endwin	ยกเลิกซับรูทีนไลบรารี curses และโครงสร้างข้อมูล
initscr	ให้ค่าเริ่มต้นซับรูทีนไลบรารี curses และโครงสร้างข้อมูล
isendwin	ส่งกลับค่า TRUE ถ้ารูทีนย่อย **endwin** ได้ถูกเรียกใช้โดยไม่มีการเรียกใช้ที่ตามมาไปยังรูทีนย่อย **wrefresh**
newterm	ตั้งค่าเทอร์มินัลใหม่
setupterm	ตั้งค่าโครงสร้าง TERMINAL สำหรับใช้โดย curses

คุณต้องรวมไฟล์ **curses.h** ที่ตอนต้น ของโปรแกรมใดๆ ที่เรียกใช้รูทีนย่อย curses ในการทำนี้ ให้ใช้คำสั่ง ต่อไปนี้:

```
#include <curses.h>
```

ก่อนที่คุณจะสามารถเรียกใช้รูทีนย่อยที่ดำเนินการหน้าต่าง หรือหน้าจอ คุณ ต้องเรียกใช้รูทีนย่อย **initscr** หรือ **newterm** รูทีนย่อยเหล่านี้ครั้งแรกบันทึกการตั้งค่าของเทอร์มินัลและจากนั้นเรียกใช้รูทีนย่อย **setupterm** เพื่อสร้างเทอร์มินัล curses

ถ้าคุณจำเป็นต้องหยุดทำงาน curses ชั่วคราว ใช้ shell escape หรือรูทีนย่อย ในการทำงานต่อหลังจากหลบออกชั่วคราว ให้เรียกใช้รูทีนย่อย **wrefresh** หรือ **doupdate** ก่อนออกจากโปรแกรม curses คุณ ต้องเรียกใช้รูทีนย่อย **endwin** รูทีนย่อย **endwin** เรียกคืนโหมด tty ย้ายเคอร์เซอร์ไปที่มุมล่างซ้าย ของหน้าจอ และตั้งค่าเทอร์มินัลใหม่ให้เป็นโหมดไม่แสดงให้เห็นที่เหมาะสม

โปรแกรมแบบเน้นหน้าจอที่มีการโต้ตอบมากที่สุดต้องการอินพุตครั้งละอักขระ โดยไม่มีการแสดงผลออกบนหน้าจอ ในการสร้างโปรแกรมของคุณ ที่มีอินพุตครั้งละอักขระ ให้เรียกใช้รูทีนย่อย **cbreak** และ **noecho** หลังจากเรียกใช้รูทีนย่อย **initscr** เมื่อยอมรับอินพุตประเภทนี้ โปรแกรมควร เรียกใช้รูทีนย่อยต่อไปนี้ด้วย:

- รูทีนย่อย **nonl**
- รูทีนย่อย **intrflush** ที่มีพารามิเตอร์ *Window* ถูกตั้งค่าเป็น **stdscr** และพารามิเตอร์ *Flag* ถูกตั้งค่าเป็น **FALSE** พารามิเตอร์ *Window* จำเป็นต้องใช้แต่ถูกข้ามไป คุณสามารถใช้ **stdscr** เป็นค่าของพารามิเตอร์ *Window* เนื่องจาก **stdscr** ได้ถูกสร้างขึ้นสำหรับคุณแล้ว
- รูทีนย่อย **keypad** ที่มีพารามิเตอร์ *Window* ถูกตั้งค่าเป็น **stdscr** และพารามิเตอร์ *Flag* ถูกตั้งค่าเป็น **TRUE**

รูทีนย่อย **isendwin** มีประโยชน์ในการทำให้เหมาะสมที่สุด ถ้าคุณไม่ต้องการเรียกใช้รูทีนย่อย **wrefresh** อย่างไม่จำเป็น ในการพิจารณาว่าถ้ารูทีนย่อย **endwin** ถูกเรียกใช้โดยไม่มีการเรียกใช้ที่ตามมาไปยังรูทีนย่อย **wrefresh** ให้ใช้รูทีนย่อย **isendwin**

Windows ในสภาวะแวดล้อม curses

โปรแกรม curses จะจัดการกับหน้าต่างที่ปรากฏบนจอแสดงผลของเทอร์มินัล หน้าต่าง สามารถขยายให้ใหญ่เท่ากับจอแสดงผลได้ หรือลดขนาดให้เล็กลงให้เหลือเท่าอักขระหนึ่งตัว ทั้งความยาวและความสูง

หมายเหตุ: ส่วนเสริมคือหน้าต่างที่ไม่จำกัดขนาดของหน้าจอ

ภายในโปรแกรม curses หน้าต่างคือตัวแปรที่ประกาศให้เป็นชนิด WINDOW ชนิดข้อมูล WINDOW จะถูกกำหนดอยู่ในไฟล์ **/usr/include/curses.h** ซึ่งเป็นโครงสร้างข้อมูล C คุณสร้างหน้าต่างได้โดยจัดสรรส่วนของหน่วยความจำของเครื่อง สำหรับโครงสร้างหน้าต่าง โครงสร้างนี้ อธิบายถึงคุณสมบัติของหน้าต่าง เมื่อโปรแกรมเปลี่ยนข้อมูลหน้าต่างภายในหน่วยความจำแล้ว โปรแกรมต้องใช้รูทีนย่อย **wrefresh** (หรือรูทีนที่เทียบเท่ากัน) เพื่ออัพเดตจอแสดงผลฟิสิคัลภายนอก เพื่อให้มีผลต่อการเปลี่ยนแปลงภายใน สำหรับโครงสร้างหน้าต่างที่เหมาะสม

โครงสร้างหน้าต่างดีฟอลต์

curses จัดเตรียมค่าดีฟอลต์โครงสร้างหน้าต่างเสมือนที่เรียกว่า *stdscr* stdscr จะแทนที่ในหน่วยความจำจอแสดงผลเทอร์มินัลทั้งหมด โครงสร้างหน้าต่าง stdscr จะถูกสร้างโดยอัตโนมัติ เมื่อไลบรารี curses ถูกกำหนดค่าเริ่มต้นและอธิบายถึงจอแสดงผล เมื่อไลบรารีถูกกำหนดค่าเริ่มต้นแล้ว ตัวแปร *length* และ *width* จะถูกตั้งค่าความกว้างและความยาวของจอแสดงผลแบบฟิสิคัล

โปรแกรมที่ใช้ stdscr ในครั้งแรกจะจัดการกับ stdscr จากนั้น จึงเรียกรูทีนย่อย **refresh** เพื่อรีเฟรชจอแสดงผลภายนอก ดังนั้นจึงตรงกับหน้าต่าง stdscr

นอกเหนือจาก stdscr แล้ว คุณสามารถกำหนดหน้าต่างของคุณเองได้ หน้าต่างเหล่านั้นจะรู้จักกันในนามของ *หน้าต่างที่ผู้ใช้กำหนดเอง* เพื่อแบ่งแยกหน้าต่างเหล่านั้นออกจาก stdscr เช่นเดียวกับ stdscr หน้าต่างที่ผู้ใช้กำหนดเองจะมีอยู่ในหน่วยความจำเครื่อง ซึ่งเป็นโครงสร้าง ยกเว้นสำหรับจำนวนหน่วยความจำที่พร้อมใช้งานกับโปรแกรม จะไม่มีข้อจำกัดเกี่ยวกับจำนวนของหน้าต่างที่คุณสามารถสร้างขึ้นได้ โปรแกรม curses สามารถจัดการกับดีฟอลต์ของหน้าต่าง หน้าต่างที่ผู้ใช้กำหนดเอง หรือทั้งสองแบบ

โครงสร้างหน้าต่าง Current

Curses สนับสนุนหน้าต่างเสมือนอื่นๆ ที่เรียกว่า *curscr* (หน้าจอปัจจุบัน) หน้าต่าง curscr คือการแทนค่าภายในของภาพปัจจุบัน ที่ปรากฏอยู่บนจอแสดงผลภายนอกของเทอร์มินัล

เมื่อโปรแกรมต้องการการแทนค่าภายนอกเพื่อจับคู่กับการแทนค่าภายใน โปรแกรมจะเรียกรูทีนย่อย เช่น รูทีนย่อย **wrefresh** เพื่ออัพเดตจอแสดงผลแบบฟิสิคัล (หรือรูทีนย่อย **refresh** ถ้าโปรแกรมกำลังทำงานด้วย stdscr)

curscr จะถูกสงวนไว้สำหรับใช้เป็นการภายในโดย curses และห้ามจัดการกับ curscr เป็นอันขาด

Subwindows

Curses ยังอนุญาตให้คุณสร้าง *หน้าต่างย่อย* ได้ หน้าต่างย่อยมีลักษณะเป็นสี่เหลี่ยมมุมฉากที่อยู่ภายในหน้าต่างอื่นๆ หน้าต่างย่อยย่อยคงเป็นชนิด WINDOW หน้าต่างที่มีหน้าต่างย่อยจะรู้จักกันในนามของหน้าต่างหลัก ของหน้าต่างย่อย และหน้าต่างย่อยจะรู้จักกันในนามของหน้าต่างรองของหน้าต่าง

การเปลี่ยนไปเป็นหน้าต่างหลัก หรือหน้าต่างรอง อย่างใดอย่างหนึ่งภายในพื้นที่ที่ซ้อนทับโดยหน้าต่างย่อย จะถูกทำทั้งสองหน้าต่าง หลังจากที่แก้ไขหน้าต่างย่อยแล้ว ให้เรียกรูทีนย่อย **touchline** หรือ **touchwin** บนหน้าต่างหลักก่อนที่จะรีเฟรชหน้าต่าง

รูทีนย่อย	คำอธิบาย
touchline	บังคับให้ช่วงของบรรทัดถูกรีเฟรชในครั้งถัดไปที่เรียกรูทีนย่อย **wrefresh**
touchwin	บังคับให้ทุกอักขระในอะเรย์อักขระของหน้าต่างถูกรีซอร์สในครั้งถัดไปที่เรียกรูทีนย่อย **wrefresh** รูทีนย่อย **touchwin** ไม่ได้บันทึกข้อมูลที่เหมาะสมที่สุด รูทีนย่อยนี้มีประโยชน์กับหน้าต่างที่ซ้อนทับกัน

refresh ที่เรียกบนหน้าต่างหลักยังคงรีเฟรชหน้าต่างรองด้วย หน้าต่างย่อย ยังคงสามารถเป็นหน้าต่างหลักได้ กระบวนการของการจัดชั้นของหน้าต่างภายในหน้าต่างจะเรียก *nesting*

ก่อนที่คุณสามารถลบหน้าต่างหลักได้ คุณต้องลบหน้าต่างรองทั้งหมดก่อนโดยใช้รูทีนย่อย **delwin** Curses จะส่งคืนข้อผิดพลาดหากคุณลองลบหน้าต่างก่อนที่จะลบหน้าต่างชายน์ทั้งหมด เป็นอันดับแรก

Pads

ส่วนเสริมคือชนิดของหน้าต่างที่ไม่ถูกจำกัดด้วยขนาดของจอแสดงผลของเทอร์มินัล หรือเชื่อมโยงกับส่วนของจอแสดงผลโดยเฉพาะ เนื่องจาก ส่วนเสริมมีขนาดใหญ่กว่าจอแสดงผลแบบฟิสิคัล เฉพาะส่วนของส่วนเสริมที่ผู้ใช้มองเห็นได้ในช่วงเวลาที่กำหนด

ใช้ส่วนเสริมหากคุณมีจำนวนที่มีขนาดใหญ่ของข้อมูลที่เกี่ยวข้อง ซึ่งคุณต้องการเก็บไว้ด้วยกันในหนึ่งหน้าต่าง แต่ไม่ต้องการแสดงข้อมูลทั้งหมดในครั้งเดียว

Windows ภายในส่วนเสริม เรียกว่า *ส่วนเสริมย่อย* ส่วนเสริมย่อย จะถูกจัดตำแหน่งอยู่ภายในส่วนเสริมที่ทำงานร่วมกับส่วนเสริมหลักที่สัมพันธ์กัน การจัดวางตำแหน่งนี้ ต่างจากหน้าต่างย่อย ซึ่งจะถูกจัดตำแหน่งโดยใช้หน้าจอที่ทำงานร่วมกัน

ไม่เหมือนกับหน้าต่างอื่นๆ การเลื่อนหรือการสะท้อนของอินพุตไม่ได้รีเฟรชส่วนเสริม แบบอัตโนมัติ แต่เหมือนกับหน้าต่างย่อยตรงที่ เมื่อเปลี่ยนอิมเมจของส่วนเสริมย่อยแล้ว คุณต้องเรียกรูทีนย่อย **touchline** หรือ **touchwin** บนส่วนเสริมหลักก่อนที่จะรีเฟรชหน้าต่างหลัก

คุณสามารถใช้รูทีนย่อย curses ทั้งหมดกับส่วนเสริม ยกเว้นรูทีนย่อย **newwin, subwin, wrefresh** และ **wnoutrefresh** รูทีนย่อยเหล่านี้จะถูกแทนที่ด้วยรูทีนย่อย **newpad, subpad, prefresh** และ **pnoutrefresh**

การจัดการกับข้อมูลหน้าต่างด้วย curses

เมื่อ curses ถูกเตรียมข้อมูลเบื้องต้น `stdscr` ถูก ระบุโดยอัตโนมัติ คุณสามารถจัดการ stdscr โดยใช้ไลบรารีรูทีนย่อย curses หรือคุณสามารถ สร้างหน้าต่างที่ผู้ใช้กำหนด

การสร้างหน้าต่าง

คุณสามารถสร้างหน้าต่างของคุณโดยใช้รูทีนย่อย **newwin**

แต่ละครั้งที่คุณเรียกรูทีนย่อย **newwin** curses จัดสรรโครงสร้างหน้าต่างใหม่ในหน่วยความจำ โครงสร้างนี้มี ข้อมูลทั้งหมดที่สัมพันธ์กับหน้าต่างใหม่ Curses ไม่กำหนดข้อจำกัด จำนวนของหน้าต่างที่คุณสามารถสร้างได้ จำนวนของ subwindows ที่ซ้อนกันถูกจำกัดด้วยจำนวนของหน่วยความจำที่มี มากได้ถึงค่าของ SHRT_MAX ตามที่กำหนดในไฟล์ **/usr/include/limits.h**

คุณสามารถเปลี่ยนหน้าต่างโดยไม่ต้องสนใจกับลำดับ ที่หน้าต่างถูกสร้าง การอัพเดตจอแสดงผลของเทอร์มินัลเกิดขึ้นผ่านการเรียกรูทีนย่อย **wrefresh**

หน้าต่างย่อย

คุณต้องกำหนดพิกัดสำหรับ subwindow ที่สัมพันธ์ กับจอแสดงผลของเทอร์มินัล subwindow ที่สร้างโดยใช้รูทีนย่อย **subwin** ต้องพอดีกับภายในขอบเขตของ หน้าต่างพาเรนต์ มิฉะนั้นค่า null จะถูกส่งกลับ

ส่วนเสริม

ใช้รูทีนย่อยดังต่อไปนี้เพื่อสร้าง pads:

รูทีนย่อย	คำอธิบาย
newpad	สร้างโครงสร้างข้อมูล pad
subpad	สร้างและส่งกลับตัวชี้ไปที่ subpad ภายใน pad

การลบหน้าต่าง, pads และหน้าต่างย่อย

subpad ใหม่ถูกกำหนดตำแหน่งสัมพันธ์กับพาเรนต์

เมื่อต้องการลบหน้าต่าง pad หรือ subwindow ให้ใช้รูทีนย่อย **delwin** ก่อนที่คุณจะสามารถลบหน้าต่างหรือ pad คุณต้องลบหน้าต่างหรือ pad ย่อยแล้ว มิฉะนั้นรูทีนย่อย **delwin** จะส่งกลับข้อผิดพลาด

การเปลี่ยนรูปหน้าจอหรือหน้าต่าง

เมื่อรูทีนย่อย curses เปลี่ยน ลักษณะที่ปรากฏของหน้าต่าง การแสดงผลภายในของหน้าต่างถูกอัพเดต ขณะที่จอแสดงผลยังคงไม่เปลี่ยนแปลง จนกว่าการเรียกรูทีนย่อย **wrefresh** ครั้งต่อไป รูทีนย่อย **wrefresh** ใช้ ข้อมูลในโครงสร้างหน้าต่างเพื่ออัพเดตจอแสดงผล

การรีเฟรชหน้าต่าง

เมื่อไรก็ตามที่คุณสร้างเอาต์พุตไปที่โครงสร้างหน้าต่าง หรือ pad คุณต้องรีเฟรช จอแสดงผลของเทอร์มินัลให้ตรงกับการแสดงผลภายใน การรีเฟรชทำ ดังต่อไปนี้:

- เปรียบเทียบเนื้อหาของ curscr กับเนื้อหาที่ผู้ใช้กำหนด หรือ stdscr
- อัพเดตโครงสร้าง curscr เพื่อให้ตรงกับที่ผู้ใช้กำหนดหรือ stdscr
- แสดงส่วนของการแสดงผลฟิสิคัลที่เปลี่ยนแปลงใหม่

ใช้รูทีนย่อยดังต่อไปนี้เพื่อรีเฟรชหน้าต่าง:

รูทีนย่อย	คำอธิบาย
refresh หรือ **wrefresh**	อัพเดตเทอร์มินัลและ curscr เพื่อสะท้อนการเปลี่ยนแปลงที่ทำกับหน้าต่าง
wnoutrefresh หรือ **doupdate**	อัพเดตหน้าต่างที่กำหนดและเอาต์พุตข้อมูลทั้งหมดครั้งเดียว ไปที่เทอร์มินัล รูทีนย่อยเหล่านี้มีประโยชน์สำหรับการตอบสนองที่เร็วกว่าเมื่อมี หลายการอัพเดต

รูทีนย่อย **refresh** และ **wrefresh** เรียกรูทีนย่อย **wnoutrefresh** ในครั้งแรก เพื่อคัดลอกหน้าต้างที่กำลังถูกรีเฟรชให้เป็นหน้าจอปัจจุบัน จากนั้นจึงเรียกรูทีนย่อย **doupdate** เพื่ออัพเดต จอแสดงผล

ถ้าคุณจำเป็นต้องรีเฟรชหลายหน้าต่างพร้อมกัน ให้ใช้หนึ่งในสองวิธีที่มี คุณสามารถใช้ชุดการเรียกรูทีนย่อย **wrefresh** ที่ให้ผลสลับการเรียก ไปที่รูทีนย่อย **wnoutrefresh** และ **doupdate** คุณยังสามารถเรียกรูทีนย่อย **wnoutrefresh** หนึ่ง ครั้งสำหรับแต่ละหน้าต่างแล้วเรียกรูทีนย่อย **doupdate** หนึ่งครั้ง ด้วยวิธีที่สอง เฉพาะหนึ่งการส่งเป็นชุดอย่างเร็วของเอาต์พุตถูกส่งไปที่จอแสดงผล

รูทีนย่อยที่ใช้สำหรับการรีเฟรช pads

รูทีนย่อย **prefresh** และ **pnoutrefresh** เหมือนกับรูทีนย่อย **wrefresh** และ **wnoutrefresh**

รูทีนย่อย **prefresh** อัพเดตทั้งหน้าจอปัจจุบัน และจอแสดงผลฟิสิคัล ขณะที่รูทีนย่อย **pnoutrefresh** อัพเดต curscr เพื่อสะท้อนการเปลี่ยนแปลงที่ทำกับ pad ที่ผู้ใช้กำหนด เนื่องจาก pads ไม่ใช่หน้าต่างที่เกี่ยวข้อง รูทีนย่อยเหล่านี้ ต้องการพารามิเตอร์เพิ่มเติมเพื่อระบุส่วนของ pad และหน้าจอ ที่เกี่ยวข้อง

การรีเฟรชพื้นที่ที่ไม่ได้เปลี่ยนแปลง

ระหว่างการรีเฟรช เฉพาะพื้นที่ที่มีการเปลี่ยนแปลง จะถูกเขียนใหม่บนจอแสดงผล คุณสามารถรีเฟรชพื้นที่ของจอแสดงผลที่ไม่เปลี่ยนแปลง โดยใช้รูทีนย่อย **touchwin** และ **touchline**:

รูทีนย่อย	คำอธิบาย
touchline	บังคับให้ช่วงของบรรทัดถูกรีเฟรชในครั้งถัดไปที่เรียกรูทีนย่อย **wrefresh**
touchwin	บังคับให้ทุกอักขระในอาร์เรย์อักขระของหน้าต่างถูกรีซอร์ส ในครั้งถัดไปที่เรียกรูทีนย่อย **wrefresh** รูทีนย่อย **touchwin** ไม่ได้บันทึกข้อมูลที่เหมาะสมที่สุด รูทีนย่อยนี้มีประโยชน์กับหน้าต่างที่ซ้อนทับกัน

การรวมรูทีนย่อย **touchwin** และ **wrefresh** มีประโยชน์เมื่อทำงานกับ subwindows หรือหน้าต่างที่ซ้อนกัน ในการนำหน้าต่างมาด้านหน้าจากด้านหลังของหน้าต่างอื่น ให้เรียกรูทีนย่อย **touchwin** ตามด้วยรูทีนย่อย **wrefresh**

การแสดงผลที่บิดเบือน

ถ้าข้อความถูกส่งไปที่จอแสดงผลของเทอร์มินัลด้วยรูทีนย่อย ที่ไม่ใช่ curses เช่นรูทีนย่อย **echo** หรือ **printf** หน้าต่างภายนอกอาจแสดงผลบิดเบือนได้ ในกรณีนี้ จอแสดงผลมีการเปลี่ยนแปลง แต่หน้าจอปัจจุบันไม่ถูกอัพเดตเพื่อสะท้อนการเปลี่ยนแปลงเหล่านี้ ปัญหาสามารถเกิดขึ้นเมื่อการรีเฟรชถูกเรียกบนจอแสดงผลที่บิดเบือน เนื่องจาก หลังจากหน้าจอถูกบิดเบือน ไม่มีความแตกต่างระหว่างหน้าต่างที่ถูกรีเฟรช และโครงสร้างหน้าจอปัจจุบัน ด้วยผลดังกล่าว ช่องว่างบนจอแสดงผล เกิดจากข้อความที่บิดเบือนไม่ถูกเปลี่ยนแปลง

ปัญหาเดียวกันนี้อาจเกิดได้เช่นกันเมื่อหน้าต่าง ถูกย้าย อักขระที่ส่งไปที่จอแสดงผลด้วยรูทีนย่อยที่ไม่ใช่ curses ไม่ ย้ายไปกับหน้าต่างภายใน

ถ้าหน้าจอกลายเป็นภาพบิดเบือน ให้เรียกรูทีนย่อย **wrefresh** บน curscr เพื่ออัพเดตการแสดงผลให้สะท้อนการแสดงผลฟิสิคัลปัจจุบัน

การจัดการกับเนื้อหาของหน้าต่าง

หลังจากหน้าต่างหรือ subwindow ถูกสร้าง บ่อยครั้งที่โปรแกรมต้องจัดการในบางวิธี โดยใช้รูทีนย่อยดังต่อไปนี้:

รูทีนย่อย	คำอธิบาย
box	วาดสี่เหลี่ยมในหรือรอบหน้าต่าง
copywin	จัดเตรียมการควบคุมที่แม่นยำกว่ากับรูทีนย่อย **overlay** และ **overwrite**
garbagedlines	ระบุแก่ curses ว่าบรรทัดหน้าจอถูกละเวนและควรถูก เอาออกไป ก่อนจะมีการเขียนข้อมูลที่ส่วนนั้น
mvwin	ย้าย หน้าต่างหรือ subwindow ไปที่ตำแหน่งใหม่
overlay หรือ **overwrite**	คัดลอกหน้าต่างหนึ่งเหนือหน้าต่างอื่น
ripoffline	เอาบรรทัดออกจากหน้าจอเริ่มต้น

เมื่อต้องการใช้รูทีนย่อย **overlay** และ **overwrite** สองหน้าต่างต้องซ้อนกัน รูทีนย่อย **overwrite** เป็นการทำลาย ขณะที่รูทีนย่อย **overlay** ไม่เป็น เมื่อข้อความถูกคัดลอกจากหนึ่งหน้าต่างไปหน้าต่างอื่นโดยใช้ รูทีนย่อย **overwrite** ส่วนที่ว่างจากหน้าต่างที่คัดลอก ทับส่วนของหน้าต่างที่คัดลอกข้อมูลมา รูทีนย่อย **overlay** ไม่มีการทำลาย เนื่องจากไม่ได้คัดลอกส่วนที่ว่าง จากหน้าต่างที่คัดลอก

เหมือนกับรูทีนย่อย **overlay** และ **overwrite** รูทีนย่อย **copywin** อนุญาต ให้คุณคัดลอกส่วนของหน้าต่างหนึ่งไปยังอีกหน้าต่างหนึ่ง ไม่เหมือนกับรูทีนย่อย **overlay** และ **overwrite** หน้าต่าง ไม่จำเป็นต้องซ้อนกัน เพื่อคุณจะใช้รูทีนย่อย **copywin**

เมื่อต้องการเอาบรรทัดออกจาก stdscr คุณสามารถใช้รูทีนย่อย **ripoffline** ถ้าคุณผ่านอาร์กิวเมนต์ *line* ที่มีค่าบวกให้กับรูทีนย่อยนี้ จำนวนบรรทัดที่ระบุถูกเอาออก จากด้านบนของ stdscr ถ้าคุณผ่านอาร์กิวเมนต์ *line* ที่มีค่าลบให้กับรูทีนย่อย บรรทัดจะถูกเอาออกจากด้านล่างของ stdscr

เมื่อต้องการละเว้นช่วงที่ระบุของบรรทัดก่อนการเขียน ข้อมูลใหม่ คุณสามารถใช้รูทีนย่อย **garbagedlines**

การสนับสนุนสำหรับตัวกรอง

รูทีนย่อย **filter** ถูกจัดเตรียมสำหรับแอ็พพลิเคชัน curses ที่เป็นตัวกรอง รูทีนย่อยนี้ทำให้ curses ทำงานเหมือนกับว่า stdscr เป็นบรรทัดเดียว เท่านั้น เมื่อรันกับรูทีนย่อย **filter**, curses ไม่ใช้ความสามารถของเทอร์มินัลที่ต้องการการรับรู้ของบรรทัด ว่า curses ทำงานอยู่

การควบคุมเคอร์เซอร์ด้วย curses

ส่วนนี้อธิบายถึงเคอร์เซอร์ชนิดอื่นที่มีอยู่ใน ไลบรารี curses

ชนิดของเคอร์เซอร์ดังต่อไปนี้มีอยู่ในไลบรารีเคอร์เซอร์:

โลจิคัลเคอร์เซอร์
: ตำแหน่งเคอร์เซอร์ภายในแต่ละหน้าต่าง โครงสร้างข้อมูลของหน้าต่าง ติดตามตำแหน่งของโลจิคัลเคอร์เซอร์ของตัวเอง แต่ละหน้ามีมีโลจิคัล เคอร์เซอร์

ฟิสิคัล เคอร์เซอร์
: เคอร์เซอร์ของจอแสดงผล เวิร์กสเตชันใช้เคอร์เซอร์นี้เพื่อเขียนไปที่ จอแสดงผล มีหนึ่งฟิสิคัลเคอร์เซอร์เท่านั้นต่อจอแสดงผล

คุณทำได้เพียงเพิ่มหรือลบอักขระที่โลจิคัลเคอร์เซอร์ ในหน้าต่าง รูทีนย่อยดังต่อไปนี้ถูกจัดเตรียมสำหรับการควบคุม เคอร์เซอร์:

`getbegyx`
: กำหนดพิกัดเริ่มต้นของหน้าต่าง ในตัวแปรจำนวนเต็ม *y* และ *x*

`getmaxyx`
กำหนดขนาดของหน้าต่าง ในตัวแปรจำนวนเต็ม *y* และ *x*

`getsyx`
ส่งกลับพิกัดปัจจุบันของเคอร์เซอร์หน้าจอเสมือน

`getyx`
ส่งกลับตำแหน่งของโลจิคัลเคอร์เซอร์ที่สัมพันธ์กับหน้าต่าง ที่ระบุ

`leaveok`
ควบคุมการวางตำแหน่งฟิสิคัลเคอร์เซอร์หลังการเรียกรูทีนย่อย **wrefresh**

`move`
เลื่อนโลจิคัลเคอร์เซอร์ที่สัมพันธ์กับ stdscr

`mvcur`
ย้ายฟิสิคัลเคอร์เซอร์

`setsyx`
ตั้งค่าเคอร์เซอร์หน้าจอเสมือนเป็นพิกัดที่ระบุ

`wmove`
เลื่อนเคอร์เซอร์โลจิคัลที่สัมพันธ์กับหน้าต่างที่ ผู้ใช้กำหนด

หลังจากการเรียกรูทีนย่อย **refresh** หรือ **wrefresh,** curses วางตำแหน่งฟิสิคัลเคอร์เซอร์ที่ ตำแหน่งอักขระที่อัพเดตล่าสุดในหน้าต่าง เมื่อต้องการคงตำแหน่งฟิสิคัลเคอร์เซอร์ ไว้และไม่มีการย้ายหลังการรีเฟรช เรียกรูทีนย่อย **leaveok** ด้วยพารามิเตอร์ *Window* กำหนดเป็น หน้าต่างที่ต้องการและพารามิเตอร์ *Flag* ตั้งค่าเป็น **TRUE**

การจัดการกับอักขระด้วย curses

คุณสามารถเพิ่มอักขระในหน้าต่าง curses โดยใช้ คีย์บอร์ด หรือแอ็พพลิเคชัน curses ส่วนนี้อธิบายวิธีที่คุณสามารถเพิ่ม, ลบออก หรือเปลี่ยนอักขระที่ปรากฏในหน้าต่าง curses

ขนาดอักขระ

ชุดอักขระบางชุดกำหนดอักขระหลายคอลัมน์ที่ครอบครองตำแหน่ง มากกว่าหนึ่งคอลัมน์เมื่อแสดงบนหน้าจอ

การเขียนอักขระที่มีความกว้างมากกว่าความกว้างของหน้าต่างปลายทาง ก่อให้เกิดข้อผิดพลาด

การเพิ่มอักขระในรูปภาพหน้าจอ

ไลบรารี curses จัดให้มีรูทีนย่อยจำนวนหนึ่ง ที่บันทึกการเปลี่ยนแปลงข้อความไปที่หน้าต่าง และทำเครื่องหมายพื้นที่ที่จะถูกอัพเดต ในการเรียกใช้ครั้งถัดไปในรูทีนย่อย **wrefresh**

รูทีนย่อย waddch

รูทีนย่อย **waddch** เขียนทับอักขระที่ตำแหน่งเคอร์เซอร์โลจิคัลปัจจุบัน ด้วยอักขระที่ระบุ หลังจากเขียนทับ เคอร์เซอร์โลจิคัลถูกเลื่อนหนึ่งช่องว่างไปทางขวา ถ้ารูทีนย่อย **waddch** ถูกเรียกใช้ที่ขอบขวา รูทีนย่อยเหล่านี้ยังเพิ่มอักขระขึ้นบรรทัดใหม่

อัตโนมัติ นอกจากนั้น ถ้าคุณเรียกใช้หนึ่งในรูทีนย่อยเหล่านี้ที่ด้านล่าง ของแถบการเลื่อนและรูทีนย่อย **scrollok** ถูกเปิดใช้งาน แถบถูกเลื่อนขึ้นหนึ่งบรรทัด ตัวอย่าง ถ้าคุณเพิ่มบรรทัดใหม่ ที่ด้านล่างของหน้าต่าง หน้าต่างควรต้องเลื่อนขึ้นหนึ่งบรรทัด

ถ้าอักขระที่จะเพิ่มเป็นอักขระแท็บ ขึ้นบรรทัดใหม่ หรือถอยกลับ curses จะย้ายเคอร์เซอร์อย่างเหมาะสมในหน้าต่างเพื่อให้แสดง การเพิ่มนั้น แท็บถูกตั้งค่าที่ทุกคอลัมน์ที่แปด ถ้าอักขระเป็นขึ้นบรรทัดใหม่ อันดับแรก curses ใช้รูทีนย่อย **wclrtoeol** เพื่อลบ บรรทัดปัจจุบันออกจากตำแหน่งเคอร์เซอร์โลจิคัลไปที่สิ้นสุดของบรรทัดก่อน ย้ายเคอร์เซอร์ กลุ่มรูทีนย่อย **waddch** ประกอบด้วย ต่อไปนี้:

รูทีนย่อย	คำอธิบาย
แมโคร **addch**	เพิ่มอักขระไปที่ stdscr
แมโคร **mvaddch**	ย้ายอักขระไปที่ตำแหน่งที่ระบุก่อนเพิ่มไปที่ stdscr
แมโคร **mvwaddch**	ย้ายอักขระไปที่ตำแหน่งที่ระบุก่อนเพิ่มไปที่ หน้าต่างที่ผู้ใช้กำหนดเอง
รูทีนย่อย **waddch**	เพิ่มเพิ่มไปที่หน้าต่างที่ผู้ใช้กำหนดเอง

โดยการใช้กลุ่มรูทีนย่อย **winch** และ **waddch** ร่วมกัน คุณสามารถทำสำเนาคุณสมบัติข้อความและวิดีโอ จากที่หนึ่งไปอีกที่หนึ่ง การใช้กลุ่มรูทีนย่อย **winch** คุณสามารถเรียกข้อมูลอักขระและแอ็ตทริบิวต์วิดีโอ จากนั้นคุณสามารถ ใช้หนึ่งในรูทีนย่อย **waddch** เพื่อเพิ่มอักขระ และแอ็ตทริบิวต์ของอักขระไปยังตำแหน่งอื่น

คุณยังสามารถใช้รูทีนย่อย **waddch** เพื่อเพิ่มอักขระควบคุมในหน้าต่าง อักขระควบคุมถูกเขียนใน เครื่องหมาย ^X

หมายเหตุ: การเรียกใช้รูทีนย่อย **winch** บนตำแหน่งหนึ่ง ในหน้าต่างที่มีอักขระควบคุมจะไม่ส่งกลับค่าอักขระ แต่ส่งกลับอักขระหนึ่งของการแทนอักขระควบคุมแทน

การเอาต์พุตอักขระที่ไม่ใช่อักขระควบคุมเดี่ยว

เมื่อทำการเอาต์พุตอักขระที่ไม่ใช่ตัวควบคุมเดี่ยว จะ เกิดผลการทำงานอย่างมีนัยสำคัญในการใช้รูทีนย่อย **wechochar** รูทีนย่อยเหล่านี้มีการทำงานเทียบเท่ากับการเรียกใช้ไปยัง รูทีนย่อย **waddchr** ที่สอดคล้องกันตามด้วย รูทีนย่อย **wrefresh** ที่สอดคล้อง รูทีนย่อย **wechochar** ประกอบด้วยรูทีนย่อย **wechochar** แมโคร **echochar** และรูทีนย่อย **pechochar**

บางชุดอักขระอาจมีอักขระที่ไม่มีระยะห่างบางตัว (อักขระที่ไม่มีระยะห่าง ได้แก่ อักขระที่นอกเหนือจาก '\0' ซึ่งรูทีนย่อย **wcwidth** ส่งกลับค่าความยาวศูนย์) แอ็พพลิเคชันอาจเขียน อักขระที่ไม่มีระยะห่างไปยังหน้าต่าง ทุกๆ อักขระที่ไม่มีระยะห่างในหน้าต่าง จะสัมพันธ์กับอักขระที่มีระยะห่าง และแก้ไขอักขระที่มีระยะห่าง อักขระที่ไม่มีระยะห่างในหน้าต่างไม่สามารถระบุแยกต่างหาก อักขระ ที่ไม่มีระยะห่างถูกระบุโดยนัยเมื่อการดำเนินการ Curses มีผลต่อ อักขระที่มีระยะห่างที่ซึ่งอักขระที่ไม่มีระยะห่างเชื่อมโยงอยู่

อักขระที่ไม่มีระยะห่างไม่สนับสนุนแอ็ตทริบิวต์ สำหรับอินเตอร์เฟสที่ใช้ อักขระตัวแทนและแอ็ตทริบิวต์ แอ็ตทริบิวต์จะถูกละเว้นถ้าอักขระตัวแทนเป็น อักขระที่ไม่มีระยะห่าง อักขระหลายคอลัมน์มีชุดของแอ็ตทริบิวต์เดียว สำหรับทุกคอลัมน์ การเชื่อมโยงของอักขระที่ไม่มีระยะห่างกับอักขระที่มีระยะห่าง ควบคุมโดยแอ็พพลิเคชันที่ใช้อินเตอร์เฟสอักขระตัวแทน ฟังก์ชันสตริงอักขระตัวแทนมีการเชื่อมโยงที่ไม่ขึ้นกับชุดโค้ด

ผลที่เกิดโดยทั่วไปของอักขระที่ไม่มีระยะห่างที่เชื่อมโยงกับอักขระที่มีระยะห่าง เรียกว่า *c* ดังนี้:

- อักขระที่ไม่มีระยะห่างอาจแก้ไขลักษณะที่ปรากฏของ *c* (ตัวอย่างเช่น อาจมีอักขระที่ไม่มีระยะห่างที่เพิ่มเครื่องหมาย diacritical ให้กับอักขระ อย่างไรก็ตาม ก็อาจมีอักขระที่มีระยะห่างที่มีเครื่องหมาย diacritical อยู่ในตัวเช่นกัน)
- อักขระที่ไม่มีระยะห่างอาจเชื่อม *c* ไปยัง อักขระที่ตามหลัง *c* ตัวอย่างของการใช้งานนี้คือ การสร้างการโยงกัน และการแปลงอักขระไปเป็นรูปแบบการแสดง ที่ซับซ้อน คำ หรือสัญลักษณ์แทนความหมาย

การนำไปปฏิบัติอาจจำกัดจำนวนอักขระที่ไม่มีระยะห่างที่สามารถ เชื่อมโยงกับอักขระที่มีระยะห่าง โดยมีขีดจำกัดอย่างน้อย 5

อักขระที่ซับซ้อน

อักขระซับซ้อนคือชื่อของอักขระที่เชื่อมโยง ซึ่งอาจรวม อักขระที่มีระยะห่างและยังอาจรวมอักขระที่ไม่มีระยะห่างที่เชื่อมโยงด้วย อักขระซับซ้อนที่มีระยะห่างคืออักขระซับซ้อนที่รวม อักขระที่มีระยะห่างหนึ่งช่อง และอักขระที่ไม่มีระยะห่างใดๆ ที่เชื่อมโยง ตัวอย่าง ของชุดโค้ดที่มีอักขระซับซ้อนได้แก่ ISO/IEC 10646-1:1993

อักขระซับซ้อนสามารถเขียนไปยังหน้าจอ ถ้าอักขระซับซ้อน ไม่ได้รวมอักขระที่มีระยะห่าง อักขระที่ไม่มีระยะห่างใดๆ จะถูกเชื่อมโยง กับอักขระซับซ้อนที่มีระยะห่างที่มีอยู่ที่ตำแหน่งหน้าจอที่ระบุ เมื่อแอ็พพลิเคชันอ่านข้อมูลจากหน้าจอ แอ็พพลิเคชันจะได้อักขระซับซ้อน ที่มีระยะห่าง

ชนิดข้อมูล cchar_t แทนอักขระซับซ้อนและการแปลความหมาย เมื่อ cchar_t แทนอักขระซับซ้อนที่ไม่มีระยะห่าง (ซึ่งคือ เมื่อไม่มี อักขระที่มีระยะห่างภายในอักขระซับซ้อน) จะไม่ใช้ การแปลความหมาย เมื่อเขียนไปยังหน้าจอ จะใช้การแปลความหมายที่ระบุ โดยอักขระที่มีระยะห่างที่แสดงอยู่แล้ว

อ็อบเจ็กต์ของชนิด cchar_t สามารถเตรียมข้อมูลเบื้องต้นโดยใช้รูทีนย่อย **setchar** และเนื้อหาสามารถถูกแยกออกโดยใช้รูทีนย่อย **getchar** ลักษณะการทำงานของฟังก์ชันที่รับค่า cchar_t ที่ยังไม่ได้เตรียมข้อมูลเบื้องต้นด้วยวิธีนี้ จะได้รับจากฟังก์ชัน curses ที่ มีอาร์กิวเมนต์เอาต์พุต cchar_t

อักขระพิเศษ

บางฟังก์ชันจะประมวลผลอักขระพิเศษ ในฟังก์ชันที่ไม่ย้ายเคอร์เซอร์ ตามข้อมูลที่วางบนหน้าต่าง อักขระพิเศษเหล่านี้ จะถูกใช้ภายในสตริงเท่านั้นเพื่อให้มีผลต่อการวางอักขระ ที่ตามมา การย้ายเคอร์เซอร์ที่ระบุด้านล่างไม่มีอยู่ในเคอร์เซอร์ที่ เห็นได้หลังสิ้นสุดการดำเนินการ ในฟังก์ชันที่ไม่มีการย้าย เคอร์เซอร์ อักขระพิเศษเหล่านี้สามารถใช้เพื่อให้มีผลต่อการวางอักขระที่ตามมา และการย้ายเคอร์เซอร์จริง

อักขระ	คำอธิบาย
อักขระถอยกลับ	ยกเว้นว่าเคอร์เซอร์อยู่ที่คอลัมน์ 0 อยู่แล้ว Backspace จะย้ายเคอร์เซอร์ หนึ่งคอลัมน์ไปทางทิศเริ่มต้นของบรรทัดปัจจุบัน และอักขระใดๆ ที่อยู่หลัง Backspace จะถูกเพิ่มหรือแทรกโดยเริ่มต้นที่จุดนั้น
ปัดแคร่ (Carriage return)	ยกเว้นเคอร์เซอร์อยู่ที่คอลัมน์ 0 แล้ว Carriage return ย้าย เคอร์เซอร์ไปที่จุดเริ่มต้นของบรรทัดปัจจุบัน อักขระใดๆ ที่อยู่หลัง Carriage return จะถูกเพิ่มหรือแทรกโดยเริ่มต้นที่จุดนั้น
ขึ้นบรรทัดใหม่ (newline)	ในการดำเนินการเพิ่ม curses เพิ่มอักขระเบื้องหลังลงในคอลัมน์ที่ตามมา จนกระทั่งถึงจุดสิ้นสุดบรรทัด การเลื่อนเกิดขึ้น และอักขระใดๆ หลังอักขระขึ้นบรรทัดใหม่จะถูกเพิ่ม โดยเริ่มที่จุดเริ่มต้นของบรรทัด ใหม่ ในการดำเนินการแทรก อักขระขึ้นบรรทัดใหม่จะลบส่วนที่เหลือของบรรทัดปัจจุบัน ด้วยอักขระเบื้องหลัง (โดยใช้รูทีนย่อย **wclrtoeol** ที่มีประสิทธิภาพ) และย้ายเคอร์เซอร์ไปที่จุดเริ่มของบรรทัดใหม่ เมื่อ การเลื่อนถูกเปิดใช้งาน การเคลื่อนเคอร์เซอร์ไปข้างหน้าที่บรรทัดใหม่อาจทำให้เกิดการเลื่อน อักขระใดๆ หลังอักขระขึ้นบรรทัดใหม่จะถูกแทรกที่จุดเริ่มต้น ของบรรทัดใหม่
แท็บ (Tab)	ฟังก์ชัน **filter** อาจขัดขวางการประมวลผลนี้ อักขระแท็บในข้อความจะย้ายอักขระที่ตามมาไปที่ตำแหน่งแท็บหยุด ถัดไป โดยดีฟอลต์ แท็บหยุดจะอยู่ที่คอลัมน์ที่ 0, 8, 16 ตามลำดับ ใน การดำเนินการแทรกหรือเพิ่ม curses จะแทรกหรือเพิ่มอักขระเบื้องหลังตามลำดับ ลงในคอลัมน์ที่ตามมาจนกระทั่งถึงแท็บหยุดถัดไป ถ้าไม่มี แท็บหยุดอีกต่อไปในบรรทัดปัจจุบัน จะเกิดการขึ้นต้นใหม่และการเลื่อน

อักขระควบคุม

ฟังก์ชัน curses ที่ทำหน้าที่ในการประมวลผลอักขระพิเศษโดยหลักการจะ แปลงอักขระควบคุมเป็นอักขระ ('^') ตามด้วย อักขระที่สอง (ซึ่งเป็นตัวอักษรตัวพิมพ์ใหญ่ถ้าเป็นตัวอักษร) และเขียนสตริงนี้ ไปยังหน้าต่างแทนอักขระควบคุม ฟังก์ชันที่ เรียกข้อความจากหน้าต่างจะไม่เรียกอักขระควบคุมต้นฉบับ

ลายเส้น:

คุณสามารถใช้ตัวแปรต่อไปนี้เพื่อเพิ่มอักขระ ลายเส้นบนหน้าจอด้วยรูทีนย่อย **waddch** เมื่อกำหนดสำหรับเทอร์มินัล ตัวแปร จะมีบิต **A_ALTCHARSET** ถูกเปิดใช้ มิฉะนั้น อักขระดีฟอลต์ที่แสดงใน ตารางต่อไปนี้จะถูกเก็บในตัวแปร

ชื่อตัวแปร	อักขระดีฟอลต์	รายละเอียด Glyph
ACS_ULCORNER	+	มุมซ้ายบน
ACS_LLCORNER	+	มุมซ้ายล่าง
ACS_URCORNER	+	มุมขวาบน
ACS_LRCORNER	+	มุมขวาล่าง
ACS_RTEE	+	tee ขวา
ACS_LTEE	+	tee ซ้าย
ACS_BTEE	+	tee ล่าง
ACS_TTEE	+	tee บน
ACS_HLINE	—	เส้นแนวนอน
ACS_VLINE	|	เส้นแนวดิ่ง
ACS_PLUS	+	บวก
ACS_S1	-	เส้นสแกน 1
ACS_S9	_	เส้นสแกน 9
ACS_DIAMOND	+	ข้าวหลามตัด
ACS_CKBOARD	:	กระดานหมากรุก (จุด)
ACS_DEGREE	,	สัญลักษณ์องศา
ACS_PLMINUS	#	บวก/ลบ
ACS_BULLET	o	สัญลักษณ์แสดงหัวข้อย่อย
ACS_LARROW	<	ลูกศรชี้ทางซ้าย
ACS_RARROW	>	ลูกศรชี้ทางขวา
ACS_DARROW	v	ลูกศรชี้ลง
ACS_UARROW	^	ลูกศรชี้ขึ้น
ACS_BOARD	#	ตารางสี่เหลี่ยม
ACS_LANTERN	#	สัญลักษณ์ตะเกียง
ACS_BLOCK	#	บล็อกลูกบาศก์ทึบ

รูทีนย่อย waddstr

รูทีนย่อย **waddstr** เพิ่ม สตริงอักขระที่สิ้นสุดด้วย null ไปยังหน้าต่าง โดยเริ่มต้นด้วยอักขระ ปัจจุบัน ถ้าคุณกำลังเพิ่มอักขระเดี่ยว ใช้รูทีนย่อย **waddch** มิฉะนั้น ใช้รูทีนย่อย **waddstr** ต่อไปนี้คือส่วนหนึ่งของกลุ่มรูทีนย่อย **waddstr**:

รูทีนย่อย	คำอธิบาย
แมโคร **addstr**	เพิ่มสตริงอักขระไปที่ stdscr
แมโคร **mvaddstr**	ย้ายเคอร์เซอร์โลจิคัลไปที่ตำแหน่งที่ระบุก่อนเพิ่ม สตริงอักขระไปที่ stdscr
รูทีนย่อย **waddstr**	เพิ่มสตริงอักขระไปยังหน้าต่างที่ผู้ใช้กำหนดเอง
แมโคร **wmvaddstr**	ย้ายเคอร์เซอร์โลจิคัลไปที่ตำแหน่งที่ระบุก่อนเพิ่ม สตริงอักขระไปยันหน้าต่างที่ผู้ใช้กำหนดเอง

รูทีนย่อย winsch

รูทีนย่อย **winsch** แทรกอักขระที่ระบุก่อนอักขระปัจจุบัน ในหน้าต่าง อักขระทั้งหมดทางขวาของอักขระที่แทรก จะถูกย้ายไปหนึ่งช่องว่างทางขวา เป็นผลให้อักขระที่อยู่ขวาสุดบน บรรทัดอาจหายไป ตำแหน่งของเคอร์เซอร์โลจิคัลและฟิสิคัลไม่เปลี่ยนแปลง หลังการย้าย รูทีนย่อย **winsch** ประกอบด้วย ต่อไปนี้:

รูทีนย่อย	คำอธิบาย
แมโคร **insch**	แทรกอักขระใน stdscr
แมโคร **mvinsch**	ย้ายเคอร์เซอร์โลจิคัลไปยังตำแหน่งที่สถานที่ระบุใน stdscr ก่อน แทรกอักขระ
แมโคร **mvwinsch**	ย้ายเคอร์เซอร์โลจิคัลไปยังตำแหน่งที่ระบุในหน้าต่างที่ ผู้ใช้กำหนดเองก่อนแทรกอักขระ
รูทีนย่อย **winsch**	แทรกอักขระในหน้าต่างที่ผู้ใช้กำหนดเอง

รูทีนย่อย winsertln

รูทีนย่อย **winsertln** แทรกบรรทัดว่างเหนือบรรทัด ปัจจุบันในหน้าต่าง รูทีนย่อย **insertln** แทรกบรรทัดใน stdscr บรรทัดล่างของหน้าต่างจะหายไป รูทีนย่อย **winsertln** ดำเนินการ แบบเดียวกันในหน้าต่างที่ผู้ใช้กำหนดเอง

รูทีนย่อย wprintw

รูทีนย่อย **wprintw** แทน ลำดับของอักขระ (เริ่มต้นด้วยอักขระปัจจุบัน) ด้วยเอาต์พุต ที่จัดรูปแบบ รูปแบบเหมือนกับสำหรับคำสั่ง **printf** กลุ่ม **printw** ประกอบด้วย ต่อไปนี้:

รูทีนย่อย	คำอธิบาย
แมโคร **mvprintw**	ย้ายเคอร์เซอร์โลจิคัลไปยังตำแหน่งที่สถานที่ระบุใน stdscr ก่อน แทนที่อักขระใดๆ
แมโคร **mvwprintw**	ย้ายเคอร์เซอร์โลจิคัลไปยังตำแหน่งที่ระบุในหน้าต่างที่ ผู้ใช้กำหนดเองก่อนแทนที่อักขระใดๆ
แมโคร **printw**	แทนที่ลำดับอักขระใน stdscr
รูทีนย่อย **wprintw**	แทนที่ลำดับอักขระในหน้าต่างที่ ผู้ใช้กำหนดเอง

รูทีนย่อย **wprintw** ทำการ เรียกใช้ไปยังรูทีนย่อย **waddch** เพื่อแทนที่อักขระ

แมโคร unctrl

แมโคร **unctrl** ส่งกลับการแทน ค่าอักขระควบคุมที่ระบุเป็นแบบที่พิมพ์ได้ ซึ่งแสดงในรูปแบบ ^X แมโคร **unctrl** ส่งกลับอักขระการพิมพ์คงเดิม

การเปิดใช้งานการเลื่อนข้อความ

ใช้รูทีนย่อย ต่อไปนี้เพื่อเปิดใช้งานการเลื่อน:

รูทีนย่อย	คำอธิบาย
idlok	อนุญาตให้ curses ใช้คุณลักษณะการแทรก/การลบบรรทัดด้วยฮาร์ดแวร์
scrollok	เปิดให้หน้าต่างสามารถเลื่อนเมื่อเคอร์เซอร์ถูกย้ายออกนอกขอบ ด้านขวาของบรรทัดสุดท้ายของหน้าต่าง
setscrreg or **wsetscrreg**	ตั้งค่าขอบเขตการเลื่อนด้วยซอฟต์แวร์ภายในหน้าต่าง

การเลื่อนเกิดขึ้นเมื่อโปรแกรมหรือผู้ใช้ย้ายเคอร์เซอร์ออกจาก ขอบด้านล่างของหน้าต่าง เพื่อให้เกิดการเลื่อน อันดับแรกคุณต้องใช้รูทีนย่อย **scrollok** เพื่อเปิดใช้งานการเลื่อนสำหรับหน้าต่าง หน้าต่าง ถูกเลื่อนถ้าการเลื่อนถูกเปิดใช้งานและมีกรณีใดต่อไปนี้เกิดขึ้น:

- เคอร์เซอร์ถูกย้ายออกนอกขอบหน้าต่าง
- อักขระขึ้นบรรทัดใหม่ถูกพบที่บรรทัดสุดท้าย
- อักขระถูกแทรกในตำแหน่งท้ายของบรรทัดสุดท้าย

เมื่อหน้าต่างถูกเลื่อน curses จะอัพเดตทั้ง หน้าต่างและหน้าจอ อย่างไรก็ตาม เพื่อให้การเลื่อนฟิสิคัลมีผลต่อ เทอร์มินัล คุณต้องเรียกใช้รูทีนย่อย **idlok** ด้วยพารามิเตอร์ *Flag* ตั้งค่าเป็น **TRUE**

ถ้าการเลื่อนถูกปิดใช้งาน เคอร์เซอร์ยังคงอยู่ บนบรรทัดสุดท้ายที่ตำแหน่งที่อักขระถูกป้อนเข้า

เมื่อการเลื่อนถูกเปิดใช้งานสำหรับหน้าต่าง คุณสามารถใช้ รูทีนย่อย **setscrreg** เพื่อสร้างแถบการเลื่อนซอฟต์แวร์ ภายในหน้าต่าง คุณส่งค่ารูทีนย่อย **setscrreg** สำหรับบรรทัดบนสุดและล่างสุดของแถบ ถ้า **setscrreg** ถูกเปิดใช้งานสำหรับแถบและการเลื่อนถูกเปิดใช้งานสำหรับ หน้าต่าง การพยายามใดๆ ที่จะย้ายออกนอกบรรทัดล่างสุดที่ระบุบรรทัดทั้งหมดใน ขอบเขตจะเลื่อนขึ้นหนึ่งบรรทัด คุณสามารถใช้แมโคร **setscrreg** เพื่อกำหนดขอบเขตการเลื่อนใน stdscr มิฉะนั้น คุณใช้รูทีนย่อย **wsetscrreg** เพื่อกำหนดขอบเขตการเลื่อนในหน้าต่าง ที่ผู้ใช้กำหนดเอง

หมายเหตุ: ต่างจากรูทีนย่อย **idlok** รูทีนย่อย **setscrreg** ไม่มีข้อกำหนดขอบเขตการใช้ ความสามารถในส่วนการเลื่อนฟิสิคัลที่เทอร์มินัลอาจมี

การลบอักขระ

คุณ สามารถลบข้อความโดยการแทนที่ด้วยช่องว่าง หรือโดยการลบอักขระ ออกจากอาร์เรย์อักขระและเลื่อนอักขระส่วนที่เหลือบนบรรทัด ไปทางซ้ายหนึ่งช่องว่าง

รูทีนย่อย werase

แมโคร **erase** ทำสำเนาช่องว่างไปยังทุกตำแหน่งใน stdscr รูทีนย่อย **werase** ใส่ช่องว่าง ที่ทุกตำแหน่งในหน้าต่างที่ผู้ใช้กำหนดเอง ในการลบอักขระเดี่ยว ในหน้าต่าง ใช้รูทีนย่อย **wdelch**

รูทีนย่อย wclear

ใช้รูทีนย่อยต่อไปนี้เพื่อลบหน้าจอทั้งหมด:

รูทีนย่อย	คำอธิบาย
clear หรือ **wclear**	ลบหน้าจอและตั้งค่าแฟล็กการลบสำหรับการรีเฟรชครั้งถัดไป
clearok	พิจารณาว่า curses ลบค่าบนหน้าต่างทั้งหมดในการเรียกใช้รูทีนย่อย **refresh** หรือ **wrefresh** ครั้งถัดไปหรือไม่

รูทีนย่อย **wclear** หรือที่คล้ายกัน กับรูทีนย่อย **werase** อย่างไรก็ตาม นอกเหนือจากใส่ช่องว่างที่ทุกตำแหน่งของหน้าต่าง รูทีนย่อย **wclear** ยังเรียกใช้รูทีนย่อย **wclearok** ด้วย เป็นผลให้ หน้าจอถูกลบค่า ให้การเรียกใช้รูทีนย่อย **wrefresh** ครั้งถัดไป

กลุ่มรูทีนย่อย **wclear** ประกอบด้วยรูทีนย่อย **wclear** แมโคร **clear** และรูทีนย่อย **clearok** แมโคร **clear** ใส่ช่องว่างที่ทุกตำแหน่งใน stdscr

รูทีนย่อย wclrtoeol

แมโคร **clrtoeol** ดำเนินการใน stdscr ขณะที่รูทีนย่อย **wclrtoeol** ดำเนินการอย่างเดียวกันภายในหน้าต่างที่ผู้ใช้กำหนดเอง

รูทีนย่อย wclrtobot

แมโคร **clrtobot** ดำเนินการใน stdscr ขณะที่ **wclrtobot** ดำเนินการอย่างเดียวกันในหน้าต่างที่ผู้ใช้กำหนดเอง

รูทีนย่อย wdelch

ใช้รูทีนย่อยต่อไปนี้เพื่อลบอักขระออกจากหน้าจอ:

รูทีนย่อย	คำอธิบาย
แมโคร **delch**	ลบอักขระปัจจุบันออกจาก stdscr
แมโคร **mvdelch**	ย้ายเคอร์เซอร์โลจิคัลก่อนลบอักขระออกจาก stdscr
แมโคร **mvwdelch**	ย้ายเคอร์เซอร์โลจิคัลก่อนลบอักขระออกจากหน้าต่างที่ ผู้ใช้กำหนดเอง
รูทีนย่อย **wdelch**	ลบอักขระปัจจุบันในหน้าต่างที่ผู้ใช้กำหนดเอง

รูทีนย่อย **wdelch** ลบอักขระปัจจุบันและย้ายอักขระทั้งหมด ไปทางขวาของอักขระปัจจุบันบนบรรทัดปัจจุบันหนึ่งตำแหน่งทางซ้าย อักขระตัวสุดท้ายในบรรทัดจะถูกเติมด้วยช่องว่าง กลุ่มรูทีนย่อย **delch** ประกอบด้วยรูทีนย่อยและแมโครต่อไปนี้:

รูทีนย่อย wdeleteln

รูทีนย่อย **deleteln** ลบบรรทัดปัจจุบัน และย้ายบรรทัดทั้งหมดที่อยู่ด้านล่าง บรรทัดปัจจุบันขึ้นมาหนึ่งบรรทัด การดำเนินการนี้ลบบรรทัดสุดท้ายของหน้าต่าง

การรับอักขระ

โปรแกรมของคุณ สามารถเรียกอักขระจากคีย์บอร์ดหรือจอแสดงผล รูทีนย่อย **wgetch** เรียกอักขระจากคีย์บอร์ด รูทีนย่อย **winch** เรียกอักขระจากจอแสดงผล

รูทีนย่อย wgetch

รูทีนย่อย **wgetch** อ่าน อักขระจากคีย์บอร์ดที่ติดกับเทอร์มินัลที่เชื่อมโยงกับ หน้าต่าง ก่อนรับค่าอักขระ รูทีนย่อยเหล่านี้เรียกใช้รูทีนย่อย **wrefresh** เพื่อดูว่ามีสิ่งใดเปลี่ยนในหน้าต่าง: ตัวอย่าง ถ้าเคอร์เซอร์ย้าย หรือข้อความเปลี่ยนแปลง สำหรับข้อมูลเพิ่มเติม โปรดดูที่รูทีนย่อย **wgetch** ใน *Technical Reference: Base Operating System and Extensions, Volume 2*

กุล่มรูทีนย่อย **wgetch** ประกอบด้วยต่อไปนี้:

รูทีนย่อย	คำอธิบาย
แมโคร **getch**	รับอักขระจาก stdscr
แมโคร **mvgetch**	ย้ายเคอร์เซอร์ก่อนรับอักขระจาก stdscr
แมโคร **mvwgetch**	ย้ายเคอร์เซอร์ก่อนรับอักขระจากหน้าต่างที่ผู้ใช้กำหนดเอง
รูทีนย่อย **wgetch**	รับอักขระจากหน้าต่างที่ผู้ใช้กำหนดเอง

ในการวางอักขระที่ได้รับมาก่อนหน้านี้โดยการเรียกใช้ รูทีนย่อย **wgetch** กลับไปที่คิวอินพุต ให้ใช้ รูทีนย่อย **ungetch** อักขระถูกเรียกข้อมูลโดย การเรียกใช้รูทีนย่อย **wgetch** ครั้งถัดไป

โหมดเทอร์มินัล

เอาต์พุตของรูทีนย่อย **wgetch** บางส่วนถูกกำหนดโดยโหมดของเทอร์มินัล รายการต่อไปนี้อธิบาย การดำเนินการของรูทีนย่อย **wgetch** ในแต่ละประเภทของ โหมดเทอร์มินัล:

รูทีนย่อย	คำอธิบาย
โหมด **DELAY**	หยุดการอ่านจนกว่าระบบส่งข้อความไปยังโปรแกรม ถ้า โหมด CBREAK ถูกตั้งค่าด้วย โปรแกรมจะหยุดทำงานหลังหนึ่งอักขระ ถ้าโหมด CBREAK ไม่ได้ถูกตั้งค่า (โหมด NOCBREAK) รูทีนย่อย **wgetch** จะหยุดการอ่านหลังอักขระขึ้นบรรทัดใหม่แรก ถ้า ECHO ถูกตั้งค่า อักขระ จะแสดงไปที่หน้าต่างด้วย
โหมด **HALF-DELAY**	หยุดการอ่านจนกว่าจะพิมพ์อักขระ หรือหมดเวลาใช้งานที่ ระบุ ถ้าโหมด ECHO ถูกตั้งค่า อักขระจะถูกแสดงไปที่หน้าต่างด้วย
โหมด **NODELAY**	ส่งกลับค่าของ ERR ถ้าไม่มีอินพุตรออยู่

หมายเหตุ: เมื่อคุณใช้รูทีนย่อย **wgetch** อย่าตั้งค่าทั้งโหมด NOCBREAK และโหมด ECHO พร้อมกัน การตั้งค่า ทั้งสองโหมดทำให้เกิดผลลัพธ์ที่ไม่ต้องกันทั้งนี้ขึ้นอยู่กับสถานะของไดรเวอร์ tty เมื่อพิมพ์แต่ละอักขระ

ฟังก์ชันคีย์

ฟังก์ชันคีย์ถูกกำหนดในไฟล์ **curses.h** ฟังก์ชันคีย์สามารถถูกส่งกลับได้โดยรูทีนย่อย **wgetch** ถ้าคีย์แพ็ดถูกเปิดใช้งาน เทอร์มินัลอาจไม่สนับสนุน ฟังก์ชันคีย์ทั้งหมด ในการดูว่าเทอร์มินัลสนับสนุนคีย์ใดเฉพาะ ให้ตรวจสอบนิยามฐานข้อมูล **terminfo** ของเทอร์มินัล สำหรับรายการของ ฟังก์ชันคีย์ โปรดดูที่ รูทีนย่อย getch, mvgetch, mvwgetch หรือรูทีนย่อย wgetch ใน *Technical Reference: Base Operating System and Extensions, Volume 2*

การรับค่าฟังก์ชันคีย์

ถ้าโปรแกรมของคุณเปิดใช้งานคีย์บอร์ดด้วยรูทีนย่อย **keypad** และผู้ใช้กดฟังก์ชันคีย์ โทเค็นสำหรับฟังก์ชันคีย์นั้นจะถูกส่งกลับแทนอักขระโดยตรง ไฟล์ **/usr/include/curses.h** กำหนดฟังก์ชันคีย์ ที่เป็นไปได้ แต่ละคำสั่งการกำหนด (define) ที่มีคำนำหน้า **KEY_** และคีย์ถูกกำหนดเป็นเลขจำนวนเต็มที่ขึ้นต้นด้วยค่า 03510

ถ้าได้รับอักขระที่อาจเป็นการเริ่มต้น ของฟังก์ชันคีย์ (เช่น Escape character) curses จะตั้งค่าตัวจับเวลา (โครงสร้าง ของประเภทช่วงเวลาที่ถูกกำหนดใน **/usr/include/sys/time.h**) ถ้าไม่ได้รับส่วนที่เหลือของลำดับก่อนที่ตัวจับเวลาจะ หมดอายุ ตัวจับเวลาจะถูกส่งไป มิฉะนั้น ค่าของฟังก์ชันคีย์ จะถูกส่งกลับ ด้วยเหตุนี้ หลังจากผู้ใช้กดคีย์ Esc จะมี การหน่วงเวลาก่อน escape ถูกส่งกลับไปยังโปรแกรม หลีกเลี่ยงการใช้คีย์ Esc เท่าที่จะทำได้ เมื่อคุณเรียกใช้รูทีนย่อยอักขระเดี่ยว เช่นรูทีนย่อย **wgetch** ตัวจับเวลานี้สามารถถูกแทนที่ หรือขยายเพิ่ม โดยใช้ตัวแปรสภาวะแวดล้อม **ESCDELAY**

ตัวแปรสภาวะแวดล้อม **ESCDELAY** ตั้งค่าระยะเวลารอก่อนที่จะหมดเวลา และทำเสมือนการกดคีย์ ESC เป็น Escape character มากกว่าที่จะเป็นการผสมกับอักขระอื่นๆ ใน บัฟเฟอร์เพื่อสร้างลำดับคีย์ ค่า **ESCDELAY** ถูกวัดทุกมิลลิวินาทีที่ห้า

ถ้าตัวแปร **ESCDELAY** เป็น 0 ระบบจะสร้างการตอบกลับ Escape ทันทีโดยไม่รอง ข้อมูลเพิ่มจากบัฟเฟอร์ คุณอาจเลือกค่าใดๆ ก็ได้ตั้งแต่ 0 ถึง 99,999 ค่ากำหนดดีฟอลต์สำหรับสำหรับ **ESCDELAY** คือ 500 (1 ส่วน 10 ของวินาที)

เพื่อป้องกันรูทีนย่อย **wgetch** จากการตั้งค่าตัวจับเวลา ให้เรียกรูทีนย่อย **notimeout** ถ้า notimeout ถูกตั้งค่าเป็น TRUE, curses ไม่แยกระหว่างฟังก์ชันคีย์ และอักขระ เมื่อรับข้อมูล

รูทีนย่อย keyname

รูทีนย่อย **keyname** ส่งกลับ ตัวชี้ไปที่สตริงอักขระที่มีชื่อสัญลักษณ์สำหรับอาร์กิวเมนต์ *Key* อาร์กิวเมนต์ *Key* สามารถเป็นคีย์ที่ส่งกลับ จากรูทีนย่อย **wgetch, getch, mvgetch** หรือ **mvwgetch**

รูทีนย่อย winch

รูทีนย่อย **winch** รับอักขระที่ตำแหน่งปัจจุบัน ถ้าแอ็ตทริบิวต์ ถูกตั้งค่าสำหรับตำแหน่ง ค่าแอ็ตทริบิวต์คือ ORed ใน ค่าที่ส่งกลับ คุณสามารถใช้รูทีนย่อย **winch** เพื่อ แยกเฉพาะอักขระหรือแอ็ตทริบิวต์ เพื่อทำดังนี้ใช้ค่าคงที่ที่ กำหนดไว้แล้ว **A_CHARTEXT** และ **A_ATTRIBUTES** ด้วยตัวดำเนินการโลจิคัล & (แอมเปอร์แซนด์) ค่าคงที่เหล่านี้ถูกกำหนดในไฟล์ **curses.h** ต่อไปเป็นเป็นรูทีนย่อย **winch**:

รูทีนย่อย	คำอธิบาย
แมโคร **inch**	รับอักขระปัจจุบันจาก stdscr
แมโคร **mvinch**	ย้ายเคอร์เซอร์โลจิคัลก่อนเรียกรูทีนย่อย **inch** บน stdscr
แมโคร **mvwinch**	ย้ายเคอร์เซอร์โลจิคัลก่อนการเรียกรูทีนย่อย **winch** ในหน้าต่างที่ผู้ใช้กำหนด
รูทีนย่อย **winch**	รับอักขระปัจจุบันจากหน้าต่างที่ผู้ใช้กำหนด

รูทีนย่อย wscanw

รูทีนย่อย **wscanw** อ่านข้อมูลอักขระ แปลอักขระตามข้อกำหนดการแปลง และเก็บผลที่แปลงไว้ในหน่วยความจำ รูทีนย่อย **wscanw** ใช้รูทีนย่อย **wgetstr** เพื่ออ่านข้อมูลอักขระ ต่อไปเป็นเป็นรูทีนย่อย **wscanw**:

รูทีนย่อย	คำอธิบาย
แมโคร **mvscanw**	ย้ายเคอร์เซอร์โลจิคัลก่อนการสแกน stdscr
แมโคร **mvwscanw**	ย้ายเคอร์เซอร์โลจิคัลในหน้าต่างที่ผู้ใช้กำหนดก่อนการสแกน
แมโคร **scanw**	สแกน stdscr
รูทีนย่อย **wscanw**	สแกนหน้าต่างที่ผู้ใช้กำหนด

รูทีนย่อย **vwscanw** สแกนหน้าต่างโดยใช้รายการอาร์กิวเมนต์ตัวแปร สำหรับข้อมูลเกี่ยวกับ การจัดการรายการอาร์กิวเมนต์ตัวแปร โปรดดูที่แมโคร **varargs** ใน *Technical Reference: Base Operating System and Extensions, Volume 2*

การทำความเข้าใจกับเทอร์มินัลที่มี curses

ความสามารถของโปรแกรมของคุณจะถูกจำกัดบางส่วน ด้วยความสามารถของเทอร์มินัลที่ทำงานอยู่

ส่วนนี้แสดงข้อมูลเกี่ยวกับการกำหนดค่าเริ่มต้นให้กับเทอร์มินัล และระบุความสามารถของเทอร์มินัลเหล่านั้น

การจัดการกับเทอร์มินัลจำนวนมาก

ด้วย curses คุณสามารถใช้เทอร์มินัลที่มากกว่าหนึ่งตัว สำหรับอินพุตและเอาต์พุต รูทีนย่อย terminal อนุญาตให้คุณสร้างเทอร์มินัลใหม่ เพื่อสับเปลี่ยนการประมวลผลอินพุตและเอาต์พุต และดึงข้อมูลความสามารถของเทอร์มินัล

คุณสามารถเริ่มต้น curses บนหน้าจอดีฟอลต์ได้โดยใช้รูทีนย่อย **initscr** ถ้าแอ็พพลิเคชันของคุณส่งเอาต์พุตไปยังเทอร์มินัลที่มากกว่าหนึ่ง ให้ใช้รูทีนย่อย **newterm** เรียกรูทีนย่อย **newterm** สำหรับเทอร์มินัลแต่ละตัว และใช้รูทีนย่อย **newterm** ถ้าแอ็พพลิเคชันของคุณต้องการแสดงเงื่อนไขข้อผิดพลาด ดังนั้นจึงสามารถดำเนินการรันต่อได้ ในโหมดเชิงบรรทัด หากเทอร์มินัลไม่สามารถ สนับสนุนโปรแกรมเชิงหน้าจอ

เมื่อเสร็จสิ้นแล้ว โปรแกรมต้องเรียกรูทีนย่อย **endwin** สำหรับแต่ละเทอร์มินัลที่ใช้ ถ้าคุณเรียกรูทีนย่อย **newterm** มากกว่าหนึ่งครั้งสำหรับเทอร์มินัลตัวเดียวกัน เทอร์มินัลแรกที่อ้างถึงต้องเป็นเทอร์มินัลตัวล่าสุดที่คุณเรียกรูทีนย่อย **endwin**

รูทีนย่อย **set_term** สับเปลี่ยนการประมวลผลอินพุต และเอาต์พุตระหว่างเทอร์มินัลที่ต่างกัน

การกำหนดความสามารถของเทอร์มินัล

Curses จะจัดหารูทีนย่อยต่อไปนี้ เพื่อช่วยคุณกำหนดความสามารถของเทอร์มินัล:

รูทีนย่อย	คำอธิบาย
has_ic	กำหนดว่า เทอร์มินัลมีความสามารถในการแทรกอักขระ
has_il	กำหนดว่า เทอร์มินัลมีความสามารถในการแทรกบรรทัด
longname	ส่งคืนชื่อถ้อยคำของเทอร์มินัล

รูทีนย่อย **longname** จะส่งคืนตัวชี้ไปยังพื้นที่สถิติที่มีคำอธิบายถ้อยคำของเทอร์มินัล ปัจจุบัน พื้นที่สถิตินี้จะถูกกำหนดไว้หลังจากที่เรียกรูทีนย่อย **initscr** or **newterm** ถ้าคุณต้องการใช้ รูทีนย่อย **longname** ด้วยเทอร์มินัลจำนวนมาก การเรียกรูทีนย่อย **newterm** แต่ละครั้งจะเขียนทับพื้นที่นี้ การเรียกรูทีนย่อย **set_term** ไม่ได้เรียกคืนค่า แต่จะบันทึกพื้นที่นี้ระหว่างการเรียกรูทีนย่อย **newterm** แทน

รูทีนย่อย **has_ic** จะส่งคืนค่า TRUE ถ้าเทอร์มินัลมีความสามารถในการแทรกและลบอักขระ

รูทีนย่อย **has_il** จะส่งคืนค่า TRUE ถ้าเทอร์มินัลมีความสามารถในการแทรกและลบบรรทัด หรือสามารถจำลองความสามารถโดยใช้ส่วนของการเลื่อน ให้ใช้รูทีนย่อย **has_il** เพื่อตรวจสอบว่า มีความเหมาะสมในการเปิดการเลื่อนแบบฟิสิคัลโดยใช้รูทีนย่อย **scrollok** หรือ **idlok**

การตั้งค่าโหมดอินพุตและเอาต์พุตของเทอร์มินัล

รูทีนย่อยที่ควบคุมอินพุตและเอาต์พุต จะกำหนดวิธีที่แอ็พพลิเคชันของคุณดึงข้อมูลและแสดงข้อมูลให้กับผู้ใช้

โหมดอินพุต

อักขระอินพุตพิเศษ จะสอดแทรกอักขระการควบคุมสายงาน อักขระอินเตอร์รัปต์ อักขระลบ และอักขระ kill โหมด curse เฉพาะตัวต่อไปนี้อนุญาตให้ แอ็พพลิเคชันควบคุมเอ็ฟเฟ็กต์ของอักขระอินพุต:

โหมด Cooked

โหมดนี้ช่วยให้บรรลุเป้าหมายของการประมวลผลในแบบหนึ่งบรรทัดในหนึ่งครั้งด้วยอักขระพิเศษทั้งหมด ที่ถูกจัดการภายนอกแอ็พพลิเคชัน บรรลุเอฟเฟ็กซ์ที่เหมือนกับการประมวลผลอินพุตในโหมด canonical สถานะของแฟล็ก ISIG และ IXON จะไม่ถูกเปลี่ยนตามการป้อนโหมดนี้โดยเรียก nocbreak( ) และถูกตั้งค่าตามการป้อนโหมดนี้โดยเรียก noraw( )

การนำไปปฏิบัติจะสนับสนุนอักขระลบและอักขระ kill จากโลแคลที่ได้รับการสนับสนุนใดๆ โดยไม่พิจารณาถึงความกว้างของอักขระ

โหมด cbreak

อักขระที่พิมพ์โดยผู้ใช้ จะพร้อมใช้งานโดยทันทีในแอ็พพลิเคชัน และ curses ไม่ได้ดำเนินการกับการประมวลผลพิเศษ บนอักขระลบหรืออักขระ kill แอ็พพลิเคชันสามารถเลือกโหมด cbreak เพื่อทำการแก้ไขบรรทัดของตนเอง แต่อนุญาตให้ยกเลิกอักขระที่ถูกใช้เพื่อ ยกเลิกภารกิจ โหมดนี้จะบรรลุเอฟเฟ็กต์เดียวกับโหมด noncanonical นั่นคือ การประมวลผลอินพุต Case B (พร้อมกับ MIN ที่ตั้งค่าเป็น 1 และ ICRNL ถูกล้างข้อมูล) สถานะของแฟล็ก ISIG และ IXON ไม่ได้ถูกเปลี่ยนตามการป้อนในโหมดนี้

โหมด Half-delay

เอฟเฟ็กต์จะเหมือนกับ **cbreak** ยกเว้นว่า ฟังก์ชันอินเตอร์จะรอจนกระทั่งอักขระพร้อมใช้งาน ช่วงเวลาที่กำหนดโดยแอ็พพลิเคชันผ่านไปโดยยึดตามเหตุการณ์ที่มาก่อน โหมดนี้จะบรรลุเอฟเฟ็กต์เดียวกับโหมด noncanonical นั่นคือ การประมวลผลอินพุต Case C (พร้อมกับ TIME ที่ตั้งค่า ตามที่ระบุโดยแอ็พพลิเคชัน) สถานะของแฟล็ก ISIG และ IXON ไม่ได้ถูกเปลี่ยนตามการป้อนในโหมดนี้

โหมด Raw

โหมด Raw จะกำหนดการควบคุมสูงสุดให้กับแอ็พพลิเคชันผ่านอินพุตเทอร์มินัล แอ็พพลิเคชันจะมองเห็นอักขระแต่ละตัวที่พิมพ์ ซึ่งจะบรรลุเอฟเฟ็กต์เดียวกับโหมด noncanonical นั่นคือ การประมวลผลอินพุต Case D แฟล็ก ISIG และ IXON จะถูกล้างข้อมูลตามการป้อนในโหมดนี้

ค่าติดตั้งของอินเตอร์เฟสเทอร์มินัลจะถูกบันทึกไว้ เมื่อการประมวลผลเรียกรูทีนย่อย **initscr** หรือ **newterm** กำหนดค่าเริ่มต้น curses และเรียกคืนค่าติดตั้งเหล่านั้น เมื่อรูทีนย่อย **endwin** ถูกเรียก โหมดอินพุตเริ่มต้นสำหรับการดำเนินการ curses ไม่ได้ระบุไว้ เว้นเสียแต่ว่า การนำไปปฏิบัติที่สนับสนุนการปฏิบัติตาม curses ที่ได้รับการพัฒนาแล้ว ซึ่งโหมดอินพุตเริ่มต้นคือ โหมด **cbreak**

ลักษณะการทำงานของปุ่ม BREAK จะขึ้นอยู่กับบิตอื่นๆ ในไดรเวอร์การแสดงผล ที่ไม่ถูกตั้งค่าโดย curses

โหมด Delay

โหมด delay โดยเฉพาะต่อไปนี้จะระบุวิธีที่ฟังก์ชัน curses ส่งคืนให้กับแอ็พพลิเคชันได้อย่างรวดเร็ว เมื่อไม่มีอินพุตเทอร์มินัลที่รออยู่ เมื่อเรียกฟังก์ชัน:

Delay	**คำอธิบาย**
No Delay	ฟังก์ชันล้มเหลว
Delay	แอ็พพลิเคชันจะรอจนกว่าการนำไปปฏิบัติจะส่งข้อความผ่านไปยัง แอ็พพลิเคชัน ถ้าโหมด cbreak และโหมด Raw ถูกตั้งค่า โหมดนี้จะอยู่หลังอักขระ หนึ่งตัวอักษร มิฉะนั้น โหมดนี้จะอยู่หลังอักขระขึ้นบรรทัดใหม่ตัวแรก อักขระสิ้นสุดบรรทัด หรืออักขระสิ้นสุดไฟล์

เอฟเฟ็กต์ของโหมด No Delay สำหรับการประมวลผลฟังก์ชันคีย์ไม่ได้ระบุไว้

การประมวลผลโหมด Echo

โหมด Echo จะกำหนดว่า curses echoes พิมพ์อักขระลงบนหน้าจอ เอฟเฟ็กต์ของโหมด echo คล้ายกับเอฟเฟ็กต์ของแฟล็ก echo ในฟิลด์โหมดโลคัลของโครงสร้าง **termios** ที่เชื่อมโยงกับอุปกรณ์เทอร์มินัลที่เชื่อมต่อกับหน้าต่าง อย่างไรก็ตาม curses จะล้างข้อมูลแฟล็ก echo เมื่อเรียกใช้งาน ยับยั้งระบบปฏิบัติการจากการดำเนินการ echo เมธอดของอักขระ echo ไม่ได้มีลักษณะเฉพาะกับเมธอดของระบบปฏิบัติการ ของอักขระ echo เนื่องจาก curses จะดำเนินการกับการประมวลผลเพิ่มเติมของอินพุตเทอร์มินัล

ถ้าอยู่ในโหมด echo curses จะดำเนินการ echo ด้วยตนเอง อักขระอินพุตที่สามารถมองเห็นได้ใดๆ จะถูกเก็บอยู่ในหน้าต่างปัจจุบันหรือหน้าต่างที่ระบุไว้โดยฟังก์ชันอินพุต ที่แอ็พพลิเคชันเรียก ที่ตำแหน่งเคอร์เซอร์ของหน้าต่าง ผ่านการเรียกรูทีนย่อย **addch** พร้อมด้วยเอฟเฟ็กต์ที่เป็นผลที่ตามมา เช่น การเคลื่อนของเคอร์เซอร์และการตัดคำ

ถ้าไม่ได้อยู่ในโหมด echo echo ใดๆ ของอินพุตต้องถูกดำเนินการโดยแอ็พพลิเคชัน แอ็พพลิเคชันจะดำเนินการ echo ด้วยตนเองในพื้นที่ที่ควบคุมของหน้าจอ หรือไม่ได้ echo ทั้งนั้น ดังนั้น จึงปิดใช้งานโหมด echo

ซึ่งอาจเป็นไปไม่ได้ที่จะปิดการประมวลผล echo สำหรับการซิงโครนัส และเทอร์มินัลเน็ตเวิร์กแบบอะซิงโครนัส เนื่องจากการประมวลผล echo จะถูกทำโดยเทอร์มินัล โดยตรง แอ็พพลิเคชันที่รันบนเทอร์มินัลควรรับรู้ว่า อักขระใดๆ ที่พิมพ์จะแสดงอยู่บนหน้าจอ ที่จุดที่เคอร์เซอร์อยู่

ต่อไปนี้คือส่วนของตระกูลการประมวลผล echo ของรูทีนย่อย:

รูทีนย่อย	**คำอธิบาย**
cbreak หรือ **nocbreak**	วางเทอร์มินัลลงในหรือดึงเทอร์มินัลออกจากโหมด CBREAK
delay_output	ตั้งค่าเอาต์พุตหน่วงเวลาในหน่วยมิลลิวินาที
echo หรือ **noecho**	ควบคุมการ echo ของอักขระที่พิมพ์ลงในหน้าจอ
halfdelay	ส่งคืน ERR ถ้าไม่มีอินพุตที่ถูกพิมพ์ หลังจากการบล็อกจำนวนของเวลาที่ระบุ
nl หรือ **nonl**	พิจารณาว่า curses แปลบรรทัดใหม่ลงในการขึ้นบรรทัดใหม่ และป้อนบรรทัดบนเอาต์พุต และแปลการส่งคืนลงในบรรทัดใหม่สำหรับอินพุต
raw หรือ **noraw**	วางเทอร์มินัลลงในหรือออกจากโหมด

รูทีนย่อย **cbreak** จะดำเนินการกับเซ็ตย่อยของฟังก์ชันที่ดำเนินการโดยรูทีนย่อย **raw** ในโหมด cbreak อักขระที่พิมพ์โดยผู้ใช้จะพร้อมใช้งานกับโปรแกรมโดยทันที และการประมวลผลอักขระลบหรืออักขระ kill ไม่ได้ถูกกระทำ ไม่เหมือนกับโหมด RAW อักขระอินเตอร์รัปต์และอักขระโฟลว์ จะถูกดำเนินการ มิฉะนั้น ไดรเวอร์ tty จะบัฟเฟอร์อักขระที่พิมพ์ไว้ จนกระทั่งบรรทัดใหม่หรือขึ้นบรรทัดใหม่จะถูกพิมพ์

หมายเหตุ: โหมด CBREAK จะปิดใช้งานการแปลโดยไดรเวอร์ tty

รูทีนย่อย **delay_output** จะตั้งค่าเอาต์พุตหน่วงเวลาให้กับจำนวนของมิลลิวินาทีที่ระบุ ห้ามใช้รูทีนย่อยนี้ เนื่องจากรูทีนนี้ใช้อักขระการเสริมเต็ม แทนการหยุดตัวประมวลผล

รูทีนย่อย **echo** จะวางเทอร์มินัลลงในโหมด echo ในโหมด echo curses จะเขียนอักขระที่พิมพ์โดยผู้ใช้ลงในเทอร์มินัล ที่ตำแหน่งเคอร์เซอร์แบบฟิสิคัล รูทีนย่อย **noecho** จะใช้เทอร์มินัลที่อยู่นอกโหมด echo

รูทีนย่อย **nl** และ **nonl** ตามลำดับ จะควบคุมว่า curses แปลบรรทัดใหม่ลงในการขึ้นบรรทัดใหม่ และป้อนบรรทัดบนเอาต์พุต และ curses จะแปลการขึ้นบรรทัดใหม่ลงในบรรทัดใหม่บน อินพุต ในตอนต้น การแปลเหล่านี้จะเกิดขึ้น ด้วยการปิดใช้งานการแปลเหล่านี้ ไลบรารีรูทีนย่อย curses จะมีการควบคุมความสามารถในการป้อนบรรทัด ซึ่งส่งผลในการเคลื่อนไหวเคอร์เซอร์ได้เร็วกว่า

รูทีนย่อย **nocbreak** จะใช้เทอร์มินัลที่อยู่นอกโหมด cbreak

รูทีนย่อย **raw** จะวางเทอร์มินัลลงในโหมด raw ในโหมด raw อักขระที่พิมพ์โดยผู้ใช้จะพร้อมใช้งานโดยทันทีกับโปรแกรม นอกจากนี้ การอินเตอร์รัปต์ ออก หยุดทำงาน และการควบคุมสายงานอักขระจะส่ง การไม่ตีความแทนการสร้างสัญลักษณ์ที่ทำไว้ในโหมด cbreak รูทีนย่อย **noraw** ใช้เทอร์มินัลที่อยู่นอกโหมด raw

การใช้ไฟล์ terminfo และ termcap

เมื่อ curses ถูกกำหนดค่าเริ่มต้นแล้ว curses จะตรวจสอบตัวแปรสภาวะแวดล้อม **TERM** เพื่อระบุชนิดของเทอร์มินัล ดังนั้น curses จะมองหาคำอธิบายนิยาม ความสามารถของเทอร์มินัล ข้อมูลนี้จะถูกเก็บในไดเร็กทอรีโลคัลที่ระบุโดยตัวแปรสภาวะแวดล้อม **TERMINFO** หรือในไดเร็กทอรี **/usr/share/lib/terminfo** โปรแกรม curses ทั้งหมดจะตรวจเช็คเป็นอันดับแรกเพื่อดูว่า ตัวแปรสภาวะแวดล้อม **TERMINFO** ได้ถูกกำหนดไว้หรือไม่ ถ้าตัวแปรนี้ไม่ได้กำหนดไว้ ไดเร็กทอรี **/usr/share/lib/terminfo** จะถูกตรวจสอบ

ตัวอย่างเช่น ถ้าตัวแปร **TERM** ถูกตั้งค่าเป็น `vt100` และตัวแปร **TERMINFO** ถูกตั้งค่าเป็นไฟล์ **/usr/mark/myterms** curses จะตรวจสอบไฟล์ **/usr/mark/myterms/v/vt100** ถ้าไฟล์นี้ไม่มีอยู่ curses จะตรวจสอบไฟล์ **/usr/share/lib/terminfo/v/vt100**

นอกจากนี้ ตัวแปรสภาวะแวดล้อม **LINES** และ **COLUMNS** สามารถตั้งค่าเพื่อแทนที่คำอธิบาย เทอร์มินัล

การเขียนโปรแกรมที่ใช้รูทีนย่อย terminfo

ใช้รูทีนย่อย **terminfo** เมื่อโปรแกรมของคุณต้องทำงานกับ ฐานข้อมูล terminfo โดยตรง ตัวอย่างเช่น ใช้รูทีนย่อยเหล่านี้กับฟังก์ชันคีย์ของโปรแกรม ในกรณีอื่นๆ ทั้งหมด รูทีนย่อย curses จะเหมาะสมมากกว่า และแนะนำให้ใช้

การกำหนดค่าเริ่มต้นให้กับเทอร์มินัล

โปรแกรมของคุณควรเริ่มต้นโดยเรียกรูทีนย่อย **setupterm** โดยปกติแล้ว รูทีนย่อยนี้ถูกเรียกโดยอ้อม โดยเรียกรูทีนย่อย **initscr** หรือ **newterm** รูทีนย่อย **setupterm** จะอ่านตัวแปรที่ต้องพึ่งพาเทอร์มินัลที่กำหนดไว้ในฐานข้อมูล **terminfo** ฐานข้อมูล **terminfo** จะสอดแทรกบูลีน ตัวเลข และตัวแปรสตริง ตัวแปร **terminfo** ทั้งหมดเหล่านี้ใช้ค่าที่กำหนดไว้สำหรับเทอร์มินัลที่ระบุ หลังจากที่อ่านฐานข้อมูล รูทีนย่อย **setupterm** จะกำหนดค่าเริ่มต้นให้กับตัวแปร **cur_term** ด้วยนิยามเทอร์มินัล ขณะที่ทำงานกับเทอร์มินัลจำนวนมาก คุณสามารถใช้รูทีนย่อย **set_curterm** เพื่อตั้งค่าตัวแปร **cur_term** ให้กับเทอร์มินัลที่ระบุเฉพาะ

รูทีนย่อยอื่น **restartterm** จะคล้ายกับรูทีนย่อย **setupterm** อย่างไรก็ตาม รูทีนย่อยนี้จะถูกเรียกหลังจากที่เรียกคืนหน่วยความจำกลับไปยังสถานะก่อนหน้านี้ ตัวอย่างเช่น คุณจะเรียกรูทีนย่อย **restartterm** หลังจากเรียกรูทีนย่อย **scr_restore** รูทีนย่อย **restartterm** จะสมมุติว่า อ็อพชันอินพุตและเอาต์พุต จะเหมือนกับที่หน่วยความจำบันทึกไว้ แต่ชนิดของเทอร์มินัลและอัตราบอด อาจต่างกัน

รูทีนย่อย **del_curterm** จะเพิ่มพื้นที่ว่างที่มีข้อมูลความสามารถสำหรับเทอร์มินัลที่ระบุ

ไฟล์ส่วนหัว

ประกอบด้วยไฟล์ **curses.h** และ **term.h** ในโปรแกรมของคุณในลำดับต่อไปนี้:

```
#include <curses.h>
#include <term.h>
```

ไฟล์เหล่านี้มีนิยามสำหรับสตริง ตัวเลข และแฟล็กในฐานข้อมูล **terminfo**

การจัดการกับความสามารถของเทอร์มินัล

ส่งสตริงที่เป็นพารามิเตอร์ผ่านรูทีนย่อย **tparm** เพื่อยกตัวอย่างประกอบ ให้ใช้รูทีนย่อย **tputs** หรือ **putp** เพื่อพิมพ์สตริง **terminfo** ทั้งหมดและเอาต์พุตของรูทีนย่อย **tparm**

รูทีนย่อย	คำอธิบาย
putp	จัดเตรียมุช็อตคัตไปยังรูทีนย่อย **tputs**
tparm	ยกตัวอย่างประกอบสตริงด้วยพารามิเตอร์
tputs	ใช้ขอมูลการเสริมเต็มกับสตริงที่กำหนดไว และออกเอาต์พุต

ใช้รูทีนย่อยต่อไปนี้เพื่อขอรับและส่งความสามารถของเทอร์มินัล :

รูทีนย่อย	คำอธิบาย
tigetflag	ส่งคืนค่าของความสามารถของบูลีนที่ระบุ ถ้าความสามารถไม่ใช่บูลีน a –1 จะถูกส่งคืน
tigetnum	ส่งคืนค่าของความสามารถของตัวเลข ถ้าความสามารถไม่ใช่ตัวเลข a –2 จะถูกส่งคืน
tigetstr	ส่งคืนค่าของความสามารถของสตริงที่ระบุ ถ้าความสามารถที่ระบุไม่ใช่สตริง รูทีนย่อย **tigetstr** จะส่งคืนค่าของ (**char** *) –1

การออกจากโปรแกรม

เมื่อออกจากโปรแกรมของคุณ ให้เรียกคืนโหมด tty ด้วยสถานะเดิม หากต้องการทำสิ่งนี้ ให้เรียกรูทีนย่อย **reset_shell_mode** ถ้าโปรแกรมของคุณใช้การกำหนดแอดเดรสของเคอร์เซอร์ โปรแกรมควรเอาต์พุตสตริง **enter_ca_mode** ที่จุดเริ่มต้นทำงาน และสตริง **exit_ca_mode** เมื่อออกจากโปรแกรม

โปรแกรมที่ใช้ shell escapes ควรเรียกรูทีนย่อย **reset_shell_mode** และเอาต์พุตสตริง **exit_ca_mode** ก่อนเรียก shell หลังจากส่งคืนจาก shell โปรแกรมควรเอาต์พุตสตริง **enter_ca_mode** และเรียกรูทีนย่อย **reset_prog_mode** การประมวลผลนี้ ต่างจากการดำเนินการ curses มาตรฐาน ซึ่งเรียกรูทีนย่อย **endwin** สำหรับการออก

รูทีนย่อย low-level screen

ใช้รูทีนย่อยต่อไปนี้สำหรับการจัดการกับหน้าจอในระดับต่ำ :

รูทีนย่อย	คำอธิบาย
ripoffline	ถอดบรรทัดเดี่ยวจาก stdscr
scr_dump	ดัมพ์เนื้อหาของหน้าจอเสมือนไปยังไฟล์ที่ระบุ
scr_init	กำหนดค่าเริ่มต้นให้กับโครงสร้างข้อมูล curses จากไฟล์ที่ระบุ
scr_restore	เรียกคืนหน้าจอเสมือนให้กับเนื้อหาของไฟล์ที่ดัมพ์ไว้ก่อนหน้า

รูทีนย่อย termcap

ถ้าโปรแกรมของคุณใช้ไฟล์ **termcap** สำหรับข้อมูลเทอร์มินัล รูทีนย่อย **termcap** จะสอดแทรกวัตถุประสงค์ในการแปล พารามิเตอร์คือพารามิเตอร์เดียวกันสำหรับรูทีนย่อย **termcap** Curses จะเลียนแบบรูทีนย่อยโดยใช้ฐานข้อมูล **terminfo** รูทีนย่อยต่อไปนี้ **termcap** จะถูกจัดหาไว้:

รูทีนย่อย	คำอธิบาย
tgetent	เลียนแบบรูทีนย่อย **setupterm**
tgetflag	ส่งคืนรายการบูลีนสำหรับ termcap identifier
tgetnum	ส่งคืนรายการตัวเลขสำหรับ termcap identifier
tgetstr	ส่งคืนรายการสตริงสำหรับ termcap identifier
tgoto	ทำซ้ำรูทีนย่อย **tparm** เอาต์พุตจากรูทีนย่อย **tgoto** ควรส่งผ่านไปยังรูทีนย่อย **tputs**

การแปลงคำอธิบาย termcap เป็นคำอธิบาย terminfo

คำสั่ง **captoinfo** จะแปลงคำอธบาย **termcap** ไปเป็นคำอธิบาย **terminfo** ตัวอย่างต่อไปนี้จะแสดงภาพของวิธีการทำงานของคำสั่ง **captoinfo** :

```
captoinfo /usr/lib/libtermcap/termcap.src
```

คำสั่งนี้จะแปลงไฟล์ **/usr/lib/libtermcap/termcap.src** ไปเป็นซอร์ส **terminfo** คำสั่ง **captoinfo** จะเขียนเอาต์พุตลงในเอาต์พุตมาตรฐาน และสงวนความคิดเห็น และข้อมูลอื่นๆ ในไฟล์

การจัดการกับ TTYs

ฟังก์ชันต่อไปนี้จะบันทึกและเรียกคืนสถานะของโหมดของเทอร์มินัล:

ฟังก์ชัน	คำอธิบาย
savetty	บันทึกสถานะของโหมด tty
resetty	เรียกคืนสถานะของโหมด tty ไปเป็นสถานะครั้งล่าสุดที่รูทีนย่อย **savetty** ถูกเรียก

เทอร์มินัลแบบซิงโครนัสและแบบอะซิงโครนัสแบบเครือข่าย

ซิงโครนัส ซิงโครนัสเน็ตเวิร์ก (NWA) หรือที่ไม่ใช่มาตรฐานที่เชื่อมต่อเทอร์มินัลแบบอะซิงโครนัสโดยตรง จะถูกใช้ในสภาวะแวดล้อมแบบเมนเฟรม และสื่อสารกับโฮสต์ในโหมดบล็อก นั่นคือ ผู้ใช้พิมพ์อักขระที่เทอร์มินัล จากนั้นกดปุ่มพิเศษเพื่อเริ่มต้นการส่งข้อมูลของอักขระไปยังโฮสต์

หมายเหตุ: แม้ว่า อาจมีความเป็นไปได้ที่จะส่งบล็อกด้วยขนาดที่ไม่มีกฎเกณฑ์ไปยังโฮสต์ ซึ่งเป็นไปไม่ได้หรือสามารถกำหนดสาเหตุที่อักขระถูกส่งด้วย การเคาะแป้นพิมพ์เดี่ยว การทำเช่นนี้อาจเป็นสาเหตุของปัญหาที่รุนแรงให้กับแอ็พพลิเคชันที่ทำการใช้อินพุตอักขระเดี่ยว

เอาต์พุต

อินเตอร์เฟส curses สามารถใช้ได้สำหรับการดำเนินการทั้งหมดที่ยังคงอยู่เพื่อเอาต์พุตไปที่เทอร์มินัล ด้วยข้อยกเว้นที่อาจเกิดขึ้นได้สำหรับเทอร์มินัลบางตัว รูทีน **refresh** อาจมีการเปลี่ยนแปลงเนื้อหาบนหน้าจอทั้งหมด เพื่อดำเนินการอัพเดตใดๆ

ถ้าไม่จำเป็นต้องล้างข้อมูลบนหน้าจอก่อนดำเนินการในแต่ละส่วน ผลลัพธ์ที่ได้ไม่สามารถกำหนดได้

อินพุต

เนื่องจากลักษณะการทำงานของการดำเนินกาแบบซิงโครนัส (โหมดบล็อก) และเทอร์มินัล NWA ไม่มีความเป็นไปได้ที่จะสนับสนุนฟังก์ชันอินพุต curses โดยเฉพาะหมายเหตุต่อไปนี้:

- อินพุตอักขระเดี่ยวอาจเป็นไปไม่ได้ ซึ่งอาจจำเป็นต้องกดคีย์พิเศษ เพื่อทำให้อักขระทั้งหมดที่พิมพ์ที่เทอร์มิเตอร์ถูกส่งผ่านไปยังโฮสต์
- ซึ่งบางครั้ง เป็นไปไม่ได้ที่จะปิดใช้งาน echo อักขระ echo อาจถูกทำงานโดยตรงด้วยเทอร์มินัล สำหรับเทอร์มินัลที่มีลักษณะการทำงานในวิธีนี้ แอ็พพลิเคชัน curse ใดๆ ที่ดำเนินการอินพุตควรรับรู้ว่า อักขระใดๆ ที่พิมพ์ จะปรากฏขึ้นบนหน้าจอที่ชี้ไปที่ตำแหน่งของเคอร์เซอร์ ซึ่งไม่ได้สอดคล้องกับตำแหน่งของเคอร์เซอร์ ในหน้าต่าง

การทำงานกับสี

ถ้าเทอร์มินัลสนับสนุนสีแล้ว คุณสามารถใช้รูทีนย่อยการจัดการกับสีเพื่อสอดคล้องสีลงในโปรแกรมต้นเหตุของคุณได้

ก่อนที่จะจัดการกับสี ให้ทดสอบว่า เทอร์มินัลนั้นสนับสนุนสี หากต้องการทดสอบ คุณสามารถใช้รูทีนย่อย **has_colors** หรือรูทีนย่อย **can_change_color** รูทีนย่อย **can_change_color** ยังตรวจสอบเพื่อดูว่า โปรแกรมสามารถเปลี่ยนนิยามสีของเทอร์มินัล

ได้ รูทีนย่อยเหล่านี้ ไม่จำเป็นต้องใช้อาร์กิวเมนต์

รูทีนย่อย	คำอธิบาย
can_change_color	ตรวจสอบเพื่อดูว่า เทอร์มินัลสนับสนุนสี และเปลี่ยนนิยามของสี
has_colors	ตรวจสอบว่า เทอร์มินัลสนับสนุนสี
start_color	กำหนดค่าเริ่มต้นสีพื้นฐานแปดสี และตัวแปรโกลบอลสองตัว **COLORS** และ **COLOR_PAIRS**

หลังจากที่คุณได้พิจารณาว่า เทอร์มินัลสนับสนุนสีแล้ว ให้เรียกรูทีนย่อย **start_color** ก่อนที่คุณจะเรียกรูทีนย่อยสีอื่นๆ ซึ่งเป็นแบบฝึกหัดที่ดีในการเรียกรูทีนย่อยในทันที หลังรูทีนย่อย **initscr** และหลังการทดสอบที่เป็นผลสำเร็จ ตัวแปรโกลบอล **COLORS** จะกำหนดจำนวนสูงสุดของสี ที่เทอร์มินัลสนับสนุน ตัวแปรโกลบอลThe **COLOR_PAIRS** กำหนดจำนวนสูงสุดของคู่สีที่เทอร์มินัลสนับสนุน

การจัดการกับแอ็ตทริบิวต์วิดีโอ

โปรแกรมของคุณสามารถจัดการแอ็ตทริบิวต์วิดีโอ

แอ็ตทริบิวต์ Video, bit masks และค่าสีดีฟอลต์

Curses ช่วยให้คุณสามารถควบคุมแอ็ตทริบิวต์ดังต่อไปนี้:

A_ALTCHARSET
: สลับชุดอักขระ

A_BLINK
: การกระพริบ

A_BOLD
: สว่างหรือชัดเป็นพิเศษ

A_DIM
: Half-bright

A_NORMAL
: แอ็ตทริบิวต์ปกติ

A_REVERSE
: การแสดงผลกลับสี

A_STANDOUT
: โหมดไฮไลต์ดีที่สุดของเทอร์มินัล

A_UNDERLINE
: ขีดเส้นใต้

COLOR_PAIR(Number)
: แสดงคู่สีที่แสดงโดย *Number* คุณต้องกำหนดค่าเริ่มต้นคู่สีแล้วโดยใช้รูทีนย่อย **init_pair**

แอ็ตทริบิวต์เหล่านี้ถูกกำหนดในไฟล์ **curses.h** คุณสามารถผ่านแอ็ตทริบิวต์ไปที่รูทีนย่อย **wattron, wattroff** และ **wattrset** หรือคุณสามารถ OR แอ็ตทริบิวต์ด้วยอักขระที่ส่งไปที่รูทีนย่อย **waddch** โลจิคัล C ตัวดำเนินการ OR คือ | (สัญลักษณ์ไพพ์) มีการจัดเตรียม bit masks ดังต่อไปนี้เช่นกัน:

A_ATTRIBUTES
แยกแอ็ตทริบิวต์

A_CHARTEXT
แยกอักขระ

A_COLOR
แยกข้อมูลฟิลด์คู่สี

A_NORMAL
ปิดแอ็ตทริบิวต์วิดีโอทั้งหมด

แมโครต่อไปนี้ถูกจัดเตรียมเพื่อทำงานกับ คู่สี: **COLOR_PAIR(** *Number*) และ **PAIR_NUMBER(** *Attribute*) แมโคร **COLOR_PAIR(** *Number*) และ **A_COLOR** mask ถูกใช้โดยแมโคร **PAIR_NUMBER(** *Attribute*) เพื่อแยกหมายเลขคู่สีที่พบในแอ็ตทริบิวต์ที่ระบุโดยพารามิเตอร์ *Attribute*

ถ้าโปรแกรมของคุณใช้สี ไฟล์ **curses.h** กำหนดจำนวนแมโครที่ระบุค่าสีเริ่มต้น ดังต่อไปนี้:

สี	ค่าจำนวนเต็ม
COLOR_BLACK	0
COLOR_BLUE	1
COLOR_GREEN	2
COLOR_CYAN	3
COLOR_RED	4
COLOR_MAGENTA	5
COLOR_YELLOW	6
COLOR_WHITE	7

Curses ถือว่าสีพื้นหลังเริ่มต้นสำหรับ เทอร์มินัลทั้งหมดคือ 0 (**COLOR_BLACK**)

การตั้งค่าแอ็ตทริบิวต์วิดีโอ

แอ็ตทริบิวต์หน้าต่างปัจจุบันถูกใช้กับอักขระทั้งหมด ที่เขียนลงในหน้าต่างด้วยรูทีนย่อย **addch** แอ็ตทริบิวต์เหล่านี้ยังคงเป็นคุณสมบัติของอักขระ อักขระเก็บค่าแอ็ตทริบิวต์เหล่านี้ระหว่างการดำเนินการของเทอร์มินัล

attroff หรือ wattroff
ปิดแอ็ตทริบิวต์

attron หรือ wattron
เปิดแอ็ตทริบิวต์

attrset หรือ wattrset
ตั้งค่าแอ็ตทริบิวต์ปัจจุบันของหน้าต่าง

standout, wstandout, standend หรือ wstandend
นำหน้าต่างเข้าสู่หรือออกจากโหมดไฮไลต์ดีที่สุดของเทอร์มินัล

vidputs หรือ vidattr
เอาต์พุตสตริงที่นำเทอร์มินัลเข้าสู่โหมด video-attribute

รูทีนย่อย **attrset** ตั้งค่าแอ็ตทริบิวต์ปัจจุบันของหน้าจอเริ่มต้น รูทีนย่อย **wattrset** ตั้งค่าแอ็ตทริบิวต์ ปัจจุบันของหน้าต่างที่ผู้ใช้กำหนด

ใช้รูทีนย่อย **attron** และ **attroff** เพื่อเปิดและปิดแอ็ตทริบิวต์ที่ระบุใน stdscr โดยไม่มีผล กับแอ็ตทริบิวต์อื่น รูทีนย่อย **wattron** และ **wattroff** ดำเนินการเหมือนกับในหน้าต่างที่ผู้ใช้กำหนด

รูทีนย่อย **standout** เหมือนกับการเรียกไปที่รูทีนย่อย **attron** ด้วยแอ็ตทริบิวต์ **A_STANDOUT** ซึ่งนำ stdscr เข้าสู่โหมดไฮไลต์ดีที่สุดของเทอร์มินัล รูทีนย่อย **wstandout** เหมือนกับการเรียกไปที่รูทีนย่อย **wattron(** *Window*, **A_STANDOUT)** ซึ่งนำหน้าต่างที่ผู้ใช้กำหนดเข้าสู่โหมด ไฮไลต์ที่ดีที่สุดของเทอร์มินัล รูทีนย่อย **standend** เหมือนกับการเรียกไปที่รูทีนย่อย **attrset(0)** ซึ่งปิดแอ็ตทริบิวต์ทั้งหมดสำหรับ stdscr รูทีนย่อย **wstandend** เหมือนกับการเรียกไปที่รูทีนย่อย **wattrset(** *Window*, **0)** ซึ่งปิดแอ็ตทริบิวต์ทั้งหมดสำหรับหน้าต่างที่ระบุ

รูทีนย่อย **vidputs** เอาต์พุตสตริงที่นำเทอร์มินัลเข้าสู่โหมดแอ็ตทริบิวต์ที่ระบุ อักขระถูกเอาต์พุตผ่านรูทีนย่อย **putc** รูทีนย่อย **vidattr** เหมือนกับรูทีนย่อย **vidputs** ยกเว้นอักขระ ถูกเอาต์พุตผ่านรูทีนย่อย **putchar**

การทำงานกับคู่สี

แมโคร **COLOR_PAIR** (*Number*) ถูกกำหนดในไฟล์ **curses.h** ดังนั้น คุณสามารถจัดการแอ็ตทริบิวต์สีเช่นเดียวกับแอ็ตทริบิวต์อื่น คุณต้อง กำหนดค่าเริ่มต้นคู่สีด้วยรูทีนย่อย **init_pair** ก่อนใช้ รูทีนย่อย **init_pair** มี พารามิเตอร์ดังต่อไปนี้: *Pair*, *Foreground*, และ *Background* พารามิเตอร์ *Pair* ต้องอยู่ระหว่าง 1 และ **COLOR_PAIRS** –1 พารามิเตอร์ *Foreground* และ *Background* ต้องอยู่ระหว่าง 0 และ **COLORS** –1 ตัวอย่างเช่น เมื่อต้องการกำหนดค่าเริ่มต้น คู่สี 1 เป็นพื้นหน้าสีดำและพื้นหลังสีเขียวแกมน้ำเงิน คุณจะใช้ คำสั่งดังต่อไปนี้:

```
init_pair(1, COLOR_BLACK, COLOR_CYAN);
```

จากนั้นคุณควรตั้งค่าแอ็ตทริบิวต์สำหรับหน้าต่างดัง นี้:

```
wattrset(win, COLOR_PAIR(1));
```

จากนั้นถ้าคุณเขียนสตริง `Let's add Color` ไปที่เทอร์มินัล สตริงแสดงเป็นอักขระสีดำ บนพื้นหลังสีเขียวแกมน้ำเงิน

การแยกแอ็ตทริบิวต์

คุณสามารถใช้ผลจากการเรียกรูทีนย่อย **winch** เพื่อแยกข้อมูลแอ็ตทริบิวต์ รวมทั้งหมายเลข คู่สี ตัวอย่างดังต่อไปนี้ใช้ค่าที่ส่งกลับโดยการเรียกไปที่รูทีนย่อย **winch** ด้วยตัวดำเนินการโลจิคัล AND ในภาษา C (**&**) และ **A_ATTRIBUTES** bit mask เพื่อแยก แอ็ตทริบิวต์ที่กำหนดให้กับตำแหน่งปัจจุบันในหน้าต่าง ผลลัพธ์ จากการดำเนินการนี้ถูกใช้กับแมโคร **PAIR_NUMBER** เพื่อแยกหมายเลขคู่สี และหมายเลข 1 ถูกพิมพ์บนหน้าจอ

```
win = newwin(10, 10, 0, 0);
init_pair(1, COLOR_RED, COLOR_YELLOW);
wattrset(win, COLOR_PAIR(1));
waddstr(win, "apple");

number = PAIR_NUMBER((mvwinch(win, 0, 0) & A_ATTRIBUTES));
wprintw(win, "%d\n", number);
wrefresh(win);
```

สีและเสียง

ไลบรารี curses จัดเตรียมรูทีนย่อยการเตือนดังต่อไปนี้เพื่อแจ้งเตือน ผู้ใช้:

`beep`
เสียงเตือนที่ได้ยินได้บนเทอร์มินัล

`flash`
แสดงการเตือนที่มองเห็นได้บนเทอร์มินัล

การตั้งค่าอ็อพชัน curses

ตัวเลือก curses ทั้งหมดถูกปิดโดยค่าเริ่มต้น ดังนั้น จึงไม่จำเป็นที่จะเปิด ก่อนการเรียกรูทีนย่อย **endwin** รูทีนย่อยดังต่อไปนี้อนุญาตให้คุณ ตั้งค่าตัวเลือกต่างๆ กับ curses:

`curs_set`
ตั้งค่าการมองเห็นได้ของเคอร์เซอร์เป็น มองไม่เห็น ปกติ หรือเห็นได้ชัดเจน

`idlok`
ระบุว่า curses สามารถใช้คุณลักษณะการแทรกและลบบรรทัด ของฮาร์ดแวร์ที่มาให้

`intrflush`
ระบุว่าคีย์อินเตอร์รัปต์ (interrupt, quit หรือ suspend) ฟลัชเอาต์พุตทั้งหมดในไดรเวอร์ tty หรือไม่ ค่าเริ่มต้นของตัวเลือกนี้ได้รับสืบทอด จากไดรเวอร์ tty

แป้นพิมพ์
ระบุว่า curses เรียกคืนข้อมูลจากแป้นพิมพ์ ของเทอร์มินัล ถ้าเปิดใช้งาน ผู้ใช้สามารถกดฟังก์ชันคีย์ (เช่นแป้นลูกศร) และรูทีนย่อย **wgetch** ส่งกลับค่าเดียวที่แสดงฟังก์ชันคีย์นั้น ถ้าปิดการใช้งาน curses จะไม่ปฏิบัติกับฟังก์ชันคีย์เป็นพิเศษ และโปรแกรมของคุณต้องแปล ลำดับ escape สำหรับรายการของฟังก์ชันคีย์เหล่านี้ โปรดดูที่รูทีนย่อย **wgetch**

`typeahead`
สั่งให้ curses ตรวจสอบชนิดล่วงหน้าใน descriptor ของไฟล์ทางเลือก

โปรดดูที่รูทีนย่อย **wgetch** และ การทำความเข้าใจกับเทอร์มินัลที่มี Curses สำหรับคำอธิบายของอ็อพชัน curses เพิ่มเติม

การจัดการกับซอฟต์เลเบล

Curses จัดให้มีรูทีนย่อยสำหรับการปรับเปลี่ยนซอฟต์ ฟังก์ชันคีย์เลเบล

เลเบลเหล่านี้แสดงอยู่ที่ด้านล่างของหน้าจอและทำให้ แอ็พพลิเคชัน เช่นเอดิเตอร์ มีลักษณะที่เป็นมิตรกับผู้ใช้มากขึ้น ในการใช้ซอฟต์เลเบล คุณต้องเรียกใช้รูทีนย่อย **slk_init** ก่อนเรียกใช้ รูทีนย่อย **initscr** หรือ **newterm**

รูทีนย่อย	คำอธิบาย
slk_clear	ลบซอฟต์เลเบลออกจากหน้าจอ
slk_init	เตรียมข้อมูลซอฟต์ฟังก์ชันคีย์เลเบลเบื้องต้น
slk_label	ส่งกลับค่าเลเบลปัจจุบัน
slk_noutrefresh	รีเฟรชซอฟต์เลเบล รูทีนย่อยนี้ทำงานเทียบเท่ากับ รูทีนย่อย **wnoutrefresh**
slk_refresh	รีเฟรชซอฟต์เลเบล รูทีนย่อยนี้ทำงานเทียบเท่ากับ รูทีนย่อย **refresh**
slk_restore	เรียกคืนซอฟต์เลเบลไปยังหน้าจอหลังการเรียกใช้รูทีนย่อย **slk_clear**
slk_set	ตั้งค่าซอฟต์เลเบล
slk_touch	อัพเดตซอฟต์เลเบลในการเรียกใช้รูทีนย่อย **slk_noutrefresh** ครั้งถัดไป

ในการจัดการซอฟต์เลเบล curses ลดขนาดของ **stdscr** ทีละหนึ่งบรรทัด โดยสำรองบรรทัดนี้เพื่อใช้โดยฟังก์ชันซอฟต์เลเบล การสำรองนี้หมายความว่าตัวแปรสภาวะแวดล้อม **LINES** จะถูกลดค่าด้วยเช่นกัน เทอร์มินัลจำนวนมากสนับสนุนซอฟต์เลเบลในตัว ถ้าซอฟต์เลเบลในตัวได้รับการสนับสนุน curses จะใช้ซอฟต์เลเบล มิฉะนั้น curses จะจำลองซอฟต์เลเบลด้วยซอฟต์แวร์

เนื่องจากเทอร์มินัลจำนวนมากที่สนับสนุนซอฟต์เลเบลมี 8 เลเบล curses จะทำตามมาตรฐานเดียวกัน สตริงเลเบลถูกจำกัดที่ 8 อักขระ Curses จัดเตรียมเลเบลโดยใช้หนึ่งในสองรูปแบบ: 3-2-3 (3 ซ้าย, 2 กลาง, 3 ขวา) หรือ 4-4 (4 ซ้าย, 4 ขวา)

ในการระบุสตริงสำหรับเลเบลเฉพาะ ให้เรียกใช้รูทีนย่อย **slk_set** รูทีนย่อยนี้ยังกำหนดให้ curses จัดชิดซ้าย จัดขวา หรือตรงกลางสำหรับสตริงบนเลเบล ในการรับ ชื่อเลเบลก่อนที่จะถูกจัดแนวโดยรูทีนย่อย **slk_set** ให้ใช้รูทีนย่อย **slk_label** รูทีนย่อย **slk_clear** และ **slk_restore** ลบและ เรียกคืนซอฟต์เลเบลตามลำดับ โดยปกติ ในการอัพเดตซอฟต์เลเบล โปรแกรมของคุณควรเรียกใช้รูทีนย่อย **slk_noutrefresh** สำหรับแต่ละ เลเบล จากนั้นใช้การเรียกใช้เดียวสำหรับรูทีนย่อย **slk_refresh** เพื่อดำเนินการเอาต์พุตที่แท้จริง ในการส่งเอาต์พุตซอฟต์เลเบลทั้งหมดในการเรียกใช้ รูทีนย่อย **slk_noutrefresh** ครั้งถัดไป ให้ใช้รูทีนย่อย **slk_touch**

ความเข้ากันได้ของ Curses

ส่วนนี้อธิบายปัญหาเกี่ยวกับความเข้ากันได้

ต่อไปนี้เป็นเรื่องความเข้ากันได้ที่จำเป็นต้องนำมาพิจารณา:

- ใน AIX 4.3, curses เข้ากันไม่ได้กับ AT&T System V Release 3.2 curses
- แอ็พพลิเคชันที่ คอมไพล์ รีบาวด์ หรือเชื่อมโยงใหม่ อาจจำเป็นต้องเปลี่ยนแปลงซอร์สโค้ด เพื่อให้เข้ากันได้กับ AIX Version 4 ของ curses ไลบรารี curses ไม่มีหรือใช้ฟังก์ชัน AIX extended curses
- แอ็พพลิเคชันที่ต้องการการสนับสนุนมัลติไบต์อาจสามารถคอมไพล์และเชื่อมโยง กับ extended curses อย่างไรก็ตามไม่แนะนำการใช้ไลบรารี extended curses นอกจากสำหรับแอ็พพลิเคชันที่ต้องการการสนับสนุนมัลติไบต์

รายการของรูทีนย่อย curses เพิ่มเติม

ส่วนต่อไปนี้อธิบายรูทีนย่อย curses เพิ่มเติม:

ส่วนต่อไปนี้อธิบายรูทีนย่อย curses เพิ่มเติม:

- การจัดการหน้าต่าง
- การจัดการอักขระ
- การจัดการเทอร์มินัล
- การจัดการสี
- ยูทิลิตีเบ็ดเตล็ด

การจัดการหน้าต่าง

ใช้รูทีนย่อยดังต่อไปนี้เพื่อจัดการหน้าต่าง:

รูทีนย่อย	คำอธิบาย
scr_dump	เขียนเนื้อหาปัจจุบันของหน้าจอเสมือนไปที่ไฟล์ ที่ระบุ
scr_init	ใช้เนื้อหาของไฟล์ที่ระบุเพื่อกำหนดค่าเริ่มต้น โครงสร้างข้อมูล curses
scr_restore	ตั้งค่าหน้าจอเสมือนเป็นเนื้อหาของไฟล์ที่ระบุ

การจัดการอักขระ

ใช้รูทีนย่อยดังต่อไปนี้เพื่อจัดการอักขระ:

รูทีนย่อย	คำอธิบาย
echochar, wechochar หรือ **pechochar**	ทำหน้าที่เหมือนกับการเรียกไปที่รูทีนย่อย **addch** (หรือ **waddch**) ที่ตามด้วยการเรียกไปที่รูทีนย่อย **refresh** (หรือ **wrefresh**)
flushinp	ฟลัชอักขระ type-ahead ที่พิมพ์โดยผู้ใช้แต่ยังไม่ได้อ่าน โดยโปรแกรม
insertln หรือ **winsertln**	แทรกบรรทัดว่างในหน้าต่าง
keyname	ส่งกลับตัวชี้ไปที่สตริงอักขระที่มีชื่อสัญลักษณ์ สำหรับพารามิเตอร์ *Key*
meta	ระบุว่าการส่งกลับอักขระ 8-บิตสำหรับรูทีนย่อย **wgetch** กระทำได้หรือไม่
nodelay	ทำการเรียกไปที่รูทีนย่อย **wgetch** ให้เป็นการเรียก แบบ nonblocking ถ้าอินพุตไม่พร้อม รูทีนย่อย **wgetch** ส่งกลับ **ERR**
scroll	เลื่อนหน้าต่างขึ้นหนึ่งบรรทัด
unctrl	ส่งกลับการแสดงอักขระที่พิมพ์ได้ อักขระควบคุม ถูกคั่นด้วย ^ (คาเร็ต)
vwprintw	การดำเนินการเหมือนกับรูทีนย่อย **wprintw** แต่รับรายการตัวแปรของอาร์กิวเมนต์
vwscanw	การดำเนินการเหมือนกับรูทีนย่อย **wscanw** แต่รับรายการตัวแปรของอาร์กิวเมนต์

การจัดการเทอร์มินัล

ใช้รูทีนย่อยดังต่อไปนี้เพื่อจัดการเทอร์มินัล:

รูทีนย่อย	คำอธิบาย
def_prog_mode	ระบุโหมดเทอร์มินัลปัจจุบันเป็นโหมด in-curses
def_shell_mode	บันทึกโหมดเทอร์มินัลปัจจุบันเป็นโหมด not-in-curses
del_curterm	ฟรีพื้นที่ที่ชี้โดยตัวแปร *oterm*
notimeout	ป้องกันรูทีนย่อย **wgetch** จากการตั้งค่า ตัวจับเวลาเมื่อแปลลำดับ escape อินพุต
pechochar	เหมือนกับการเรียกรูทีนย่อย **waddch** ตามด้วยการเรียกไปที่รูทีนย่อย **prefresh**
reset_prog_mode	คืนค่าเทอร์มินัลลงในโหมดโปรแกรม in-curses
reset_shell_mode	คืนค่าเทอร์มินัลไปที่โหมด shell (โหมด out-of-curses) รูทีนย่อย **endwin** ดำเนินการนี้โดยอัตโนมัติ
restartterm	ตั้งค่าโครงสร้าง TERMINAL สำหรับใช้โดย curses รูทีนยอยนี้เหมือน กับรูทีนย่อย **setupterm** เรียกรูทีนย่อย **restartterm** หลังจากคืนค่าหน่วยความจำ ไปที่ภาวะก่อนหน้า ตัวอย่างเช่น เรียกรูทีนย่อยนี้หลังจากเรียกไปที่รูทีนย่อย **scr_restore**

การจัดการสี

ใช้รูทีนย่อยดังต่อไปนี้เพื่อจัดการสี:

รูทีนย่อย	คำอธิบาย
color_content	ส่งกลับองค์ประกอบของสี
init_color	เปลี่ยนสีเป็นองค์ประกอบที่ต้องการ
init_pair	กำหนดค่าเริ่มต้นคู่สีเป็นสีพื้นหน้าและ พื้นหลังที่ระบุ
pair_content	ส่งกลับสีพื้นหน้าและพื้นหลังสำหรับตัวเลข คู่สีที่ระบุ

ยูทิลีตีเบ็ดเตล็ด

มียูทิลิตีเบ็ดเตล็ดดังต่อไปนี้:

ยูทิลีตี	คำอธิบาย
baudrate	เคียวรีเทอร์มินัลปัจจุบันและส่งกลับความเร็วเอาต์พุต
erasechar	ส่งกลับอักขระลบที่เลือกโดยผู้ใช้
killchar	ส่งกลับอักขระ line-kill ที่เลือกโดยผู้ใช้

การดีบักโปรแกรม

มีโปรแกรมดีบักหลายโปรแกรมที่พร้อมใช้งานสำหรับการดีบักโปรแกรมของคุณ: **adb, dbx, dex, softdb** และโปรแกรม kernel debug โปรแกรม **adb** ช่วยให้คุณดีบักไบนารีไฟล์ที่รันได้ และตรวจสอบไฟล์ข้อมูล non-ASCII

โปรแกรม **dbx** เปิดใช้งาน การดีบักระดับซอร์สของโปรแกรมภาษา C, C++ และ FORTRAN เช่นเดียวกับ การดีบักภาษา assembler ของโปรแกรมเรียกทำงานที่ระดับเครื่อง (**dex**) มี X interface สำหรับโปรแกรมดีบัก **dbx**จัดเตรียมหน้าต่างสำหรับการดูซอร์ส บริบท และตัวแปรของโปรแกรมแอ็พพลิเคชัน โปรแกรมดีบัก **softdb** ทำงานเหมือนโปรแกรมดีบัก **dex** แต่ **softdb** ถูกใช้กับ AIX Software Development Environment Workbench โปรแกรมดีบักเคอร์เนลถูกใช้เพื่อช่วยระบุข้อผิดพลาดในโค้ด ที่รันอยู่ในเคอร์เนล

บทความดังต่อไปนี้มีข้อมูลเกี่ยวกับโปรแกรมดีบัก **adb** and **dbx**:
ข้อมูลที่เกี่ยวข้อง:
คำสั่ง adb

ภาพรวมโปรแกรมดีบัก adb

คำสั่ง **adb** จัดเตรียมโปรแกรมดีบัก อเนกประสงค์ คุณสามารถใช้คำสั่งนี้เพื่อตรวจสอบอ็อบเจ็กต์และไฟล์คอร์และ จัดเตรียมสภาพแวดล้อมที่ควบคุมสำหรับรันโปรแกรม

ขณะที่คำสั่ง **adb** รันอยู่ คำสั่งรับอินพุตมาตรฐาน และเขียนไปที่เอาต์พุตมาตรฐาน คำสั่งไม่รู้จักคีย์ Quit หรือ Interrupt ถ้ามีการใช้คีย์เหล่านี้คำสั่ง **adb** จะรอคำสั่งใหม่

การเริ่มต้นใช้งานโปรแกรมดีบัก adb

ส่วนนี้อธิบายวิธีเริ่มต้นโปรแกรมการดีบัก **adb** จากไฟล์ต่างๆ โดยใช้พร้อมต์ **adb** ใช้คำสั่งเชลล์จากภายในโปรแกรม **adb** และหยุดการทำงานโปรแกรม **adb**

การเริ่มต้น adb ด้วยไฟล์โปรแกรม

คุณสามารถเริ่มการทำงานโปรแกรมดีบัก **adb** โดยไม่มีชื่อไฟล์ ในกรณีนี้โปรแกรม **adb** จะค้นหาไฟล์ **a.out** เริ่มต้นในไดเร็กทอรี ทำงานปัจจุบันของคุณและเตรียมไฟล์สำหรับการดีบัก ดังนั้นคำสั่ง:

```
adb
```

เหมือนกับการป้อน:

```
adb a.out
```

โปรแกรม **adb** เริ่มการทำงานด้วยไฟล์ **a.out** และรอคำสั่ง+ ถ้าไฟล์ **a.out** ไม่มีอยู่โปรแกรม **adb** เริ่มทำงานโดยไม่มีไฟล์ และไม่แสดงข้อความแสดงข้อผิดพลาด

การเริ่มต้น adb ด้วยอิมเมจไฟล์ Ccore

คุณสามารถใช้โปรแกรมดีบัก **adb** เพื่อตรวจสอบไฟล์ core image ของโปรแกรมที่ทำให้เกิดข้อผิดพลาดของระบบ ที่ไม่สามารถกู้คืนได้ ไฟล์ Core image รักษาเร็กคอร์ดของเนื้อหาของรีจิสเตอร์ CPU สแต็ก และพื้นที่หน่วยความจำของโปรแกรมของคุณขณะเกิดข้อผิดพลาด ดังนั้น ไฟล์ core image จัดเตรียมวิธีในการระบุสาเหตุของข้อผิดพลาด

เพื่อตรวจสอบไฟล์ core image ด้วยโปรแกรมที่เกี่ยวข้อง คุณต้อง ให้ชื่อของทั้งไฟล์ core และไฟล์โปรแกรม บรรทัดคำสั่งมีฟอร์ม:

adb *ProgramFile CoreFile*

โดยที่ *ProgramFile* เป็นชื่อไฟล์ของโปรแกรม ที่ทำให้เกิดข้อผิดพลาด และ *CoreFile* เป็นชื่อไฟล์ ของไฟล์ core image ที่สร้างโดยระบบ จากนั้นโปรแกรม **adb** ใช้ข้อมูลจากทั้งสองไฟล์เพื่อจัดเตรียมการตอบสนองกับคำสั่งของคุณ

ถ้าคุณไม่ให้ชื่อไฟล์ของไฟล์ core image โปรแกรม **adb** จะค้นหาไฟล์ core เริ่มต้นที่ชื่อ **core** ในไดเร็กทอรีทำงานปัจจุบันของคุณ ถ้าพบไฟล์ดังกล่าว โปรแกรม **adb** โปรแกรมจะตรวจสอบว่าไฟล์ core เป็นของ *ProgramFile* หรือไม่ ถ้าใช่ โปรแกรม **adb** จะใช้ไฟล์นี้ มิฉะนั้น โปรแกรม **adb** จะละเว้นไฟล์ core โดยแจ้งข้อความแสดงข้อผิดพลาดที่เหมาะสม

> **หมายเหตุ:** คำสั่ง **adb** ไม่สามารถใช้เพื่อตรวจสอบอ็อบเจ็กต์ 64-บิตและรูปแบบ AIX 4.3 core **adb** ยังคงทำงานได้กับรูปแบบ pre-AIX 4.3 core บน AIX 4.3 ผู้ใช้สามารถทำให้เคอร์เนลสร้าง core dumps ในรูปแบบ pre-AIX 4.3 โดยใช้ smitty

การเริ่มต้น adb ด้วยไฟล์ข้อมูล

โปรแกรม **adb** จัดเตรียมวิธีดูเนื้อหาของ ไฟล์ที่มีรูปแบบและโครงสร้างที่หลากหลาย คุณสามารถใช้โปรแกรม **adb** เพื่อตรวจสอบไฟล์ข้อมูลโดยใช้ชื่อ ไฟล์ข้อมูลในตำแหน่งของ program หรือ core file ตัวอย่างเช่น ในการตรวจสอบ ไฟล์ข้อมูลชื่อ **outdata** ให้ป้อน:

```
adb outdata
```

โปรแกรม **adb** จะเปิดไฟล์ชื่อ `outdata` และให้ผู้ใช้ตรวจสอบเนื้อหาไฟล์ วิธีการตรวจสอบไฟล์นี้ มีประโยชน์ถ้าไฟล์มีข้อมูลที่ไม่ใช่ ASCII คำสั่ง **adb** อาจแสดงคำเตือน เมื่อคุณให้ชื่อของไฟล์ข้อมูลที่ไม่ใช่ ASCII ในตำแหน่งของไฟล์โปรแกรม ซึ่งปกติเกิดขึ้นเมื่อเนื้อหา ของไฟล์ข้อมูลเหมือนกับไฟล์โปรแกรม เช่นไฟล์ core ไฟล์ data ไม่สามารถถูกดำเนินการได้

การเริ่มต้น adb ด้วยอ็อพชันการเขียน

ถ้าคุณเปิดโปรแกรมหรือไฟล์ข้อมูลด้วยแฟล็ก **-w** ของคำสั่ง **adb** คุณสามารถ ทำการเปลี่ยนแปลงและแก้ไขไฟล์ ตัวอย่างเช่น คำสั่ง:

```
adb -w sample
```

เปิดไฟล์โปรแกรม `sample` เพื่อการเขียน จากนั้นคุณสามารถใช้คำสั่ง **adb** เพื่อตรวจสอบและ ปรับเปลี่ยนไฟล์นี้ แฟล็ก **-w** ทำให้โปรแกรม **adb** สร้างไฟล์ที่กำหนด ถ้าไฟล์ยังไม่มีอยู่ ตัวเลือก ยังให้คุณเขียนข้อมูลได้โดยตรงไปที่หน่วยความจำหลังจากการโปรแกรมที่กำหนด

การใช้พร้อมต์

หลังจากคุณได้เริ่มโปรแกรม **adb** คุณสามารถกำหนด พร้อมต์ของคุณใหม่ด้วยคำสั่งย่อย **$P**

เมื่อต้องการเปลี่ยนพร้อมต์ **[adb:scat]>>** เป็น **Enter a debug command--->** ให้ป้อน:

```
$P"Enter a debug command--->"
```

เครื่องหมายคำพูดไม่จำเป็น เมื่อทำการกำหนดพร้อมต์ใหม่จากบรรทัดคำสั่ง **adb**

การใช้คำสั่งเชลล์จากภายในโปรแกรม adb

คุณสามารถรันคำสั่งเชลล์โดยไม่ต้องออกจากโปรแกรม **adb** โดยใช้คำสั่ง **adb** escape (**!**) (เครื่องหมายตกใจ) คำสั่ง escape มีรูปแบบ:

! *Command*

ในรูปแบบนี้ *Command* เป็นคำสั่งเชลล์ที่คุณ ต้องการรัน คุณต้องจัดเตรียมอาร์กิวเมนต์ที่จำเป็นกับคำสั่ง โปรแกรม **adb** ส่งผ่านคำสั่งนี้ไปที่เชลล์ระบบที่เรียก คำสั่ง เมื่อโปรแกรมเสร็จสิ้น เชลล์ส่งกลับการควบคุมไปที่โปรแกรม **adb** ตัวอย่างเช่น เมื่อต้องการแสดงวันที่ ให้ป้อน คำสั่งดังต่อไปนี้:

```
! date
```

ระบบแสดงวันที่และคืนการควบคุมไปที่โปรแกรม **adb**

การออกจากโปรแกรมดีบัก adb

คุณสามารถหยุดโปรแกรม **adb** และกลับไปที่เชลล์ระบบ โดยใช้คำสั่งย่อย **$q** หรือ **$Q** คุณยังสามารถหยุดโปรแกรม **adb** โดยพิมพ์ลำดับคีย์ Ctrl-D คุณไม่สามารถหยุดโปรแกรม **adb** โดยการกด คีย์ Interrupt หรือ Quit คีย์เหล่านี้ทำให้ **adb** รอ คำสั่งใหม่

การควบคุมการเรียกใช้งานโปรแกรม

ส่วนนี้อธิบายคำสั่งและคำสั่งย่อยที่จำเป็นในการเตรียมโปรแกรมสำหรับ การดีบัก ดำเนินการโปรแกรม ตั้งค่า แสดง และลบจุดหยุด ดำเนินโปรแกรมต่อ ใช้คำสั่งทีละคำสั่งผ่านโปรแกรม หยุดโปรแกรม และปิด โปรแกรม

การเตรียมโปรแกรมสำหรับการดีบักด้วยโปรแกรม adb

คอมไพล์โปรแกรมโดยใช้คำสั่ง **cc** กับไฟล์ เช่น **adbsamp2** โดยป้อนคำสั่งดังนี้:

```
cc adbsamp2.c -o adbsamp2
```

เมื่อต้องการเริ่มเซสชันดีบัก ให้ป้อน:

```
adb adbsamp2
```

ภาษา C ไม่สร้างเลเบลคำสั่งสำหรับโปรแกรม ดังนั้น คุณไม่สามารถอ้างอิงถึงคำสั่งภาษา C เป็นรายคำสั่งได้ เมื่อใช้โปรแกรมดีบัก เพื่อให้การใช้คำสั่งดำเนินการได้อย่างมีประสิทธิภาพ คุณต้องคุ้นเคยกับ คำสั่งที่คอมไพลเลอร์ C สร้าง และวิธีที่คำสั่งเหล่านี้ สัมพันธ์กับแต่ละคำสั่งของภาษา C เทคนิคหนึ่งที่มีประโยชน์คือสร้าง รายการภาษาแอสเซมเบลอร์ของโปรแกรม C ของคุณก่อนการใช้โปรแกรม **adb** จากนั้น อ้างอิงรายการขณะที่คุณใช้โปรแกรมดีบัก เมื่อต้องการสร้างรายการภาษาแอสเซมเบลอร์ ใช้แฟล็ก **-S** หรือ **-q***List* ของคำสั่ง **cc**

ตัวอย่างเช่น เมื่อต้องการสร้างรายการภาษาแอสเซมเบลอร์ของโปรแกรมตัวอย่าง, **adbsamp2.c**, ใช้คำสั่งดังต่อไปนี้:

```
cc -S adbsamp2.c -o adbsamp2
```

คำสั่งนี้สร้างไฟล์ **adbsamp2.s** ที่มีรายการภาษา แอสเซมเบลอร์สำหรับโปรแกรม และคอมไพล์โปรแกรมเป็นไฟล์ เรียกทำงาน **adbsamp2**

การรันโปรแกรม

คุณสามารถดำเนินการโปรแกรมโดยใช้คำสั่งย่อย **:r** หรือ **:R** คำสั่งมีรูปแบบ:

[*Address*][**,***Count*] **:r** [*Arguments*]

หรือ

[*Address*][**,***Count*] **:R** [*Arguments*]

ในรูปแบบนี้พารามิเตอร์ *Address* ให้แอดเดรส ซึ่งเริ่มการรันโปรแกรม พารามิเตอร์ *Count* คือจำนวนของจุดหยุดที่จะข้ามก่อนถึงตำแหน่งที่ต้องการ และพารามิเตอร์ *Arguments* จัดเตรียมอาร์กิวเมนต์บรรทัดคำสั่ง เช่นชื่อไฟล์และตัวเลือก เพื่อส่งผ่านให้กับโปรแกรม

ถ้าคุณไม่ระบุค่า *Address* โปรแกรม **adb** จะใช้จุดเริ่มต้นของโปรแกรม เมื่อต้องการรันโปรแกรม จากจุดเริ่มต้นให้ป้อน:

```
:r
```

ถ้าคุณระบุค่า *Count* โปรแกรม **adb** ละเว้นจุดหยุดทั้งหมดจนกว่าจะถึงหมายเลขที่กำหนด ตัวอย่างเช่น เมื่อต้องการข้ามห้าจุดหยุด ให้ใช้คำสั่ง:

```
,5:r
```

ถ้าคุณมีอาร์กิวเมนต์ให้แยกแต่ละรายการด้วยช่องว่างอย่างน้อยหนึ่งช่อง อาร์กิวเมนต์ ถูกส่งผ่านไปที่โปรแกรมในแบบเดียวกับเชลล์ส่งผ่านอาร์กิวเมนต์บรรทัดคำสั่ง ไปที่โปรแกรม คุณสามารถใช้สัญลักษณ์การเปลี่ยนทิศทางเชลล์

คำสั่งย่อย **:R** ส่งผ่านอาร์กิวเมนต์คำสั่งผ่าน เชลล์ก่อนเริ่มการดำเนินการของโปรแกรม คุณสามารถใช้อักขระการจับคู่รูปแบบในอาร์กิวเมนต์เพื่ออ้างอิงถึงหลายไฟล์หรือค่าอินพุตอื่น เชลล์ขยายอาร์กิวเมนต์ที่มีอักขระการจับคู่รูปแบบก่อน ส่งไปที่โปรแกรม คุณลักษณะนี้มีประโยชน์ถ้าโปรแกรมต้องการ ชื่อไฟล์หลายชื่อ ตัวอย่างเช่น คำสั่งดังต่อไปนี้ส่งอาร์กิวเมนต์ **[a-z]*** ไปที่เชลล์ซึ่งถูกขยายเป็นรายการของ ชื่อไฟล์ที่ตรงกัน ก่อนถูกส่งไปที่โปรแกรม:

```
:R [a-z]*.s
```

คำสั่งย่อย **:r** และ **:R** เอาเนื้อหา ของรีจิสเตอร์ทั้งหมดและทำลายปัจจุบันก่อนการเริ่มต้น โปรแกรม การดำเนินการนี้หยุดสำเนาของโปรแกรมก่อนหน้าที่อาจ รันอยู่

การตั้งค่า breakpoints

เมื่อต้องการตั้งค่าจุดหยุดในโปรแกรม ให้ใช้คำสั่งย่อย **:b** จุดหยุดหยุดการดำเนินการ เมื่อโปรแกรมทำงานมาถึงแอดเดรสที่ระบุ จากนั้นการควบคุมส่งกลับไปที่โปรแกรมดีบัก **adb** คำสั่งมีฟอร์ม:

[*Address*] [**,***Count*] **:b** [*Command*]

ในรูปแบบนี้พารามิเตอร์ *Address* ต้องเป็นแอดเดรสคำสั่งเครื่อง ที่ถูกต้อง พารามิเตอร์ *Count* เป็น จำนวนครั้งที่คุณต้องการให้จุดหยุดข้ามไปก่อนที่จะ ทำให้โปรแกรมหยุด และพารามิเตอร์ *Command* เป็นคำสั่ง **adb** ที่คุณต้องการใช้งานแต่ละครั้งที่คำสั่งถูกดำเนินการ (ไม่ว่าจุดหยุดจะหยุดโปรแกรม หรือไม่) ถ้าคำสั่งที่ระบุตั้งค่า . (จุด) เป็น 0 จุดหยุดจะทำให้หยุดทำงาน

ตั้งค่าจุดหยุดเพื่อหยุดการกระทำการของโปรแกรมที่ตำแหน่งจำเพาะในโปรแกรม เช่นจุดเริ่มต้นของฟังก์ชัน เพื่อที่คุณสามารถดูที่เนื้อหาของ รีจิสเตอร์และหน่วยความจำ ตัวอย่างเช่น เมื่อดีบักตัวอย่างโปรแกรม **adbsamp2** คำสั่งต่อไปนี้ตั้งค่าจุดหยุดที่จุดเริ่มต้น ของฟังก์ชันที่ชื่อ **f**:

```
.f :b
```

จุดหยุดถูกนำมาใช้เป็นการควบคุมเข้าสู่ฟังก์ชันและก่อน ที่เฟรมสแต็กถูกสร้าง

จุดหยุดที่มีการนับถูกใช้ภายในฟังก์ชันที่ถูกเรียกหลายครั้ง ระหว่างการดำเนินการของโปรแกรม หรือภายในคำสั่งที่ตรงกับคำสั่ง **for** หรือ **while** จุดหยุด ดังกล่าวอนุญาตให้โปรแกรมดำเนินต่อเพื่อรันจนฟังก์ชันหรือ คำสั่งที่กำหนดถูกดำเนินการตามจำนวนครั้งที่ระบุ ตัวอย่างเช่น คำสั่งดังต่อไปนี้ตั้งค่าจุดหยุดในครั้งที่สองที่ฟังก์ชัน **f** ถูกเรียกในโปรแกรม **adbsamp2**:

```
.f,2 :b
```

จุดหยุดไม่หยุดฟังก์ชันจนถึงครั้งที่สองที่ฟังก์ชัน ถูกรัน

การแสดง breakpoints

ใช้คำสั่งย่อย **$b** เพื่อแสดงตำแหน่งและจำนวนของ แต่ละจุดหยุดที่กำหนดในปัจจุบัน คำสั่งนี้แสดงรายการของจุดหยุด ตามแอดเดรสและจำนวนนับหรือคำสั่งที่ระบุสำหรับจุดหยุด ตัวอย่างเช่น ต่อไปนี้เป็นการตั้งค่าสองจุดหยุดในไฟล์ **adbsamp2** แล้วใช้คำสั่งย่อย **$b** เพื่อแสดง จุดหยุดเหล่านั้น:

```
.f+4:b
.f+8:b$v
$b
จุดพัก
count  brkpt              command
1      .f+8               $v
1      .f+4
```

เมื่อโปรแกรมรัน โปรแกรมจะหยุดที่จุดหยุดแรกที่พบ เช่น `.f+4` ถ้าคุณใช้คำสั่งย่อย **:c** เพื่อดำเนินการกระทำการต่อ โปรแกรมจะหยุดอีกครั้งที่จุดหยุดถัดไป และเริ่มคำสั่งย่อย **$v** คำสั่งและลำดับ ตอบสนองมีลักษณะเหมือนกับตัวอย่างดังต่อไปนี้:

```
:r
adbsamp2: running
breakpoint        .f+4:         st       r3,32(r1)
:c
adbsamp2: running
variables
b = 268435456
d = 236
e = 268435512
m = 264
breakpoint        .f+8           l       r15,32(r1)
```

การลบ breakpoints

เมื่อต้องการใช้คำสั่งย่อย **:d** เพื่อลบจุดหยุดจาก โปรแกรม ป้อน:

Address **:d**

ในรูปแบบนี้พารามิเตอร์ *Address* ให้ข้อมูล แอดเดรสของจุดหยุดที่จะลบ

ตัวอย่างเช่น เมื่อดีบักตัวอย่างโปรแกรม **adbsamp2** การป้อนคำสั่งดังต่อไปนี้ลบจุดหยุดที่จุดเริ่มต้นของฟังก์ชัน **f**:

```
.f:d
```

การเรียกทำงานโปรแกรมต่อ

เมื่อต้องการใช้คำสั่งย่อย **:c** เพื่อดำเนินการดำเนินการ ของโปรแกรมต่อไป หลังจากได้ถูกหยุดโดยการเข้าสู่จุดหยุด:

[*Address*] [,*Count*] **:c** [*Signal*]

ในรูปแบบนี้พารามิเตอร์ *Address* ให้แอดเดรส ของคำสั่งที่จะดำเนินการต่อ พารามิเตอร์ *Count* ให้จำนวนของจุดหยุดที่จะละเว้น และพารามิเตอร์ *Signal* เป็นจำนวนของสัญญาณที่ส่งไปที่โปรแกรม

ถ้าคุณไม่ระบุพารามิเตอร์ *Address* โปรแกรม จะเริ่มที่คำสั่งถัดไปหลังจากจุดหยุด ถ้าคุณกำหนด พารามิเตอร์ *Count* โปรแกรมดีบัก **adb** ละเว้นจุดหยุด *Count* แรก

ถ้าโปรแกรมถูกหยุดโดยใช้คีย์ Interrupt หรือ Quit สัญญาณนี้ ถูกส่งผ่านไปที่โปรแกรมโดยอัตโนมัติเมื่อเริ่มใหม่ เพื่อป้องกันสัญญาณนี้ไม่ให้ถูกส่ง ให้ป้อนคำสั่งในรูปแบบ:

[*Address*] [,*Count*] **:c 0**

อาร์กิวเมนต์คำสั่ง **0** ป้องกันสัญญาณจากการถูกส่ง ไปที่ subprocess

โปรแกรมแบบขั้นตอนเดียว

ใช้คำสั่งย่อย **:s** เพื่อรันโปรแกรมในขั้นตอนเดียว หรือครั้งละหนึ่งคำสั่ง คำสั่งนี้เรียกคำสั่งและส่งกลับการควบคุม ไปที่โปรแกรมดีบัก **adb** คำสั่ง มีฟอร์ม:

[A*address*] [,*Count*] **:s** [*Signal*]

ในรูปแบบนี้พารามิเตอร์ *Address* ให้แอดเดรส ของคำสั่งที่คุณต้องการสั่งการ และพารามิเตอร์ *Count* เป็นจำนวนครั้งที่คุณต้องการทำคำสั่งซ้ำ ถ้าไม่มี subprocess ปัจจุบันพารามิเตอร์ *ObjectFile* ถูกรันเป็น subprocess ในกรณีนี้ไม่มีสัญญาณที่สามารถถูกส่งได้และ ตัวเตือนของบรรทัดจะถูกจัดการเป็นอาร์กิวเมนต์ส่งไปที่ subprocess ถ้าคุณไม่ได้ กำหนดค่าสำหรับพารามิเตอร์ *Address* โปรแกรม **adb** จะใช้แอดเดรสปัจจุบัน ถ้าคุณระบุพารามิเตอร์ *Count* โปรแกรม **adb** ทำการเรียกแต่ละคำสั่งที่ต่อเนื่องกันต่อไป จนกว่าคำสั่งพารามิเตอร์ *Count* จะถูกรัน จุดหยุดจะถูกละเว้นขณะทำการรันทีละคำสั่ง ตัวอย่างเช่น คำสั่งดังต่อไปนี้เรียกห้าคำสั่งแรกในฟังก์ชัน **main**:

```
.main,5:s
```

การหยุดโปรแกรมด้วยคีย์อินเตอร์รัปต์และ quit

ใช้คีย์ Interrupt หรือ Quit เพื่อหยุดการรันโปรแกรมเมื่อต้องการ การกดคีย์ใดคีย์หนึ่งจะหยุดโปรแกรมปัจจุบันและส่งกลับการควบคุม ไปที่โปรแกรม **adb** คีย์เหล่านี้มีประโยชน์กับโปรแกรม ที่มีการวนรอบไม่รู้จบหรือข้อผิดพลาดโปรแกรมอื่น

เมื่อคุณกดปุ่ม Interrupt หรือ Quit เพื่อหยุดโปรแกรม โปรแกรม **adb** บันทึกสัญญาณโดยอัตโนมัติ ถ้าคุณเริ่มโปรแกรม อีกครั้งโดยใช้คำสั่ง **:c** โปรแกรม **adb** ส่งสัญญาณไปที่โปรแกรมโดยอัตโนมัติ คุณลักษณะนี้ มีประโยชน์เมื่อทำการทดสอบโปรแกรมที่ใช้สัญญาณนี้เป็นส่วนหนึ่งของการประมวลผล เมื่อต้องการรันโปรแกรมต่อโดยไม่มีการส่งสัญญาณ ให้ใช้คำสั่ง:

```
:c 0
```

อาร์กิวเมนต์คำสั่ง **0** (ศูนย์) ป้องกันสัญญาณจาก การถูกส่งไปที่โปรแกรม

การหยุดโปรแกรม

เมื่อต้องการหยุดการทำงานของโปรแกรมที่คุณกำลังดีบัก ให้ใช้คำสั่งย่อย **:k** คำสั่งนี้หยุดกระบวนการที่สร้างสำหรับโปรแกรมและส่งกลับการควบคุม ไปที่โปรแกรมดีบัก **adb** คำสั่งล้างเนื้อหาปัจจุบันของรีจิสเตอร์ยูนิตระบบ และสแต็ก และเริ่มโปรแกรมอีกครั้ง ตัวอย่างดังต่อไปนี้แสดงการใช้ของคำสั่งย่อย **:k** เพื่อล้างกระบวนการปัจจุบันจากโปรแกรม **adb**:

```
:k
560:    killed
```

การใช้นิพจน์ adb

ส่วนนี้อธิบายถึงการใช้นิพจน์ **adb**

การใช้จำนวนเต็มในนิพจน์

ขณะที่สร้างนิพจน์ คุณสามารถใช้เลขจำนวนเต็มได้ในสามรูปแบบคือ: เลขฐานสิบ เลขฐานแปด และเลขฐานสิบหก เลขจำนวนเต็มฐานสิบต้องขึ้นต้อนด้วยดิจิตเลขฐานสิบที่ไม่ใช่ศูนย์ เลขฐานแปดต้องขึ้นต้อนด้วย 0 (ศูนย์) และมีดิจิตฐานแปดเท่านั้น (0–7) เลขฐานสิบหกต้องขึ้นด้วยคำนำหน้า 0x และสามารถมีดิจิตเลขฐานสิบและตัวอักษรจาก a ถึง f (ทั้งตัวพิมพ์ใหญ่และตัวพิมพ์เล็ก) ต่อไปนี้ คือตัวอย่างของตัวเลขที่ถูกต้อง:

เลขฐานสิบ	เลขฐานแปด	เลขฐานสิบหก
34	042	0x22
4090	07772	0xffa

การใช้สัญลักษณ์ในนิพจน์

สัญลักษณ์คือชื่อของตัวแปรโกลบอล และฟังก์ชันที่นิยามไว้ภายในโปรแกรมที่ต้องการดีบัก สัญลักษณ์มีค่าเท่ากับแอดเดรสของตัวแปร หรือฟังก์ชันที่กำหนดไว้ ซึ่งจะเก็บอยู่ในตารางสัญลักษณ์ของโปรแกรม และพร้อมใช้งานหากตารางสัญลักษณ์ไม่ถูกแสดงจากไฟล์โปรแกรม

ในนิพจน์ คุณสามารถสะกดสัญลักษณ์ได้อย่างถูกต้องเหมือนกับที่อยู่ในซอร์สโปรแกรม หรือเหมือนกับที่เก็บอยู่ในตารางสัญลักษณ์ สัญลักษณ์ในตารางสัญลักษณ์จะมีความยาวอักขระไม่เกิน 8 ตัวอักษร

เมื่อคุณใช้คำสั่งย่อย **?** โปรแกรม **adb** จะใช้สัญลักษณ์ที่พบในตารางสัญลักษณ์ของไฟล์โปรแกรม ที่สร้างการกำหนดแอดเดรสในเชิงสัญลักษณ์ ดังนั้น คำสั่งย่อย **?** จะกำหนดชื่อฟังก์ชัน ขณะที่แสดงข้อมูลในบางครั้ง ซึ่งจะไม่เกิดเหตุการณ์นี้ขึ้น หากคำสั่งย่อย **?** ถูกใช้สำหรับข้อความ (คำสั่ง) และคำสั่ง **/** ถูกใช้สำหรับข้อมูล

ตัวแปรโลคัลอาจถูกกำหนดแอดเดรสได้หากซอร์สโปรแกรมในภาษา C ถูกคอมไพล์ด้วยแฟล็ก **-g**

ถ้าซอร์สโปรแกรมในภาษา C ไม่ได้ถูกคอมไพล์โดยใช้แฟล็ก **-g** ตัวแปรโลคัลจะไม่สามารถกำหนดแอดเดรสได้ คำสั่งต่อไปนี้แสดงค่าของตัวแปรโลคัล **b** ในฟังก์ชันตัวอย่าง :

```
.sample.b / x - value of local variable.
.sample.b = x - Address of local variable.
```

การใช้ตัวดำเนินการในนิพจน์

คุณสามารถรวมเลขจำนวนเต็ม สัญลักษณ์ ตัวแปร และชื่อรีจิสเตอร์ ด้วยตัวดำเนินการต่อไปนี้ได้:

ตัวดำเนินการ Unary:

~ (tilde)	ตัวดำเนินการเพื่อจัดการกับข้อมูลในระดับบิต (Bitwise complementation)
- (เส้นประ)	การปฏิเสธเลขจำนวนเต็ม
* (เครื่องหมายดอกจัน)	ส่งคืนเนื้อหาของตำแหน่ง

ตัวดำเนินการ Binary:

+ (plus)	การบวก
- (ลบ)	การลบ
* (เครื่องหมายดอกจัน)	การคูณ
% (เปอร์เซ็นต์)	การหารเลขจำนวนเต็ม
& (เครื่องหมายแอมเปอร์แซนด์)	การเชื่อมในระดับบิต
] (เครื่องหมายวงเล็บขวา)	ถอนการเชื่อมในระดับบิต
^ (คาเร็ต)	มอดูโล
# (เครื่องหมายตัวเลข)	ปัดขึ้นไปเป็นจำนวนถัดไป

โปรแกรมดีบัก **adb** ใช้หลักการคำนวณแบบ 32 บิต ค่าที่เกิน 2,147,483,647 (เลขฐานสิบ) จะถูกแสดงเป็นค่าติดลบ ตัวอย่างต่อไปนี้ จะแสดงผลลัพธ์ของค่าสองค่าที่ต่างกันในตัวแปร *n* และแสดงค่าทั้งในรูปแบบเลขฐานสิบ และเลขฐานสิบหก:

```
2147483647>n<
n=D
    2147483647<
n=X
    7fffffff
2147483648>n<
n=D
    -2147483648<
n=X
    80000000
```

ตัวดำเนินการ unary จะมีการนำหน้าที่สูงกว่าตัวดำเนินการไบนารี ตัวดำเนินการไบนารีทั้งหมด จะมีการนำหน้าที่เหมือนกัน และถูกประเมินผลตามลำดับ จากซ้ายไปขวา ดังนั้น โปรแกรม **adb** จะประเมินผลนิพจน์ไบนารีต่อไปนี้ ตามที่แสดงไว้:

```
2*3+4=d
    10
4+2*3=d
    18
```

คุณสามารถเปลี่ยนการนำหน้าของการดำเนินการในนิพจน์ได้โดยใช้เครื่องหมายวงเล็บ ตัวอย่างต่อไปนี้แสดงวิธีการเปลี่ยนนิพจน์ก่อนหน้านี้โดยใช้เครื่องหมายวงเล็บ:

```
4+(2*3)=d
    10
```

ตัวดำเนินการ unary หรือ * (เครื่องหมายดอกจัน) จะใช้แอดเดรสที่กำหนดไว้เป็นตัวชี้ไปยังเซ็กเมนต์ข้อมูล นิพจน์ที่ใช้ตัวดำเนินการนี้ จะเท่ากับค่าที่ชี้โดยตัวชี้นั้น ตัวอย่างเช่น นิพจน์:

```
*0x1234
```

มีค่าเท่ากับแอดเดรสข้อมูล 0x1234 ตามตัวอย่างดังนี้:

```
0x1234
```

มีค่าเท่ากับ 0x1234

การกำหนดโปรแกรมดีบัก adb เอง

ส่วนนี้อธิบายวิธีที่คุณสามารถกำหนดโปรแกรมดีบัก **adb** เอง

การรวมคำสั่งบนบรรทัดเดียว

คุณสามารถให้คำสั่งมากกว่าหนึ่งคำสั่งบนหนึ่งบรรทัด โดยแยกคำสั่ง ด้วย; (เซมิโคลอน) คำสั่งถูกดำเนินการครั้งเดียว เริ่มต้นทางด้านซ้าย เปลี่ยนเป็นแอดเดรสและรูปแบบปัจจุบันเลื่อนออกไปสู่คำสั่งถัดไป ถ้ามีข้อผิดพลาดเกิดขึ้น คำสั่งที่เหลือจะถูกละเว้น ตัวอย่างเช่น ลำดับดังต่อไปนี้แสดงทั้ง ตัวแปร **adb** แล้วเรียกใช้รูทีนย่อยที่จุดหนึ่งในโปรแกรม **adbsamp2**:

```
$v;$c
ตัวแปร
b = 10000000
d = ec
e = 10000038
m = 108
t = 2f8.
f(0,0) .main+26.
main(0,0,0) start+fa
```

การสร้างสคริปต์ adb

คุณสามารถกำหนดโปรแกรมดีบัก **adb** ให้อ่านคำสั่ง จากไฟล์ข้อความแทนอ่านจากแป้นพิมพ์โดยเปลี่ยนทิศทางของ ไฟล์อินพุต เมื่อคุณเริ่มโปรแกรม **adb** เมื่อต้องการเปลี่ยนทิศทาง อินพุตมาตรฐาน ให้ใช้สัญลักษณ์การเปลี่ยนทิศทางอินพุต, < (น้อยกว่า) และระบุ ชื่อไฟล์ ตัวอย่างเช่น ใช้คำสั่งดังต่อไปนี้เพื่ออ่านคำสั่ง จากสคริปต์ไฟล์:

```
adb sample <script
```

ไฟล์ต้องมีคำสั่งย่อย **adb** ที่ถูกต้อง ใช้ไฟล์ สคริปต์โปรแกรม **adb** เมื่อชุดคำสั่งเดียวกัน สามารถถูกใช้สำหรับไฟล์อ็อบเจ็กต์ที่ต่างกัน สคริปต์สามารถแสดงเนื้อหาของ ไฟล์คอร์หลังหลังจากโปรแกรมมีข้อผิดพลาด ตัวอย่างดังต่อไปนี้ แสดงไฟล์ที่มี คำสั่งที่แสดงข้อมูลเกี่ยวกับข้อผิดพลาดของข้อผิดพลาด เมื่อไฟล์นั้นถูกใช้ เป็นอินพุตกับโปรแกรม **adb** โดยใช้คำสั่งดังต่อไปนี้ เพื่อดีบักไฟล์ **adbsamp2** เอาต์พุตที่ระบุ ถูกสร้าง

```
120$w
4095$s.
f:b:
r
=ln"======= adb Variables ======="
$v
=ln"======= Address Map ======="
$m
=ln"======= C Stack Backtrace ======="
```

```
$C
=1n"======= C External Variables ======="
$e
=1n"======= Registers ======="
$r
0$s
=1n"======= Data Segment ======="<
b,10/8xna
```

```
$ adb adbsamp2 <script
```

```
adbsamp2: running
breakpoint .f:   b  .f+24
    ======= adb Variables =======
ตัวแปร
0 = TBD
1 = TBD
2 = TBD
9 = TBD
b = 10000000
d = ec
e = 10000038
m = 108
t = 2f8
    ======= Address Map =======
[0]? map  .adbsamp2.
b1 = 10000000  e1 = 100002f8  f1 = 0
b2 = 200002f8  e2 = 200003e4  f2 = 2f8
[0]/ map  .-.
b1 = 0    e1 = 0    f1 = 0
b2 = 0    e2 = 0    f2 = 0
    ======= C Stack Backtrace =======.
f(0,0) .main+26.
main(0,0,0) start+fa
    ======= C External Variables =======Full word.
errno: 0.
environ:  3fffe6bc.
NLinit:  10000238.
main: 100001ea.
exit: 1000028c.
fcnt: 0
```

```
.loop .count:  1.
f:   100001b4.
NLgetfile: 10000280.
write: 100002e0.
NLinit. .X:  10000238 .
NLgetfile. .X:   10000280 .
cleanup: 100002bc.
exit: 100002c8 .
exit . .X:  1000028c . .
cleanup . .X:  100002bc
```

```
    ======= Registers =======
mq  20003a24  .errno+3634
cs  100000 gt
ics  1000004
```

```
pc  100001b4 .f
r15  10000210 .main+26
r14  20000388  .main
r13  200003ec  .loop .count
r12  3fffe3d0
r11  3fffe44c
r10  0
r9  20004bcc
r8  200041d8  .errno+3de8
r7  0
r6  200030bc  .errno+2ccc
r5  1
r4  200003ec  .loop .count
r3  f4240
r2  1
r1  3fffe678
r0  20000380  .f.
f:   b  .f+24
    ======= Data Segment =======
10000000:  103 5313  3800  0  0  2f8 0  ec
10000010:  0  10 1000  38 0  0  0  1f0
10000020:  0  0  0  0  1000  0  2000  2f8
10000030:  0  0  0  0  4  6000  0  6000
10000040:  6e10  61d0  9430  a67 6730  6820  c82e  8
10000050:  8df0  94 cd0e  60 6520  a424  a432  c84e
10000060:  8  8df0  77 cd0e  64 6270  8df0  86
10000070:  cd0e  60 6520  a424  a432  6470  8df0  6a
10000080:  cd0e  64 c82e  19 8df0  78 cd0e  60
10000090:  6520  a424  a432  c84e  19 8df0  5b cd0e
100000a0:  64 cd2e  5c 7022  d408  64 911 c82e
100000b0:  2e 8df0  63 cd0e  60 6520  a424  a432
100000c0:  c84e  2e 8df0  46 cd0e  64 15 6280
100000d0:  8df0  60 cd0e  68 c82e  3f 8df0  4e
100000e0:  cd0e  60 6520  a424  a432  c84e  3f 8df0
100000f0:  31 cd0e  64 c820  14 8df0  2b cd0e
10000100:
```

การตั้งค่าความกว้างของเอาต์พุต

ใช้คำสั่งย่อย **$w** เพื่อตั้งค่าความกว้างสูงสุด (เป็น อักขระ) ของแต่ละบรรทัดของเอาต์พุตที่สร้างโดยโปรแกรม **adb** คำสั่งมีฟอร์ม:

*Width***$w**

ในรูปแบบนี้พารามิเตอร์ *Width* เป็นจำนวนเต็ม ที่ระบุความกว้างเป็นอักขระของการแสดงผล คุณสามารถระบุความกว้าง ที่เหมาะสมกับอุปกรณ์แสดงผลของคุณ เมื่อโปรแกรม **adb** ถูกเรียกใช้ครั้งแรก ความกว้างเริ่มต้นคือ 80 อักขระ

คำสั่งนี้สามารถถูกใช้เมื่อการเปลี่ยนทิศทางเอาต์พุตไปที่พรินเตอร์รายบรรทัดหรือ อุปกรณ์เอาต์พุตพิเศษ ตัวอย่างเช่น คำสั่งดังต่อไปนี้ตั้งค่าความกว้างการแสดงผล เป็น 120 อักขระ ความกว้างสูงสุดทั่วไปสำหรับพรินเตอร์รายบรรทัด:

```
120$w
```

การตั้งค่า offset สูงสุด

โปรแกรมดีบัก **adb** โดยปกติแสดงหน่วยความจำและ แอดเดรสไฟล์เป็นผลรวมของสัญลักษณ์และออฟเซ็ต รูปแบบนี้ช่วยในการ เชื่อมโยงคำสั่งและข้อมูลบนจอแสดงผลด้วยฟังก์ชันหรือ ตัวแปรเฉพาะ เมื่อโปรแกรม **adb** เริ่มต้นจะตั้งค่า ออฟเซ็ตสูงสุดเป็น 255 ดังนั้นแอดเดรสเชิงสัญลักษณ์ถูกกำหนดให้เฉพาะกับ คำสั่งหรือข้อมูลที่เกิดขึ้นน้อยกว่า 256 ไบต์จากจุดเริ่มต้นของฟังก์ชันหรือตัวแปร คำสั่งหรือข้อมูลนอกเหนือจากจุดนี้จะถูกกำหนด แอดเดรสตัวเลข

ในหลายโปรแกรม ขนาดของฟังก์ชันหรือตัวแปรใหญ่กว่า 255 ไบต์ ด้วยเหตุผลนี้โปรแกรม **adb** อนุญาต ให้คุณเปลี่ยนออฟเซ็ตสูงสุดเพื่อให้เหมาะกับโปรแกรมที่ใหญ่กว่า คุณสามารถเปลี่ยน ออฟเซ็ตสูงสุดโดยใช้คำสั่งย่อย **$s**

คำสั่งย่อยมีรูปแบบ:

*Offset***$s**

ในรูปแบบนี้พารามิเตอร์ *Offset* เป็นจำนวนเต็ม ที่ระบุออฟเซ็ตใหม่ ตัวอย่างเช่นคำสั่งดังต่อไปนี้เพิ่ม ออฟเซ็ตสูงสุดที่เป็นไปได้เป็น 4095:

```
4095$s
```

คำสั่งและข้อมูลทั้งหมดที่น้อยกว่า 4096 ไบต์ถูกกำหนด แอดเดรสเชิงสัญลักษณ์ คุณสามารถปิดใช้งานการกำหนดแอดเดรสเชิงสัญลักษณ์ทั้งหมดโดย ตั้งค่าออฟเซ็ตสูงสุดเป็นศูนย์ แอดเดรสทั้งหมดถูกกำหนดเป็นค่าตัวเลขแทน

การตั้งค่ารูปแบบดีฟอลต์

เมื่อต้องการเปลี่ยนแปลงรูปแบบเริ่มต้นสำหรับตัวเลขที่ใช้ในคำสั่งให้ใช้ **$d** หรือคำสั่งย่อย **$o** (ฐานแปด) รูปแบบเริ่มต้นแจ้งวิธีกับโปรแกรมดีบัก **adb** ในการแปลตัวเลข ที่ไม่ได้เริ่มต้นด้วย 0 (ฐานแปด) หรือ 0x (เลขฐานสิบหก) และวิธีในการแสดงตัวเลข เมื่อไม่มีการระบุรูปแบบ ใช้คำสั่งเหล่านี้ในการทำงานกับการรวมกัน ของตัวเลข ฐานสิบ ฐานแปด และเลขฐานสิบหก

คำสั่งย่อย **$o** ตั้งค่าฐานเป็น 8 และตั้งค่า รูปแบบเริ่มต้นสำหรับตัวเลขที่ใช้ในคำสั่งเป็นฐานแปด หลังจากคุณป้อน คำสั่งย่อยนั้นโปรแกรม **adb** แสดงตัวเลขทั้งหมด ในรูปแบบฐานแปด ยกเว้นที่ระบุในรูปแบบอื่น

รูปแบบสำหรับคำสั่งย่อย **$d** คือคำสั่ง *Radix***$d** ซึ่งพารามิเตอร์ *Radix* คือค่าใหม่ของเลขฐาน ถ้าพารามิเตอร์ *Radix* ไม่ถูกระบุ คำสั่งย่อย **$d** ตั้งค่าฐานเป็นค่าเริ่มต้น 16 เมื่อคุณเริ่มโปรแกรม **adb** ครั้งแรกรูปแบบเริ่มต้นคือเลขฐานสิบหก ถ้าคุณเปลี่ยนรูปแบบเริ่มต้น คุณสามารถคืนค่ารูปแบบได้ โดยป้อนคำสั่งย่อย **$d** โดยตัวของมันเอง:

```
$d
```

เมื่อต้องการกำหนดรูปแบบเริ่มต้นเป็นเลขฐานสิบ ให้ใช้คำสั่งดังต่อไปนี้:

```
0xa$d
```

การเปลี่ยนโหมด disassembly

ใช้คำสั่งย่อย **$i** และ **$n** เพื่อบังคับโปรแกรมดีบัก **adb** เพื่อถอดแยกคำสั่ง โดยใช้ชุดคำสั่งและคำย่อที่ระบุ คำสั่งย่อย **$i** ระบุชุดคำสั่งที่จะถูกใช้สำหรับถอดแยกภาษาแอสเซมบลี คำสั่งย่อย **$n** ระบุคำย่อที่จะถูกใช้ในการถอดแยกภาษาแอสเซมบลี

ถ้าไม่มีการป้อนค่า คำสั่งเหล่านี้แสดงการตั้งค่าปัจจุบัน

คำสั่งย่อย **$i** ยอมรับค่าดังต่อไปนี้:

com
ระบุชุดคำสั่งสำหรับโหมด intersection ทั่วไปของ PowerPC® และ POWER® family

pwr
ระบุชุดคำสั่งและคำย่อสำหรับการนำ POWER ของ POWER Architecture ไปปฏิบัติ

pwrx
ระบุชุดคำสั่งและ mnemonics สำหรับการนำ POWER2 ไปใช้ของ POWER family

ppc
ระบุชุดคำสั่งและคำย่อสำหรับ PowerPC

601
ระบุชุดคำสั่งและคำย่อสำหรับ PowerPC 601 RISC Microprocessor

603
ระบุชุดคำสั่งและคำย่อสำหรับ PowerPC 603 RISC Microprocessor

604
ระบุชุดคำสั่งและคำย่อสำหรับ PowerPC 604™ RISC Microprocessor

ANY
ระบุคำสั่งที่ถูกต้อง สำหรับชุดคำสั่ง ที่ซ้อนทับ ตัวย่อจะมีค่าเริ่มต้นเป็นตัวย่อ PowerPC

คำสั่งย่อย **$n** ยอมรับ ค่าดังต่อไปนี้:

pwr
ระบุชุดคำสั่งและคำย่อสำหรับการนำ POWER ของ POWER Architecture ไปปฏิบัติ

ppc
ระบุคำย่อสำหรับสถาปัตยกรรม PowerPC

การคำนวณตัวเลขและการแสดงข้อความ

คุณสามารถทำการคำนวณทางคณิตศาสตร์ ขณะอยู่ในโปรแกรมดีบัก **adb** โดยใช้คำสั่งย่อย = (เครื่องหมายเท่ากับ) คำสั่งนี้กำหนดโปรแกรม **adb** ให้แสดง ค่าของนิพจน์ในรูปแบบที่ระบุ

คำสั่งแปลงตัวเลขจากฐานหนึ่ง ไปเป็นฐานอื่น ตรวจสอบซ้ำเกี่ยวกับคณิตศาสตร์ที่ดำเนินการโดยโปรแกรม และแสดงแอดเดรสที่ซับซ้อนในแบบง่าย ตัวอย่างเช่น คำสั่ง ดังต่อไปนี้แสดงเลขฐานสิบหก 0x2a เป็นเลขฐานสิบ 42:

```
0x2a=d
        42
```

เช่นเดียวกัน คำสั่งดังต่อไปนี้แสดง 0x2a เป็นอักขระ ASCII * (ดอกจัน):

```
0x2a=c
        *
```

นิพจน์ในคำสั่งสามารถใช้การรวมกันของสัญลักษณ์และตัวดำเนินการ ตัวอย่างเช่น คำสั่งต่อไปนี้จะคำนวณค่าโดยใช้ เนื้อหาของรีจิสเตอร์ **r0** และ **r1** และตัวแปร **adb b**

```
<r0-12*<r1+<b+5=X
    8fa86f95
```

คุณยังสามารถคำนวณค่าของสัญลักษณ์ภายนอกเพื่อตรวจสอบค่าเลขฐานสิบหก ของแอดเดรสสัญลักษณ์ภายนอก โดยการป้อน:

```
main+5=X
    2000038d
```

คำสั่งย่อย = (เครื่องหมายเท่ากับ) ยังสามารถแสดง สตริงตามตัวอักษร ใช้คุณลักษณะนี้ในสคริปต์โปรแกรม **adb** เพื่อ แสดงข้อคิดเห็นเกี่ยวกับสคริปต์ตามการดำเนินการคำสั่งของสคริปต์ ตัวอย่างเช่น คำสั่งย่อยดังต่อไปนี้สร้างที่ว่างสามบรรทัดแล้วพิมพ์ข้อความ `C Stack Backtrace`:

```
=3n"C Stack Backtrace"
```

การแสดงและการจัดการไฟล์ต้นฉบับด้วยโปรแกรม adb

ส่วนนี้อธิบายวิธีใช้โปรแกรม **adb** เพื่อ แสดงและจัดการไฟล์ต้นฉบับ

การแสดงคำสั่งและข้อมูล

โปรแกรม **adb** จัดเตรียมหลายคำสั่งย่อยสำหรับ การแสดงคำสั่งและข้อมูลของโปรแกรมที่กำหนดและข้อมูลของ ไฟล์ข้อมูลที่กำหนด คำสั่งย่อยและรูปแบบของคำสั่งคือ:

แสดงแอดเดรส

Address [, *Count*] **=** *Format*

แสดงคำสั่ง

Address [, *Count*] **?** *Format*

แสดงค่าของตัวแปร

Address [, *Count*] **/** *Format*

ในรูปแบบนี้ สัญลักษณ์และตัวแปรมีความหมายดังต่อไปนี้:

`Address`
ให้ที่อยู่ของคำสั่งหรือรายการข้อมูล

`Count`
ให้จำนวนของรายการที่จะแสดง

`Format`
กำหนดวิธีแสดงรายการ

= แสดงแอดเดรสของรายการ

? แสดงคำสั่งในเซ็กเมนต์ข้อความ

/ แสดงค่าของตัวแปร

การจัดฟอร์มแอดเดรส

ในโปรแกรม **adb** แอดเดรสเป็นค่า 32-บิตที่บ่งชี้ แอดเดรสหน่วยความจำ อย่างไรก็ตาม ข้อมูลนี้ถูกแสดงได้ใน ฟอร์มดังต่อไปนี้:

`Absolute address`

ค่า 32-บิตถูกแสดงโดยตัวเลขฐานสิบหก 8-หลัก หรือค่าที่เท่ากันในระบบตัวเลขฐานอื่น

ชื่อสัญลักษณ์

ตำแหน่งของสัญลักษณ์ที่กำหนดในโปรแกรมสามารถถูกแสดง ตามชื่อของสัญลักษณ์นั้นในโปรแกรม

`Entry points`

entry point เข้าสู่รูทีนถูกแสดงโดยชื่อของรูทีนนำหน้า ด้วย . (จุด) ตัวอย่างเช่น เมื่ออ้างอิงถึงแอดเดรสของจุดเริ่มต้น ของรูทีน `main` ให้ใช้เครื่องหมายดังนี้:

```
.main
```

การย้ายตำแหน่ง

จุดอื่นในโปรแกรมสามารถถูกอ้างอิงได้โดยใช้การย้ายตำแหน่ง จาก entry points ในโปรแกรม ตัวอย่างเช่น เครื่องหมายดังต่อไปนี้อ้างอิง คำสั่งซึ่งคือ 4 ไบต์จาก entry point สำหรับสัญลักษณ์ `main`:

```
.main+4
```

การแสดงแอดเดรส

ใช้คำสั่งย่อย = (เครื่องหมายเท่ากับ) เพื่อแสดงแอดเดรสใน รูปแบบที่กำหนด คำสั่งนี้แสดงคำสั่งและแอดเดรสข้อมูลใน รูปแบบที่ธรรมดากว่าและสามารถแสดงผลลัพธ์ของนิพจน์เชิงคำนวณ ตัวอย่างเช่น การป้อน:

```
main=an
```

แสดงแอดเดรสของสัญลักษณ์ `main`:

```
10000370:
```

ตัวอย่างดังต่อไปนี้แสดงคำสั่งที่แสดง (ฐานสิบ) ผลรวม ของตัวแปรจำนวนเต็ม **b** และค่าเลขฐานสิบหก 0x2000, พร้อมด้วยเอาต์พุต:

```
<b+0x2000=D
    268443648
```

ถ้ามีการกำหนดการนับ ค่าเดิมจะถูกทำซ้ำตามจำนวนที่กำหนด ตัวอย่าง ดังต่อไปนี้แสดงคำสั่งที่แสดงค่าของ `main` สองครั้งและเอาต์พุตผลลัพธ์:

```
main,2=x
    370 370
```

ถ้าไม่มีการกำหนดแอดเดรส แอดเดรสปัจจุบันจะถูกใช้ หลังจากรันคำสั่ง ด้านบนหนึ่งครั้ง (ตั้งค่าแอดเดรสปัจจุบันเป็น `main`) คำสั่งดังต่อไปนี้ทำซ้ำฟังก์ชัน:

```
,2=x
    370 370
```

ถ้าคุณไม่ได้ระบุรูปแบบโปรแกรมดีบัก **adb** ใช้รูปแบบสุดท้ายที่ถูกใช้กับคำสั่งนี้ ตัวอย่างเช่น ในลำดับ ของคำสั่งดังต่อไปนี้ ทั้ง `main` และ `one` ถูกแสดงในเลขฐานสิบหก:

```
main=x
  370
one=
  33c
```

การแสดง C stack backtrace

เมื่อต้องการติดตามพาธของฟังก์ชันที่ใช้งานอยู่ทั้งหมด ใช้คำสั่งย่อย **$c** คำสั่งย่อยนี้แสดงชื่อของฟังก์ชันทั้งหมดที่ถูกเรียกและยังไม่ส่งกลับการควบคุม และยังแสดงแอดเดรสจาก แต่ละฟังก์ชันที่ถูกเรียกและอาร์กิวเมนต์ที่สองไปที่แต่ละฟังก์ชัน ตัวอย่างเช่น ลำดับคำสั่งดังต่อไปนี้ตั้งค่าจุดหยุดที่ฟังก์ชันแอดเดรส .f+2 ในโปรแกรม **adbsamp2** จุดหยุดเรียกคำสั่งย่อย **$c** โปรแกรมถูกเริ่มต้น รันไปถึงจุดหยุด แล้วแสดง backtrace ของฟังก์ชันภาษา C ที่เรียก:

```
.f+2:b$c
:r
adbsamp2: running
.f(0,0) .main+26
.main(0,0,0) start+fa
breakpoint              f+2:          tgte     r2,r2
```

โดยค่าเริ่มต้นคำสั่งย่อย **$c** แสดงการเรียกทั้งหมด เมื่อต้องการแสดงการเรียกที่น้อยลง ให้ระบุจำนวนของการเรียกที่จะแสดง ตัวอย่างเช่น คำสั่งดังต่อไปนี้แสดงเฉพาะหนึ่งในฟังก์ชันที่ใช้งานอยู่ ที่จุดหยุดก่อนหน้านี้:

```
,1$c
.f(0,0) .main+26
```

การเลือกรูปแบบข้อมูล

รูปแบบ คือตัวอักษรหรืออักขระที่กำหนด วิธีที่ข้อมูลจะถูกแสดง ต่อไปนี้เป็นรูปแบบที่ใช้ทั่วไป:

- `a` แอดเดรสสัญลักษณ์ปัจจุบัน
- `b` หนึ่งไบต์ในฐานแปด (แสดงข้อมูลที่สัมพันธ์กับคำสั่ง หรือ ไบต์สูงหรือต่ำของรีจิสเตอร์)
- `c` หนึ่งไบต์เป็นอักขระ (ตัวแปร char)
- `d` Halfword เป็นเลขฐานสิบ (ตัวแปร short)
- `D` Fullword เป็นเลขฐานสิบ (ตัวแปร long)
- `i` คำสั่งเครื่องในรูปแบบตัวย่อ
- `n` บรรทัดใหม่
- `o` Halfword เป็นฐานแปด (ตัวแปร short)
- `O` Fullword เป็นฐานแปด (ตัวแปร long)
- `r` ช่องว่าง
- `s` สตริงอักขระปิดท้ายด้วย null (อาร์เรย์ที่ปิดท้ายด้วย null ของตัวแปร char)
- `t` แท็บแนวนอน
- `u` Halfword เป็น unsigned integer (ตัวแปร short)
- `x` Halfword เป็นเลขฐานสิบหก (ตัวแปร short)
- `X` Fullword เป็นเลขฐานสิบหก (ตัวแปร long)

ตัวอย่างเช่น คำสั่งดังต่อไปนี้สร้างเอาต์พุตดังกล่าว เมื่อใช้โปรแกรมตัวอย่าง **adbsamp**:

```
main=o
    1560

main=O
    4000001560

main=d
    880

main=D
    536871792

main=x
    370

main=X
    20000370

main=u
    880
```

รูปแบบสามารถถูกใช้โดยตัวเองหรือรวมกับรูปแบบอื่นเพื่อ แสดงการรวมกันของข้อมูลฟอร์มต่างกัน คุณสามารถรวมรูปแบบ **a, n, r** และ **t** กับรูปแบบอื่นเพื่อทำให้การแสดงผลอ่านได้ง่ายขึ้น

การเปลี่ยนการแม็พหน่วยความจำ

คุณสามารถเปลี่ยนค่าของแม็พหน่วยความจำโดยใช้คำสั่งย่อย **?m** และ **/m** คำสั่งเหล่านี้ค่าที่ระบุให้กับรายการแม็พที่ตรงกัน คำสั่งมีรูปแบบ:

```
[,count] ?m b1 e1 f1
[,count] /m b1 e1 f2
```

ตัวอย่างดังต่อไปนี้แสดงผลลัพธ์ของคำสั่งเหล่านี้บแม็พหน่วยความจำ ที่แสดงด้วยคำสั่งย่อย **$m** ในตัวอย่าง ก่อนหน้า:

```
,0?m    10000100         10000470       0
/m      100      100     100
$m
  [0] :   ?map :   'adbsamp3'
 b1 = 0x10000100,  e1 = 10000470,  f1 = 0
 b2 = 0x20000600,  e2 = 0x2002c8a4, f2 = 0x600

 [1] :  ?map :   'shr.o' in library '/usr/ccs/lib/libc.a'
 b1 = 0xd00d6200,  e1 = 0xd01397bf,  f1 = 0xd00defbc
 b2 = 0x20000600,  e2 = 0x2002beb8, f2 = 0x4a36c

  [-] :  /map :  '-'
 b1 = 100,    e1 = 100,   f1 = 100
 b2 = 0,    e2 = 0,     f2 = 0
```

เมื่อต้องการเปลี่ยนค่าเซ็กเมนต์ข้อมูล ให้เพิ่ม * (ดอกจัน) หลับ / หรือ ?

```
,0?*m   20000270         20000374       270
/*m     200      200     200
$m
```

```
 [0] : ?map :   'adbsamp3'
b1 = 0x10000100,  e1 = 10000470,  f1 = 0
b2 = 0x20000270,  e2 = 0x20000374, f2 = 0x270

[1] :  ?map :   'shr.o' in library '/usr/ccs/lib/libc.a'
b1 = 0xd00d6200,  e1 = 0xd01397bf,  f1 = 0xd00defbc
b2 = 0x20000600,  e2 = 0x2002beb8, f2 = 0x4a36c

 [-] :  /map :  '-'
b1 = 100,    e1 = 100,   f1 = 100
b2 = 0,    e2 = 0,     f2 = 0
```

การแพตช์ไบนารีไฟล์

คุณสามารถทำการแก้ไขหรือเปลี่ยนแปลงไฟล์ รวมถึงไบนารีไฟล์ที่รันได้ โดยเริ่มต้นโปรแกรม **adb** ด้วยแฟล็ก **–w** และโดยใช้คำสั่งย่อย **w** และ **W** ()

การหาค่าในไฟล์

ค้นหาค่าในไฟล์โดยใช้คำสั่งย่อย **I** และ **L** คำสั่งย่อย มีฟอร์ม:

?I *Value*

OR

/I *Value*

การค้นหาเริ่มต้นที่แอดเดรสปัจจุบันและค้นหานิพจน์ที่ระบุ โดย *Value* คำสั่งย่อย **I** ค้นหา ค่า 2–ไบต์ คำสั่งย่อย **L** ค้นหา ค่า 4–ไบต์

คำสั่งย่อย **?I** เริ่มการค้นหาที่แอดเดรสปัจจุบัน และต่อไปจนถึงพบข้อมูลครั้งแรกหรือจนถึงตำแหน่งสิ้นสุดไฟล์ ถ้าพบ ค่าแอดเดรสปัจจุบันถูกตั้งค่าเป็นแอดเดรสของค่า ตัวอย่างเช่น คำสั่งดังต่อไปนี้ค้นหาการตำแหน่งแรกของสัญลักษณ์ **f** ในไฟล์ **adbsamp2**:

```
?I .f.
write+a2
```

พบค่าที่ **.write+a2** และแอดเดรสปัจจุบัน ถูกตั้งค่าที่แอดเดรสนั้น

การเขียนลงในไฟล์

เขียนไปที่ไฟล์โดยใช้คำสั่งย่อย **w** และ **W** คำสั่งย่อย มีฟอร์ม:

[*Address*] **?w** *Value*

ในรูปแบบนี้พารามิเตอร์ *Address* เป็นแอดเดรส ของค่าที่คุณต้องการเปลี่ยนแปลง และพารามิเตอร์ *Value* เป็นค่าใหม่ คำสั่งย่อย **w** เขียนค่า 2–ไบต์ คำสั่งย่อย **W** เขียนค่า 4–ไบต์ ตัวอย่างเช่น คำสั่งดังต่อไปนี้เปลี่ยนคำว่า "This" เป็น "The":

```
?I .Th.
?W .The.
```

คำสั่งย่อย **W** เปลี่ยนอักขระทั้งสี่ตัว

การทำการเปลี่ยนแปลงกับหน่วยความจำ

ทำการเปลี่ยนแปลงกับหน่วยความจำเมื่อโปรแกรมถูกรัน ถ้าคุณได้ใช้คำสั่งย่อย **:r** กับจุดหยุดเพื่อเริ่มการดำเนินการ คำสั่งย่อย **w** ต่อมาจะทำให้โปรแกรม **adb** เขียนข้อมูลไปที่โปรแกรมในหน่วยความจำไม่ได้เขียนไปที่ไฟล์ คำสั่งนี้ถูกใช้เพื่อทำการเปลี่ยนแปลงกับข้อมูลโปรแกรมขณะที่รัน เช่น การเปลี่ยนค่าชั่วคราวของ แฟล็กหรือตัวแปรของโปรแกรม

การใช้ตัวแปร adb

โปรแกรมดีบัก **adb** สร้างชุดของตัวแปร ของตัวเองโดยอัตโนมัติ เมื่อโปรแกรมเริ่มการทำงาน ตัวแปรเหล่านี้ถูกตั้งค่าเป็นแอดเดรส และขนาดของส่วนต่างๆ ของโปรแกรมไฟล์ตามที่กำหนด ในตารางดังต่อไปนี้:

0 ค่าล่าสุดที่พิมพ์

1 ส่วนที่ย้ายออกล่าสุดของซอร์สคำสั่ง

2 ค่าก่อนหน้าของตัวแปร **1**

9 จำนวนของ **$<** ล่าสุดหรือคำสั่ง **$<<**

b แอดเดรสฐานของเซ็กเมนต์ข้อมูล

d ขนาดของเซ็กเมนต์ข้อมูล

e แอดเดรสรายการของโปรแกรม

m ตัวเลข "Magic"

s ขนาดของเซ็กเมนต์สแตก

t ขนาดของเซ็กเมนต์ข้อความ

โปรแกรมดีบัก **adb** อ่านโปรแกรมไฟล์เพื่อค้นหา ค่าสำหรับตัวแปรเหล่านี้ ถ้าไฟล์ไม่ใช่ไฟล์ของโปรแกรม โปรแกรม **adb** จะปล่อยค่าไว้เป็น undefined

เมื่อต้องการแสดงค่าที่โปรแกรมดีบัก **adb** กำหนดให้กับตัวแปรเหล่านี้ ให้ใช้คำสั่งย่อย **$v** คำสั่งย่อยนี้แสดง ชื่อตัวแปรตามด้วยค่าของตัวแปรในรูปแบบปัจจุบัน คำสั่งย่อย แสดงตัวแปรที่มีค่าไม่เป็น 0 (ศูนย์) และถ้าตัวแปรมี ค่าเซ็กเมนต์ที่ไม่ใช่ศูนย์ ค่าของตัวแปรจะถูกแสดงเป็นแอดเดรส หรือไม่แล้ว จะถูกแสดงเป็นตัวเลข ตัวอย่างดังต่อไปนี้แสดงการใช้ คำสั่งเพื่อแสดงค่าตัวแปรสำหรับโปรแกรมตัวอย่าง **adbsamp**:

```
$v
ตัวแปร
0 = undefined
1 = undefined
2 = undefined
9 = undefined
b = 10000000
d = 130
e = 10000038
```

```
m = 108
t = 298
```

ระบุค่าปัจจุบันของตัวแปร **adb** ใน นิพจน์โดยนำหน้าชื่อตัวแปรด้วย < (เครื่องหมายน้อยกว่า) ตัวอย่าง ดังต่อไปนี้แสดงค่าปัจจุบันของตัวแปรฐาน **b**:

```
<b=X
10000000
```

สร้างตัวแปรของคุณเองหรือเปลี่ยนค่าของตัวแปรที่มีอยู่แล้วโดย กำหนดค่าให้กับชื่อตัวแปรด้วย > (เครื่องหมายมากกว่า) การกำหนดค่า มีฟอร์ม:

Expression > *VariableName*

โดยที่พารามิเตอร์ *Expression* คือค่าที่จะ ถูกกำหนดให้กับตัวแปรและพารามิเตอร์ *VariableName* คือตัวแปรที่รับค่า พารามิเตอร์ *VariableName* ต้องเป็นตัวอักษรเดี่ยว ตัวอย่างเช่น การกำหนดค่า:

```
0x2000>b
```

กำหนดค่าเลขฐานสิบหก 0x2000 ให้กับตัวแปร **b** แสดงเนื้อหาของ **b** อีกครั้งเพื่อแสดงว่าการกำหนดค่า เกิดขึ้น:

```
<b=X
 2000
```

การหาแอดเดรสปัจจุบัน

โปรแกรม **adb** มีสองค่าพิเศษที่ติดตาม แอดเดรสล่าสุดที่ใช้ในคำสั่งและแอดเดรสล่าสุดที่พิมพ์ด้วย คำสั่ง ตัวแปร . (จุด) ตัวแปร เรียกอีกอย่างว่า แอดเดรสปัจจุบัน มีแอดเดรสล่าสุดที่ใช้ในคำสั่ง ตัวแปร ″ (เครื่องหมายคำพูดคู่) มีแอดเดรสล่าสุดที่พิมพ์ด้วย คำสั่ง . และ ″ โดยปกติมีแอดเดรสเหมือนกัน ยกเว้นคำสั่งโดยนัย เช่น newline และอักขระ ^ (คาเร็ท) ถูกใช้ อักขระเหล่านี้เพิ่มและ ลดตัวแปร . โดยอัตโนมัติ แต่ไม่เปลี่ยนแปลงตัวแปร) (วงเล็บปิด)

ทั้งตัวแปร . และ ″ สามารถ ถูกใช้ในนิพจน์ทั้งหมด ไม่จำเป็นต้องมี < (เครื่องหมายน้อยกว่า) ตัวอย่างเช่น คำสั่งต่อไปนี้แสดงตัวแปรเหล่านี้ที่เริ่มต้นของการดีบัก ด้วยโปรแกรม **adbsamp**:

```
.=
    0.
=
    0
```

การแสดงตัวแปรภายนอก

ใช้คำสั่งย่อย **$e** เพื่อแสดงค่าของตัวแปรภายนอกทั้งหมด ในโปรแกรม **adb** ตัวแปรภายนอกคือตัวแปรในโปรแกรมของคุณที่มีสโคปโกลบอลหรือได้ถูกกำหนดภายนอกฟังก์ชัน และรวมถึงตัวแปร ที่กำหนดในไลบรารีรูทีนที่ใช้โดยโปรแกรมของคุณ เช่นเดียวกับตัวแปร ภายนอกทั้งหมดของไลบรารีที่แบ่งใช้

คำสั่งย่อย **$e** มีประโยชน์ในการในการรับรายการ ของชื่อสำหรับตัวแปรที่มีทั้งหมดหรือผลรวมของค่าของตัวแปร คำสั่ง แสดงหนึ่งชื่อบนแต่ละบรรทัดพร้อมกับค่าของตัวแปร (ถ้ามี) บนบรรทัดเดียวกัน ถ้าพารามิเตอร์ *Count* ถูกระบุ เฉพาะตัวแปรภายนอกที่สัมพันธ์กับไฟล์จะถูกพิมพ์

ตัวอย่างต่อไปนี้แสดงการตั้งค่า breakpoint ในโปรแกรมตัวอย่าง **adbsamp2** ที่เรียกใช้คำสั่งย่อย **$e** และเอาต์พุตผลลัพธ์เมื่อโปรแกรมรัน (ต้องแน่ใจว่าลบ breakpoints ใดๆ ที่คุณตั้งไว้ก่อนหน้านี้):

```
.f+2:b,0$e
:r
adbsamp2: running
_errno: 0
_environ:  3fffe6bc
__NLinit:  10000238
_main: 100001ea
_exit: 1000028c
_fcnt: 0
_loop_count:  1
_f:   100001b4
_NLgetfile: 10000280
_write: 100002e0
__NLinit__X:  10000238
_NLgetfile__X:   10000280
__cleanup: 100002bc
__exit: 100002c8
_exit__X:  1000028c
__cleanup__X:  100002bc
breakpoint .f+2:  st r2,1c(r1)
```

การแสดงการแม็พแอดเดรส

โปรแกรม **adb** เตรียมชุดของแม็พสำหรับข้อความ และเซ็กเมนต์ข้อมูลในโปรแกรมของคุณและใช้แม็พเหล่านี้เพื่อเข้าถึงรายการ ที่คุณร้องขอให้แสดง ใช้คำสั่งย่อย **$m** เพื่อแสดง เนื้อหาของแอดเดรสแม็พ คำสั่งย่อยแสดงแม็พสำหรับเซ็กเมนต์ทั้งหมดในโปรแกรมและใช้ ข้อมูลที่นำมาจากโปรแกรมและคอร์ไฟล์หรือรับตรงมา จากหน่วยความจำ

คำสั่งย่อย **$m** แสดงข้อมูลเหมือนดัง ต่อไปนี้:

```
$m
  [0] : ?map :   'adbsamp3'
  b1 = 0x10000200,  e1 = 0x10001839,  f1 = 0x10000200
  b2 = 0x2002c604,  e2 = 0x2002c8a4,  f2 = 0x600

  [1] : ?map :   'shr.o' in library 'lib/libc.a'
  b1 = 0xd00d6200,  e1 = 0xd013976f,  f1 = 0xd00defbc
  b2 = 0x20000600,  e2 = 0x2002bcb8,  f2 = 0x4a36c

  [-] :  /map :  '-'
  b1 = 0x0000000,  e1 = 0x00000000,  f1 = 0x00000000
  b2 = 0x0000000,  e2 = 0x00000000,  f2 = 0x00000000
```

การแสดงกำหนดพารามิเตอร์ address-mapping สำหรับข้อความ (b1, e1 และ f1) และเซ็กเมนต์ข้อมูล (b2, e2 และ f2) สำหรับสองไฟล์ที่กำลังถูกใช้โดยโปรแกรมดีบัก **adb** ตัวอย่างนี้แสดงค่าของ โปรแกรมตัวอย่าง **adbsamp3** เท่านั้น ชุดที่สองของค่าแม็พสำหรับคอร์ไฟล์ที่ใช้อยู่ เนื่องจากไม่มีการใช้อยู่ ตัวอย่างแสดงชื่อไฟล์เป็น – (ขีด)

ค่าที่แสดงในวงเล็บเหลี่ยมสามารถถูกใช้เป็นพารามิเตอร์ *Count* ในคำสั่งย่อย **?e** และ **?m**

ข้อมูลอ้างอิงโปรแกรมดีบัก adb

โปรแกรมดีบัก **adb** ใช้แอดเดรส นิพจน์ ตัวดำเนินการคำสั่งย่อยและตัวแปรเพื่อจัดการและควบคุมข้อมูล

แอดเดรสโปรแกรมดีบัก adb

แอดเดรสในไฟล์ที่สัมพันธ์กับแอดเดรสที่เขียน ถูกกำหนดโดยการแม็พที่สัมพันธ์กับไฟล์นั้น แต่ละการแม็พถูกแสดงโดยสองทริปเปิล (*B1*, *E1*, *F1*) และ (*B2*, *E2*, *F2*) พารามิเตอร์ *FileAddress* ที่ตรงกับพารามิเตอร์ *Address* ที่เขียน ถูกคำนวณดังนี้:

B1<=*Address*<*E1*=>*FileAddress*=*Address*+*F1*−*B1*

OR

B2<=*Address*<*E2*=>*FileAddress*=*Address*+*F2*−*B2*

ถ้าพารามิเตอร์ *Address* ที่ร้องขอ ไม่ได้อยู่ระหว่าง *B1* และ *E1* หรือ ไม่อยู่ระหว่าง *B2* และ *E2* พารามิเตอร์ *Address* จะไม่ถูกต้อง ในบางกรณีเช่น โปรแกรมที่มี I และ D space แยกกัน สองเซ็กเมนต์สำหรับไฟล์อาจซ้อนทับกัน ถ้า ? (เครื่องหมายคำถาม) หรือ / (สแลช) คำสั่งย่อยตามด้วย * (เครื่องหมายดอกจัน) เฉพาะ ทริปเปิดที่สองจะถูกใช้

การตั้งค่าเริ่มต้นของการแม็พทั้งสองใช้ได้สำหรับ ไฟล์ **a.out** และ **core** ปกติ ถ้าไฟล์ ใดไฟล์หนึ่งไม่ได้เป็นชนิดที่ต้องการ พารามิเตอร์ *B1* สำหรับ ไฟล์นั้นจะถูกกำหนดเป็นค่า 0 พารามิเตอร์ *E1* ถูกกำหนด เป็นขนาดไฟล์สูงสุด และพารามิเตอร์ *F1* ถูกกำหนด เป็นค่า 0 ด้วยวิธีนี้ไฟล์ทั้งหมดสามารถถูกตรวจสอบโดยไม่ต้องมี การแปลแอดเดรส

นิพจน์โปรแกรมดีบัก adb

นิพจน์ดังต่อไปนี้ได้รับการสนับสนุนโดยโปรแกรมดีบัก**adb**:

. (จุด)
ระบุแอดเดรสสุดท้ายที่ใช้โดยคำสั่งย่อย แอดเดรสสุดท้าย หรือเรียกอีกอย่างเป็นแอดเดรสปัจจุบัน

+ (บวก)
เพิ่มค่าของ . (จุด) ด้วยส่วนเพิ่มปัจจุบัน

^ (caret)
ลดค่าของ . (จุด) ด้วยส่วนเพิ่มปัจจุบัน

" (เครื่องหมายคำพูดคู่)
ระบุแอดเดรสสุดท้ายที่พิมพ์โดยคำสั่ง

จำนวนเต็ม
ระบุเลขฐานแปดถ้าพารามิเตอร์นี้เริ่มด้วย **0o** เลขฐานสิบหกนำหน้าด้วย **0x** หรือ # หรือเลขฐานสิบ ถ้านำหน้าด้วย **0t** หรือมิฉะนั้นนิพจน์นี้ระบุตัวเลขที่แปลใน ฐานปัจจุบัน ค่าเริ่มต้นฐานคือ 16

` Cccc '
ระบุค่า ASCII ได้ถึง 4 อักขระ\(backslash) สามารถถูกใช้เพื่อ escape เครื่องหมาย ' (อะโพสโทรฟี)

< Name

อ่านค่าปัจจุบันของพารามิเตอร์ *Name* ชื่อ *Name* เป็นชื่อตัวแปรหรือ ชื่อรีจิสเตอร์ คำสั่ง **adb** รักษาจำนวนตัวแปรที่ตั้งชื่อโดยตัวอักษรหรือตัวเลขเดี่ยว ถ้าพารามิเตอร์ *Name* เป็นชื่อรีจิสเตอร์ ค่าของรีจิสเตอร์ ได้รับจากเฮดเดอร์ระบบในพารามิเตอร์ *CoreFile* ใช้คำสั่งย่อย **$r** เพื่อดูชื่อรีจิสเตอร์ที่ถูกต้อง

สัญลักษณ์

ระบุลำดับของลำดับตัวอักษรตัวพิมพ์ใหญ่หรือตัวพิมพ์เล็ก เครื่องหมายขีดเส้นใต้ หรือตัวเลข แม้ว่าลำดับไม่สามารถขึ้นต้นด้วยตัวเลข ค่าของพารามิเตอร์ *Symbol* ถูกนำมาจากตารางสัญลักษณ์ในพารามิเตอร์ *ObjectFile* (เครื่องหมายขีดล่าง) เริ่มต้นถูกนำมาเป็น ส่วนเติมหน้าให้กับพารามิเตอร์ *Symbol* ถ้าจำเป็น

. สัญลักษณ์

ระบุ entry point ของฟังก์ชันทั้งตั้งชื่อโดยพารามิเตอร์ *Symbol*

Routine.Name

ระบุแอดเดรสของพารามิเตอร์ *Name* ในรูทีนภาษา C ที่ระบุ ทั้งพารามิเตอร์ *Routine* และ *Name* เป็นพารามิเตอร์ *Symbol* ถ้าพารามิเตอร์ *Name* ถูกเว้นไป ค่าเป็นแอดเดรส ของ C stack frame ที่เปิดใช้งานล่าสุดตามพารามิเตอร์ *Routine*

(นิพจน์)

ระบุค่าของนิพจน์

โอเปอเรเตอร์โปรแกรมดีบัก adb

ชื่อ จำนวนเต็ม สัญลักษณ์ ตัวแปร และรีจิสเตอร์ สามารถนำมา รวมกันด้วยตัวดำเนินการดังต่อไปนี้:

โอเปอเรเตอร์ Unary

***** *Expression*

ส่งกลับเนื้อหาของตำแหน่งที่กำหนดโดยพารามิเตอร์ *Expression* พารามิเตอร์ *CoreFile*

@ *Expression*

ส่งกลับเนื้อหาของตำแหน่งที่กำหนดโดยพารามิเตอร์ *Expression* พารามิเตอร์ *ObjectFile*

- *Expression*

ทำการหักจำนวนเต็ม

~ *Expression*

ทำ bit-wise ให้สมบูรณ์

*Expression*

ทำการหักแบบโลจิคัล

โอเปอเรเตอร์ไบนารี

Expression1 **+***Expression2*

ทำการบวกจำนวนเต็ม

Expression1 **-***Expression2*

ทำการลบจำนวนเต็ม

Expression1 ***Expression2*

ทำการคูณจำนวนเต็ม

Expression1 **%** *Expression2*
ทำการหารจำนวนเต็ม

Expression1 **&** *Expression2*
ทำการรวม bit-wise

Expression1 **|** *Expression2*
ทำการแยก bit-wise

Expression1 **#** *Expression2*
ปัดค่าพารามิเตอร์ *Expression1* ไปเป็น ผลคูณถัดไปของพารามิเตอร์ *Expression2*

ตัวดำเนินการไบนารีเป็น left-associative และมีการ เชื่อมโยงน้อยกว่าตัวดำเนินการยูนารี

คำสั่งย่อยโปรแกรมดีบัก adb

คุณสามารถแสดงเนื้อหาของข้อความหรือเซ็กเมนต์ข้อมูล ด้วย **?** (เครื่องหมายคำถาม) หรือ **/** (สแลช) คำสั่งย่อย **=** (เครื่องหมายเท่ากับ) คำสั่งย่อย แสดงแอดเดรสที่กำหนดในรูปแบบที่ระบุ **?** และ **/** คำสั่งย่อย ตามด้วย * (เครื่องหมายดอกจัน) ได้

?*Format*
แสดงเนื้อหาของพารามิเตอร์ *ObjectFile* เริ่มต้นที่พารามิเตอร์ *Address* ค่าของ. (จุด) เพิ่มโดยผลรวมของส่วนเพิ่มสำหรับแต่ละ ตัวอักษรรูปแบบ

/*Format*
แสดงเนื้อหาของพารามิเตอร์ *CoreFile* เริ่มต้นที่พารามิเตอร์ *Address* ค่าของ. (จุด) เพิ่มโดยผลรวมของส่วนเพิ่มสำหรับแต่ละ ตัวอักษรรูปแบบ

=*Format*
แสดงค่าของพารามิเตอร์ *Address* ตัวอักษรรูปแบบ **i** และ **s** ไม่มีความหมาย สำหรับคำสั่งนี้

พารามิเตอร์ *Format* ประกอบด้วย หนึ่งอักขระหรือมากกว่านั้นซึ่งระบุลักษณะการพิมพ์ แต่ละอักขระรูปแบบ อาจถูกนำหน้าด้วยเลขทศนิยมที่เป็นจำนวนนับซ้ำสำหรับ อักขระรูปแบบ ขณะดำเนินการผ่านขั้นตอน รูปแบบ . (จุด) เพิ่มตามจำนวนที่กำหนดสำหรับแต่ละตัวอักษรรูปแบบ ถ้าไม่มีการกำหนดรูปแบบ จะใช้รูปแบบล่าสุด

ตัวอักษรรูปแบบที่พร้อมใช้งาน มีดังนี้:

a พิมพ์ค่าของ . (จุด) ในฟอร์ม สัญลักษณ์ สัญลักษณ์ถูกตรวจสอบเพื่อประกันว่ามีชนิดที่เหมาะสม

b พิมพ์ไบต์ที่แอดเดรสในฐานปัจจุบัน ไม่มีเครื่องหมาย

c พิมพ์อักขระที่แอดเดรส

C พิมพ์อักขระที่แอดเดรสโดยใช้ระเบียบ escape ดังต่อไปนี้:
- พิมพ์อักขระควบคุมเช่น ~ (tilde) ตามด้วย อักขระการพิมพ์ที่เกี่ยวข้อง
- พิมพ์อักขระที่ไม่สามารถพิมพ์ได้ เช่น ~ (tilde) <*Number*> โดยที่ *Number* จะระบุค่าเลขฐานสิบหกของอักขระ อักขระ ~ พิมพ์เป็น ~ ~ (tilde tilde)

d พิมพ์เป็นทศนิยม

D พิมพ์เป็น long decimal

f พิมพ์ค่า 32-บิต เป็นตัวเลข floating-point

F พิมพ์ double floating point

i *Number*
พิมพ์ตามคำสั่ง *Number* คือจำนวนไบต์ ที่ใช้โดยคำสั่ง

n พิมพ์บรรทัดใหม่

o พิมพ์ 2 ไบต์แบบฐานแปด

O พิมพ์ 4 ไบต์แบบฐานแปด

p พิมพ์ค่าที่แอดเดรสในฟอร์มสัญลักษณ์โดยใช้กฎเหมือนกับ การค้นหาสัญลักษณ์ ตามตัวอักษรรูปแบบ **a**

q พิมพ์ 2 ไบต์ในฐานปัจจุบัน ไม่มีเครื่องหมาย

Q พิมพ์ 4 ไบต์ไม่มีเครื่องหมายในฐานปัจจุบัน

r พิมพ์ช่องว่าง

s *Number*
พิมพ์อักขระที่แอดเดรสจนถึงอักขระศูนย์

S *Number*
พิมพ์สตริงโดยใช้ระเบียบ ~ (tilde) escape ตัวแปร *Number* ระบุความยาวของสตริงรวมทั้งตัวปิดท้าย ศูนย์

t แท็บไปที่แท็บหยุดที่เหมาะสมถัดไป เมื่อนำหน้าโดยจำนวนเต็ม ตัวอย่าง คำสั่งรูปแบบ **8t** ย้ายไปที่แท็บหยุด 8-สเปซถัดไป

u พิมพ์ตัวเลขทศนิยมไม่มีเครื่องหมาย

U พิมทศนิยม long unsigned

x พิมพ์ 2 ไบต์แบบเลขฐานสิบหก

X พิมพ์ 4 ไบต์แบบเลขฐานสิบหก

Y พิมพ์ 4 ไบต์ในรูปแบบวันที่

/ สัญลักษณ์ข้อมูล Local หรือ global

? สัญลักษณ์ข้อความ Local หรือ global

= สัญลักษณ์สัมบูรณ์ Local หรือ global

"..."
พิมพ์สตริงที่ปิดล้อม

^ ลด . (จุด) ด้วยส่วนเพิ่มปัจจุบัน ไม่มีการพิมพ์

+ เพิ่ม . (จุด) ด้วยค่า 1 ไม่มี การพิมพ์

- ลด . (จุด) ลดด้วยค่า 1 ไม่มีการพิมพ์

ขึ้นบรรทัดใหม่ (newline)
ทำซ้ำคำสั่งก่อนหน้าที่เพิ่มด้วย *Count* 1

[?/]l *Value Mask*

คำที่เริ่มต้นด้วย . (จุด) ถูกมาสก์ด้วยค่า *Mask* และที่เปรียบเทียบกับพารามิเตอร์ *Value* จนกระทั่งพบค่าที่ตรงกัน ถ้า **L** ถูกใช้ จะใช้การจับคู่ 4 ไบต์ต่อครั้ง แทน 2 ไบต์ ถ้า ไม่พบข้อมูลตรงกัน แล้ว . (จุด) จะไม่เปลี่ยนแปลง หรือมิฉะนั้น . (จุด) ถูกกำหนดให้กับตำแหน่งที่ตรง ถ้าไม่ระบุพารามิเตอร์ *Mask* จะใช้ค่า -1

[?/]w*Value...*

เขียนพารามิเตอร์ *Value* 2 ไบต์ลงใน ตำแหน่งที่แอดเดรส ถ้าคำสั่งคือ **W** เขียน 4 ไบต์ ถ้าคำสั่งคือ **V**, เขียน 1 ไบต์ ข้อจำกัดการจัดแนว อาจถูกใช้เมื่อใช้คำสั่ง **w** หรือ **W**

[,*Count*][?/]m B1 E1 F1[?/]

บันทึกค่าใหม่สำหรับพารามิเตอร์ *B1*, *E1* และ *F1* ถ้าน้อยกว่าสามนิพจน์ ที่กำหนด พารามิเตอร์แม็พที่เหลือจะไม่ถูกเปลี่ยนแปลง ถ้า **?** (เครื่องหมายคำถาม) หรือ **/** (สแลช) ตามด้วย * (เครื่องหมายดอกจัน) เซ็กเมนต์ที่สอง (*B2*, *E2*, *F2*) ของการแม็พจะถูกเปลี่ยน ถ้ารายการถูกปิดท้ายด้วย **?** หรือ **/**, ไฟล์ (*ObjectFile* หรือ *CoreFile* ตามลำดับ) ถูกใช้สำหรับการร้องขอต่อมา (ตัวอย่างเช่นคำสั่ง **/m?** ทำให้ **/** อ้างถึง *ObjectFile*) ถ้ามีการระบุพารามิเตอร์ *Count* คำสั่ง **adb** เปลี่ยนแม็พที่สัมพันธ์กับไฟล์หรือไลบรารีนั้นเท่านั้น คำสั่ง **$m** แสดงจำนวนที่เกี่ยวข้องกับ ไฟล์ ถ้าพารามิเตอร์ *Count* ไม่ถูกระบุ จะใช้ค่าดีฟอลต์ 0

>*Name*

กำหนด . (จุด) ให้กับตัวแปรหรือรีจิสเตอร์ที่ระบุโดยพารามิเตอร์ *Name*

! เรียกเชลล์เพื่ออ่านบรรทัดดังต่อไปนี้ **!** (เครื่องหมาย ตกใจ)

$ *Modifier*

คำสั่งเบ็ดเตล็ด ค่าที่พร้อมใช้งานสำหรับ *Modifier* คือ:

<*File* อ่านคำสั่งจากไฟล์ที่ระบุและส่งกลับไปที่อินพุตมาตรฐาน ถ้า count ถูกกำหนดเป็น 0 คำสั่งจะถูกละเว้น ค่าของ count ถูกกำหนดไว้ในตัวแปร **adb 9** ก่อนคำสั่งแรกในพารามิเตอร์ *File* ถูก ดำเนินการ

<<*File* อ่านคำสั่งจากไฟล์ที่ระบุและส่งกลับไปที่อินพุตมาตรฐาน คำสั่ง <<*File* สามารถถูก ใช้ในไฟล์โดยไม่ทำให้ไฟล์ถูกปิด ถ้า count ถูกกำหนด เป็น 0 คำสั่งจะถูกละเว้น ค่าของ countถูกกำหนดไว้ในตัวแปร **adb 9**ก่อนคำสั่งแรกใน *File* ถูกดำเนินการ ตัวแปร **adb 9** ถูกบันทึกระหว่างการดำเนินคำสั่ง <<*File* และคืนค่าเมื่อ <<*File* สมบูรณ์ มีข้อจำกัดกับ จำนวนของคำสั่ง <<*File* ที่สามารถถูกเปิดได้ในหนึ่งครั้ง

>*File* ส่งเอาต์พุตไปที่ไฟล์ที่ระบุ ถ้าพารามิเตอร์ *File* ถูกละเว้น เอาต์พุตส่งกลับไปที่เอาต์พุตมาตรฐาน พารามิเตอร์ *File* ถูกสร้างถ้าไม่มีอยู่

b พิมพ์จุดหยุดและ counts และคำสั่งสั่งที่เกี่ยวข้องทั้งหมด

c ซ้อนย้อนกลับสแต็ก ถ้าพารามิเตอร์ *Address* ถูกกำหนด จะถูกใช้เป็นแอดเดรสของเฟรมปัจจุบัน (แทนการใช้รีจิสเตอร์ตัวชี้เฟรม) ถ้าตัวอักษรรูปแบบ **C** ถูกใช้ชื่อและค่าของตัวแปร automatic และ static ทั้งหมดจะถูกพิมพ์สำหรับแต่ละฟังก์ชันที่ใช้งานอยู่ ถ้าพารามิเตอร์ *Count* ถูกกำหนด เฉพาะจำนวนของเฟรมที่ระบุโดยพารามิเตอร์ *Count* จะถูกพิมพ์

d กำนหดฐานปัจจุบันเป็นค่า *Address* หรือ ค่า 16 ถ้าไม่มีการระบุแอดเดรส

e พิมพ์ชื่อและค่าของตัวแปรภายนอก ถ้ามีการระบุ count เฉพาะตัวแปรภายนอกที่สัมพันธ์กับไฟล์จะถูกพิมพ์

f พิมพ์รีจิสเตอร์ floating-point เป็นเลขฐานสิบหก

i *ชุดคำสั่ง*

เลือกชุดคำสั่งเครื่องเพื่อใช้สำหรับถอดแยกภาษาแอสเซมบลี

I เปลี่ยนไดเร็กทอรีเริ่มต้นตามที่ระบุโดยแฟล็ก **–I** ไปเป็นค่าพารามิเตอร์ *Name*

m พิมพ์แอดเดรสแม็พ

n *mnem_set*
เลือกตัวช่วยจำที่ถูกใช้สำหรับถอดแยกภาษาแอสเซมบลี

o ตั้งค่าฐานปัจจุบันเป็นค่า 8

q ออกจากคำสั่ง **adb**

r พิมพ์รีจิสเตอร์ทั่วไปและคำสั่งที่แอดเดรสโดย **iar** และตั้งค่า . (จุด) เป็น **iar** พารามิเตอร์ *Number***$r** พิมพ์รีจิสเตอร์ที่ระบุโดยตัวแปร *Number* พารามิเตอร์ *Number*,*Count***$r** พิมพ์รีจิสเตอร์ *Number*+*Count*–1,...,*Number*

s ตั้งค่าจำกัดสำหรับสัญลักษณ์ที่ตรงกับค่า *Address* ค่าเริ่มต้นคือ 255

v พิมพ์ตัวแปรที่ไม่ใช่ศูนย์ทั้งหมดเป็นฐานแปด

w ตั้งค่าความกว้างหน้าเอาต์พุตสำหรับพารามิเตอร์ *Address* ค่าดีฟอลต์คือ 80

P *Name*
ใช้ค่า *Name* เป็นสตริงพร้อมต์

? พิมพ์ process ID สัญญาณที่ทำให้ stoppage หรือ termination และรีจิสเตอร์ของ **$r**

: *Modifier*
จัดการ subprocess modifiers ที่พร้อมใช้งานคือ:

b*Command*
ตั้งค่าจุดหยุดที่พารามิเตอร์ *Address* จุดหยุดรันพารามิเตอร์ *Count* –1 ครั้งก่อน ทำให้หยุด แต่ละครั้งที่พบจุดหยุด คำสั่งที่ระบุ รัน ถ้าคำสั่งตั้งค่า . (จุด) เป็น 0 จุดหยุดจะทำให้หยุดทำงาน

c*Signal* ดำเนิน subprocess ที่มีสัญญาณที่ระบุต่อ ถ้าพารามิเตอร์ *Address* ถูกกำหนด subprocess จะดำเนินต่อที่แอดเดรสนี้ ถ้าไม่มีการระบุสัญญาณ สัญญาณที่ทำให้ subprocess หยุดจะถูก ส่ง การข้ามจุดหยุดเหมือนกับ **r** modifier

d ลบจุดหยุดที่พารามิเตอร์ *Address*

k หยุด subprocess ปัจจุบัน ถ้ามีรันอยู่

r รันพารามิเตอร์ *ObjectFile* เป็น subprocess ถ้าพารามิเตอร์ *Address* ถูกกำหนดชัดเจน โปรแกรมจะเข้าสู่การทำงานที่จุดนี้ มิฉะนั้นโปรแกรมจะถูกเข้าทำงานที่ entry point มาตรฐาน พารามิเตอร์ *Count* ระบุ จำนวนจุดหยุดที่จะถูกละเว้นก่อนการหยุด อาร์กิวเมนต์ที่ส่งให้กับ subprocess สามารถกำหนดให้ที่บรรทัดเดียวกับคำสั่ง อาร์กิวเมนต์เริ่มต้นด้วย < หรือ > จะสร้าง อินพุตหรือเอาต์พุตมาตรฐานสำหรับคำสั่ง บนรายการที่ส่งไปที่ subprocess สัญญาณทั้งหมด ถูกเปิด

s*Signal* ดำเนินการต่อ subprocess ทีละหนึ่งขั้นถึงตัวเลขที่ระบุ ในพารามิเตอร์ *Count* ถ้าไม่มี subprocess ปัจจุบันพารามิเตอร์ *ObjectFile* ถูกรันเป็น subprocess ใน กรณีนี้ไม่สามารถส่งสัญญาณได้ ตัวเตือนของบรรทัดจะถูกจัดการเป็นอาร์กิวเมนต์ ส่งไปที่ subprocess

ตัวแปรโปรแกรมดีบัก adb

คำสั่ง **adb** จัดเตรียม ตัวแปร เมื่อโปรแกรม **adb** เริ่มทำงาน ตัวแปรดังต่อไปนี้ ถูกตั้งค่าจากส่วนหัวระบบในไฟล์คอร์ที่ระบุ ถ้าพารามิเตอร์ *CoreFile* ไม่ใช่ไฟล์ **core** ค่าเหล่านี้จะถูกตั้งค่าจากพารามิเตอร์ *ObjectFile*:

0 ค่าล่าสุดที่พิมพ์

1 ส่วนที่ย้ายออกล่าสุดของซอร์สคำสั่ง

2 ค่าก่อนหน้าของตัวแปร **1**

9 จำนวนของ **$<** ล่าสุดหรือคำสั่งย่อย **$<<**

b แอดเดรสฐานของเซ็กเมนต์ข้อมูล

d ขนาดของเซ็กเมนต์ข้อมูล

e แอดเดรสรายการของโปรแกรม

m ตัวเลข "Magic"

s ขนาดของเซ็กเมนต์สแต็ก

t ขนาดของเซ็กเมนต์ข้อความ

ตัวอย่างโปรแกรม adb: adbsamp

โปรแกรมตัวอย่างดังต่อไปนี้ใช้ในตัวอย่างนี้:

```
/* Program Listing for adbsamp.c */
char  str1[ ] = "This is a character string";
int one = 1;
int number = 456;
long lnum = 1234;
float fpt = 1.25;
char str2[ ] = "This is the second character string";
main()
{
        one = 2;
        printf("First String = %s\n",str1);
        printf("one = %d\n",one);
        printf("Number = %d\n",lnum);
        printf("Floating point Number = %g\n",fpt);
        printf("Second String = %s\n",str2);
}
```

คอมไพล์โปรแกรมโดยใช้คำสั่ง **cc** กับไฟล์ **adbsamp** ดังนี้:

```
cc -g adbsamp.c -o adbsamp
```

เมื่อต้องการเริ่มเซสชันดีบัก ให้ป้อน:

```
adb adbsamp
```

ตัวอย่างโปรแกรม adb: adbsamp2

โปรแกรมตัวอย่างดังต่อไปนี้ใช้ในตัวอย่างนี้:

```
/*program listing for adbsamp2.c*/
int     fcnt,loop_count;

f(a,b)
int a,b;
{
        a = a+b;
        fcnt++;
        return(a);
}
main()
{
        loop_count = 0;
        while(loop_count <= 100)
        {
                loop_count = f(loop_count,1);
                printf("%s%d\n","Loop count is: ", loop_count);
                printf("%s%d\n","fcnt count is: ",fcnt);
        }
}
```

คอมไพล์โปรแกรมโดยใช้คำสั่ง **cc** ไปที่ไฟล์ **adbsamp2** ด้วยคำสั่ง ดังต่อไปนี้:

```
cc -g adbsamp2.c -o adbsamp2
```

เมื่อต้องการเริ่มเซสชันดีบัก ให้ป้อน:

```
adb adbsamp2
```

ตัวอย่างโปรแกรม adb: adbsamp3

โปรแกรมตัวอย่างดังต่อไปนี้ **adbsamp3.c** มีการเรียกซ้ำไม่สิ้นสุดของการเรียก subfunction

ถ้าคุณรันโปรแกรม จนสมบูรณ์โปรแกรมจะทำให้เกิดข้อผิดพลาด memory fault และจบการทำงาน

```
int     fcnt,gcnt,hcnt;
h(x,y)
int x,y;
{
        int hi;
        register int hr;
        hi = x+1;
        hr = x-y+1;
        hcnt++;
        hj:
        f(hr,hi);
}
g(p,q)
int p,q;
{
        int gi;
        register int gr;
        gi = q-p;
        gr = q-p+1;
        gcnt++;
        gj:
```

```
        h(gr,gi);
}
f(a,b)
int a,b;
{
        int fi;
        register int fr;
        fi = a+2*b;
        fr = a+b;
        fcnt++;
        fj:
        g(fr,fi);
}
main()
{
        f(1,1);
}
```

คอมไพล์โปรแกรมโดยใช้คำสั่ง **cc** ไปที่ไฟล์ **adbsamp3** ด้วยคำสั่ง ดังต่อไปนี้:

```
cc -g adbsamp3.c -o adbsamp3
```

เมื่อต้องการเริ่มเซสชันดีบัก ให้ป้อน:

```
adb adbsamp3
```

ตัวอย่างของไดเร็กทอรีและดัมพ์ i-node ในการดีบัก adb

ตัวอย่างนี้แสดงวิธีสร้างสคริปต์ **adb** เพื่อใช้แสดงเนื้อหาการแม็พไดเร็กทอรีและ i-node ของระบบไฟล์ ในตัวอย่าง ไดเร็กทอรีถูกตั้งชื่อ **dir** และมีไฟล์หลากหลาย

ระบบไฟล์เชื่อมโยงกับ ไฟล์อุปกรณ์ /dev/hd3 (**/tmp**) ซึ่งจะมีสิทธิที่จำเป็น ที่จะถูกอ่านโดยผู้ใช้

ในการแสดงไดเร็กทอรี ให้สร้างสคริปต์ที่เหมาะสม โดยปกติไดเร็กทอรีมีอย่างน้อยหนึ่งรายการ แต่ละรายการประกอบด้วยหมายเลข i-node ที่ไม่ได้ระบุ (i-number) และชื่อไฟล์ยาว 14 อักขระ คุณสามารถ แสดงข้อมูลนี้โดยรวมคำสั่งไว้ในไฟล์สคริปต์ของคุณ โปรแกรมการดีบัก **adb** คาดหวังอ็อบเจ็กต์ไฟล์ต้องเป็นไฟล์รูปแบบ **xcoff** นี้ไม่ใช่กรณีเดียวกับไดเร็กทอรี โปรแกรม **adb** บ่งชี้ว่า เนื่องจากไม่ใช่ไฟล์รูปแบบ **xcoff** ไดเร็กทอรีมีความยาวข้อความเป็น 0 ใช้คำสั่ง **m** เพื่อชี้ไปยังโปรแกรม **adb** ที่ไดเร็กทอรีนี้ มีความยาวข้อความมากกว่า 0 ดังนั้น แสดงรายรายการ ในเซสชัน **adb** ของคุณโดยการป้อน:

```
,0?m 360 0
```

ตัวอย่าง คำสั่งต่อไปนี้แสดง 20 รายการแรกโดยคั่นหมายเลข i-node และชื่อไฟล์ด้วยอักขระแท็บ:

```
0,20?ut14cn
```

คุณสามารถเปลี่ยนหมายเลขที่สอง 20 เพื่อระบุจำนวนรายการในไดเร็กทอรี ถ้าคุณใส่คำสั่ง ต่อไปนี้ที่ตอนเริ่มต้นของสคริปต์ โปรแกรม **adb** จะแสดงสตริงเป็นส่วนหัวสำหรับแต่ละคอลัมน์ของหมายเลข:

```
="inumber"8t"Name"
```

เมื่อคุณสร้างสคริปต์ไฟล์แล้ว ให้เปลี่ยนเส้นทาง เป็นอินพุตเมื่อคุณเริ่มทำงานโปรแกรม **adb** ด้วยชื่อ ของไดเร็กทอรีของคุณ ตัวอย่าง คำสั่งต่อไปนี้จะเริ่มทำงานโปรแกรม **adb** บนไดเร็กทอรี geo โดยใช้อินพุต คำสั่งจากไฟล์สคริปต์ ddump :

```
adb geo - <ddump
```

เครื่องหมายลบ (-) หลีกเลี่ยงมิให้โปรแกรม **adb** เปิดไฟล์ core โปรแกรม **adb** อ่านคำสั่งจากไฟล์สคริปต์

ในการแสดงตาราง i-node ของระบบไฟล์ สร้าง สคริปต์ใหม่ จากนั้นเริ่มทำงานโปรแกรม **adb** ด้วย ชื่อไฟล์ของอุปกรณ์ที่เชื่อมโยงกับระบบไฟล์ ตาราง i-node ของระบบไฟล์มีโครงสร้างที่ซับซ้อน แต่ละรายการประกอบด้วย:

- ค่า 1 word สำหรับแฟล็กสถานะ
- ค่า 1 ไบต์สำหรับลิงก์หมายเลข
- ค่า 2 ไบต์สำหรับ ID ผู้ใช้และกลุ่ม
- ค่า 1 ไบต์ และ 1 word สำหรับขนาด
- ค่า 8 word สำหรับตำแหน่งบนดิสก์ของ บล็อกของไฟล์
- ค่า 2 word สำหรับวันที่สร้างและวันที่ แก้ไข

ต่อไปนี้คือตัวอย่างเอาต์พุตการดัมพ์ไดเร็กทอรี:

```
        inumber Name
0:       26    .
        2..
        27    .estate
        28    adbsamp
        29    adbsamp.c
        30    calc.lex
        31    calc.yacc
        32    cbtest
        68    .profile
        66    .profile.bak
        46    adbsamp2.c
        52    adbsamp2
        35    adbsamp.s
        34    adbsamp2.s
        48    forktst1.c
        49    forktst2.c
        50    forktst3.c
        51    lpp&us1.name
        33    adbsamp3.c
        241   sample
        198   adbsamp3
        55    msgqtst.c
        56    newsig.c
```

ตาราง i-node เริ่มต้นที่แอดเดรส 02000 คุณ สามารถแสดงรายการแรกโดยใส่คำสั่งต่อไปนี้ในไฟล์สคริปต์ ของคุณ:

```
02000,-1?on3bnbrdn8un2Y2na
```

คำสั่งระบุอักขระขึ้นบรรทัดใหม่ (new-line) หลายตัว สำหรับการแสดงเอาต์พุตเพื่อให้อ่านง่าย

ในการใช้ไฟล์สคริปต์กับตาราง i-node ของไฟล์ **/dev/hd3** ป้อนคำสั่งต่อไปนี้:

```
adb /dev/hd3 - <script
```

แต่ละรายการในการแสดงผลมีรูปแบบ:

```
02000: 073145
    0163 0164 0141
    0162 10356
    28770 8236 25956 27766 25455 8236 25956 25206
    1976 Feb 5 08:34:56 1975 Dec 28 10:55:15
```

ตัวอย่างของการจัดรูปแบบข้อมูลในการดีบัก adb

ในการแสดงแอดเดรสปัจจุบันหลังคำสั่งเครื่อง แต่ละคำสั่ง ป้อน:

```
main , 5 ? ia
```

คำสั่งนี้สร้างเอาต์พุตต่อไปนี้เมื่อใช้ กับโปรแกรมตัวอย่าง **adbsamp**:

```
.main:
.main:          mflr 0
.main+4:        st r0,  0x8(r1)
.main+8:        stu rs,    (r1)
.main+c:        li l  r4,  0x1
.main+10:       oril  r3, r4, 0x0
.main+14:
```

ในการทำให้ชัดเจนขึ้นว่าแอดเดรสปัจจุบัน ไม่ได้อยู่ในคำสั่งที่ปรากฏบนบรรทัดเดียวกัน เพิ่มอักขระการจัดรูปแบบ บรรทัดใหม่ (n) ในคำสั่ง:

```
.main , 5 ? ian
```

นอกจากนั้น คุณสามารถใส่หมายเลขหน้าอักขระการจัด รูปแบบเพื่อระบุจำนวนครั้งที่ทำซ้ำสำหรับรูปแบบนั้น

ในการพิมพ์รายการคำสั่งและรวมแอดเดรส หลังจากทุกๆ คำสั่งที่สี่ ให้ใช้คำสั่งต่อไปนี้:

```
.main,3?4ian
```

คำสั่งนี้สร้างเอาต์พุตต่อไปนี้เมื่อ ใช้กับโปรแกรมตัวอย่าง **adbsamp**:

```
.main:
                mflr 0
                st  r0, 0x8(r1)
                stu r1, -56(r1)
                lil r4, 0x1

.main+10:
                oril r3, r4, 0x0
                bl .f
                l r0, 0x40(r1)
                ai r1, r1, 0x38

.main+20:
                mtlr r0
                br
                Invalid opcode
                Invalid opcode

.main+30:
```

ขอให้ระมัดระวังเมื่อคุณใส่หมายเลข

คำสั่งต่อไปนี้ แม้จะคล้ายกับคำสั่งก่อนหน้า แต่ไม่ได้ให้เอาต์พุตเดียวกัน:

```
main,3?i4an
```

```
.main:
.main:          mflr  0
.main+4:        .main+4:        .main+4:        .main+4:
                st  r0, 0x8(r1)
.main+8:        .main+8:        .main+8:        .main+8:
                stu  r1,    (r1)
.main+c:        .main+c:        .main+c:        .main+c:
```

คุณสามารถรวมการร้องขอเพื่อให้มีการแสดง ที่ซับซ้อนยิ่งขึ้น ตัวอย่าง การป้อนคำสั่งต่อไปนี้จะแสดงการช่วยจำ คำสั่งตามด้วยค่าเลขฐานสิบหกที่เทียบเท่า:

```
.main,-1?i^xn
```

ในตัวอย่างนี้ การแสดงเริ่มที่ `main` แอดเดรส จำนวนนับที่เป็นค่าลบ (-1) ทำให้เรียกใช้คำสั่ง แบบไม่สิ้นสุด ดังนั้นการแสดงผลจะยังคงทำต่อไปจนเงื่อนไขข้อผิดพลาด (เช่น end-of-file (สิ้นสุดไฟล์)) เกิดขึ้น ในรูปแบบ `i` แสดง คำสั่งช่วยจำที่ตำแหน่งนั้น `^` (caret) ย้ายแอดเดรสปัจจุบันกลับไปที่เริ่มต้นคำสั่ง และ `x` แสดงคำสั่งอีกครั้งเป็นเลขฐานสิบหก สุดท้าย `n` ส่งอักขระขึ้นบรรทัดใหม่ไปยังเทอร์มินัล เอาต์พุตจะมีลักษณะคล้ายที่แสดงต่อไปนี้ เพียงแต่ยาวกว่า:

```
.main:
.main:          mflr  0
                7c0802a6
                  st  r0, 0x8(r1)
                9001008
                  st r1, -56(r1)
                9421ffc8
                  lil r4, 0x1
                38800001
                  oril r3, r4, 0x0
                60830000
                  bl  -  .f
                4bffff71
                  l r0, 0x40(r1)
                80010040
                  ai r1, r1, 0x38
                30210038
                  mtlr r0
                7c0803a6
```

ตัวอย่างต่อไปนี้แสดงวิธีรวมรูปแบบ ในคำสั่งย่อย **?** หรือ **/** เพื่อแสดง ประเภทของค่าที่ต่างกันเมื่อถูกเก็บร่วมกันในโปรแกรมเดียวกัน ซึ่งใช้โปรแกรม **adbsamp** สำหรับคำสั่งที่จะมีการใช้ตัวแปร ทำงาน อันดับแรกคุณต้องตั้งค่าจุดหยุด (breakpoint) เพื่อหยุดทำงานโปรแกรม และ จากนั้นรันโปรแกรมต่อจนกระทั่งพบจุดหยุด ใช้คำสั่ง **:b** เพื่อตั้งค่าจุดหยุด:

```
.main+4:b
```

ใช้คำสั่ง **$b** เพื่อแสดงว่า จุดหยุดถูกตั้งค่า:

```
$b
จุดพัก
count  bkpt      command
1    .main+4
```

รันโปรแกรมจนกระทั่งพบจุดหยุดโดยการป้อน:

```
:r
adbsamp: running
breakpoint   .main+4:    st r0, 0x8(r1)
```

ในตอนนี้คุณสามารถแสดงเงื่อนไขของโปรแกรมเมื่อ หยุดทำงาน ในการแสดงค่าของแต่ละตัวแปร ให้กำหนดชื่อและ รูปแบบที่เกี่ยวข้องในคำสั่ง / (เครื่อหมายทับ) ตัวอย่าง คำสั่งต่อไปนี้แสดงเนื้อหาของ str1 เป็นสตริง:

```
str1/s
str1:
str1:     This is a character string
```

คำสั่งต่อไปนี้แสดงเนื้อหาของ number เป็นเลขจำนวนเต็ม:

```
number/D
number:
number:    456
```

คุณสามารถเลือกดูตัวแปรในรูปแบบ ต่างๆ ตัวอย่าง คุณสามารถแสดง **lnum** ตัวแปรแบบยาวเป็นจำนวนฐานสิบ ฐานแปด และฐานสิบหกที่ยาว 4 ไบต์ได้โดยการ ป้อนคำสั่ง:

```
lnum/D
lnum:
lnum:     1234

lnum/O
lnum:
lnum:     2322

lnum/X
lnum:
lnum:     4d2
```

คุณยังสามารถตรวจสอบตัวแปรในรูปแบบอื่นๆ ตัวอย่าง คำสั่งต่อไปนี้แสดงบางตัวแปรเป็นค่าฐานสิบหกแปดชุด บนหนึ่งบรรทัด และต่อเนื่องกันเป็นจำนวนห้าบรรทัด:

```
str1,5/8x
str1:
str1:   5468  6973  2069  7320  6120  6368  6172  6163
         7465  7220  7374  7269  6e67  0   0   0     0

number:  0        1c8   0     0      0    4d2    0     0
         3fa0     0   0     0    5468  6973  2069  7320
         7468  6520  7365  636f  6e64  2063  6861  7261
```

เนื่องจากข้อมูลมีการผสมกันของค่าตัวเลข และสตริง ให้แสดงแต่ละค่าทั้งตัวเลขและอักขระเพื่อดู ว่าสตริงที่แท้จริงอยู่ที่ตำแหน่งใด คุณสามารถทำสิ่งนี้ได้ด้วยคำสั่งหนึ่งคำสั่ง:

```
str1,5/4x4^8Cn
str1:
str1:    5468 6973   2069  7320  This is
          6120  6368  6172  6163   a charac
          7465  7220  7374  7269   ter stri
          6e67  0      0    0       ng~@~@~@~@~@~@
          0    1c8   0       0      ~@~@~A~<c8>~@~@~@~@
```

ในกรณีนี้ คำสั่งแสดงค่าสี่ค่าเป็น เลขฐานสิบหก จากนั้นแสดงค่าเดียวกันเป็นอักขระ ASCII แปดอักขระ อักขระ ^ (caret) ถูกใช้สี่ครั้งก่อนการแสดงอักขระเพื่อตั้งค่า แอดเดรสปัจจุบันกลับไปเป็นแอดเดรสเริ่มต้นสำหรับบรรทัดนั้น

ในการทำให้การแสดงผลอ่านได้ง่ายขึ้น คุณสามารถแทรก แท็บระหว่างค่าและอักขระ และกำหนดแอดเดรสสำหรับแต่ละบรรทัด:

```
str1,5/4x4^8t8Cna
str1:
str1:     5468  6973  2069  7320      This is
str1+8:   6120  6368  6172  6163      a charac
str1+10:  7465  7220  7374  7269      ter stri
str1+18:  6e67     0    0    1        ng~@~@~@~@~@~A

number:
number:    0      1c8   0     0       ~@~@~A~<c8>~@~@~@~@
fpt:
```

ตัวอย่างของการติดตามหลายฟังก์ชันในการดีบัก adb

ตัวอย่างต่อไปนี้แสดงวิธีทำงานโปรแกรม ภายใต้การควบคุม **adb** และดำเนินการดีบักเบื้องต้น ดังอธิบายในส่วนต่อไปนี้

หมายเหตุ: โปรแกรมตัวอย่างที่ใช้ในส่วนนี้ **adbsamp3** มีการเรียกซ้ำไม่รู้จบของการเรียกใช้ฟังก์ชันย่อย ถ้าคุณรันโปรแกรมจนสมบูรณ์โปรแกรมจะทำให้เกิดข้อผิดพลาด memory fault และจบการทำงาน

ซอร์สโปรแกรมสำหรับตัวอย่างนี้จะถูกเก็บไว้ในไฟล์ที่ชื่อ **adbsamp3.c** คอมไพล์โปรแกรมนี้เป็นไฟล์ที่เรียกทำงานได้ชื่อ **adbsamp3** โดยใช้ คำสั่ง **cc**

```
cc adbsamp3.c -o adbsamp3
```

เริ่มต้นโปรแกรม adb

ในการทำงานเซสชันและเปิดโปรแกรมไฟล์ใช้ คำสั่งต่อไปนี้ (ไม่ใช่ไฟล์ core):

```
adb adbsamp3
```

การตั้งค่า breakpoints

อันดับแรก ตั้งค่าจุดหยุดที่จุดเริ่มต้นของแต่ละฟังก์ชัน โดยใช้ฟังก์ชันย่อย **:b** :

```
.f:b
.g:b
.h:b
```

การแสดงชุดของคำสั่ง

ถัดไปแสดงห้าคำสั่งแรกในฟังก์ชัน **f**:

```
.f,5?ia
.f:
.f:             mflr  r0
.f+4:           st  r0, 0x8(r1)
.f+8:           stu  r1, -64(r1)
.f+c:           st  r3,  0x58(r1)
.f+10:          st  r4,  0x5c(r1)
.f+14:
```

แสดงห้าคำสั่งในฟังก์ชัน **g** โดยไม่มีแอดเดรส:

```
.g,5?i
.g:    mflr  r0
        st  r0, 0x8(r1)
        stu  r1, -64(r1)
        st  r3,  0x58(r1)
        st  r4,  0x5c(r1)
```

การเริ่มต้นโปรแกรม adsamp3

เริ่มทำงานโปรแกรมโดยการป้อนคำสั่งต่อไปนี้:

```
:r
adbsamp3: running
breakpoint   .f:       mflr  r0
```

โปรแกรม **adb** ทำงานโปรแกรม ตัวอย่างจนกระทั่งถึงจุดหยุดแรกที่โปรแกรมจะหยุดทำงาน

การลบ breakpoint

เนื่องจากการทำงานโปรแกรมไปที่จุดนี้ไม่มี ข้อผิดพลาดเกิดขึ้น คุณสามารถเอาจุดหยุดแรกออก:

```
.f:d
```

การรันโปรแกรมต่อ

ใช้คำสั่งย่อย **:c** เพื่อดำเนิน โปรแกรมต่อ:

```
:c
adbsamp3: running
breakpoint   .g:       mflr  r0
```

โปรแกรม **adb** เริ่มทำงานโปรแกรม **adbsamp3** ต่อที่คำสั่งถัดไป การดำเนินการของโปรแกรม ยังคงทำต่อไปจนกระทั่งถึงจุดหยุดถัดไป ซึ่งโปรแกรมจะหยุดทำงาน

การติดตามพาธของการเรียกทำงาน

ติดตามพาธของการดำเนินงานโดยการป้อน:

การแสดงค่าตัวแปร

แสดงเนื้อหาของตัวแปรจำนวนเต็ม `fcnt` โดยป้อนคำสั่ง:

```
fcnt/D
fcnt:
fcnt:     1
```

การข้าม breakpoints

ถัดไป ดำเนินการทำงานโปรแกรมต่อ และข้ามจุดหยุด 10 จุดแรกโดยการป้อน:

```
,10:c
adbsamp3: running
breakpoint   .g:       mflr  r0
```

โปรแกรม **adb** เริ่มทำงานโปรแกรม **adbsamp3** และแสดงข้อความที่กำลังทำงานอีกครั้ง ซึ่งจะไม่หยุดทำงานโปรแกรมจนว่าจะครบจุดหยุด 10 จุดแน่นอน เพื่อให้แน่ใจว่าจุดหยุดเหล่านี้ถูกข้าม แสดงการติดตามย้อนกลับ (backtrace) อีกครั้ง:

```
$c
.g(0,0) .f+2a
.f(2,11) .h+28
.h(10,f) .g+2a
.g(11,20) .f+2a
.f(2,f) .h+28
.h(e,d) .g+2a
.g(f,1c) .f+2a
.f(2,d) .h+28
.h(c,b) .g+2a
.g(d,18) .f+2a
.f(2,b) .h+28
.h(a,9) .g+2a
.g(b,14) .f+2a
.f(2,9) .h+28
.h(8,7) .g+2a
.g(9,10) .f+2a
.f(2,7) .h+28
.h(6,5) .g+2a
.g(7,c) .f+2ae
.f(2,5) .h+28
.h(4,3) .g+2a
.g(5,8) .f+2a
.f(2,3) .h+28
.h(2,1) .g+2a
.g(2,3) .f+2a
.f(1,1) .main+e
.main(0,0,0) start+fa
```

ภาพรวมโปรแกรมดีบักสัญลักษณ์ dbx

โปรแกรมดีบักสัญลักษณ์ **dbx** ช่วยให้คุณสามารถดีบักแอ็พพลิเคชันโปรแกรมที่สองระดับ: ระดับซอร์ส และที่ระดับภาษแอสเซมเบลอร์ การดีบักที่ระดับซอร์สอนุญาตให้คุณดีบัก โปรแกรมภาษา C, C++ หรือ FORTRAN ของคุณ

การดีบักที่ระดับภาษาแอสเซมเบลอร์ อนุญาตให้คุณดีบักโปรแกรมเรียกทำงานที่ระดับเครื่อง คำสั่งที่ใช้ สำหรับการดีบักระดับเครื่องเหมือนกับที่ใช้สำหรับการดีบัก ระดับซอร์ส

การใช้โปรแกรมดีบัก **dbx** คุณสามารถดูการทำงานของแอ็พพลิเคชันโปรแกรมที่คุณต้องการดีบักทีละบรรทัด หรือกำหนดจุดหยุดในโปรแกรมเชิงอ็อบเจ็กต์ที่จะหยุดโปรแกรมดีบัก คุณยังสามารถค้นหาและแสดงส่วนของไฟล์ต้นฉบับสำหรับ แอ็พพลิเคชันโปรแกรม

ส่วนดังต่อไปนี้มีข้อมูลวิธีดำเนิน งานต่างๆ ด้วยโปรแกรมดีบัก **dbx**:

การใช้โปรแกรมดีบัก dbx

ส่วนนี้มีข้อมูลเกี่ยวกับวิธีใช้โปรแกรมดีบัก **dbx**

การเริ่มต้นโปรแกรมดีบัก dbx

โปรแกรม **dbx** สามารถเริ่มต้นได้ด้วยแฟล็กที่หลากหลาย ซึ่งมีวิธีการส่วนใหญ่โดยทั่วไปสามวิธีในการเริ่มต้นเซสชันดีบักด้วยโปรแกรม **dbx** คือ:

- การรันคำสั่ง **dbx** บน ไฟล์อ็อบเจ็กต์ที่ระบุ
- การใช้แฟล็ก **–r** เพื่อรันคำสั่ง **dbx** บนโปรแกรมที่จบแบบผิดปกติ
- การใช้แฟล็ก **–a** เพื่อรันคำสั่ง **dbx** บนกระบวนการที่กำลังทำงาน

เมื่อคำสั่ง **dbx** ถูกเริ่มต้น คำสั่งจะตรวจสอบไฟล์ **.dbxinit** ในไดเร็กทอรีปัจจุบันของผู้ใช้และในไดเรกทอรี **$HOME** ของผู้ใช้ ถ้าไฟล์ **.dbxinit** มีอยู่แล้ว คำสั่งย่อยจะทำงานที่จุดเริ่มต้นของเซสชันดีบัก ถ้าไฟล์ **.dbxinit** มีอยู่แล้วทั้งในไดเร็กทอรี home และไดเร็กทอรีปัจจุบัน ดังนั้นไดเร็กทอรีทั้งสองจะอ่านตามลำดับ เนื่องจากไฟล์ **.dbxinit** ในไดเร็กทอรีปัจจุบันถูกอ่านครั้งล่าสุด คำสั่งย่อยของไฟล์นั้นสามารถแทนที่คำสั่งย่อยเหล่านั้นในไดเร็กทอรี home

ถ้าไม่ได้ระบุอ็อบเจ็กต์ไฟล์ไว้โปรแกรม **dbx** จะถามชื่อของอ็อบเจ็กต์ไฟล์ที่ต้องการตรวจสอบ ค่าดีฟอลต์คือ **a.out** ถ้าไฟล์ **core** มีอยู่ในไดเร็กทอรีปัจจุบัน หรือระบุพารามิเตอร์ *CoreFile* ไว้โปรแกรม **dbx** จะรายงานตำแหน่งที่โปรแกรมเกิดข้อบกพร่อง ตัวแปร รีจิสเตอร์ และหน่วยความจำที่พักอยู่ในอิมเมจหลักอาจถูกตรวจสอบ จนกว่าการประมวลผลของอ็อบเจ็กต์ไฟล์จะเริ่มต้นขึ้น ณ จุดนั้น โปรแกรมดีบัก **dbx** จะพร้อมต์รับคำสั่ง

การดีบักอิมเมจแกนที่ไม่มีโมดูลการพึ่งพา

สำหรับการเริ่มต้นด้วย AIX 5.3 โปรแกรม**dbx** จะมีความสามารถในการตรวจสอบอิมเมจหลักแม้ว่าโมดูลที่ต้องพึ่งพาตั้งแต่หนึ่งโมดูลขึ้นไปจะไม่สามารถเข้าถึงได้ก็ตาม ในระหว่างการกำหนดค่าเริ่มต้น ข้อความแจ้งเตือนจะถูกแสดง สำหรับโมดูลที่ต้องพึ่งพาที่ขาดหายไป

ในการดำเนินการปกติ โปรแกรม **dbx** จะใช้ข้อมูลที่มีอยู่ในตารางสัญลักษณ์ของโมดูลที่ต้องพึ่งพา และส่วนของข้อความ เนื่องจากข้อมูลบางส่วนนั้นหายไป เซสชัน **dbx** ที่มีโมดูลที่ต้องพึ่งพาที่หายไป จะมีข้อจำกัดดังต่อไปนี้:

- ความพยายามทั้งหมดที่ทำโดยผู้ใช้เพื่ออ่านเนื้อหาของหน่วยความจำที่อยู่ในส่วนของข้อความ ของโมดูลที่ต้องพึ่งพาที่ขาดหายไปจะส่งผลทำให้เกิดข้อความแสดงความผิดพลาด ข้อความแสดงความผิดพลาดจะเหมือนกับข้อความที่เกิดขึ้นเมื่อข้อมูลไม่สามารถเข้าถึงได้ เนื่องจากข้อความนั้นไม่ได้อยู่ในไฟล์หลัก
- ผู้ใช้ไม่สามารถขอรับข้อมูลที่เกี่ยวข้องกับสัญลักษณ์ที่อ่านจากตารางสัญลักษณ์ ของโมดูลที่ต้องพึ่งพาที่ขาดหายไป ลักษณะการทำงานของโปรแกรม **dbx** จะคล้ายคลึงกับกรณีที่ตารางสัญลักษณ์ของโมดูลที่ต้องพึ่งพา ที่ถูกแสดง
- กรอบสแต็กที่สอดคล้องกับรูทีนภายในโมดูลที่ต้องพึ่งพาที่ขาดหายไป จะแสดงขึ้นดังนี้:

  ```
  .()
  ```

นอกจากนี้ แอดเดรสคำสั่งเครื่องภายในรูทีนที่ไม่รู้จัก และชื่อของโมดูลที่ต้องพึ่งพาที่หายไป จะแสดงขึ้น

ผู้ใช้ยังมีอ็อพชันของการเปลี่ยนทิศทางโปรแกรม **dbx** กับโมดูลที่ต้องพึ่งพาที่สามารถเข้าถึงได้โดยใช้แฟล็ก **–p** สำหรับข้อมูลเพิ่มเติม โปรดดู คำสั่ง **dbx** ใน *Commands Reference, Volume 2*

การดีบักอิมเมจแกนที่มีโมดูลการพึ่งพาที่ไม่ตรงกัน

สำหรับการเริ่มต้นด้วย AIX 5.3 โปรแกรม **dbx** จะตรวจพบว่า โมดูลที่ต้องพึ่งพาใดๆ ที่อ้างอิงอยู่ในไฟล์หลักแตกต่างจากการสร้างไฟล์หลัก ในระหว่างการกำหนดค่าเริ่มต้น ข้อความแจ้งเตือนจะแสดงขึ้น สำหรับโมดูลที่ต้องพึ่งพาที่ไม่ตรงกัน

ผู้ใช้ควรรับทราบว่า ข้อมูลใดๆ ที่แสดงโดยโปรแกรม **dbx** ซึ่งเป็นไปตามการอ้างอิงเนื้อหาของโมดูลที่ต้องพึ่งพาที่ไม่ตรงกัน อาจเชื่อถือไม่ได้ ในความพยายามแจ้งเตือนผู้ใช้ถึงข้อมูลที่ไม่ควรเชื่อถือ โปรแกรม **dbx** จะส่งข้อความแจ้งเตือน ไม่ว่าข้อมูลที่สามารถเป็นคำถามได้จะแสดงขึ้น

หากปิดใช้งานฟังก์ชันนี้ และโปรแกรม **dbx** บังคับให้ใช้โมดูลที่ต้องพึ่งพาที่ไม่ตรงกัน เป็นโมดูลที่ต้องพึ่งพาที่ขาดหายไป ผู้ใช้สามารถเอ็กซ์พอร์ตตัวแปรสภาวะแวดล้อม *DBX_MISMATCH_MODULE* ด้วยค่า DISCARD ได้ สำหรับตัวแปรที่เอ็กซ์พอร์ตนี้ โปรแกรม **dbx** ยังคงแจ้งเตือนผู้ใช้ที่ไม่ตรงกัน แต่ดำเนินการต่อไปได้หากโมดูลที่ต้องพึ่งพาที่ไม่ตรงกันไม่สามารถเข้าถึงได้

ผู้ใช้ยังมีอ็อพชันของการเปลี่ยนทิศทางโปรแกรม **dbx** ให้กับโมดูลที่ต้องพึ่งพาการจับคู่โดยใช้แฟ็ลก **–p** สำหรับข้อมูลเพิ่มเติม โปรดดู คำสั่ง **dbx** ใน *Commands Reference, Volume 2*

การรันคำสั่งเชลล์จาก dbx

คุณสามารถรันคำสั่ง shell โดยไม่ต้องออกจากโปรแกรมดีบักโดยใช้คำสั่งย่อย **sh**

ถ้าป้อน **sh** โดยไม่ระบุคำสั่งใดๆ shell จะถูกป้อนเพื่อใช้งานจนกว่า shell นั้นจะออกไป ณ ช่วงเวลาที่การควบคุมส่งค่าไปยังโปรแกรม **dbx**

การแก้ไขบรรทัดรับคำสั่งใน dbx

คำสั่ง **dbx** จะมีคุณลักษณะในการแก้ไขบรรทัดรับคำสั่ง ซึ่งคล้ายกับที่ได้จัดเตรียมไว้โดย **Korn Shell** โหมด **vi** จัดเตรียมคุณลักษณะการแก้ไขที่เหมือนกับ **vi** ขณะที่โหมด **emacs** อนุญาตให้คุณควบคุมซึ่งคล้ายกับ **emacs**

You can turn these features on by using **dbx** subcommand **set –o** or **set edit**. ดังนั้น หากต้องการเปิดการแก้ไขบรรทัดรับคำสั่งในรูปของ **vi** คุณต้องพิมพ์คำสั่งย่อย `set edit vi` หรือ `set -o vi`

คุณยังสามารถใช้ตัวแปรสภาวะแวดล้อม **EDITOR** เพื่อตั้งค่าโหมดการแก้ไขได้

คำสั่ง **dbx** จะบันทึกคำสั่งที่ป้อนลงในไฟล์ประวัติ **.dbxhistory** ถ้าตัวแปรสภาวะแวดล้อม **DBXHISTFILE** ไม่ได้ตั้งค่าไว้ ไฟล์ประวัติที่ใช้คือ **$HOME/.dbxhistory**

ตามค่าดีฟอลต์ คำสั่ง **dbx** จะบันทึกข้อความของคำสั่ง 128 สุดท้ายที่ป้อน ตัวแปรสภาวะแวดล้อม **DBXHISTSIZE** สามารถนำมาใช้เพื่อเพิ่มข้อจำกัดนี้

การใช้การควบคุมโปรแกรม

โปรแกรมดีบัก **dbx** อนุญาตให้คุณตั้งค่าจุดพัก (ตำแหน่งการหยุด) ในโปรแกรม หลังจากที่ป้อนโปรแกรม **dbx** แล้ว คุณสามารถระบุบรรทัดหรือแอดเดรสที่เป็นจุดพัก จากนั้น รันโปรแกรมที่คุณต้องการดีบักด้วยโปรแกรม **dbx** โปรแกรมจะหยุดทำงานและรายงาน เมื่อโปรแกรมมาถึงจุดพัก จากนั้น คุณสามารถใช้คำสั่ง **dbx** เพื่อตรวจสอบสถานะของโปรแกรมของคุณ

ทางเลือกในการตั้งค่าจุดพักคือ รันโปรแกรมหนึ่งบรรทัดหรือหนึ่งคำสั่งต่อครั้ง โพรซีเดอร์จะรู้จักกันในนามของการก้าวทีละขั้น

การตั้งค่าและการลบ breakpoints

ใช้คำสั่งย่อย **stop** เพื่อตั้งค่าจุดพักในโปรแกรม **dbx** คำสั่งย่อย **stop** จะหยุดแอ็พพลิเคชันโปรแกรม เมื่อตรงตามเงื่อนไขบางอย่าง:

- *ตัวแปร* จะถูกเปลี่ยนไป เมื่อพารามิเตอร์ *variable* ถูกระบุไว้
- *เงื่อนไข* จะเป็นจริงเมื่อแฟล็ก **if** *Condition* ถูกนำมาใช้
- *โพรซีเดอร์* ถูกเรียก เมื่อแฟล็ก **ใน** *โพรซีเดอร์* ถูกนำมาใช้
- หมายเลขบรรทัด *SourceLine* จะมาถึง เมื่อ แฟล็ก **at** *SourceLine* ถูกนำมาใช้

> **หมายเหตุ:** ตัวแปร *SourceLine* สามารถระบุเป็นเลขจำนวนเต็ม หรือสตริงชื่อไฟล์ที่ตามด้วย : (โคลอน) และเลขจำนวนเต็ม

หลังจากคำสั่งเหล่านี้โปรแกรม **dbx** จะตอบกลับข้อความการรายงาน ID เหตุการณ์ที่เชื่อมโยงด้วยจุดพัก พร้อมกับการตีความคำสั่งของคุณ คุณสามารถเชื่อมโยงคำสั่งย่อย dbx กับ ID เหตุการณ์ที่ระบุได้โดยใช้คำสั่งย่อย **addcmd** คำสั่งย่อย dbx ที่เชื่อมโยงนี้ถูกออกคำสั่ง เมื่อมาถึงจุดพัก จุด trance หรือจุดดูที่สอดคล้องกับเหตุการณ์นี้ ใช้คำสั่งย่อย **delcmd** เพื่อลบคำสั่งย่อย **dbx** ที่เชื่อมโยงจาก ID เหตุการณ์ที่ระบุไว้

การรันโปรแกรม

คำสั่งย่อย **run** จะเริ่มต้นโปรแกรมของคุณ ซึ่งจะบอกให้โปรแกรม **dbx** เริ่มต้นรันอ็อบเจ็กต์ไฟล์ อ่านอาร์กิวเมนต์ใดๆ ตามที่ได้พิมพ์ไว้บนบรรทัดรับคำสั่ง shell คำสั่งย่อย **rerun** มีรูปแบบเดียวกับคำสั่งย่อย **run** แต่ต่างกันตรงที่ ถ้ามีการส่งผ่านอาร์กิวเมนต์แล้ว รายการอาร์กิวเมนต์จากการประมวลผลก่อนหน้า จะถูกนำมาใช้ หลังจากที่โปรแกรมของคุณเริ่มต้นขึ้นแล้ว โปรแกรมจะดำเนินการต่อ จนกระทั่งหนึ่งในเหตุการณ์ต่อไปนี้จะเกิดขึ้น:

- โปรแกรมมาถึงจุดพัก
- สัญญาณที่เกิดขึ้นไม่ได้ถูกละเว้น เช่น **INTERRUPT** หรือ **QUIT**
- เหตุการณ์แบบหลายกระบวนการจะเกิดขึ้นขณะที่เปิดใช้งาน การดีบักแบบหลายกระบวนการ
- โปรแกรมดำเนินการรูทีนย่อย **load, unload** หรือ **loadbind**

  > **หมายเหตุ:** โปรแกรม **dbx** จะข้ามเงื่อนไขนี้ไป หากตั้งค่าตัวแปรดีบัก **$ignoreload** นี้ไว้ นี่คือค่าดีฟอลต์ สำหรับตัวอย่างเพิ่มเติม โปรดดู คำสั่งย่อย **set**
- โปรแกรมเสร็จสิ้น

ในแต่ละกรณี โปรแกรมดีบัก **dbx** จะรับการควบคุม และแสดงข้อความที่อธิบายสาเหตุที่โปรแกรมหยุดทำงาน

มีหลายวิธีที่ใช้ดำเนินการกับโปรแกรมต่อ หากโปรแกรมหยุดทำงาน:

คำสั่ง	คำอธิบาย
cont	ดำเนินการกับโปรแกรมต่อจากจุดที่โปรแกรมหยุดทำงาน
detach	ดำเนินการกับโปรแกรมต่อจากจุดที่โปรแกรมหยุดทำงาน ออกจากโปรแกรมดีบัก วิธีการนี้มีประโยชน์หลังจากที่คุณได้แพ็ตช์โปรแกรมแล้ว และต้องการดำเนินการต่อโดยไม่ดีบักโปรแกรม
return	ดำเนินการกับการประมวลผลต่อจนกว่าจะพบกับการส่งคืนค่า *โพรซีเดอร์* หรือจนกว่าโพรซีเดอร์ปัจจุบันจะส่งคืนค่า ถ้าไม่ได้ระบุ *โพรซีเดอร์* ไว้
skip	ดำเนินการกับการประมวลผลต่อจนกว่าจะสิ้นสุดโปรแกรม หรือจนกว่า *จำนวน* + 1 จุดพักจะถูกเรียกใช้งาน
step	รันหนึ่งบรรทัดของซอร์ส หรือรันตาม *จำนวน* ของบรรทัดของซอร์สที่ระบุไว้
next	รันจนถึงบรรทัดของซอร์สถัดไป หรือรันตาม *จำนวน* ของบรรทัดของซอร์สที่ระบุไว้

เมธอดทั่วไปของการดีบักคือ การดีบักทีละขั้นตอน ตลอดทั้งโปรแกรม หนึ่งบรรทัดต่อครั้ง คำสั่งย่อย **step** และ **next** จะช่วยให้พบกับวัตถุประสงค์นั้น ความแตกต่างระหว่างคำสั่งสองคำสั่งนี้คือ ลักษณะที่ปรากฏขึ้นเท่านั้น เมื่อบรรทัดของซอร์สถัดไปที่ต้องการรัน เกี่ยวข้องกับการเรียกโปรแกรมย่อย ในกรณีนี้ คำสั่งย่อย **step** จะหยุดทำงานในโปรแกรมย่อย คำสั่งย่อย **next** จะรันจนกว่าโปรแกรมย่อยจะเสร็จสิ้น และหยุดทำงานที่คำสั่งถัดไปหลังจากการเรียก

ตัวแปรดีบัก **$stepignore** สามารถนำมาใช้เพื่อแก้ไขลักษณะการทำงานของคำสั่งย่อย **step** ได้ โปรดดู คำสั่ง **dbx** ใน *Commands Reference, Volume 2* สำหรับข้อมูลเพิ่มเติม

ไม่มีหมายเลขเหตุการณ์ที่เชื่อมโยงกับการหยุดทำงานเหล่านี้ เนื่องจากไม่มีเหตุการณ์ถาวรที่เชื่อมโยงกับการหยุดทำงานโปรแกรม

ถ้าโปรแกรมของคุณมีหลายเธรด เธรดเหล่านั้นจะรันตามปกติในระหว่างคำสั่งย่อย **cont, next, nexti** และ **step** คำสั่งเหล่านี้ทำหน้าที่รันเธรด (เธรดที่หยุดการประมวลผลแล้วโดยมาถึงจุดพัก) แม้ว่า หากเธรดอื่นรันโค้ดที่กำลังก้าวตามขั้นตอน การดำเนินการ **cont, next, nexti** หรือ **step** ยังคงดำเนินการต่อ จนกว่าเธรดที่รันอยู่จะถูกเรียกใช้งานโค้ดนั้น

ถ้าคุณต้องการให้คำสั่งย่อยเหล่านี้เรียกใช้งานเธรดที่รันอยู่เท่านั้น คุณสามารถตั้งค่าโปรแกรมดีบัก **dbx** ให้กับตัวแปร **$hold_next** ซึ่งอาจเป็นสาเหตุทำให้โปรแกรมดีบัก **dbx** พักเธรดของผู้ใช้อื่นทั้งหมด ในระหว่างคำสั่งย่อย **cont, next, nexti** และ **step**

> **หมายเหตุ:** ถ้าคุณใช้คุณลักษณะนี้ โปรดจำว่า เธรดที่พักจะไม่สามารถรีลีสล็อกใดๆ ได้ตามสิทธิ์ที่มี เธรดอื่นที่ต้องการหนึ่งในล็อกเหล่านี้อาจ deadlock โปรแกรมของคุณ

การแยกเอาต์พุต dbx จากเอาต์พุตของโปรแกรม

ใช้คำสั่งย่อย **screen** เพื่อดีบักโปรแกรมที่ถูกวางแนวหน้าจอแล้ว เช่น เท็กซ์เอดิเตอร์ หรือโปรแกรมกราฟิก คำสั่งย่อยนี้จะเปิด Xwindow สำหรับคำสั่ง **dbx** แบบโต้ตอบ โปรแกรมจะดำเนินการต่อในหน้าต่างเดิมที่ถูกสร้าง ถ้าไม่ได้ใช้คำสั่งย่อย **screen** เอาต์พุตโปรแกรม **dbx** จะถูกผสานกันด้วยการวางแนวหน้าจอเอาต์พุตโปรแกรม

การเรียกทำงานการติดตาม

คำสั่งย่อย **trace** จะบอกโปรแกรม **dbx** ให้พิมพ์ข้อมูลเกี่ยวกับสถานะของโปรแกรมที่ต้องการดีบัก ขณะที่โปรแกรมนั้นกำลังทำงานอยู่ คำสั่งย่อย **trace** สามารถลดระดับความเร็วของโปรแกรมให้ช้าลงได้ ขึ้นอยู่กับจำนวนงานที่โปรแกรม **dbx** ต้องทำ ซึ่งมีรูปแบบของการ trace โปรแกรมอยู่ด้วยกันห้ารูปแบบ:

- คุณสามารถ trace โปรแกรมแบบทีละขั้นตอน พิมพ์บรรทัดของซอร์สแต่ละบรรทัดที่ประมวลผล ตัวแปรดีบัก **$stepignore** สามารถนำมาใช้เพื่อแก้ไขลักษณะการทำงานของคำสั่งย่อย **trace** ได้ โปรดดู คำสั่งย่อย **set** สำหรับข้อมูลเพิ่มเติม
- คุณสามารถจำกัดการพิมพ์บรรทัดของซอร์สได้ เมื่อโพรซีเดอร์ที่ระบุนั้นแอ็คทีฟ คุณยังสามารถระบุเงื่อนไขที่เป็นทางเลือกเพื่อควบคุม เมื่อรายละเอียดการติดตามถูกสร้างขึ้น

- คุณสามารถแสดงข้อความได้ในแต่ละครั้งที่โพรซีเดอร์นั้นถูกเรียกหรือส่งคืนค่า
- คุณสามารถพิมพ์บรรทัดของซอร์สที่ระบุได้ เมื่อโปรแกรมมาถึงบรรทัดนั้น
- คุณสามารถพิมพ์ค่านิพจน์ได้ เมื่อโปรแกรมมาถึงบรรทัดของซอร์สที่ระบุ

การลบเหตุการณ์ trace จะเหมือนกับการลบเหตุการณ์หยุดทำงาน เมื่อคำสั่งย่อย **trace** ถูกประมวลผล ID เหตุการณ์ที่เชื่อมโยงจะถูกแสดงขึ้นพร้อมกับการแทนค่าภายในของเหตุการณ์

การแสดงและการจัดการไฟล์ต้นฉบับด้วยโปรแกรมดีบักการ dbx

ส่วนนี้อธิบายกระบวนการของการแสดงและการจัดการ ไฟล์ต้นฉบับด้วยโปรแกรมดีบัก dbx

คุณสามารถใช้โปรแกรมดีบัก **dbx** เพื่อค้นหาและแสดงส่วนของไฟล์ต้นฉบับสำหรับโปรแกรม

คุณไม่จำเป็นต้องใช้รายการซอร์สปัจจุบันเพื่อทำการค้นหา โปรแกรมดีบัก **dbx** ติดตามไฟล์ปัจจุบัน โพรซีเดอร์ปัจจุบัน และบรรทัดปัจจุบัน ถ้าไฟล์คอร์มีอยู่ บรรทัดปัจจุบันและ ไฟล์ปัจจุบันถูกตั้งค่าเริ่มต้นเป็นบรรทัดและไฟล์ที่มีคำสั่งซอร์ส ที่กระบวนการสิ้นสุด เป็น true เฉพาะเมื่อกระบวนการที่หยุด ในตำแหน่งที่คอมไพล์สำหรับการดีบัก

การเปลี่ยนพาธของไดเร็กทอรีต้นทาง

โดยค่าเริ่มต้นโปรแกรมดีบัก **dbx** ค้นหาไฟล์ต้นฉบับของโปรแกรมที่กำลังถูกดีบักในไดเร็กทอรี ดังต่อไปนี้:

- ไดเร็กทอรีที่ไฟล์ต้นฉบับอยู่ เมื่อถูกคอมไพล์ ไดเร็กทอรี นี้ถูกค้นหาเฉพาะถ้าคอมไพลเลอร์กำหนดพาธซอร์สในอ็อบเจ็กต์
- ไดเร็กทอรี Current
- ไดเร็กทอรีที่โปรแกรมอยู่ในปัจจุบัน

คุณสามารถเปลี่ยนรายการของไดเร็กทอรีที่จะถูกค้นหา โดยใช้ตัวเลือก **–I** บนบรรทัดการเรียกใช้ **dbx** หรือส่งคำสั่งย่อย **use** ภายในโปรแกรม **dbx** ตัวอย่างเช่น ถ้าคุณย้ายไฟล์ต้นฉบับไปที่ตำแหน่งใหม่ตั้งแต่ทำการคอมไพล์ คุณอาจต้องการใช้หนึ่งในคำสั่งเหล่านี้เพื่อระบุตำแหน่งเก่า ตำแหน่งใหม่ และตำแหน่งชั่วคราวบางส่วน

การแสดงไฟล์ปัจจุบัน

คำสั่งย่อย **list** อนุญาตให้คุณแสดงรายการบรรทัดซอร์ส

สัญลักษณ์ **$** (เครื่องหมายดอลลาร์) และ @ (เครื่องหมาย at) แสดง *SourceLineExpression* และมีประโยชน์กับคำสั่งย่อย **list**, **stop** และ **trace** สัญลักษณ์ **$** แสดงบรรทัดถัดไปที่จะถูกรัน สัญลักษณ์ @ แสดงบรรทัดถัดไป ที่จะถูกแสดงรายการ

คำสั่งย่อย **move** เปลี่ยนหมายเลขบรรทัดถัดไปที่จะถูกแสดง

การเปลี่ยนไฟล์หรือโพรซีเดอร์ปัจจุบัน

ใช้คำสั่งย่อย **func** และ **file** เพื่อ เปลี่ยนไฟล์ปัจจุบัน โพรซีเดอร์ปัจจุบัน และบรรทัดปัจจุบันภายในโปรแกรม **dbx** โดยไม่ต้องรันส่วนใดของโปรแกรมของคุณ

ค้นหาไฟล์ปัจจุบันเพื่อหาข้อความที่ตรงกับ นิพจน์ทั่วไป ถ้าพบ บรรทัดปัจจุบันถูกตั้งค่าเป็นบรรทัด ที่มีข้อความตรงกัน ไวยากรณ์ของคำสั่งย่อย search คือ:

/ *RegularExpression* **[/]**
ค้นหาไปข้างหน้า ในไฟล์ต้นฉบับปัจจุบัน ตามนิพจน์ที่กำหนด

? *RegularExpression* **[?]**
ค้นหาย้อนกลับ ในไฟล์ต้นฉบับปัจจุบัน ตามนิพจน์ที่กำหนด

ถ้าคุณทำซ้ำการค้นหาโดยไม่มีอาร์กิวเมนต์ คำสั่ง **dbx** ค้นหานิพจน์ทั่วไป ก่อนหน้าอีกครั้ง การค้นหาวนผ่านจุดสิ้นสุดหรือจุดเริ่มต้น ของไฟล์

คุณยังสามารถเรียกโปรแกรมแก้ไขข้อความภายนอกเพื่อใช้กับ ไฟล์ต้นฉบับของคุณโดยใช้คำสั่งย่อย **edit** คุณสามารถแทนค่าตัวแก้ไขเริ่มต้น (**vi**) โดยตั้งค่า ตัวแปรสภาวะแวดล้อม **EDITOR** เป็นตัวแก้ไขที่คุณต้องการ ก่อนการเริ่มโปรแกรม **dbx**

โปรแกรม **dbx** ดำเนินการควบคุม กระบวนการต่อเมื่อเซสชันการแก้ไขสมบูรณ์

การดีบักโปรแกรมที่เกี่ยวข้องกับหลายเธรด

โปรแกรมที่เกี่ยวข้องกับเธรดผู้ใช้หลายเธรดเรียกรูทีนย่อย **pthread_create** เมื่อกระบวนการเรียกรูทีนย่อยนี้ ระบบปฏิบัติการสร้างเธรดของการกระทำการใหม่ภายในกระบวนการ เมื่อดีบักโปรแกรมแบบมัลติเธรด จำเป็น ต้องทำงานกับแต่ละเธรดของกระบวนการ โปรแกรม **dbx** ทำงาน กับ user threads เท่านั้น: ในเอกสาร **dbx** คำว่า *thread* คำเดียวโดยปกติหมายถึง *user thread* โปรแกรม **dbx** กำหนดหมายเลขเธรดเฉพาะ ให้กับแต่ละเธรดในกระบวนการที่ถูกดีบัก และยังสนับสนุน แนวคิดของการรันเธรดปัจจุบัน:

เธรดที่รันอยู่
user thread มีหน้าที่ในการหยุดโปรแกรมโดย การเข้าสู่จุดหยุด คำสั่งย่อยที่ single-step โปรแกรมทำงานกับ เธรดที่รันอยู่

เธรด Current
user thread ที่คุณกำลังตรวจสอบ คำสั่งย่อยที่แสดงข้อมูล ทำงานในบริบทของเธรดปัจจุบัน

โดยค่าเริ่มต้น เธรดที่รันอยู่และเธรดปัจจุบันเหมือนกัน คุณสามารถ เลือกเธรดปัจจุบันโดยใช้คำสั่งย่อย **thread** เมื่อคำสั่งย่อย **thread** แสดงเธรด บรรทัดของเธรดปัจจุบันจะขึ้นต้นด้วย > ถ้าเธรดที่รันอยู่ ไม่เหมือนกับเธรดปัจจุบัน, บรรทัดจะถูกนำหน้าด้วย *

การดีบักโปรแกรมที่เกี่ยวข้องกับหลายกระบวนการ

โปรแกรมที่เกี่ยวข้องกับหลายกระบวนการเรียกรูทีนย่อย **fork** และ **exec** เมื่อโปรแกรม forks ระบบปฏิบัติการจะสร้าง กระบวนการอื่นที่มีอิมเมจเหมือนกับต้นฉบับ กระบวนการต้นฉบับ เรียกว่ากระบวนการ parent กระบวนการที่สร้างเรียกว่ากระบวนการ child

เมื่อกระบวนการดำเนินรูทีนย่อย **exec** โปรแกรมใหม่เข้าแทนที่โปรแกรมต้นฉบับ ภายใต้สภาวะปกติ โปรแกรมดีบัก ดีบักเฉพาะกระบวนการ parent อย่างไรก็ตาม โปรแกรม **dbx** สามารถติดตามการกระทำการและดีบักกระบวนการใหม่ เมื่อคุณใช้คำสั่งย่อย **multproc** คำสั่งย่อย **multproc** เปิดใช้งาน การดีบัก multiprocess

เมื่อการดีบัก multiprocess ถูกเปิดใช้งานและ fork เกิดขึ้น กระบวนการ parent และ child จะถูกหยุด เทอร์มินัลเสมือน Xwindow แยกถูกเปิดขึ้นมาสำหรับเวอร์ชันใหม่ของโปรแกรม **dbx** เพื่อควบคุม การรันของกระบวนการ child:

```
(dbx) multproc on
(dbx) multproc
การดีบัก multi-process ถูกเปิดใช้งาน
(dbx) run
```

เมื่อ fork เกิดขึ้น การกระทำการถูกหยุดใน parent และโปรแกรม **dbx** แสดงสถานะของโปรแกรม:

```
application forked, child pid = 422, process stopped, awaiting input
stopped due to fork with multiprocessing enabled in fork at 0x1000025a (fork+0xe)
(dbx)
```

จากเทอร์มินัลเสมือน Xwindow อีกหน้าต่างถูกเปิด เพื่อดีบักกระบวนการ child:

```
debugging child, pid=422, process stopped, awaiting input
stopped due to fork with multiprocessing enabled in fork at 0x10000250
10000250 (fork+0x4) )80010010    1       r0,0x10(r1)
(dbx)
```

ที่จุดนี้ เซสชันการดีบักที่แยกกัน กำลังรันอยู่ เซสชันการดีบักสำหรับกระบวนการ child รักษาจุดหยุดทั้งหมด จากกระบวนการ parent แต่เฉพาะกระบวนการ parent เท่านั้นที่ถูกรันซ้ำได้

เมื่อโปรแกรมดำเนินการรูทีนย่อย **exec** ในโหมดการดีบัก multiprocess โปรแกรมแทนที่ตัวเอง และข้อมูลสัญลักษณ์ ดั้งเดิมจะหมดอายุไป จุดหยุดทั้งหมดถูกลบเมื่อรูทีนย่อย **exec** รัน; โปรแกรมใหม่ถูกหยุดและจำแนก สำหรับการดีบักที่สำคัญ โปรแกรม **dbx** เชื่อมต่อ ตัวเองเข้ากับอิมเมจโปรแกรมใหม่ สร้างรูทีนย่อยเพื่อระบุ ชื่อของโปรแกรมใหม่ รายงานชื่อ แล้ว พร้อมต์เพื่อรับอินพุต พร้อมต์มีลักษณะดังต่อไปนี้:

```
(dbx) multproc
Multi-process debugging is enabled
(dbx) run
Attaching to program from exec . . .
Determining program name . . .
Successfully attached to /home/user/execprog . . .
Reading symbolic information . . .
(dbx)
```

ถ้าโปรแกรมแบบมัลติเธรด forks กระบวนการชายน์ใหม่จะมี หนึ่งเธรดเท่านั้น กระบวนการควรเรียกรูทีนย่อย **exec** มิฉะนั้น ข้อมูลสัญลักษณ์ดั้งเดิมจะถูกคงไว้ และคำสั่งย่อยที่สัมพันธ์กับเธรด (เช่น **thread**) แสดงอ็อบเจ็กต์ของ กระบวนการ parent ซึ่งใช้งานไม่ได้แล้ว ถ้ารูทีนย่อย **exec** ถูกเรียก ข้อมูลสัญลักษณ์ดั้งเดิมถูกกำหนดค่าซ้ำ และคำสั่งย่อยที่สัมพันธ์ กับเธรดแสดงอ็อบเจ็กต์ในกระบวนการ child ใหม่

เป็นไปได้ที่จะติดตามกระบวนการ child ของ fork โดยไม่เปิด Xwindow ใหม่โดยใช้แฟล็ก **child** ของคำสั่งย่อย **multproc** เมื่อ กระบวนการมีการ fork ถูก สร้าง **dbx** ติดตามกระบวนการ child แฟล็ก **parent** ของคำสั่งย่อย **multproc** ทำให้ **dbx** หยุดทำงาน เมื่อโปรแกรม forks แต่จากนั้นจะติดตาม parent ทั้งแฟล็ก **child** และ **parent** ติดตามกระบวนการที่ดำเนินการ แฟล็กเหล่านี้มี ประโยชน์มากสำหรับการดีบักโปรแกรม เมื่อ Xwindows ไม่ได้รันอยู่

การตรวจสอบข้อมูลโปรแกรม

ส่วนนี้อธิบายวิธีตรวจสอบ ทดสอบ และปรับเปลี่ยนข้อมูล โปรแกรม

การจัดการกับสัญญาณ

โปรแกรมดีบัก **dbx** สามารถรอจับหรือ ละเว้นสัญญาณ ก่อนที่จะถูกส่งไปที่โปรแกรมของคุณ แต่ละครั้ง ที่โปรแกรมของคุณจะได้รับสัญญาณโปรแกรม **dbx** จะถูกแจ้งเตือน ถ้าสัญญาณจะถูกละเว้น สัญญาณจะถูกส่งผ่านไปที่โปรแกรม ของคุณ หรือไม่แล้วโปรแกรม **dbx** หยุดการทำงาน ของโปรแกรมและแจ้งคุณว่ามีการจับสัญญาณได้ โปรแกรม **dbx** ไม่สามารถละเว้นสัญญาณ **SIGTRAP** ถ้ามาจากกระบวนการภายนอกกระบวนการดีบัก ในโปรแกรม แบบมัลติเธรด สัญญาณสามารถส่งไปยังเธรดเฉพาะผ่านรูทีนย่อย **pthread_kill** โดยค่าเริ่มต้นโปรแกรม **dbx** หยุดทำงานและแจ้งคุณว่าได้มีการจับ สัญญาณ ถ้าคุณร้องขอสัญญาณจะถูกส่งไปที่โปรแกรมของคุณ โดยใช้คำสั่งย่อย **ignore,** โปรแกรม **dbx** ละเว้นสัญญาณและส่งผ่านไปที่เธรด ใช้ **catch** และ **ignore** เพื่อเปลี่ยน การจัดการเริ่มต้น

ในตัวอย่างดังต่อไปนี้ โปรแกรมใช้ **SIGGRANT** และ **SIGREQUEST** เพื่อ จัดการการจัดสรรทรัพยากร เพื่อให้โปรแกรม **dbx** ดำเนิน ต่อในแต่ละครั้งที่ได้รับสัญญาณ ให้ป้อน:

```
(dbx) ignore GRANT
(dbx) ignore SIGREQUEST
(dbx) ignore
CONT CLD ALARM KILL GRANT REQUEST
```

โปรแกรมดีบัก **dbx** สามารถบล็อก สัญญาณที่ส่งมาที่โปรแกรมของคุณ ถ้าคุณตั้งค่าตัวแปร **$sigblock** โดยค่าเริ่มต้น สัญญาณที่ได้รับผ่านโปรแกรม **dbx** ถูกส่งไปที่ ซอร์สโปรแกรมหรืออ็อบเจ็กต์ไฟล์ที่ระบุโดยพารามิเตอร์ **dbx** *ObjectFile* ถ้าตัวแปร **$sigblock** ถูกตั้งค่าโดยใช้คำสั่งย่อย **set** สัญญาณ ที่ได้รับโดยโปรแกรม **dbx** จะไม่ถูกส่งไปที่ ซอร์สโปรแกรม ถ้าคุณต้องการให้สัญญาณถูกส่งไปที่โปรแกรม ให้ใช้คำสั่งย่อย **cont** และระบุสัญญาณ เป็นตัวถูกดำเนินการ

คุณสามารถใช้คุณลักษณะนี้เพื่ออินเตอร์รัปต์การกระทำการ ของโปรแกรมที่รันอยู่ภายใต้โปรแกรมดีบัก **dbx** สถานะโปรแกรมสามารถถูกตรวจสอบก่อนการดำเนินการต่อตามปกติ ถ้าตัวแปร **$sigblock** ไม่ได้ถูกตั้งค่า, การอินเตอร์รัปต์ การกระทำการทำให้สัญญาณ **SIGINT** ถูกส่งไปที่ โปรแกรม ซึ่งทำให้เกิดการกระทำการ เมื่อดำเนินต่อ ไปที่สาขาไปที่ ตัวจัดการสัญญาณ ถ้ามีอยู่

โปรแกรมตัวอย่างดังต่อไปนี้แสดงวิธีกระทำการ โดยใช้โปรแกรมดีบัก **dbx** เปลี่ยนแปลง เมื่อตัวแปร **$sigblock** ถูกตั้งค่า:

```
#include <signal.h>
#include <stdio.h>
void inthand( ) {
        printf("\nSIGINT received\n");
        exit(0);
}

main( )
{
        signal(SIGINT, inthand);
        while (1) {
                    printf(".");
                  fflush(stdout);
                sleep(1);
        }
}
```

เซสชันตัวอย่างดังต่อไปนี้โดยมีโปรแกรม **dbx** ใช้โปรแกรมก่อนหน้าเป็น ไฟล์ต้นฉบับ ในการรันครั้งแรกของโปรแกรม ตัวแปร **$sigblock** ไม่ถูกตั้งค่า ระหว่างการรีรันตัวแปร **$sigblock** ถูกตั้งค่า ข้อคิดเห็นถูกแทรกระหว่างวงเล็บมุมทางด้านขวา:

```
dbx version 3.1.
Type ‘help’ for help.
reading symbolic information ...
```

```
(dbx)run
.........^C                      <User pressed Ctrl-C here!>
interrupt in sleep at 0xd00180bc
0xd00180bc (sleep+0x40) 80410014         1        r2,0x14(r1)
(dbx) cont

SIGINT received

execution completed
(dbx) set $sigblock
(dbx) rerun
[ looper ]
..............^C                  <User pressed Ctrl-C here!>
interrupt in sleep at 0xd00180bc
0xd00180bc (sleep+0x40) 80410014         1        r2,0x14(r1)
(dbx) cont
....^C   <Program did not receive signal, execution continued>

interrupt in sleep at 0xd00180bc
0xd00180bc (sleep+0x40) 80410014         1        r2,0x14(r1)
(dbx) cont 2                     <End program with a signal 2>

SIGINT received

execution completed
(dbx)
```

การเรียกใช้โพรซีเดอร์

คุณสามารถเรียกโพรซีเดอร์โปรแกรมของคุณจากโปรแกรม **dbx** เพื่อทดสอบอาร์กิวเมนต์ที่ต่างกัน คุณสามารถ เรียกรูทีนวินิจฉัยที่จัดรูปแบบข้อมูลเพื่อช่วยในการดีบักได้เช่นกัน ใช้คำสั่งย่อย **call** หรือคำสั่งย่อย **print** เพื่อเรียกโพรซีเดอร์

การแสดงสแต็กการติดตาม

เมื่อต้องการแสดงการเรียกโพรซีเดอร์ก่อนโปรแกรม หยุด ให้ใช้คำสั่ง **where**

ในตัวอย่างดังต่อไปนี้ อ็อบเจ็กต์ไฟล์ที่รันได้ **hello** ประกอบด้วยสองไฟล์ต้นฉบับและ สามโพรซีเดอร์ รวมโพรซีเดอร์มาตรฐาน `main` โปรแกรมหยุดที่จุดหยุดในโพรซีเดอร์ `sub2`

```
(dbx) run
[1] stopped in sub2 at line 4 in file "hellosub.c"
(dbx) where
sub2(s = "hello", n = 52), line 4 in "hellosub.c"
sub(s = "hello", a = -1, k = delete), line 31 in "hello.c"
main(), line 19 in "hello.c"
```

การติดตามสแต็กแสดงการเรียกในลำดับ ย้อนกลับ เริ่มจากด้านล่าง เหตุการณ์ดังต่อไปนี้เกิดขึ้น:

1. Shell เรียก `main`
2. `main` เรียกโพรซีเดอร์ `sub` ที่บรรทัด 19 ด้วยค่า `s = "hello"`, `a = -1`, และ `k = delete`
3. `sub` เรียกโพรซีเดอร์ `sub2` ที่บรรทัด 31 ด้วยค่า `s = "hello"` และ `n = 52`
4. โปรแกรมที่หยุดในโพรซีเดอร์ `sub2` ที่บรรทัด 4

ส่วนของการติดตามสแต็กจากตัวเลขเฟรม 0 ถึง เฟรมหมายเลข 1 สามารถถูกแสดงโดยใช้ `where 0 1`

```
(dbx)run
[1] stopped in sub2 at line 4 in file "hellosub.c"
(dbx) where 0 1
sub2(s = "hello", n = 52), line 4 in "hellosub.c"
sub(s = "hello", a = -1, k = delete), line 31 in "hello.c"
```

หมายเหตุ: ตั้งค่าตัวแปรโปรแกรมดีบัก **$noargs** เพื่อ ปิดการแสดงอาร์กิวเมนต์ที่ส่งไปที่โพรซีเดอร์ ตั้งค่าตัวแปรโปรแกรม ดีบัก **$stack_details** เพื่อแสดง หมายเลขเฟรมและการตั้งค่ารีจิสเตอร์สำหรับแต่ละฟังก์ชันหรือโพรซีเดอร์ที่ที่ใช้งานอยู่

คุณยังสามารถแสดงส่วนของสแต็ก ด้วยคำสั่งย่อย **up, down** และ **frame**

การแสดงและการแก้ไขตัวแปร

เมื่อต้องการแสดงนิพจน์ ใช้คำสั่งย่อย **print** เมื่อต้องการพิมพ์ชื่อและค่าของตัวแปร ให้ใช้คำสั่งย่อย **dump** ถ้า โพรซีเดอร์ที่กำหนดเป็น period, ตัวแปรที่ใช้งานอยู่ทั้งหมดจะถูกพิมพ์ ถ้าพารามิเตอร์ *PATTERN* ถูกระบุ ไม่เพียงแสดงสัญลักษณ์ที่ระบุ แต่สัญลักษณ์ทั้งหมดที่ ตรงกับ *PATTERN* จะถูกพิมพ์ เมื่อต้องการปรับเปลี่ยน ค่าของตัวแปร ใช้คำสั่งย่อย **assign**

ในตัวอย่างดังต่อไปนี้ โปรแกรม C มี ตัวแปร automatic integer `x` มีค่า 7, และพารามิเตอร์ `s` และ `n` ในโพรซีเดอร์ `sub2`:

```
(dbx) print x, n
7 52
(dbx) assign x = 3*x
(dbx) print x
21
(dbx) dump
sub2(s = "hello", n = 52)
x = 21
```

การแสดงข้อมูลที่เกี่ยวกับเธรด

เมื่อต้องการแสดงข้อมูล user threads, mutexes, conditions, และ attribute objects ให้ใช้คำสั่งย่อย **thread, mutex, condition** และ **attribute** คุณยังสามารถใช้คำสั่งย่อย **print** กับอ็อบเจ็กต์เหล่านี้ ในตัวอย่างดังต่อไปนี้ เธรดที่รันอยู่คือ เธรด 1 ผู้ใช้ตั้งค่าเธรดปัจจุบันเป็นเธรด 2 แสดง เธรด พิมพ์ข้อมูลบนเธรด 1 และสุดท้ายพิมพ์ข้อมูล บนอ็อบเจ็กต์ thread-related ทั่วไป

```
(dbx) thread current 2
(dbx) thread
 thread  state-k   wchan state-u   k-tid mode held scope function
*$t1     run             running   12755   u   no   pro  main
>$t2     run             running   12501   k   no   sys  thread_1
```

```
(dbx) print $t1
(thread_id = 0x1, state = run, state_u = 0x0, tid = 0x31d3, mode = 0x1, held = 0x0, priority = 0x3c,
    policy = other, scount = 0x1, cursig = 0x5, attributes = 0x200050f8)
```

```
(dbx) print $a1,$c1,$m2
(attr_id = 0x1, type = 0x1, state = 0x1, stacksize = 0x0, detachedstate = 0x0, process_shared = 0x0,
 contentionscope = 0x0, priority = 0x0, sched = 0x0, inherit = 0x0, protocol = 0x0, prio_ceiling = 0x0)
(cv_id = 0x1, lock = 0x0, semaphore_queue = 0x200032a0, attributes = 0x20003628)
(mutex_id = 0x2, islock = 0x0, owner = (nil), flags = 0x1, attributes = 0x200035c8)
```

ขอบเขตของชื่อ

ชื่อแยกแยะก่อนโดยใช้สโคป static ของฟังก์ชันปัจจุบัน สโคปไดนามิกถูกใช้ ถ้าชื่อไม่ได้ถูกกำหนด ในสโคปแรก ถ้าการค้นหาสแตติกและไดนามิก ไม่พบข้อมูล สัญลักษณ์แบบสุ่มถูกเลือกและข้อความ การใช้ QualifiedName ถูกพิมพ์ คุณสามารถเขียนทับโพรซีเดอร์การแยกแยะชื่อโดยกำหนดคุณสมบัติ ตัวระบุด้วยชื่อบล็อก (เช่น *Module.Variable*) ไฟล์ต้นฉบับถูกจัดการเช่นเดียวกับโมดูลที่ตั้งชื่อด้วยชื่อไฟล์ โดยไม่มีส่วนเติมท้าย ตัวอย่างเช่นตัวแปร *x* ซึ่งถูก ประกาศในโพรซีเดอร์ sub ภายในไฟล์ **hello.c** มีชื่อที่ระบุแบบเต็ม **hello.sub.x** ตัวโปรแกรมเองมีจุดสำหรับ ชื่อ

คำสั่งย่อย **which** และ **whereis** มีประโยชน์ในการระบุว่า สัญลักษณ์ใดที่พบเมื่อ มีสัญลักษณ์ที่มีชื่อเหมือนกัน

การใช้โอเปอเรเตอร์และ modifiers ในนิพจน์

โปรแกรม **dbx** สามารถแสดง ช่วงกว้างของนิพจน์ ระบุนิพจน์ที่มีชุดย่อยทั่วไป ของไวยากรณ์ C พร้อมด้วยส่วนขยาย FORTRAN บางส่วน

*** (asterisk) หรือ ^ (caret)**
แสดงการยกเลิกอ้างอิงทางอ้อมหรือการยกเลิกอ้างอิงตัวชี้

[ ] (brackets) หรือ ( ) (parentheses)
หมายถึงนิพจน์ subscript array

. (จุด)
ใช้ตัวดำเนินการอ้างอิงฟิลด์นี้กับตัวชี้และโครงสร้าง ซึ่งทำให้โอเปอเรเตอร์ C -> (ลูกศร) ไม่จำเป็น แม้ว่าจะสามารถใช้ได้

& (เครื่องหมายแอมเปอร์แซนด์)
รับแอดเดรสของตัวแปร

.. (สองจุด)
แยกขอบเขตบนและล่าง เมื่อระบุส่วนย่อย ของระบุ ตัวอย่างเช่น: **n[1..4]**.

ชนิดของการดำเนินการต่อไปนี้ใช้ได้ ในนิพจน์:

พีชคณิต
=, -, *,/ (การหารแบบมีจุดทศนิยม) **div** (การหารจำนวนเต็ม) **mod, exp** (การยกกำลัง)

Bitwise
-, I, bitand, xor, ~, <<, >>

โลจิคัล
or, and, not, II, &&

การเปรียบเทียบ
<, >, <=, >=, <> or **!**=, = or ==

อื่นๆ
sizeof

นิพจน์โลจิคัลและการเปรียบเทียบ ใช้เป็น เงื่อนไขได้ในคำสั่งย่อย **stop** และ **trace**

การตรวจสอบชนิดของนิพจน์

โปรแกรมดีบัก **dbx** ตรวจสอบ ชนิดของนิพจน์ คุณสามารถเขียนทับค่าชนิดนิพจน์โดยใช้ ตัวดำเนินการเปลี่ยนชื่อหรือแคสติง การเปลี่ยนชื่อชนิดมีสามรูปแบบ:

- *Typename* (*Expression*)
- *Expression* \ *Typename*
- (*Typename*) *Expression*

> **หมายเหตุ:** เมื่อคุณ แคสท์ไปเป็น(หรือมาจาก) ค่า structure, union, หรือ class การแคสท์เป็นแบบจัดมาจากทางซ้าย อย่างไรก็ตาม เมื่อแคสท์จากคลาสไปเป็นคลาสฐาน กฎไวยากรณ์ C++ เป็นดังนี้

ตัวอย่างเช่น เมื่อต้องการเปลี่ยนชื่อตัวแปร x ซึ่ง x เป็นจำนวนเต็มมีค่า 97 ให้ป้อน:

```
(dbx) print char (x), x \ char, (char) x, x,
'a' 'a' 'a' 97
```

ตัวอย่างดังต่อไปนี้แสดงวิธีที่คุณสามารถใช้ รูปแบบการเปลี่ยนชื่อชนิด (*Typename*) *Expression* :

```
print (float) i
print ((struct qq *) void_pointer)->first_element
```

ข้อบังคับต่อไปนี้ใช้กับ C-style typecasting สำหรับโปรแกรมดีบัก **dbx**:

- ชนิด Fortran (integer*1, integer*2, integer*4, logical*1, logical*2, logical*4, and so on) ไม่ได้รับการ สนับสนุนเป็นตัวดำเนินการแคสท์
- ถ้าตัวแปรที่ใช้งานอยู่มีชื่อเหมือนกับ ชนิดฐานหรือชนิดที่ผู้ใช้กำหนด, ชนิดจะไม่สามารถถูก ใช้เป็นตัวดำเนินการแคสท์ สำหรับ C-style typecasting

คำสั่งย่อย **whatis** พิมพ์ การประกาศของตัวระบุ ซึ่งคุณสามารถระบุถึงได้ ด้วยชื่อบล็อก

ใช้ **$$***TagName* construct เพื่อพิมพ์การประการศของ enumeration, structure หรือ union tag

ชนิดของนิพจน์คำสั่งย่อย **assign** ต้องตรงกับ ชนิดตัวแปรที่คุณกำหนด ถ้าชนิดไม่ตรง ข้อความแสดงข้อผิดพลาด จะถูกแสดง เปลี่ยนชนิดนิพจน์โดยใช้การเปลี่ยนชื่อชนิด ปิดใช้งานการตรวจสอบชนิดโดยการตั้งค่าตัวแปรโปรแกรมดีบัก **dbx** **$unsafeassign** พิเศษ

การโฟลด์ตัวแปรเป็นตัวพิมพ์เล็กและตัวพิมพ์ใหญ่

โดยค่าเริ่มต้นโปรแกรม **dbx** โฟลด์ สัญลักษณ์จากภาษาปัจจุบัน ถ้าภาษาปัจจุบัน คือ C, C++ หรือ undefined สัญลักษณ์จะไม่ถูกโฟลด์ ถ้าภาษาปัจจุบัน คือ Fortran สัญลักษณ์จะถูกโฟลด์เป็นตัวพิมพ์เล็ก ภาษาปัจจุบันมี เป็นไม่ได้กำหนด ถ้าโปรแกรมอยู่ในส่วนโค้ดที่ไม่ได้ ถูกคอมไพล์ด้วยแฟล็ก **debug** คุณสามารถ เขียนทับการจัดการค่าเริ่มต้นด้วยคำสั่งย่อย **case**

การใช้คำสั่งย่อย **case** โดยไม่มีอาร์กิวเมนต์ แสดงโหมดตัวพิมพ์ปัจจุบัน

คอมไพลเลอร์ Fortran แปลงสัญลักษณ์โปรแกรมทั้งหมด เป็นตัวพิมพ์เล็ก คอมไพลเลอร์ C ไม่แปลง อย่างไรก็ตามบางคอมไพลเลอร์ของ Fortran อาจไม่สร้างสัญลักษณ์ตัวพิมพ์เล็กเสมอไป ตัวอย่างเช่น กำหนด โพรซีเดอร์ชื่อ **proc1** ในโมดูลชื่อ **mod2** คอมไพลเลอร์ XLF Fortran สร้างสัญลักษณ์ **__mod2_MOD_proc1** ซึ่งเป็นตัวพิมพ์ผสมกัน ใน สถานการณ์เช่นนี้ คุณต้องเปลี่ยน **ตัวพิมพ์**ใน โปรแกรม **dbx** ให้เป็นตัวพิมพ์ **ผสม**

การเปลี่ยนเอาต์พุตการพิมพ์ด้วยตัวแปรโปรแกรมดีบักพิเศษ

ใช้คำสั่งย่อย **set** เพื่อตั้งค่าตัวแปรโปรแกรมดีบัก **dbx** พิเศษดังต่อไปนี้ เพื่อรับผลที่ต่างไปจากคำสั่งย่อย **print**:

$hexints
พิมพ์นิพจน์จำนวนเต็มแบบเลขฐานสิบหก

$hexchars
พิมพ์นิพจน์อักขระแบบเลขฐานสิบหก

$hexstrings
พิมพ์แอดเดรสของสตริงอักขระ ไม่ใช่ ตัวสตริงเอง

$octints
พิมพ์นิพจน์จำนวนเต็มแบบเลขฐานแปด

$expandunions
พิมพ์ฟิลด์ภายใน union

$pretty
แสดงชนิด C และ C++ แบบซับซ้อนในรูปแบบ **pretty**

$print_dynamic
พิมพ์ชนิดไดนามิกของอ็อบเจ็กต์ C++

$show_vft
พิมพ์ Virtual Function Table ขณะพิมพ์อ็อบเจ็กต์ C++

ตั้งค่าและยกเลิกการตั้งค่าตัวแปรโปรแกรมดีบัก เพื่อรับผลลัพธ์ที่ต้องการ ตัวอย่างเช่น:

```
(dbx) whatis x; whatis i; whatis s
int x;
char i;
char *s;
(dbx) print x, i, s
375 'c' "hello"
(dbx) set $hexstrings; set $hexints; set $hexchars
(dbx) print x, i, s
0x177 0x63 0x3fffe460
(dbx) unset $hexchars; set $octints
(dbx) print x, i
0567 'c'
(dbx) whatis p
struct info p;
(dbx) whatis struct info
struct info {
    int x;
    double position[3];
    unsigned char c;
    struct vector force;
};
(dbx) whatis struct vector
struct vector {
    int a;
```

```
    int b;
    int c;
};
(dbx) print p
(x = 4, position = (1.3262493258532527e-315, 0.0, 0.0),
c = '\0', force = (a = 0, b = 9, c = 1))(dbx) set $pretty="on"
(dbx) print p
{
    x = 4
    position[0] = 1.3262493258532527e-315
    position[1] = 0.0
    position[2] = 0.0
    c = '\0'
    force = {
        a = 0
        b = 9
        c = 1
    }
}
(dbx) set $pretty="verbose"
(dbx) print p
x = 4
position[0] = 1.3262493258532527e-315
position[1] = 0.0
position[2] = 0.0
c = '\0'
force.a = 0
force.b = 9
force.c = 1
```

เมื่อ show_vft ไม่ได้ถูกตั้งค่าและอ็อบเจ็กต์ ถูกพิมพ์โดยใช้คำสั่งย่อย Virtual Function Table (VFT) จะไม่ถูกพิมพ์ ถ้ามีการตั้งค่า VFT จะถูกแสดง ตัวอย่างเช่น:

```
(dbx) p *d
    B1:(int_in_b1 = 91)
    B2:(int_in_b2 = 92)
(int_in_d = 93)
(dbx) p *b2
(int_in_b2 = 20)
(dbx)set $show_vft
(dbx) p *d
    B1:(B1::f1(), int_in_b1 = 91)
    B2:(D::f2(), int_in_b2 = 92)
(int_in_d = 93)
(dbx) p *b2
(B2::f2(), int_in_b2 = 20)
(dbx)
```

เมื่อ print_dynamic ไม่ได้ถูกตั้งค่า อ็อบเจ็กต์ จะถูกแสดงเป็นต่อแม่แบบ static type's (ข้อมูลที่ถูกกำหนดใน ซอร์สโค้ด) หรือไม่แล้วจะถูกแสดงเป็นต่อแม่แบบ dynamic type's (เป็นอ็อบเจ็กต์ก่อนที่จะถูกแคสท์) ตัวอย่างเช่น:

```
(dbx) r
[1] stopped in main at line 57
   57   A *obj1 = new A();
(dbx) n
```

```
stopped in main at line 58
   58   A *obj2 = new B();
(dbx) n
stopped in main at line 59
   59   cout<<" a = "<<obj2->a<<" b = "<<obj2->b<<endl;
(dbx) p *obj2
(a = 1, b = 2)
(dbx)set $print_dynamic
(dbx) print *obj2
    A:(a = 1, b = 2)
(c = 3, d = 4)
(dbx)
```

การดีบักที่ระดับเครื่องด้วย dbx

คุณสามารถใช้โปรแกรมดีบัก **dbx** เพื่อ ตรวจสอบโปรแกรมที่ระดับภาษาแอสเซมบลี คุณสามารถแสดงและปรับเปลี่ยน แอดเดรสหน่วยความจำ แสดงคำสั่ง assembler คำสั่ง single-step ตั้งค่าจุดหยุดและติดตามเหตุการณ์ที่แอดเดรสหน่วยความจำและแสดงรีจิสเตอร์

ในคำสั่งและตัวอย่าง แอดเดรส เป็นนิพจน์ที่ประเมินค่าแอดเดรสหน่วยความจำ รูปแบบธรรมดาที่สุด ของแอดเดรสคือจำนวนเต็มและนิพจน์ที่รับแอดเดรสของตัวระบุ ด้วยตัวดำเนินการ **&** (ampersand) คุณยังสามารถระบุ แอดเดรสเป็นนิพจน์ปิดด้วยวงเล็บในคำสั่งระดับเครื่อง แอดเดรสสามารถประกอบด้วยแอดเดรสอื่นและตัวดำเนินการ **+** (บวก), **-** (ลบ) และเปลี่ยนทิศทาง (ยูนารี *)

ส่วนต่อไปนี้มีข้อมูลเพิ่มเติม เกี่ยวกับการดีบักที่ระดับเครื่องด้วยโปรแกรม **dbx**

การใช้ลงทะเบียนเครื่อง

ใช้ **รีจิสเตอร์** คำสั่งย่อยเพื่อดูค่าของรีจิสเตอร์เครื่อง รีจิสเตอร์ถูกแบ่งเป็นสามกลุ่ม: general-purpose, floating-point, และ system-control

รีจิสเตอร์ General-purpose

รีจิสเตอร์ General-purpose แสดงโดย **$r***Number* โดยที่ *Number* แสดง จำนวนของรีจิสเตอร์

> **หมายเหตุ:** ค่ารีจิสเตอร์ อาจถูกตั้งค่าเป็นค่าเลขฐานสิบหก `0xdeadbeef` นี่เป็นค่าการกำหนดค่าเริ่มต้นที่กำหนดให้กับรีจิสเตอร์ general-purpose ทั้งหมด เมื่อทำการการกำหนดค่าเริ่มต้นกระบวนการ

รีจิสเตอร์ Floating-point

รีจิสเตอร์ Floating-point แสดงโดย **$fr***Number* โดยที่ *Number* แสดงตัวเลข รีจิสเตอร์ รีจิสเตอร์ Floating-point ไม่ถูกแสดงตามค่าเริ่มต้น คุณสามารถยกเลิกการตั้งค่า ตัวแปรโปรแกรมดีบัก **$noflregs** เพื่อเปิดใช้งานการแสดงผล รีจิสเตอร์ floating-point (**unset $noflregs**) คุณยังสามารถอ้างอิง รีจิสเตอร์ floating-point โดยพิมพ์เมื่อใช้รีจิสเตอร์เหล่านี้กับคำสั่งย่อย **print** และ **assign $frNumber** มีค่าเริ่มต้นเป็นชนิด double **$frNumberh** อ้างอิงรีจิสเตอร์ floating-point เป็นชนิด **_Decimal32** **$frNumberd** อ้างอิงรีจิสเตอร์ floating-point เป็นชนิด **_Decimal64** ข้อมูลต่อไปนี้เป็นตัวอย่าง ของชนิดต่างๆ ของรีจิสเตอร์ floating-point:

```
(dbx) print $fr0
```

```
1.10000002

(dbx) print $fr0h

1.100001

(dbx) print $fr0d

1.10000062

(dbx) assign $fr0 = 9.876

(dbx) assign $fr0h = 9.876df

(dbx) assign $fr0d = 9.876dd
```

รีจีสเตอร์ Vector

รีจิสเตอร์ Vector แสดงโดย $vr*Number* โดยที่ *Number* แสดงจำนวนของรีจิสเตอร์ รีจิสเตอร์ Vector ไม่ถูกแสดงโดยค่าเริ่มต้น และแสดงเฉพาะบนโพรเซสเซอร์ ที่สนับสนุน Vector Processing Unit

คุณสามารถยกเลิกการตั้งค่าตัวแปรโปรแกรมดีบัก **$novregs** เพื่อเปิดใช้งานการแสดงผล รีจิสเตอร์ vector ด้วย (`unset $novregs`) คุณ ยังสามารถอ้างอิงรีจิสเตอร์ vector โดยพิมพ์เมื่อมีการใช้งานด้วยคำสั่งย่อย **print** และ **assign** $vr*Number* ค่าเริ่มต้นชนิดของ vector คือ int $vr*Number*f อ้างอิงชนิดของ vector เป็น float $vr*Number*s อ้างอิงชนิดของ vector เป็น short $vr*Number*c อ้างอิงชนิดของ vector เป็น char

ข้อมูลต่อไปนี้เป็นตัวอย่างของชนิดต่างๆ ของรีจิสเตอร์ vector:

```
(dbx) print $vr20

((1066192077, 1074161254, 1078355558, 1082340147))

(dbx) print $vr20f

((1.10000002, 2.0999999, 3.0999999, 4.0999999))

(dbx) print $vr20s

((16268, 52429, 16390, 26214, 16454, 26214, 16515, 13107))

(dbx) assign $vr20f[3] = 9.876

(dbx) print $vr20f ((1.10000002, 2.0999999, 3.0999999, 9.8760004))
```

รีจิสเตอร์ System-control

รีจิสเตอร์ system-control ที่สนับสนุนถูกแสดงโดย:

- รีจิสเตอร์แอดเดรสคำสั่งเครื่อง **$iar** หรือ **$pc**
- รีจิสเตอร์ Condition Status, **$cr**
- รีจิสเตอร์ Multiplier Quotient, **$mq**
- รีจิสเตอร์ Machine State, **$msr**
- รีจิสเตอร์ Link, **$link**
- รีจิสเตอร์ Count, **$ctr**
- รีจิสเตอร์ Fixed Point Exception, **$xer**
- รีจิสเตอร์ Transaction ID, **$tid**
- รีจิสเตอร์ Floating-Point Status, **$fpscr**

การตรวจสอบแอดเดรสหน่วยความจำ

ใช้รูปแบบคำสั่งดังต่อไปนี้เพื่อพิมพ์เนื้อหาของ หน่วยความจำซึ่งเริ่มต้นที่แอดเดรสแรกและต่อเนื่องไปถึงแอดเดรสที่สอง หรือจนถึงจำนวนรายการที่ระบุโดยตัวแปร *Count* ถูกแสดง *Mode* ระบุวิธีที่หน่วยความจำจะ ถูกพิมพ์

Address, *Address* **/** [*Mode*][> *File*]

Address **/** [*Count*][*Mode*] [> *File*]

ถ้าตัวแปร *Mode* ถูกละเว้น โหมดก่อนหน้าที่ระบุจะถูกนำมาใช้ใหม่ โหมดแรกเริ่มคือ **X** ต่อไปนี้คือโหมดที่สนับสนุน:

`b` พิมพ์ไบต์ในแบบฐานแปด

`c` พิมพ์ไบต์เป็นอักขระ

`D` พิมพ์ long word เป็นทศนิยม

`d` พิมพ์ short word เป็นทศนิยม

`Df` พิมพ์ตัวเลข decimal float ความเที่ยงตรงสองเท่า

`DDf`
พิมพ์ตัวเลข decimal float ความเที่ยงตรงสี่เท่า

`f` พิมพ์ตัวเลข floating-point ความเที่ยงตรงหนึ่งเท่า

`g` พิมพ์ตัวเลข floating-point ความเที่ยงตรงสองเท่า

`Hf` พิมพ์ตัวเลข decimal float ความเที่ยงตรงหนึ่งเท่า

`h` พิมพ์ไบต์ในแบบเลขฐานสิบหก

`i` พิมพ์คำสั่งเครื่อง

`lld`
พิมพ์ตัวเลขทศนิยมที่มีเครื่องหมาย 8 ไบต์

`llo`
พิมพ์ตัวเลขฐานแปดที่ไม่มีเครื่องหมาย 8 ไบต์

`llu`
พิมพ์ตัวเลขทศนิยมที่ไม่มีเครื่องหมาย 8 ไบต์

`llx`
พิมพ์ตัวเลขฐานสิบหกที่ไม่มีเครื่องหมาย 8 ไบต์

`O` พิมพ์ long word เป็นฐานแปด

`o` พิมพ์ short word เป็นฐานแปด

`q` พิมพ์ตัวเลข extended-precision floating-point

`s` พิมพ์สตริงของอักขระที่ปิดท้ายด้วย null ไบต์

`X` พิมพ์ long word เป็นเลขฐานสิบหก

`x` พิมพ์ short word เป็นเลขฐานสิบหก

ในตัวอย่างดังต่อไปนี้ นิพจน์ในวงเล็บ สามารถถูกใช้เป็นแอดเดรส:

```
(dbx) print &x
0x3fffe460
(dbx) &x/X
3fffe460: 31323300
(dbx) &x,&x+12/x
3fffe460: 3132 3300 7879 7a5a 5958 5756 003d 0032
(dbx) ($pc)/2i
100002cc (sub)  7c0802a6        mflr     r0
100002d0 (sub + 0x4)    bfc1fff8        stm       r30,-8(r1)
```

การรันโปรแกรมที่ระดับเครื่อง

คำสั่งสำหรับการดีบักโปรแกรมของคุณที่ระดับเครื่อง เหมือนกับที่ระดับสัญลักษณ์ คำสั่งย่อย **stopi** หยุดการทำงานของเครื่องเมื่อถึง แอดเดรส เงื่อนไขเป็นจริง หรือตัวแปรถูกเปลี่ยนแปลง คำสั่งย่อย **tracei** เหมือนกับคำสั่งติดตาม สัญลักษณ์ คำสั่งย่อย **stepi** เรียกใช้หนึ่งหรือ *Number* ที่ระบุของ คำสั่งเครื่อง

ถ้าคุณดำเนินคำสั่งย่อย **stepi** อื่นที่จุดนี้ คุณจะหยุดที่แอดเดรส 0x10000618 ที่ระบุเป็น entry point ของโพรซีเดอร์ `printf` ถ้าคุณไม่ตั้งใจจะ หยุดการทำงานที่แอดเดรสนี้ คุณสามารถใช้คำสั่งย่อย **return** เพื่อดำเนินการเรียกใช้ที่คำสั่งต่อไปได้ใน `sub` ที่แอดเดรส 0x100002e0 ที่จุดนี้คำสั่งย่อย **nexti** จะดำเนินการเรียกใช้ต่อไปที่ 0x10000428 โดยอัตโนมัติ

ถ้าโปรแกรมของคุณมีหลายเธรด ชื่อเธรดสัญลักษณ์ของเธรด ที่รันอยู่ถูกแสดงเมื่อโปรแกรมหยุดทำงาน ตัวอย่างเช่น:

```
หยุดใน sub ที่ 0x100002d4 ($t4)
10000424 (sub+0x4) 480001f5 bl 0x10000618 (printf)
```

การดีบัก fdpr reordered executables

คุณสามารถดีบักโปรแกรมที่ได้ถูกจัดลำดับใหม่ด้วย **fdpr** (feedback directed program restructuring เป็นส่วนหนึ่งของ Performance Toolbox for AIX) ที่ระดับคำสั่ง ถ้าตัวเลือก optimization **-R0** หรือ **-R2** ถูกใช้ ข้อมูลเพิ่มเติมถูกจัดเตรียม เปิดใช้ **dbx** เพื่อแม็พแอดเดรสคำสั่งที่จัดลำดับใหม่ล่าสุดกับแอดเดรส ที่ตรงกันในคำสั่งโปรแกรมต้นฉบับดังนี้:

```
0xRRRRRRRR = fdpr[0xYYYYYYYY]
```

ในตัวอย่างนี้ 0xRRRRRRRR คือแอดเดรสที่ถูกจัดลำดับใหม่และ 0xYYYYYYYY คือ แอดเดรสดั้งเดิม นอกจากนี้ **dbx** ใช้รายการ traceback ในพื้นที่คำสั่งดั้งเดิมเพื่อค้นหาชื่อโพรซีเดอร์ที่เกี่ยวข้อง สำหรับข้อความ `stopped in` คำสั่งย่อย **func** และ traceback

```
(dbx) stepi
stopped in proc_d at 0x1000061c = fdpr[0x10000278]
0x1000061c (???) 9421ffc0       stwu   r1,-64(r1)
(dbx)
```

ในตัวอย่างก่อนหน้านี้ **dbx** บ่งชี้ว่า โปรแกรมถูกหยุดทำงานในรูทีนย่อย `proc_d` ที่แอดเดรส `0x1000061c` ในส่วนข้อความที่จัดลำดับใหม่ ซึ่งเดิมอยู่ที่แอดเดรส `0x10000278` สำหรับข้อมูล เพิ่มเติมเกี่ยวกับ **fdpr** โปรดดูที่คำสั่ง **fdpr**

การแสดงคำสั่งแอสเซมบลี

คำสั่งย่อย **listi** สำหรับคำสั่ง **dbx** แสดงชุดคำสั่งที่ระบุจาก ไฟล์ต้นฉบับ ในโหมดเริ่มต้น โปรแกรม **dbx** แสดง คำสั่งสำหรับสถาปัตยกรรมซึ่งโปรแกรมรันอยู่ คุณสามารถ ลบล้างโหมดเริ่มต้นด้วยตัวแปร **$instructionset** และ **$mnemonics** ของคำสั่งย่อย **set** สำหรับคำสั่ง **dbx**

สำหรับข้อมูลเพิ่มเติมเกี่ยวกับการแสดงคำสั่งหรือคำสั่ง disassembling โปรดดูที่คำสั่งย่อย **listi** สำหรับคำสั่ง **dbx** สำหรับข้อมูลเพิ่มเติมเกี่ยวกับ การลบล้างค่าโหมดเริ่มต้น โปรดดูที่ตัวแปร **$instructionset** และ **$mnemonics** ของคำสั่งย่อย **set** สำหรับคำสั่ง **dbx**

การกำหนดสภาวะแวดล้อมการดีบัก dbx เอง

คุณสามารถกำหนดลักษณะการทำงานการดีบักเองโดยการสร้าง aliases คำสั่งย่อยโดยระบุตัวเลือกในไฟล์ **.dbxinit** คุณสามารถอ่านคำสั่งย่อย **dbx** จากไฟล์โดยใช้ แฟล็ก **–c**

ส่วนต่อไปนี้มีข้อมูลเพิ่มเติม เกี่ยวกับตัวเลือกการกำหนดเอง:

การกำหนดพร้อมต์ dbx ใหม่

พร้อมต์ **dbx** โดยปกติคือชื่อ ที่ใช้เพื่อเริ่มโปรแกรม **dbx** ถ้าคุณระบุ `/usr/ucb/dbx a.out` บนบรรทัดคำสั่ง พร้อมต์ จะเป็น `/usr/ucb/dbx`

คุณสามารถเปลี่ยนพร้อมต์ด้วยคำสั่งย่อย **prompt** หรือโดยระบุพร้อมต์อื่นใน บรรทัด **prompt** ของไฟล์ **.dbxinit** การเปลี่ยนพร้อมต์ในไฟล์ **.dbxinit** ทำให้ใช้พร้อมต์ของคุณแทนค่าเริ่มต้นแต่ละครั้งที่คุณกำหนด ค่าเริ่มต้นโปรแกรม **dbx**

ตัวอย่างเช่น เมื่อต้องการกำหนดค่าเริ่มต้นโปรแกรม **dbx** กับพร้อมต์ดีบัก `debug-->` ให้ป้อนบรรทัดต่อไปนี้ในไฟล์ **.dbxinit** ของคุณ:

```
prompt "debug-->"
```

การสร้างนามแฝงคำสั่งย่อย dbx

คุณสามารถสร้างคำสั่งของคุณเองจากชุด คำสั่งย่อยพื้นฐาน **dbx** คำสั่งดังต่อไปนี้อนุญาตให้คุณสร้าง alias ผู้ใช้จากอาร์กิวเมนต์ที่ระบุ คำสั่งทั้งหมดในสตริงแทนที่ สำหรับ alias ต้องเป็นคำสั่งย่อยพื้นฐาน **dbx** จากนั้นคุณสามารถใช้ aliases ของคุณในตำแหน่งที่เป็นพื้นฐาน **dbx**

คำสั่งย่อย **alias** ที่ไม่มีอาร์กิวเมนต์แสดง aliases ปัจจุบันที่มีผลอยู่ หนึ่งอาร์กิวเมนต์คำสั่งจะแสดงสตริงการแทนที่ที่สัมพันธ์กับ alias นั้น

alias [*AliasName*[*CommandName*]]

alias *AliasName "CommandString"*

alias *AliasName* **(***Parameter1, Parameter2, . . .* **)** *"CommandString"*

สองรูปแบบแรกของคำสั่งย่อย **alias** ถูกใช้เพื่อแทนที่สตริงการแทนที่สำหรับ **alias** แต่ละครั้งที่ถูกใช้ รูปแบบที่สามของ aliasing คือความสามารถ ทางแมโครที่จำกัด แต่ละพารามิเตอร์ที่ระบุในคำสั่งย่อย **alias** ถูกแทนที่ในสตริงการแทนที่

aliases และชื่อคำสั่งย่อยที่สัมพันธ์กันดังต่อไปนี้ เป็นค่าเริ่มต้น:

`attr`
attribute

`bfth`
stop (ในเธรดที่กำหนดที่ฟังก์ชันที่ระบุ)

`blth`
stop (ในเธรดที่กำหนดที่บรรทัดซอร์สที่ระบุ)

`c` cont

`cv` condition

`d` delete

`e` edit

`h` help

`j` สถานะ

`l` list

`m` map

`mu` mutex

`n` next

`p` **print**

`q` quit

`r` run

`s` step

`st` stop

`t` **where**

`th` thread

คุณสามารถเอา alias ออกด้วยคำสั่ง **unalias**

การใช้ไฟล์ .dbxinit

แต่ละครั้งที่คุณเริ่มเซสชันการดีบักโปรแกรม **dbx** ค้นหาไฟล์การกำหนดค่าพิเศษชื่อ **.dbxinit** ซึ่งมีรายการของคำสั่งย่อย **dbx** ที่จะดำเนินการ คำสั่งย่อยเหล่านี้ถูกดำเนินการก่อนโปรแกรม **dbx** เริ่มอ่านคำสั่งย่อยจากอินพุตมาตรฐาน เมื่อคำสั่ง **dbx** ถูกเริ่มต้น คำสั่งจะตรวจสอบไฟล์ **.dbxinit** ในไดเร็กทอรีปัจจุบันของผู้ใช้และในไดเร็กทอรี **$HOME** ของผู้ใช้ ถ้าไฟล์ **.dbxinit** มีอยู่แล้ว คำสั่งย่อยจะทำงานที่จุดเริ่มต้นของเซสชันดีบัก ถ้าไฟล์ **.dbxinit** มีอยู่แล้วทั้งในไดเร็กทอรี home และไดเร็กทอรีปัจจุบัน ดังนั้นไดเร็กทอรีทั้งสองจะอ่านตามลำดับ เนื่องจากไฟล์ **.dbxinit** ในไดเร็กทอรีปัจจุบันถูกอ่านครั้งล่าสุด คำสั่งย่อยของไฟล์นั้นสามารถแทนที่คำสั่งย่อยเหล่านั้นในไดเร็กทอรี home

โดยปกติไฟล์ **.dbxinit** มีคำสั่งย่อย **alias** แต่ สามารถมีคำสั่งย่อย **dbx** ที่ถูกต้องได้ ตัวอย่างเช่น:

```
$ cat .dbxinit
alias si "stop in"
prompt "dbg-->"
$ dbx a.out
dbx version 3.1
Type 'help' for help.
reading symbolic information .. .
dbg--> alias
si   stop in
t    where . . .
dbg-->
```

การอ่านคำสั่งย่อย dbx จากไฟล์

ตัวเลือกการร้องขอ **-c** และไฟล์ **.dbxinit** จัดเตรียมกลไกสำหรับการดำเนินการคำสั่งย่อย **dbx** ก่อนการอ่านจากอินพุตมาตรฐาน เมื่อตัวเลือก **-c** ถูกระบุโปรแกรม **dbx** ไม่ค้นหา ไฟล์ **.dbxinit** ใช้คำสั่งย่อย **source** เพื่ออ่านคำสั่งย่อย **dbx** จาก ไฟล์เมื่อเซสชันการดีบักเริ่มขึ้น

หลังจากดำเนินการรายการของคำสั่งในไฟล์ **cmdfile** โปรแกรม **dbx** แสดงพร้อมต์ และรออินพุต

คุณยังสามารถใช้ตัวเลือก **-c** เพื่อ ระบุรายการของคำสั่งย่อยที่จะถูกดำเนินการเมื่อเริ่มการทำงานโปรแกรม **dbx**

การดีบัก spinlocks

คุณสามารถใช้โปรแกรม **dbx** เพื่อดีบัก spinlocks ในการทำดังนั้น ตั้งค่า ตัวแปรสภาวะแวดล้อม AIXTHREAD_SPINLOCKS เป็น ON

การพัฒนาสำหรับปลั๊กอินเฟรมเวิร์ก dbx

dbx จัดเตรียม ปลั๊กอินเฟรมเวิร์กสำหรับผู้พัฒนาซึ่งต้องการเพิ่มคำสั่งย่อย **dbx** และ event handlers ใหม่

ผู้ใช้ dbx สามารถสร้างปลั๊กอินที่เพิ่มความสามารถ **dbx** ด้วยคำสั่งจำเพาะแอ็พพลิเคชันหรือไลบรารี เพื่อช่วยในการดีบัก

หมายเหตุ:

1. เนื่องจากคำสั่ง **dbx** ค่าเริ่มต้นเป็นกระบวนการ 64 บิต ปลั๊กอินทั้งหมดจำเป็นต้องถูกคอมไพล์เป็น 64 บิตเพื่อใช้กับ คำสั่ง **dbx** เพื่อโหลดปลั๊กอิน 32 บิต ให้ใช้คำสั่ง **dbx** เวอร์ชัน 32 บิต ซึ่งคือคำสั่ง **dbx32**
2. ระวังในการสับสนกับ **dbx callback routines** และ **plug-in interface routines**
3. **dbx callback routines** คือชุดของเซอร์วิสที่เสนอ โดย **dbx** แก่ปลั๊กอิน ปลั๊กอินให้การเข้าถึงแก่รูทีนเหล่านี้ ผ่านชุดของตัวชี้ฟังก์ชัน
4. **plug-in interface routines** คือชุดของเมธอด **dbx** ที่จำเป็น ในการนำไปใช้โดยปลั๊กอิน

รูปแบบไฟล์

แต่ละปลั๊กอินต้องเป็นอ็อบเจ็กต์ไฟล์ที่แบ่งใช้

การตั้งชื่อ

ในการเปลี่ยนทิศทางอินพุตคำสั่งย่อยอย่างถูกต้อง **dbx** แต่ละปลั๊กอินจำเป็นต้องมี *ชื่อที่ไม่ซ้ำกัน*

ชื่อไฟล์ของปลั๊กอินสื่อสารชื่อที่ไม่ซ้ำกันนี้กับ **dbx** ขณะการกำหนดค่าเริ่มต้น **dbx** ค้นหา ชุดของไดเร็กทอรีที่กำหนดไว้ล่วงหน้าและจำเพาะผู้ใช้สำหรับไฟล์ซึ่ง ชื่อฐานตรงกับนิพจน์ทั่วไป:

```
^libdbx_.+\.so$
```

ตารางดังต่อไปนี้แสดงตัวอย่างของชื่อไฟล์ที่ถูกต้อง และไม่ถูกต้องสำหรับ **ปลั๊กอิน dbx** *ชื่อที่ไม่ซ้ำกัน* ที่เหมาะสมถูกแสดงสำหรับตัวอย่างที่ถูกต้องทั้งหมด:

ชื่อไฟล์	ค่าที่ถูกต้อง	ชื่อที่ไม่ซ้ำกัน
libdbx_sample.so	ใช่	ตัวอย่าง
libdbx_xyz.so	ใช่	xyz
libdbx_my_app.so	ใช่	my_app
libdbx.so	ไม่ใช่	
libdbx_.so	ไม่	
libdbx_sample.so.plugin	ไม่	
plugin_libdbx_sample.so	ไม่	

ตำแหน่ง

dbx อนุญาตให้ผู้ใช้ระบุรายการของ ไดเร็กทอรีเพื่อการค้นหาโดยใช้ตัวแปรสภาวะแวดล้อม DBX_PLUGIN_PATH แต่ละไดเร็กทอรีในรายการควรถูกแยกโดยโคลอน ใน ตัวอย่างดังต่อไปนี้ โคลอนแยกสองไดเร็กทอรี

```
$ export dbx_PLUGIN_PATH=$HOME/dbx_plugins:/mnt/share/dbx_plugins
```

ขณะการกำหนดค่าเริ่มต้น **dbx** ค้นหาปลั๊กอิน **dbx** ยังค้นหาไดเร็กทอรีของไฟล์เรียกทำงาน (ถ้ารู้จัก) ไดเร็กทอรีนี้ถูกค้นหาหลังจากไดเร็กทอรีที่ผู้ใช้กำหนด ถูกค้นหา

หมายเหตุ: เมื่อคุณใช้ **dbx** เพื่อ เชื่อมต่อกับกระบวนการ ไม่สามารถระบุพาธเต็มไปที่ ไฟล์เรียกทำงานได้

การโหลด

ปลั๊กอินถูกโหลดหนึ่งในวิธีดังต่อไปนี้:

- ปลั๊กอินสามารถถูกโหลดและกำหนดค่าเริ่มต้นโดยอัตโนมัติโดยการนำไปไว้ใน ไดเร็กทอรีที่ถูกค้นหาโดย **dbx** ซึ่งเกิดขึ้นขณะการกำหนดค่าเริ่มต้น **dbx**
- ปลั๊กอินสามารถถูกโหลดและกำหนดค่าเริ่มต้นด้วยตัวเองโดยการระบุ ตำแหน่งให้กับคำสั่งย่อย **pluginload dbx** ซึ่งสามารถเกิดขึ้นได้ทุกขณะระหว่าง เซสชัน **dbx**

หลังจากโหลดปลั๊กอิน อัตโนมัติหรือด้วยตัวเองสำเร็จ ข้อความ เหมือนดังต่อไปนี้ถูกแสดง:

```
(dbx) pluginload /home/user/dbx_plugins/libdbx_sample.so
 plug-in "/home/user/dbx_plugins/libdbx_sample.so"  loaded
```

ปลั๊กอินทั้งหมดที่มี *ชื่อที่ไม่ซ้ำกัน* เหมือนกับ ปลั๊กอินที่ใช้งานอยู่ในขณะนี้ถูกละเว้นและข้อความคำเตือนเหมือนดัง ต่อไปนี้จะถูกแสดง

```
(dbx) pluginload /mnt/share/dbx_plugins/libdbx_sample.so

could not load plug-in
"/mnt/share/dbx_plugins/libdbx_sample.so":
plug-in "/home/user/dbx_plugins/libdbx_sample.so" already loaded.
```

การยกเลิกการโหลด

ปลั๊กอินทั้งหมด ไม่ว่าจะถูกโหลดอย่างไร สามารถถูกยกเลิกการโหลดด้วยตัวเอง โดยการระบุชื่อกับคำสั่งย่อย **pluginunload dbx** หลังจากปลั๊กอินถูกยกเลิกการโหลดสำเร็จ ข้อความเหมือนดังข้อความต่อไปนี้จะถูกแสดง

```
(dbx) pluginunload sample
plug-in "/home/user/dbx_plugins/libdbx_sample.so" unloaded.
```

การควบคุมเวอร์ชัน

If changes are made to the plug-in framework that would otherwise break the compatibility of the existing plug-ins with earlier versions, a new version identifier is created. This process is true for any significant changes or additions done to the **Plug-in Interface** or **Plug-in dbx callback routine.**

เพื่อลดความจำเป็นในการเปลี่ยนแปลงเวอร์ชันปลั๊กอินบ่อยๆ บาง **รูทีน Plug-in dbx callback** ต้องการพารามิเตอร์เพิ่มเติมที่แสดง ขนาดของบัฟเฟอร์ การกระทำดังกล่าวถูกใช้สำหรับพารามิเตอร์บัฟเฟอร์ ที่มาจากโครงสร้างระบบ ซึ่งขนาดไม่ได้ถูกควบคุมโดย **dbx** ซึ่งอนุญาตให้ขนาดของโครงสร้างระบบ เปลี่ยนแปลงได้โดยไม่จำเป็นต้องมีการอัพเดตเป็นปลั๊กอินเวอร์ชันล่าสุด

ขณะนี้ ตัวระบุเวอร์ชันคือ DBX_PLUGIN_VERSION_1 เท่านั้น

ไฟล์ส่วนหัว

ปลั๊กอิน dbx ผู้พัฒนาสามารถค้นหาต้นแบบฟังก์ชัน ข้อกำหนดโครงสร้างข้อมูล และข้อกำหนดแมโครใน ไฟล์ header ดังต่อไปนี้:

```
/usr/include/sys/dbx_plugin.h
```

ปลั๊กอินอินเตอร์เฟส

อ้างอิงถึงไฟล์ header `dbx_plugin.h` สำหรับต้นแบบ และข้อกำหนดสำหรับรูทีน **Plug-in Interface**

แต่ละปลั๊กอิน *ต้อง* นำไปใช้และเอ็กซ์พอร์ต รูทีนดังต่อไปนี้ทั้งหมด:

- `int dbx_plugin_version(void)`
- `int dbx_plugin_session_init(dbx_plugin_session_t session, constdbx_plugin_service_t *servicep)`
- `void dbx_plugin_session_command(dbx_plugin_session_t session, int argc, char *const argv[])`
- `void dbx_plugin_session_event(dbx_plugin_session_t session, int event, dbx_plugin_event_info_t *event_infop)`

int dbx_plugin_version(void)

รูทีนนี้ควรส่งกลับตัวระบุเวอร์ชัน **dbx Plug-in** ที่ตรงกับเวอร์ชันปลั๊กอินที่เข้ากันได้ ขณะนี้ ตัวระบุเวอร์ชันคือ `DBX_PLUGIN_VERSION_1` เท่านั้น

int dbx_plugin_session_init(dbx_plugin_session_t session, constdbx_plugin_service_t *servicep)

รูทีนนี้ควรดำเนินการกำหนดค่าเริ่มต้นที่จำเป็นสำหรับปลั๊กอิน เพื่อทำงานอย่างถูกต้อง ก่อนการส่งกลับการควบคุมกลับไปที่ **dbx** ซึ่งรวมถึงการตั้งค่า aliases สำหรับปลั๊กอินคำสั่งย่อยถ้าต้องการ

รูทีนนี้ควรสร้างเซสชันปลั๊กอินที่สัมพันธ์กับ ตัวระบุเซสชันที่กำหนด กับแอ็พพลิเคชันโปรแกรมหรือไฟล์คอร์ เมื่อต้องการระบุกระบวนการหรือไฟล์คอร์ ตัวระบุเซสชันถูกใช้โดย **dbx** ในการเรียก **Plug-in Interface** และ โดยปลั๊กอินสำหรับการร้องขอ **plugin dbx callback routine** รูทีนนี้ยังยอมรับโครงสร้างรูทีน callback

รูทีนนี้ควรส่งกลับค่าศูนย์สำหรับการกำหนดค่าเริ่มต้นสำเร็จ ถ้าการกำหนดค่าเริ่มต้นไม่สำเร็จ **dbx** ยกเลิกการโหลด และละทิ้งปลั๊กอิน

void dbx_plugin_session_command(dbx_plugin_session_t session, int argc, char *const argv[])

รูทีนนี้ควรยอมรับอินพุตจากผู้ใช้ **dbx** ในรูปของอาร์กิวเมนต์ที่จัดเตรียมให้กับคำสั่งย่อย **plugin** ไวยากรณ์ของคำสั่งย่อย **plugin** เป็นดังต่อไปนี้:

`plugin` *`Name`* `[`*`arg0 arg1 arg2 ... argn`*`]`

ซึ่งอนุญาตให้ผู้ใช้ **dbx** จัดเตรียมอินพุต ให้กับปลั๊กอินเดี่ยว ปลั๊กอินมีการควบคุมเต็มที่กับข้อมูลที่รับ เป็นอินพุต

คำสั่งย่อย **plugin** ส่งคำสั่งที่ระบุโดย พารามิเตอร์ *arg** parameters ไปที่ปลั๊กอินที่ระบุโดยพารามิเตอร์ *Name* (ตัวอย่างเช่น ชื่อปลั๊กอินอาจเป็น `libdbx_`*`Name`*`.so`) ใช้รูทีนนี้ **dbx** passes *arg0* through *argn* กับ ปลั๊กอิน *argv[0]* corresponds to *arg0*, *argv[1]* to *arg1*, และต่อไป

ในกรณีส่วนใหญ่ *arg0* จะแสดงชื่อของ คำสั่งย่อย ที่กำหนดโดยปลั๊กอินและ *arg1* ถึง *argn* จะ แสดงแฟล็กหรืออาร์กิวเมนต์เพิ่มเติม อย่างไรก็ตาม นี่ไม่ใช่ข้อกำหนด

ผู้พัฒนาจะถูกสนับสนุนให้ใช้คำสั่งย่อย **help** ซึ่งแสดงข้อมูลการใช้งานสำหรับปลั๊กอิน

void dbx_plugin_session_event(dbx_plugin_session_t session, int event, dbx_plugin_event_info_t *event_infop)

ในการตอบสนองกับเหตุการณ์โปรแกรมแอ็พพลิเคชัน รูทีนนี้ควร ดำเนินการประมวลผลภายในทั้งหมดที่จำเป็นต่อปลั๊กอิน รูทีน ถูกร้องขอหนึ่งครั้งโดย **dbx** กับการเกิดขึ้นของ แต่ละเหตุการณ์ ตารางดังต่อไปนี้อธิบายชนิดเหตุการณ์ซึ่ง ปลั๊กอินถูกแจ้งเตือน:

ID (เหตุการณ์)	ข้อมูลที่เกี่ยวข้อง (event_infop)	สาเหตุ
DBX_PLUGIN_EVENT_RESTART	ไม่มี	ผู้ใช้ **dbx** เรียกใช้ คำสั่งย่อย **run**
DBX_PLUGIN_EVENT _EXIT	รหัสจบการทำงาน	โปรแกรมแอ็พพลิเคชันสิ้นสุดผ่าน รูทีน จบการทำงาน
DBX_PLUGIN_EVENT _TERM	การสิ้นสุดหมายเลขสัญญาณ	โปรแกรมแอ็พพลิเคชันสิ้นสุด เนื่องจาก สัญญาณไม่ได้รับการจัดการ
DBX_PLUGIN_EVENT _LOAD	dbx_plugin_modinfo_t structure ของโมดูลที่โหลด	โมดูลถูกโหลดลงใน โปรแกรมแอ็พพลิเคชัน
DBX_PLUGIN_EVENT _UNLOAD	dbx_plugin_modinfo_t structure ของโมดูลที่ยกเลิกการโหลด	โมดูลถูกยกเลิกการโหลดจาก โปรแกรมแอ็พพลิเคชัน
DBX_PLUGIN_EVENT_BP	ไม่มี	โปรแกรมแอ็พพลิเคชันหยุดทำงานเนื่องจาก ผู้ใช้หรือ จุดหยุด หรือข้อมูล watchpoint **dbx** ภายใน
DBX_PLUGIN_EVENT_SIGNAL	หมายเลขสัญญาณ	โปรแกรมแอ็พพลิเคชันที่หยุด เนื่องจาก การส่งมอบสัญญาณ
DBX_PLUGIN_EVENT_SWTHRD	Handle ของ pthread ปัจจุบัน	ผู้ใช้ **dbx** เรียกใช้คำสั่งย่อย **thread current <handle>** ที่มีผลในการเปลี่ยนแปลง ใน pthread ปัจจุบัน

เหตุการณ์ DBX_PLUGIN_EVENT_BP และ DBX_PLUGIN_EVENT_SIGNAL แสดง ว่าโปรแกรมแอ็พพลิเคชันถูกเริ่มทำงานแล้วแต่ได้หยุดการทำงานไป เหตุการณ์เหล่านี้ เพื่อแสดงว่าข้อมูลที่แคชซึ่งปลั๊กอินครอบครองอยู่ อาจใช้ไม่ได้อีกต่อไป ในการแจ้งเตือนของเหตุการณ์เหล่านี้ จะมีประสิทธิภาพมากกว่า สำหรับปลั๊กอิน เพียง invalidate ข้อมูลที่แคชแทนการ รีเฟรชข้อมูล การรีเฟรชที่สมบูรณ์ของข้อมูลที่แคชควรเกิดขึ้น เฉพาะเมื่อข้อมูลมีความจำเป็น เรื่องนี้สำคัญเป็นพิเศษ เนื่องจาก บางสัญญาณอาจถูกละเว้นโดย **dbx** และบางจุดหยุด อาจเป็นจุดหยุดภายใน ถ้าผู้ใช้ไม่มีโอกาสในการรัน คำสั่งย่อยs ก่อนโปรแกรมแอ็พพลิเคชันเริ่มอีกครั้ง การ รีเฟรชข้อมูลซ้ำๆ เป็นการสิ้นเปลืองทรัพยากร

```
void dbx_plugin_session_destroy(dbx_plugin_session_t session)
```

รูทีนนี้ควรทำงาน การล้างค่าขั้นสุดท้ายและการจัดการหน่วยความจำ ที่จำเป็นโดยปลั๊กอิน

รูทีน dbx callback

dbx callback routines ดังต่อไปนี้จัดเตรียม สำหรับแต่ละปลั๊กอินผ่านรูทีน `dbx_plugin_session_init`

รูทีน dbx **session** callback อนุญาตให้คุณรับคุณลักษณะ ของเซสชัน dbx **dbx** ใส่ข้อมูลในพารามิเตอร์ *flagsp*

```
typedef int (*dbx_plugin_session_service_t)(dbx_plugin_session_t session,
                                        dbx_plugin_session_flags_t *flagsp).
```

พารามิเตอร์รูทีน dbx **session** callback มีดังนี้:

พารามิเตอร์
คำอธิบาย

session
ตัวระบุเซสชัน

flagsp
คุณลักษณะเซสชันในการรวมกันของ:

- DBX_PLUGIN_SESSION_64BIT
  ถ้าตั้งค่า เซสชันแสดงโปรแกรม แอ็พพลิเคชัน 64 บิต มิฉะนั้น เซสชันแสดงโปรแกรม แอ็พพลิเคชัน 32 บิต
- DBX_PLUGIN_SESSION_CORE
  ถ้าตั้งค่า เซสชันแสดง ไฟล์คอร์ มิฉะนั้น เซสชันแสดงกระบวนการที่ทำงานอยู่

รหัสที่ส่งกลับของรูทีน dbx **session** callback มีดังนี้:

- DBX_PLUGIN_SUCCESS
- DBX_PLUGIN_BAD_SESSION เซสชันไม่ถูกต้อง
- DBX_PLUGIN_BAD_POINTER *flagsp* เป็น NULL

การประมวลผล

รูทีน dbx **process** callback อนุญาตให้คุณรับข้อมูลเกี่ยวกับ กระบวนการที่กำลังถูกดีบัก **dbx** กำหนดข้อมูล พารามิเตอร์ *infop*

```
typedef int (*dbx_plugin_process_service_t)(dbx_plugin_session_t session,
                                            dbx _plugin_procinfo_t *infop,
                                            size_t procinfo_size)
```

พารามิเตอร์รูทีน dbx **process** callback มีดังนี้:

พารามิเตอร์
คำอธิบาย

session
ตัวระบุเซสชัน

infop
Allocated dbx_plugin_procinfo_t structure

procinfo_size
Size of dbx_plugin_procinfo_t structure

รหัสที่ส่งกลับของรูทีน dbx **process** callback มีดังนี้:

- DBX_PLUGIN_SUCCESS
- DBX_PLUGIN_BAD_SESSION *เซสชัน* ไม่ถูกต้อง
- DBX_PLUGIN_BAD_POINTER *infop* เป็น NULL
- DBX_PLUGIN_BAD_ARG *procinfo_size* ไม่ถูกต้อง

- DBX_PLUGIN_UNAVAILABLE กระบวนการไม่ทำงานหรือข้อมูลไม่อยู่ในคอร์

fds

รูทีน dbx **fds** callback ทำให้คุณสามารถรับข้อมูล ในไฟล์ descriptors สำหรับกระบวนการ คุณสามารถ:

- เรียกซ้ำเพื่อรับข้อมูลแยกในแต่ละไฟล์ descriptor หรือ
- เรียกเพียงครั้งเดียวเพื่อรับจำนวนไฟล์ descriptors รวมและเรียก อีกครั้งเพื่อรับข้อมูลในไฟล์ descriptors ทั้งหมดพร้อมกัน

ถ้าปลั๊กอินส่งบัฟเฟอร์ non-NULL *infop*, **dbx** กำหนดค่าบัฟเฟอร์ด้วยจำนวนรายการ ที่ร้องขอใน **countp* เริ่มด้วยไฟล์ descriptor ที่อ้างอิงโดย **indexp*

ถ้าปลั๊กอินส่ง **countp* ที่มากกว่า จำนวนของรายการที่เหลือ **dbx** เรียกข้อมูลรายการ ที่เหลือทั้งหมด **dbx** อัพเดต *countp* เพื่อสะท้อน จำนวนรายการจริงที่เรียกข้อมูลและ *indexp* เพื่อสะท้อน ดัชนีโมดูลถัดไป ถ้าไฟล์ descriptor สุดท้ายถูกเรียกคืน *indexp* ถูกตั้งค่า เป็น -1 ถ้าปลั๊กอินส่งผ่านบัฟเฟอร์ NULL infop, *indexp* และ *countp* ยังคง อัพเดต — ถ้า *infop* เป็นค่า non-NULL

```
typedef int (*dbx_plugin_fds_service_t)(dbx_plugin_session_t session,
                                        dbx_plugin_fdinfo_t *infop,
                                        size_t fdinfo_size,
                                        unsigned int *indexp,
                                        unsigned int *countp)
```

พารามิเตอร์รูทีน dbx **fds** callback มีดังนี้:

พารามิเตอร์
คำอธิบาย

session
ตัวระบุเซสชัน

infop
อาร์เรย์ของ dbx_plugin_fdinfo_t structures หรือ NULL ที่จัดสรร

fdinfo_size
ขนาดของ dbx_plugin_fdinfo_t structure เดี่ยว

indexp
ไฟล์ descriptor เริ่มต้น/ต่อไป (โดยที่ศูนย์ตรงกับ ไฟล์ descriptor แรก)

countp
จำนวนของไฟล์ descriptors

รหัสที่ส่งกลับของรูทีน dbx **fds** callback คือ:

- DBX_PLUGIN_SUCCESS
- DBX_PLUGIN_BAD_SESSION *เซสชัน* ไม่ถูกต้อง
- DBX_PLUGIN_BAD_POINTER *indexp* มีค่า NULL หรือ countp เป็น NULL
- DBX_PLUGIN_BAD_ARG *fdinfo_size* ไม่ถูกต้องหรือ * *countp* == 0
- DBX_PLUGIN_UNAVAILABLE กระบวนการไม่ทำงานหรือข้อมูลไม่อยู่ในคอร์

โมดูล

รูทีน dbx **modules** callback อนุญาตให้คุณรับข้อมูล ในโมดูลที่โหลดสำหรับกระบวนการ คุณสามารถ:

- เรียกซ้ำเพื่อรับข้อมูลแยกในแต่ละโมดูล หรือ
- เรียกเพียงครั้งเดียวเพื่อรับจำนวนโมดูล รวมและเรียก อีกครั้งเพื่อรับข้อมูลในโมดูลทั้งหมดพร้อมกัน

ถ้าปลั๊กอินส่งผ่านบัฟเฟอร์ non-NULL infop, **dbx** กำหนดค่า บัฟเฟอร์ด้วยจำนวนของรายการที่ร้องขอใน **countp* เริ่มต้นด้วยโมดูลที่อ้างอิงโดย **indexp*

ถ้าปลั๊กอินส่ง **countp* ที่มากกว่า จำนวนของรายการที่เหลือ **dbx** เรียกข้อมูลรายการ ที่เหลือทั้งหมด **dbx** อัพเดต *countp* เพื่อสะท้อน จำนวนรายการจริงที่เรียกข้อมูลและอัพเดต *indexp* เพื่อสะท้อน ดัชนีโมดูลถัดไป ถ้าโมดูลสุดท้ายถูกเรียกคืน *indexp* ถูก ตั้งค่าเป็น -1 ถ้าปลั๊กอินส่งผ่านบัฟเฟอร์ NULL infop, *indexp* และ *countp* ยังคง อัพเดต — ถ้า *infop* เป็นค่าที่ไม่ใช่ NULL

หมายเหตุ: รูทีนนี้จัดสรรหน่วยความจำเพื่อเก็บค่าชื่อไฟล์และ สตริงอักขระสมาชิก ผู้เรียกต้องฟรีหน่วยความจำนี้เมื่อ ไม่จำเป็นต้องใช้

```
typedef int (*dbx_plugin_modules_service_t)(dbx_plugin_session_t session,
                                            dbx_plugin_modinfo_t *infop,
                                            size_t modinfo_size,
                                            unsigned int *indexp,
                                            unsigned int *countp)
```

พารามิเตอร์รูทีน dbx **modules** callback มีดังนี้:

พารามิเตอร์
คำอธิบาย

session
ตัวระบุเซสชัน

infop
อาร์เรย์ของ dbx_plugin_modinfo_t structures หรือ NULL ที่จัดสรร

modinfo_size
ขนาดของ dbx_plugin_modinfo_t structure เดี่ยว

indexp
โมดูล เริ่มต้น/ต่อไป (โดยที่ศูนย์ตรงกับ โมดูลแรก)

countp
จำนวนของโมดูล

รหัสที่ส่งกลับของรูทีน dbx **modules** callback คือ:

- DBX_PLUGIN_SUCCESS
- DBX_PLUGIN_BAD_SESSION *เซสชัน* ไม่ถูกต้อง
- DBX_PLUGIN_BAD_POINTER *indexp* มีค่า NULL หรือ countp เป็น NULL
- DBX_PLUGIN_BAD_ARG *modinfo_size* ไม่ถูกต้องหรือ * *countp* == 0

ขอบเขต

รูทีน dbx **regions** callback ทำให้คุณสามารถรับข้อมูล ในขอบเขตหน่วยความจำสำหรับกระบวนการ

ขอบเขตที่เรียกคืนรวมถึง:

- Main thread stack region (DBX_PLUGIN_REGION_STACK)
- User data region (DBX_PLUGIN_REGION_DATA)
- Process private data region (DBX_PLUGIN_REGION_SDATA)
- Memory mapped region (DBX_PLUGIN_REGION_MMAP)
- Shared memory region (DBX_PLUGIN_REGION_SHM)

คุณสามารถ:

- เรียกซ้ำเพื่อรับข้อมูลแยกในหนึ่งขอบเขต หรือ
- เรียกเพียงครั้งเดียวเพื่อรับจำนวนขอบเขต รวมและเรียก อีกครั้งเพื่อรับข้อมูลในขอบเขตทั้งหมดพร้อมกัน

ถ้าปลั๊กอินส่งผ่านบัฟเฟอร์ non-NULL *infop* กำหนดค่า **dbx** บัฟเฟอร์ที่มีจำนวนของรายการ ที่ร้องขอใน **countp* เริ่มต้นด้วย ขอบเขตที่อ้างอิงโดย **indexp*

ถ้าปลั๊กอินส่ง **countp* ที่มากกว่า จำนวนของรายการที่เหลือ **dbx** เรียกข้อมูลรายการ ที่เหลือทั้งหมด **dbx** อัพเดต *countp* เพื่อ สะท้อน จำนวนรายการจริงที่เรียกข้อมูลและ *indexp* เพื่อสะท้อน ดัชนีขอบเขตถัดไป

ถ้าขอบเขตสุดท้ายถูกเรียกคืน *indexp* ถูกตั้งค่าเป็น -1 ถ้าปลั๊กอินส่งผ่านบัฟเฟอร์ NULL *infop*, *indexp* และ *countp* ยังคง อัพเดต — ถ้า *infop* เป็นค่า non-NULL

หมายเหตุ: ในขณะนี้ รูทีนนี้ถูกนำไปใช้เฉพาะกับเซสชัน ที่แสดงไฟล์คอร์ ข้อมูลที่เพียงพอไม่พร้อมใช้งานกับ **dbx** สำหรับ เซสชันที่แสดงกระบวนการที่ทำงานอยู่ การเรียก ใช้เซสชันดังกล่าวจะส่งกลับ DBX_PLUGIN_UNAVAILABLE

```
typedef int (*dbx_plugin_regions_service_t)(dbx_plugin_session_t session,
                                            dbx_plugin_reginfo_t *infop,
                                            size_t reginfo_size,
                                            unsigned int *indexp,
                                            unsigned int *countp)
```

พารามิเตอร์รูทีน dbx **regions** callback มีดังนี้:

พารามิเตอร์
คำอธิบาย

session
ตัวระบุเซสชัน

infop
อาร์เรย์ของ dbx_plugin_region_t structures หรือ NULL ที่จัดสรร

reginfo_size
ขนาดของ dbx_plugin_reginfo_t structure เดี่ยว

indexp
ขอบเขต เริ่มต้น/ต่อไป (โดยที่ศูนย์ตรงกับขอบเขตแรก)

countp
จำนวนขอบเขต

รูทีน dbx **regions** callback ส่งกลับรหัสที่ส่งกลับเป็น:

- DBX_PLUGIN_SUCCESS
- DBX_PLUGIN_BAD_SESSION *เซสชัน* ไม่ถูกต้อง
- DBX_PLUGIN_BAD_POINTER *indexp* มีค่า NULL หรือ *countp* เป็น NULL
- DBX_PLUGIN_BAD_ARG *reginfo_size* ไม่ถูกต้องหรือ * *countp* == 0
- เซสชัน DBX_PLUGIN_UNAVAILABLE แสดงกระบวนการที่ทำงานอยู่และ ขอบเขตที่ไม่สามารถเข้าถึงได้

เธรด

รูทีน dbx **threads** callback อนุญาตให้คุณรับข้อมูลเกี่ยวกับ เธรดเคอร์เนลในการประมวลผล

คุณสามารถ:
- เรียกซ้ำเพื่อรับข้อมูลแยกกันบนหนึ่งเธรด หรือ
- เรียกเพียงครั้งเดียวเพื่อรับจำนวนเธรด รวมและเรียก อีกครั้งเพื่อรับข้อมูลในเธรดทั้งหมดพร้อมกัน

ถ้าปลั๊กอินส่งผ่านบัฟเฟอร์ non-NULL *infop* กำหนดค่า **dbx** บัฟเฟอร์ที่มีจำนวนของรายการ ที่ร้องขอใน **countp* เริ่มต้นด้วยเธรดที่อ้างอิงโดย **indexp*

ถ้าปลั๊กอินส่งผ่าน **countp* ที่มากกว่าหรือเท่ากับ จำนวนของรายการที่เหลือ **dbx** เรียกคืน รายการที่เหลือทั้งหมดและอัพเดต *countp* เพื่อสะท้อนจำนวน จริงของรายการที่เรียกคืน

ถ้ารายการสุดท้ายถูกเรียกคืน และ *countp* น้อยกว่า ค่าที่ส่ง *indexp* ถูกตั้งค่าเป็น -1 มิฉะนั้น *indexp* ถูก อัพเดตเพื่อแสดงข้อมูล thread id สำหรับการร้องขอถัดไป

หมายเหตุ: ถ้า ค่าของ *countp* ที่ส่งเท่ากับจำนวนของรายการที่มี *countp* จะยังคงเหมือนเดิม แต่ *indexp* ไม่ถูก ตั้งค่าเป็น -1

ถ้าปลั๊กอินส่งผ่านบัฟเฟอร์ NULL *infop*, *indexp* และ *countp* ถูก อัพเดต — ถ้า *infop* เป็นค่า non-NULL

```
typedef int (*dbx_plugin_threads_service_t)(dbx_plugin_session_t session,
                                        dbx _plugin_thrdinfo_t *infop,
                                        size_t thrdinfo_size,
                                        tid64_t *indexp,
                                        unsigned int *countp)
```

พารามิเตอร์รูทีน dbx **threads** callback มีดังนี้:

พารามิเตอร์
คำอธิบาย

session
ตัวระบุเซสชัน

infop
อาร์เรย์ของ dbx_plugin_thrdinfo_t structures หรือ NULL ที่จัดสรร

thrdinfo_size
ขนาดของ dbx_plugin_thrdinfo_t structure เดี่ยว

indexp
thread id เริ่มต้น/ถัดไป (โดยที่ในอินพุต ศูนย์หมายถึง เธรดแรก)

countp
จำนวนของเธรด

รหัสที่ส่งกลับของรูทีน dbx **threads** callback คือ:

- DBX_PLUGIN_SUCCESS
- DBX_PLUGIN_BAD_SESSION *เซสชัน* ไม่ถูกต้อง
- DBX_PLUGIN_BAD_POINTER *indexp* มีค่า NULL หรือ countp เป็น NULL
- DBX_PLUGIN_BAD_ID **indexp* เป็น ID ที่ไม่ถูกต้อง
- DBX_PLUGIN_BAD_ARG *thrdinfo_size* ไม่ถูกต้องหรือ *countp ==0
- DBX_PLUGIN_UNAVAILABLE กระบวนการไม่ทำงานหรือรายการไม่อยู่ในคอร์

pthreads

รูทีน dbx **pthreads** callback อนุญาตให้คุณรับข้อมูล บน pthreads ในกระบวนการ รวมทั้งเธรดเคอร์เนลที่เกี่ยวข้อง

คุณสามารถ:
- เรียกซ้ำเพื่อรับข้อมูลแยกในหนึ่ง pthread หรือ
- เรียกเพียงครั้งเดียวเพื่อรับจำนวน pthread รวมและเรียก อีกครั้งเพื่อรับข้อมูลใน pthread ทั้งหมดพร้อมกัน

ถ้าปลั๊กอินส่งผ่านบัฟเฟอร์ non-NULL *infop*, กำหนดค่า **dbx** บัฟเฟอร์ที่มีจำนวนของรายการ ที่ร้องขอใน **countp* เริ่มต้นด้วย pthread ที่อ้างอิงโดย **indexp*

ถ้าปลั๊กอินส่ง **countp* ที่มากกว่า จำนวนของรายการที่เหลือ **dbx** เรียกข้อมูลรายการ ที่เหลือทั้งหมด **dbx** อัพเดต *countp* เพื่อสะท้อน จำนวนรายการจริงที่เรียกข้อมูลและ *indexp* เพื่อแสดงข้อมูล pthread handle สำหรับการร้องขอถัดไป

ถ้ารายการสุดท้ายถูกเรียกคืน *indexp* ถูกตั้งค่าเป็น -1 ถ้าปลั๊กอินส่งผ่านบัฟเฟอร์ NULL *infop*, *indexp* และ *countp* ยังคง อัพเดต — ถ้า *infop* เป็นค่า non-NULL

ถ้า pthread แรกถูกร้องขอและ *countp* ถูกอัพเดต เป็นศูนย์ กระบวนการไม่ถูก pthread

```
typedef int (*dbx_plugin_pthreads_service_t)(dbx_plugin_session_t session,
                                             dbx_plugin_pthinfo_t *infop,
                                             size_t pthrdinfo_size,
                                             pthdb_pthread_t *indexp,
                                             unsigned int *countp)
```

พารามิเตอร์รูทีน dbx **pthreads** callback มีดังนี้:

พารามิเตอร์
คำอธิบาย

session
ตัวระบุเซสชัน

infop
อาร์เรย์ของ dbx_plugin_pthinfo_t structures หรือ NULL ที่จัดสรร

pthrdinfo_size
ขนาดของ dbx_plugin_pthrdinfo_t structure เดี่ยว

indexp
pthread handle เริ่มต้น/ถัดไป (โดยที่ ในอินพุต ศูนย์หมายถึง pthread แรกและ DBX_PLUGIN_PTHREAD_CURRENT หมายถึง pthread ปัจจุบันใน **dbx**)

countp
จำนวนของ pthreads

รหัสที่ส่งกลับของรูทีน dbx **pthreads** callback คือ:

- DBX_PLUGIN_SUCCESS
- DBX_PLUGIN_BAD_SESSION *เซสชัน* ไม่ถูกต้อง
- DBX_PLUGIN_BAD_POINTER *indexp* มีค่า NULL หรือ *countp* เป็น NULL
- DBX_PLUGIN_BAD_ARG *pthrdinfo_size* ไม่ถูกต้องหรือ * *countp* == 0

get_thread_context

รูทีน dbx **get_thread_context** callback อนุญาตให้คุณ อ่านวัตถุประสงค์ทั่วไปของเธรดเคอร์เนล วัตถุประสงค์พิเศษ และรีจิสเตอร์ floating point **dbx** กำหนดค่าพารามิเตอร์ *contextp*

```
typedef int (*dbx_plugin_reg_service_t)(dbx_plugin_session_t session,
                                        uint64_t reg_flags,
                                        uint64_t id,
                                        dbx_plugin_context_t *contextp,
                                        size_t context_size)
```

พารามิเตอร์รูทีน dbx **get_thread_context** callback มีดังนี้:

พารามิเตอร์
คำอธิบาย

session
ตัวระบุเซสชัน

reg_flags
โลจิคัล OR ของอย่างน้อยหนึ่งใน DBX_PLUGIN_REG_GPRS, DBX_PLUGIN_REG_SPRS, DBX_PLUGIN_REG_FPRS, DBX_PLUGIN_REG_EXT

id Kernel thread tid (tid64_t)

contextp

dbx_plugin_context_t structure ที่จัดสรร

context_size

ขนาดของ dbx_plugin_context_t structure ถ้าแฟล็กรีจิสเตอร์ DBX_PLUGIN_REG_EXT ถูกใช้ ควรใช้ขนาดของ dbx_plugin_extctx_t structure dbx_plugin_extctx_t structure เป็นเวอร์ชันที่ขยาย ของ dbx_plugin_context_t structure

รหัสที่ส่งกลับรูทีน dbx **get_thread_context** callback มีดังนี้:

- DBX_PLUGIN_SUCCESS.
- DBX_PLUGIN_BAD_SESSION *เซสชัน* ไม่ถูกต้อง
- DBX_PLUGIN_BAD_ID *ID* ไม่ถูกต้อง
- DBX_PLUGIN_BAD_ARG *reg_flags* ไม่ถูกต้องหรือ *context_size* ไม่ถูกต้อง
- DBX_PLUGIN_BAD_POINTER *contextp* เป็น NULL
- กระบวนการ DBX_PLUGIN_UNAVAILABLE ไม่ได้ทำงานอยู่หรือเธรดอยู่ใน โหมดเคอร์เนลและรีจิสเตอร์ไม่สามารถเข้าถึงได้

set_thread_context

รูทีน dbx **set_thread_context** callback อนุญาตให้คุณ เขียนข้อมูลไปที่วัตถุประสงค์ทั่วไปของเธรดเคอร์เนล วัตถุประสงค์พิเศษและ รีจิสเตอร์ floating point

```
typedef int (*dbx_plugin_reg_service_t) (dbx_plugin_session_t session,
                    uint64_t reg_flags,
                                uint64_t id,
                                dbx_plugin_context_t *contextp,
                                size_t context_size)
```

พารามิเตอร์รูทีน dbx **set_thread_context** callback มีดังนี้:

พารามิเตอร์

คำอธิบาย

session

ตัวระบุเซสชัน

reg_flags

โลจิคัล OR ของอย่างน้อยหนึ่งใน DBX_PLUGIN_REG_GPRS, DBX_PLUGIN_REG_SPRS, DBX_PLUGIN_REG_FPRS, DBX_PLUGIN_REG_EXT

id Kernel thread tid (tid64_t)

contextp

dbx_plugin_context_t structure ที่จัดสรร

context_size

ขนาดของ dbx_plugin_context_t structure ถ้าแฟล็กรีจิสเตอร์ DBX_PLUGIN_REG_EXT ถูกใช้ ควรใช้ขนาดของ dbx_plugin_extctx_t structure dbx_plugin_extctx_t structure เป็นเวอร์ชันที่ขยาย ของ dbx_plugin_context_t structure

รหัสที่ส่งกลับรูทีน dbx **set_thread_context** callback มีดังนี้:

- DBX_PLUGIN_SUCCESS
- DBX_PLUGIN_BAD_SESSION *เซสชัน* ไม่ถูกต้อง
- DBX_PLUGIN_BAD_ID *ID* ไม่ถูกต้อง
- DBX_PLUGIN_BAD_ARG *reg_flags* ไม่ถูกต้องหรือ *context_size* ไม่ถูกต้อง
- DBX_PLUGIN_BAD_POINTER *contextp* เป็น NULL
- กระบวนการ DBX_PLUGIN_UNAVAILABLE ไม่ได้ทำงานอยู่หรือเธรดอยู่ใน โหมดเคอร์เนลและรีจิสเตอร์ไม่สามารถเข้าถึงได้

get_pthread_context

รูทีน dbx **get_pthread_context** callback อนุญาตให้คุณ อ่านวัตถุประสงค์ทั่วไปของ pthread วัตถุประสงค์พิเศษ และรีจิสเตอร์ floating point **dbx** กำหนดค่าพารามิเตอร์ *contextp*

```
typedef int (*dbx_plugin_reg_service_t)(dbx_plugin_session_t session,
                                        uint64_t reg_flags,
                                        uint64_t id,
                                        dbx_plugin_context_t *contextp,
                                        size_t context_size)
```

พารามิเตอร์รูทีน dbx **get_pthread_context** callback มีดังนี้:

พารามิเตอร์
คำอธิบาย

session
ตัวระบุเซสชัน

reg_flags
โลจิคัล OR ของอย่างน้อยหนึ่งใน DBX_PLUGIN_REG_GPRS, DBX_PLUGIN_REG_SPRS, DBX_PLUGIN_REG_FPRS, DBX_PLUGIN_REG_EXT

id pthread handle (pthdb_pthread_t)

contextp
dbx_plugin_context_t structure ที่จัดสรร

context_size
ขนาดของ dbx_plugin_context_t structure ถ้าแฟล็กรีจิสเตอร์ DBX_PLUGIN_REG_EXT ถูกใช้ ควรใช้ขนาดของ dbx_plugin_extctx_t structure dbx_plugin_extctx_t structure เป็นเวอร์ชันที่ขยาย ของ dbx_plugin_context_t structure

รหัสที่ส่งกลับรูทีน dbx **get_pthread_context** callback มีดังนี้:

- DBX_PLUGIN_SUCCESS
- DBX_PLUGIN_BAD_SESSION *เซสชัน* ไม่ถูกต้อง
- DBX_PLUGIN_BAD_ID *ID* ไม่ถูกต้อง
- DBX_PLUGIN_BAD_ARG *reg_flags* ไม่ถูกต้องหรือ *context_size* ไม่ถูกต้อง
- DBX_PLUGIN_BAD_POINTER *contextp* เป็น NULL

- กระบวนการ DBX_PLUGIN_UNAVAILABLE ไม่ได้ทำงานอยู่หรือเธรดอยู่ใน โหมดเคอร์เนลและรีจิสเตอร์ไม่สามารถเข้าถึงได้

set_pthread_context

รูทีน dbx **set_pthread_context** callback อนุญาตให้คุณ เขียนข้อมูลไปที่วัตถุประสงค์ทั่วไปของ pthread วัตถุประสงค์พิเศษ และ รีจิสเตอร์ floating point

```
typedef int (*dbx_plugin_reg_service_t)(dbx_plugin_session_t session,
                                        uint64_t reg_flags,
                                        uint64_t id,
                                        dbx_plugin_context_t *contextp,
                                        size_t context_size)
```

พารามิเตอร์รูทีน dbx **set_pthread_context** callback มีดังนี้:

พารามิเตอร์
คำอธิบาย

session
ตัวระบุเซสชัน

reg_flags
โลจิคัล OR ของอย่างน้อยหนึ่งใน DBX_PLUGIN_REG_GPRS, DBX_PLUGIN_REG_SPRS, DBX_PLUGIN_REG_FPRS, DBX_PLUGIN_REG_EXT

id Pthread handle (pthdb_pthread_t)

contextp
dbx_plugin_context_t structure ที่จัดสรร

context_size
ขนาดของ dbx_plugin_context_t structure ถ้าแฟล็กรีจิสเตอร์ DBX_PLUGIN_REG_EXT ถูกใช้ ควรใช้ขนาดของ dbx_plugin_extctx_t structure dbx_plugin_extctx_t structure เป็นเวอร์ชันที่ขยาย ของ dbx_plugin_context_t structure

รหัสที่ส่งกลับรูทีน dbx **set_pthread_context** callback มีดังนี้:

- DBX_PLUGIN_SUCCESS
- DBX_PLUGIN_BAD_SESSION *เซสชัน* ไม่ถูกต้อง
- DBX_PLUGIN_BAD_ID *ID* ไม่ถูกต้อง
- DBX_PLUGIN_BAD_ARG *reg_flags* ไม่ถูกต้องหรือ *context_size* ไม่ถูกต้อง
- DBX_PLUGIN_BAD_POINTER *contextp* เป็น NULL
- กระบวนการ DBX_PLUGIN_UNAVAILABLE ไม่ได้ทำงานอยู่หรือเคอร์เนลเธรด ที่สามารถเข้าถึงได้กับ pthread อยู่ใน โหมดเคอร์เนลและรีจิสเตอร์ไม่สามารถเข้าถึงได้

read_memory

รูทีน dbx **read_memory** callback อนุญาตให้คุณอ่าน จากพื้นที่แอดเดรสของกระบวนการ **dbx** กำหนด ค่าพารามิเตอร์บัฟเฟอร์

```
typedef int (*dbx_plugin_mem_service_t)(dbx_plugin_session_t session,
                                        uint64_t addr,
                                        void *buffer,
                                        size_t len)
```

พารามิเตอร์รูทีน dbx **read_memory** callback มีดังนี้:

พารามิเตอร์
คำอธิบาย

session
ตัวระบุเซสชัน

addr
แอดเดรสที่จะทำการอ่าน

buffer
บัฟเฟอร์ที่จัดสรรเพื่อเก็บเนื้อหาหน่วยความจำ

len
จำนวนของไบต์ที่อ่าน

รหัสที่ส่งกลับรูทีน dbx **read_memory** callback มีดังนี้:

- DBX_PLUGIN_SUCCESS
- DBX_PLUGIN_BAD_SESSION *เซสชัน* ไม่ถูกต้อง
- DBX_PLUGIN_BAD_POINTER *buffer* เป็น NULL
- DBX_PLUGIN_UNAVAILABLE ไม่สามารถอ่านข้อมูลจาก *addr*

write_memory

รูทีน dbx **write_memory** callback อนุญาตให้คุณเขียน ข้อมูลไปที่พื้นที่แอดเดรสของกระบวนการ

```
typedef int (*dbx_plugin_mem_service_t)(dbx_plugin_session_t session,
                                        uint64_t addr,
                                        void *buffer,
                                        size_t len)
```

พารามิเตอร์รูทีน dbx **write_memory** callback มีดังนี้:

พารามิเตอร์
คำอธิบาย

session
ตัวระบุเซสชัน

addr
แอดเดรสที่จะทำการเขียนข้อมูล

buffer
บัฟเฟอร์ที่จัดสรรและกำหนดค่าเริ่มต้น

len
จำนวนของไบต์ที่จะเขียน

รหัสที่ส่งกลับรูทีน dbx **write_memory** callback มีดังนี้:

- DBX_PLUGIN_SUCCESS
- DBX_PLUGIN_BAD_SESSION *เซสชัน* ไม่ถูกต้อง
- DBX_PLUGIN_BAD_POINTER *buffer* เป็น NULL
- DBX_PLUGIN_UNAVAILABLE ไม่สามารถเขียนข้อมูลไปที่ *addr*

locate_symbol

รูทีน dbx **locate_symbol** callback อนุญาตให้คุณแปลง ชื่อสัญลักษณ์ไปเป็นแอดเดรส

ปลั๊กอินต้องกำหนดค่าเริ่มต้นฟิลด์ *name* และ *mod* ของแต่ละรายการในอาร์เรย์พารามิเตอร์สัญลักษณ์ ฟิลด์ *name* ระบุชื่อของสัญลักษณ์ที่จะระบุตำแหน่ง ฟิลด์ *mod* ระบุดัชนีโมดูลของโมดูลซึ่งควรมี การค้นหา เขตข้อมูล *mod* ถูกกำหนดค่าเริ่มต้นเป็น -1 แสดงว่าโมดูลทั้งหมด ควรถูกค้นหา

dbx กำหนดค่าเขตข้อมูล *addr* สัญลักษณ์ที่ไม่รู้จักมีแอดเดรสเป็นศูนย์ ถ้าสัญลักษณ์ถูกค้นพบ และโมดูลทั้งหมดถูกค้นหาแล้ว **dbx** อัพเดตฟิลด์ *mod* ด้วยดัชนีโมดูลจริงของสัญลักษณ์

```
typedef int (*dbx_plugin_sym_service_t)(dbx_plugin_session_t session,
                                        dbx_plugin_sym_t *symbols,
                                        size_t syminfo_size,
                                        unsigned int count)
```

พารามิเตอร์รูทีน dbx **locate_symbol** callback มีดังนี้:

พารามิเตอร์
คำอธิบาย

session
ตัวระบุเซสชัน

symbols
อาร์เรย์ที่จัดสรรของ dbx_plugin_sym_t structures ที่มีฟิลด์ *name* และ *mod* ถูกกำหนดค่าเริ่มต้น

syminfo_size
ขนาดของ dbx_plugin_sym_t structure

count
จำนวนของสัญลักษณ์ที่ค้นหา

รหัสที่ส่งกลับรูทีน dbx **locate_symbol** callback มีดังนี้:

- DBX_PLUGIN_SUCCESS
- DBX_PLUGIN_BAD_SESSION *เซสชัน* ไม่ถูกต้อง
- DBX_PLUGIN_BAD_ARG *syminfo_size* ไม่ถูกต้อง
- DBX_PLUGIN_BAD_POINTER *symbols* เป็น NULL

what_function

รูทีน dbx **what_function** callback อนุญาตให้คุณแปลง แอดเดรสข้อความไปเป็นสัญลักษณ์

ปลั๊กอินต้องกำหนดค่าเริ่มต้นฟิลด์ *addr* ของแต่ละรายการ ในอาร์เรย์พารามิเตอร์สัญลักษณ์ ฟิลด์ *addr* ระบุ แอดเดรสคำสั่งเครื่อง ภายในฟังก์ชันที่จะถูกจำแนก

dbx กำหนดค่าฟิลด์ *name* แอดเดรส ข้อความที่ไม่รู้จักมีชื่อเป็น NULL **dbx** กำหนดค่า ฟิลด์ *mod* ด้วยดัชนีโมดูลจริงของแอดเดรสข้อความ

```
typedef int (*dbx_plugin_sym_service_t)(dbx_plugin_session_t session,
                                        dbx_plugin_sym_t *symbols,
                                        size_t syminfo_size,
                                        unsigned int count)
```

พารามิเตอร์รูทีน dbx **what_function** callback มีดังนี้:

พารามิเตอร์
คำอธิบาย

session
ตัวระบุเซสชัน

symbols
อาร์เรย์ที่จัดสรรของ dbx_plugin_sym_t structures ที่มีฟิลด์ addr ที่กำหนดค่าเริ่มต้นด้วยแอดเดรสข้อความ

syminfo_size
ขนาดของ dbx_plugin_sym_t structure

count
จำนวนแอดเดรสที่แปลง

รหัสที่ส่งกลับรูทีน dbx **what_function** callback มีดังนี้:

- DBX_PLUGIN_SUCCESS
- DBX_PLUGIN_BAD_SESSION *เซสชัน* ไม่ถูกต้อง
- DBX_PLUGIN_BAD_ARG *syminfo_size* ไม่ถูกต้อง
- DBX_PLUGIN_BAD_POINTER *symbols* เป็น NULL

print

รูทีน dbx **print** callback อนุญาตให้คุณแสดง เอาต์พุตข้อมูลแจ้งให้ทราบหรือเอาต์พุตข้อผิดพลาด

```
typedef int (*dbx_plugin_print_service_t)(dbx_plugin_session_t session,
                                          int print_mode,
                                          char *message)
```

พารามิเตอร์รูทีน dbx **print** callback มีดังนี้:

พารามิเตอร์
คำอธิบาย

session
ตัวระบุเซสชัน

print_mode
DBX_PLUGIN_PRINT_MODE_OUT หรือ DBX_PLUGIN_PRINT_MODE_ERR

message
สตริงอักขระสำหรับ **dbx** ที่จะแสดง

รหัสที่ส่งกลับของรูทีน dbx **print** callback คือ:

- DBX_PLUGIN_SUCCESS
- DBX_PLUGIN_BAD_SESSION *เซสชัน* ไม่ถูกต้อง
- DBX_PLUGIN_BAD_ARG *print_mode* ไม่ถูกต้อง
- DBX_PLUGIN_BAD_POINTER *message* เป็น NULL

alias

รูทีน dbx **alias** callback อนุญาตให้คุณสร้าง alias สำหรับปลั๊กอิน คำสั่งย่อย

ไวยากรณ์ของ **ปลั๊กอิน dbx** คำสั่งย่อย ต้องการใช้ผู้ใช้ dbx พิมพ์ส่วนเติมหน้า ของ `plugin Name` สำหรับแต่ละการเรียกคำสั่งย่อย ปลั๊กอิน เมื่อต้องการจัดเตรียมวิธีลดเวลาการเรียกดังกล่าว **dbx** อนุญาต ให้ปลั๊กอินสร้าง aliases ใหม่

พารามิเตอร์ *alias* และ *expansion* ควรจัดเตรียม คำอธิบายของ alias ใหม่ ไวยากรณ์เหมือนกับไวยากรณ์ ที่กำหนดสำหรับ alias **dbx** subcommand

ต่อไปนี้เป็นตัวอย่างการร้องขอรูทีน dbx **alias** callback :

```
alias("intprt", "plugin xyz interpret");
alias("intprt2(addr, count, format)", "addr / count format; plugin xyz interpret addr");
```

หมายเหตุ: ถ้าคุณพยายามสร้าง alias ที่มีชื่อเหมือนกัน กับ alias ที่มีอยู่ การร้องขอถูกปฏิเสธและข้อความคำเตือน ถูกแสดง ผู้พัฒนาปลั๊กอินถูกกระตุ้นให้สร้าง alias ซึ่งอนุญาตให้ผู้ใช้แก้ไขความขัดแย้งของ alias วิธีหนึ่ง เพื่อบรรลุการแก้ปัญหานี้ คือ โดยการอ่านข้อกำหนด alias จาก configuration file ที่ถูกแพ็กเกจมากับปลั๊กอิน

```
typedef int (*dbx_plugin_alias_service_t)(dbx_plugin_session_t session,
                                          const char *alias,
                                          const char *expansion)
```

พารามิเตอร์รูทีน dbx **alias** callback มีดังนี้:

พารามิเตอร์
คำอธิบาย

session
ตัวระบุเซสชัน

alias
สตริงอักขระที่แสดงชื่อ alias และพารามิเตอร์ทางเลือก

expansion
สตริงอักขระที่แสดงส่วนขยาย alias

รหัสที่ส่งกลับของรูทีน dbx **alias** callback คือ:

- DBX_PLUGIN_SUCCESS
- DBX_PLUGIN_BAD_SESSION *เซสชัน* ไม่ถูกต้อง
- DBX_PLUGIN_BAD_ARG *alias* ไม่ถูกต้อง
- DBX_PLUGIN_BAD_POINTER *alias* เป็น NULL หรือส่วนขยายเป็น NULL
- DBX_PLUGIN_UNAVAILABLE alias ที่มีเดียวกันนี้ มีอยู่แล้ว

ตัวอย่าง

1. ตัวอย่างดังต่อไปนี้กำหนดคำสั่งย่อย **help** และคำสั่งย่อย **hello**:

ตัวอย่าง c:

```
#include <sys/dbx_plugin.h>

dbx_plugin_session_t sid;
dbx_plugin_services_t dbx;

static void usage(void);
static void hello_cmd(void);

int
dbx_plugin_version(void) {
    return DBX_PLUGIN_VERSION_1;
}

int dbx_plugin_session_init(dbx_plugin_session_t session,
                            const dbx_plugin_services_t *servicep) {
    /* record session identifier */
    sid= session;

    /* record dbx service */
    memcpy(&dbx, servicep, sizeof(dbx_plugin_services_t));

    (*(dbx.alias))(sid, "hello", "plugin example hello");
    (*(dbx.alias))(sid, "help", "plugin example help");

    return  0;

}

void
dbx_plugin_session_command(dbx_plugin_session_t session,
                           int argc,
                           char *const argv[]) {
    if (argc == 0 || (argc == 1 && strcmp(argv[0], "help") == 0)) {

 usage();
 return;
```

```
    }
    if (argc == 1 && strcmp(argv[0], "hello") == 0) {
 hello_cmd();
 return;
    }
    (*(dbx.print))(sid,DBX_PLUGIN_PRINT_MODE_ERR,
     "unrecognized command\n");

}

void
dbx_plugin_session_event(dbx_plugin_session_t session,
                         int event,
                         dbx_plugin_event_info_t *event_infop) {
    /* ignore event notifications */
}

void
dbx_plugin_session_destroy(dbx_plugin_session_t session){
    /* no clean up to perform */
}

static
void
usage(void) {
    (*(dbx.print))(sid,DBX_PLUGIN_PRINT_MODE_OUT,
                       "Subcommands for Plug-in \"example\":\n\n" \
                       "   help  - displays this output\n" \
                       "   hello - displays a greeting\n" \
     "\n");
}

static
void
hello_cmd(void) {
    (*(dbx.print))(sid,DBX_PLUGIN_PRINT_MODE_OUT,
     "Hello dbx World!\n");

}
```

example.exp:

```
dbx_plugin_version
dbx_plugin_session_init
dbx_plugin_session_command
dbx_plugin_session_event
dbx_plugin_session_destroy
```

2. **เมื่อต้องการคอมไพล์ตัวอย่างปลั๊กอิน พิมพ์:**

```
cc -q64 -o libdbx_example.so example.c -bM:Sre -bE:example.exp -bnoentry
```

รายการของคำสั่งย่อย dbx

คำสั่งและคำสั่งย่อยสำหรับโปรแกรมดีบัก **dbx** อยู่ใน *Commands Reference*

โปรแกรมดีบัก **dbx** จัดเตรียม คำสั่งย่อยสำหรับดำเนินประเภทงานดังต่อไปนี้:

การตั้งค่าและการลบ breakpoints

คำสั่งย่อย	คำอธิบาย
clear	เอาการหยุดในบรรทัดซอร์สที่กำหนดออก
cleari	เอาจุดหยุดทั้งหมดที่แอดเดรสออก
delete	เอาการติดตามและการหยุดที่ตรงตามหมายเลขที่ระบุออก
status	แสดง **การติดตาม** ที่แอ็คทีฟอยู่และคำสั่งย่อย **stop**
stop	หยุดกระทำการของโปรแกรมแอ็พพลิเคชัน

การรันโปรแกรมของคุณ

คำสั่งย่อย	คำอธิบาย
cont	รันโปรแกรมต่อจากจุดหยุดปัจจุบันจนกว่า โปรแกรมเสร็จสิ้นหรือพบจุดหยุดอื่น
detach	ออกจากโปรแกรมดีบัก แต่ทำการรันแอ็พพลิเคชันต่อ
down	เลื่อนฟังก์ชันลงไปในสแต็ก
goto	ทำให้บรรทัดซอร์สที่ระบุเป็นบรรทัดที่จะรันถัดไป
gotoi	เปลี่ยนแอดเดรส program counter
ถัดไป	รันโปรแกรมแอ็พพลิเคชันที่บรรทัดซอร์สถัดไป
nexti	รันโปรแกรมแอ็พพลิเคชันที่คำสั่งซอร์สถัดไป
rerun	เริ่มรันแอ็พพลิเคชัน
return	รันโปรแกรมแอ็พพลิเคชันต่อจนถึงการส่งกลับไปที่โพรซีเดอร์ที่ระบุ
run	เริ่มรันแอ็พพลิเคชัน
skip	กระทำการต่อเนื่องจากจุดหยุดปัจจุบัน
step	รันหนึ่งบรรทัดซอร์ส
stepi	รันหนึ่งคำสั่งซอร์ส
up	ย้ายฟังก์ชันขึ้นไปบนสแต็ก

การติดตามการเรียทำงานโปรแกรม

คำสั่งย่อย	คำอธิบาย
ติดตาม	พิมพ์ข้อมูลการติดตาม
tracei	เปิดการติดตาม
where	แสดงรายการของโพรซีเดอร์และฟังก์ชันแอ็คทีฟทั้งหมด

การจบการทำงานของโปรแกรม

คำสั่งย่อย	คำอธิบาย
quit	ออกจากโปรแกรมดีบัก **dbx**

การแสดงซอร์สไฟล์

คำสั่งย่อย	คำอธิบาย
edit	ร้องขอตัวแก้ไขบนไฟล์ที่ระบุ
file	เปลี่ยนไฟล์ต้นฉบับปัจจุบันไปเป็นไฟล์ที่ระบุ
func	เปลี่ยนฟังก์ชันปัจจุบันเป็นฟังก์ชันหรือโพรซีเดอร์ที่ระบุ
list	แสดงบรรทัดของไฟล์ต้นฉบับปัจจุบัน
listi	แสดงคำสั่งจากแอ็พพลิเคชัน
move	เปลี่ยนบรรทัดถัดไปที่จะถูกแสดง
/ (Search)	ค้นหารูปแบบไปข้างหน้าในไฟล์ต้นฉบับปัจจุบัน
? (Search)	ค้นหารูปแบบย้อนกลับในไฟล์ต้นฉบับปัจจุบัน
use	ตั้งค่ารายการของไดเร็กทอรีที่จะถูกค้นหาเมื่อค้นหาไฟล์

การพิมพ์และการแก้ไขตัวแปร นิพจน์ และชนิด

คำสั่งย่อย	คำอธิบาย
assign	กำหนดค่าให้กับตัวแปร
case	เปลี่ยนวิธีซึ่ง **dbx** แปลสัญลักษณ์
dump	แสดงชื่อและค่าของตัวแปรในโพรซีเดอร์ที่ระบุ
print	พิมพ์ค่าของนิพจน์หรือรันโพรซีเดอร์และพิมพ์ รหัสที่ส่งกลับ
set	กำหนดค่าให้กับตัวแปร nonprogram
unset	ลบตัวแปร nonprogram
whatis	แสดงการประกาศของคอมโพเนนต์โปรแกรมแอ็พพลิเคชัน
whereis	แสดงการระบุคุณสมบัติเต็มของสัญลักษณ์ทั้งหมดซึ่งมีชื่อชื่อ ตรงกับตัวระบุที่กำหนด
which	แสดงการระบุคุณสมบัติแบบเต็มของตัวระบุที่กำหนด

การดีบักเธรด

คำสั่งย่อย	คำอธิบาย
attribute	แสดงข้อมูลเกี่ยวกับอ็อบเจ็กต์แอ็ตทริบิวต์ ทั้งหมดหรือตามที่เลือก
condition	แสดงข้อมูลเกี่ยวกับตัวแปรเงื่อนไข ทั้งหมดหรือที่เลือก
mutex	แสดงข้อมูลเกี่ยวกับ mutex ทั้งหมดหรือตามที่เลือก
thread	แสดงและควบคุมเธรด
tstophwp	ตั้งค่าการหยุด watchpoint ของฮาร์ดแวร์ระดับเธรด
ttracehwp	ตั้งค่าการติดตาม watchpoint ของฮาร์ดแวร์ระดับเธรด
tstop	ตั้งค่าการหยุด breakpoint ระดับซอร์สสำหรับเธรด
tstopi	ตั้งค่าการหยุด breakpoint ระดับคำสั่งสำหรับเธรด
ttrace	ตั้งค่าการติดตามระดับซอร์สสำหรับเธรด
ttracei	ตั้งค่าการติดตามระดับคำสั่งสำหรับเธรด
tnext	รันเธรดขึ้นไปที่บรรทัดซอร์สถัดไป
tnexti	รันเธรดขึ้นไปที่คำสั่งเครื่องถัดไป
tstep	รันเธรดหนึ่งบรรทัดซอร์ส
tstepi	รันเธรดหนึ่งคำสั่งเครื่อง
tskip	ข้ามจุดหยุดสำหรับเธรด

การดีบัก Multiprocess

คำสั่งย่อย	คำอธิบาย
multproc	เปิดใช้งานหรือปิดใช้งานการดีบัก multiprocess

การเรียกใช้โพรซีเดอร์

คำสั่งย่อย	คำอธิบาย
call	รันอ็อบเจ็กต์โค้ดที่สัมพันธ์กับโพรซีเดอร์หรือฟังก์ชันที่กำหนดชื่อ
print	พิมพ์ค่าของนิพจน์หรือรันโพรซีเดอร์และพิมพ์ รหัสที่ส่งกลับ

การจัดการสัญญาณ

คำสั่งย่อย	คำอธิบาย
catch	เริ่มจับสัญญาณก่อนสัญญาณถูกส่งไปที่โปรแกรม แอ็พพลิเคชัน
ignore	หยุดจับสัญญาณก่อนสัญญาณถูกส่งไปที่โปรแกรม แอ็พพลิเคชัน

การดีบักระดับเครื่อง

คำสั่งย่อย	คำอธิบาย
display memory	แสดงเนื้อหาของหน่วยความจำ
gotoi	เปลี่ยนแอดเดรส program counter
map	แสดงการแม็พแอดเดรสและข้อมูลโหลดเดอร์สำหรับโปรแกรม แอ็พพลิเคชัน
nexti	รันโปรแกรมแอ็พพลิเคชันที่คำสั่งเครื่องถัดไป
registers	แสดงค่าของรีจิสเตอร์อเนกประสงค์ รีจิสเตอร์ system-control รีจิสเตอร์ floating-point registersและรีจิสเตอร์คำสั่งปัจจุบันทั้งหมด
stepi	รันหนึ่งคำสั่งซอร์ส
stopi	ตั้งค่าหยุดที่ตำแหน่งที่ระบุ
tracei	เปิดการติดตาม

การดีบักการควบคุมสภาวะแวดล้อม

คำสั่งย่อย	คำอธิบาย
alias	แสดงและกำหนด aliases สำหรับคำสั่งย่อย **dbx**
help	แสดงข้อมูลวิธีใช้สำหรับคำสั่งย่อย **dbx** หรือหัวข้อ
prompt	เปลี่ยนพรอมต์ **dbx** เป็นสตริงที่ระบุ
screen	เปิด Xwindow สำหรับเอาต์พุตคำสั่ง **dbx**
sh	ผ่านคำสั่งไปที่เชลล์เพื่อดำเนินการ
source	อ่านคำสั่ง **dbx** จากไฟล์
unalias	เอา alias ออก

ภาพรวมการบันทึกข้อผิดพลาด

การประมวลผล การบันทึกข้อผิดพลาดเริ่มต้นขึ้นเมื่อโมดูลระบบปฏิบัติการตรวจพบข้อผิดพลาด

การตรวจหาข้อผิดพลาด ส่วนย่อยของโค้ด จากนั้นส่งข้อมูลข้อผิดพลาดไปที่เคอร์เนลเซอร์วิส **errsave** และ **errlast** หรือไปที่รูทีนย่อย **errlog** จากข้อมูลข้อผิดพลาดนี้ถูกเขียนลงไฟล์พิเศษ **/dev/error** จากนั้นการประมวลผลนี้เพิ่มการประทับเวลาลงในข้อมูลที่รวบรวม **errdemon** daemon ตรวจสอบ ไฟล์ **/dev/error** เพื่อดูรายการใหม่อย่างสม่ำเสมอ และเมื่อ มีการบันทึกข้อมูลใหม่ daemon จะจัดการด้วยชุดของการดำเนินการ

ก่อนที่รายการจะถูกเขียนลงบันทึกข้อผิดพลาด **errdemon** daemon เปลี่ยนเทียบเลเบลที่ส่งโดยเคอร์เนลหรือ แอ็พพลิเคชันโค้ดกับเนื้อหาของ Error Record Template Repository ถ้าเลเบลตรงกันกับรายการในที่เก็บ daemon จะรวบรวมข้อมูลเพิ่มเติมจากส่วนอื่นๆ ของระบบ

ในการสร้างรายการในบันทึกข้อผิดพลาด **errdemon** daemon เรียกข้อมูลเท็มเพลตที่เหมาะสมจากที่เก็บ ชื่อรีซอร์สของยูนิตที่ตรวจสอบข้อผิดพลาด และข้อมูลรายละเอียด รวมทั้ง ถ้าข้อผิดพลาดแสดงให้เห็นปัญหาที่เกี่ยวกับฮาร์ดแวร์ และมี vital product data (VPD) ของฮาร์ดแวร์อยู่ daemon จะเรียกข้อมูล VPD จาก Object Data Manager เมื่อคุณเข้าถึงบันทึกข้อผิด

พลาด ไม่ว่าผ่าน SMIT หรือด้วยคำสั่ง **errpt** บันทึกข้อผิดพลาดจะถูกจัดรูปแบบตาม เท็มเพลตข้อผิดพลาดในที่เก็บเท็มเพลต ข้อผิดพลาด และแสดงใน รายงานสรุป หรือรายงานรายละเอียด รายกาสามารถถูกเรียกข้อมูลโดยใช้เซอร์วิส ที่จัดไว้ให้ใน **liberrlog, errlog_open, errlog_close, errlog_find_first, errlog_find_next, errlog_find_sequence, errlog_set_direction** และ **errlog_write errlog_write** มีความสามารถในการอัพเดตที่จำกัด

รายการส่วนใหญ่ในบันทึกข้อผิดพลาดมาจากปัญหาฮาร์ดแวร์และซอฟต์แวร์ แต่ข้อความแสดงข้อมูลก็สามารถบันทึกได้

คำสั่ง **diag** ใช้บันทึกข้อผิดพลาดเพื่อวินิจฉัยปัญหาฮาร์ดแวร์ เพื่อให้วินิจฉัย ปัญหาของระบบใหม่ได้อย่างถูกต้อง ระบบจะลบ รายการที่เกี่ยวกับฮาร์ดแวร์ ที่มีอายุมากกว่า 90 วันออกจากบันทึกข้อผิดพลาด ระบบลบรายการที่เกี่ยวกับซอฟต์แวร์ ออกหลัง จากบันทึกการทำงานไว้ 30 วัน

คุณควรคุ้นเคยกับเทอมต่อไปนี้:

คำศัพท์	**คำอธิบาย**
ID ข้อผิดพลาด	โค้ดฐานสิบหก CRC 32 บิตที่ใช้ระบุความล้มเหลวเฉพาะ แต่ละเท็มเพลตของเร็กคอร์ดข้อผิดพลาด จะมี ID ข้อผิดพลาดเฉพาะ
เลเบลข้อผิดพลาด	ชื่อช่วยจำสำหรับ ID ข้อผิดพลาด
บันทึกข้อผิดพลาด	ไฟล์ที่เก็บอินสแตนซ์ของข้อผิดพลาดและความล้มเหลวที่ระบบ พบ
รายการบันทึกข้อผิดพลาด	เร็กคอร์ดในบันทึกข้อผิดพลาดระบบที่อธิบายความขัดข้องของฮาร์ดแวร์ ความขัดข้องของ ซอฟต์แวร์ หรือข้อความตัวดำเนินการ รายการบันทึกข้อผิดพลาดมี ข้อมูลความล้มเหลวที่บันทึกได้
เท็มเพลตของเร็กคอร์ดข้อผิดพลาด	รายละเอียดของข้อมูลที่แสดงเมื่อบันทึกข้อผิดพลาดถูกจัดรูปแบบ สำหรับรายงาน โดยรวมข้อมูล เกี่ยวกับประเภทและคลาสของข้อผิดพลาด สาเหตุ ที่เป็นไปได้ และการดำเนินการที่แนะนำ โดยรวม แล้ว เท็มเพลตประกอบเป็น Error Record Template Repository

ข้อมูลที่เกี่ยวข้อง:

การบันทึกข้อผิดพลาดไฟล์พิเศษ

คำสั่ง errsave

คำสั่ง errlog

คำสั่ง crontab

คำสั่ง errclear

คำสั่ง errdead

คำสั่ง errdemon

คำสั่ง errinstall

คำสั่ง errlogger

คำสั่ง errmsg

คำสั่ง errpt

คำสั่ง errstop

คำสั่ง errupdate

คำสั่ง odmadd

คำสั่ง errstop

คำสั่ง odmget

คำสั่ง snap

Error-logging facility

หน่วยบริการ error-logging บันทึกความล้มเหลวของฮาร์ดแวร์และซอฟต์แวร์ในบันทึกข้อผิดพลาด เพื่อวัตถุประสงค์ด้านข้อมูลหรือเพื่อตรวจจับข้อผิดพลาดและการดำเนินการแก้ไข

อ้างอิงข้อมูลดังต่อไปนี้ในการใช้หน่วยบริการ error-logging:

ในบาง AIX Version 4 ของคำสั่งบันทึกข้อผิดพลาด ถูกส่งมอบในแพ็กเกจการติดตั้งที่เป็นทางเลือกชื่อว่า **bos.sysmgt.serv_aid** ระบบฐาน (**bos.rte**) มีเซอร์วิสดังต่อไปนี้สำหรับบันทึกข้อผิดพลาดลงในไฟล์บันทึกข้อผิดพลาด:

- รูทีนย่อย errlog
- เซอร์วิสเคอร์เนล **errsave** และ **errlast**
- ไดรเวอร์อุปกรณ์ข้อผิดพลาด (**/dev/error**)
- **error** daemon
- คำสั่ง **errstop**

คำสั่งที่จำเป็นสำหรับการติดตั้งโปรแกรมที่ได้ไลเซนส์ (**errinstall** และ **errupdate**) ยังถูกรวมไว้ใน **bos.rte** เช่นกัน สำหรับข้อมูลเพิ่มเติมเกี่ยวกับการถ่ายโอน ล็อกไฟล์ข้อผิดพลาดของระบบของคุณไปยังระบบที่ติดตั้งแพ็กเกจ Software Service Aids โปรดดูที่ การถ่ายโอน บันทึกข้อผิดพลาดของคุณไปยังระบบอื่น

การจัดการการบันทึกข้อผิดพลาด

การบันทึกข้อผิดพลาดถูกเริ่มต้นโดยอัตโนมัติโดยสคริปต์ **rc.boot** ระหว่างการกำหนดค่าเริ่มต้นระบบ และถูกหยุดโดยอัตโนมัติ โดยสคริปต์ **shutdown** ระหว่างการปิดระบบ

การวิเคราะห์บันทึกข้อผิดพลาดทำโดยคำสั่ง **diag** วิเคราะห์รายการข้อผิดพลาดฮาร์ดแวร์ ระยะเวลาเริ่มต้นที่รายการข้อผิดพลาด ฮาร์ดแวร์คงอยู่ในบันทึกข้อผิดพลาดคือ 90 วัน ถ้าคุณเอารายการข้อผิดพลาดฮาร์ดแวร์ ที่มีอายุน้อยกว่า 90 วันออก คุณสามารถจำกัดประสิทธิผลของการวิเคราะห์ บันทึกข้อผิดพลาดนี้

การถ่ายโอนบันทึกข้อผิดพลาดของคุณไปยังระบบอื่น

คำสั่ง **errclear, errdead, errlogger, errmsg** และ **errpt** เป็นส่วนหนึ่งของแพ็กเกจ Software Service Aids ที่ติดตั้งได้เป็นทางเลือก (**bos.sysmgt.serv_aid**) คุณต้องใช้แพ็กเกจ Software Service Aids เพื่อสร้างรายงานจาก บันทึกข้อผิดพลาดหรือเพื่อลบรายการจากบันทึกข้อผิดพลาด คุณสามารถติดตั้งแพ็กเกจ Software Service Aids บนระบบของคุณ หรือคุณสามารถถ่ายโอนไฟล์ บันทึกข้อผิดพลาดของระบบที่มีแพ็กเกจ Software Service Aids ติดตั้งอยู่

ระบุพาธไฟล์บันทึกข้อผิดพลาดของระบบของคุณ โดยรันคำสั่งดังต่อไปนี้:

```
/usr/lib/errdemon -l
```

คุณสามารถถ่ายโอนไฟล์ไปที่ระบบอื่นได้หลาย วิธี คุณสามารถ:

- คัดลอกไฟล์ไปที่ระบบใส่ที่เชื่อมต่อแบบรีโมตโดยใช้คำสั่ง **cp**
- คัดลอกไฟล์ข้ามการเชื่อมต่อเครือข่ายโดยใช้คำสั่ง **rcp, ftp** หรือ **tftp**
- คัดลอกไฟล์ไปที่สื่อบันทึกแบบถอดได้โดยใช้คำสั่ง **tar** หรือ **backup** และไฟล์คืนค่าบนระบบอื่น

คุณสามารถจัดรูปแบบรายงานสำหรับบันทึกข้อผิดพลาดที่คัดลอก ไปที่ระบบของคุณจากระบบอื่นโดยใช้แฟล็ก **–i** ของคำสั่ง **errpt** แฟล็ก **–i** อนุญาตให้คุณระบุชื่อพาธของไฟล์บันทึกข้อผิดพลาดนอกจากค่าเริ่มต้น เช่นเดียวกัน คุณสามารถลบรายการออกจากไฟล์บันทึกข้อผิดพลาดที่คัดลอกไปที่ ระบบของคุณจากระบบอื่นโดยใช้แฟล็ก **–i** ของคำสั่ง **errclear**

การกำหนดค่าการบันทึกข้อผิดพลาด

คุณสามารถกำหนดชื่อและตำแหน่งของไฟล์บันทึกข้อผิดพลาด และขนาดของบัฟเฟอร์ข้อผิดพลาดภายในเองเพื่อให้เหมาะสมกับความต้องการของคุณ คุณ ยังสามารถควบคุมการบันทึกข้อผิดพลาดที่ซ้ำกัน

การแสดงรายการค่าติดตั้งปัจจุบัน

เมื่อต้องการแสดงการตั้งค่าปัจจุบัน รัน **/usr/lib/errdemon –l** ค่าสำหรับชื่อไฟล์บันทึกข้อผิดพลาด ขนาดไฟล์บันทึกข้อผิดพลาด และขนาดบัฟเฟอร์ ที่ปัจจุบันเก็บอยู่ในฐานข้อมูลการกำหนดค่า error–log แสดง บนหน้าจอของคุณ

การกำหนดตำแหน่งล็อกไฟล์เอง

เมื่อต้องการเปลี่ยนชื่อไฟล์ที่ใช้สำหรับการบันทึกข้อผิดพลาด ให้รันคำสั่ง **/usr/lib/errdemon –i** *FileName* ชื่อไฟล์ที่ระบุถูกบันทึกในฐานข้อมูลการกำหนดค่าบันทึกข้อผิดพลาด และ error daemon ถูกเริ่มระบบใหม่ทันที

การกำหนดขนาดของล็อกไฟล์เอง

เมื่อต้องการเปลี่ยนขนาดสูงสุดของไฟล์บันทึกข้อผิดพลาด พิมพ์:

```
/usr/lib/errdemon -s LogSize
```

ขนาดจำกัดที่ระบุสำหรับไฟล์บันทึกถูกบันทึก ในฐานข้อมูลการกำหนดค่า error–log และ error ถูกเริ่มระบบใหม่ ทันที ถ้าการจำกัดขนาดสำหรับไฟล์บันทึกน้อยกว่าขนาดของ ไฟล์บันทึกที่ใช้อยู่ขณะนี้ ไฟล์บันทึกปัจจุบันจะถูกเปลี่ยนชื่อโดยต่อท้ายด้วย **.old** กับชื่อไฟล์ และไฟล์บันทึกใหม่จะถูกสร้างด้วย การจำกัดขนาดที่ระบุ จำนวนของพื้นที่ที่ระบุถูกสำรองไว้สำหรับ ไฟล์บันทึกข้อผิดพลาดและไฟล์อื่นไม่สามารถนำไปใช้ได้ ดังนั้น โปรดระวัง อย่าทำให้บันทึกมีขนาดใหญ่เกินไป แต่ ถ้าคุณสร้างบันทึกเล็กเกินไป ข้อมูลสำคัญอาจถูกเขียนทับก่อนกำหนด เมื่อขนาดถึงการจำกัด ขนาดไฟล์ ไฟล์จะ *wraps* ซึ่งคือ รายการเก่าที่สุดจะถูกเขียนทับโดยรายการใหม่

การกำหนดขนาดของบัฟเฟอร์เอง

เมื่อต้องการเปลี่ยนขนาดของบัฟเฟอร์ภายใน ของไดรเวอร์อุปกรณ์บันทึกข้อผิดพลาด พิมพ์:

```
/usr/lib/errdemon -B BufferSize
```

ขนาดบัฟเฟอร์ที่ระบุถูกบันทึกในฐานข้อมูลการกำหนดค่า error–log และถ้ามีขนาดใหญ่เกินกว่าขนาดบัฟเฟอร์ที่ใช้อยู่ในปัจจุบัน บัฟเฟอร์ในหน่วยความจำจะเพิ่มขึ้นทันที ถ้ามีขนาดน้อยกว่าขนาดบัฟเฟอร์ที่ ใช้อยู่ในปัจจุบันขนาดใหม่จะถูกนำมาใช้ในครั้งต่อไปที่ error daemon ถูกเริ่มต้นหลังจากระบบถูกรีบูต ขนาดบัฟเฟอร์ ไม่สามารถทำให้เล็กกว่าค่าเริ่มต้น hard–coded 8 KB ขนาดที่คุณระบุ จะถูกปัดขึ้นเป็นผลคูณจำนวนเต็มถัดไปของขนาดเพจหน่วยความจำ (4 KB) หน่วยความจำที่ใช้สำหรับบัฟเฟอร์ในหน่วยความจำของไดรเวอร์อุปกรณ์บันทึกข้อผิดพลาด กระบวนการอื่นไม่สามารถใช้ได้ (บัฟเฟอร์ถูกกำหนดไว้ตายตัว)

โปรดระวังไม่ให้มีผลกระทบกับประสิทธิภาพระบบของคุณ จากการทำให้บัฟเฟอร์มีขนาดใหญ่เกินไป แต่ ถ้าคุณสร้างบัฟเฟอร์ที่เล็กเกินไป บัฟเฟอร์อาจเต็มถ้ารายการข้อผิดพลาดมาถึงเร็วกว่าที่ ถูกอ่านจากบัฟเฟอร์และถูกนำไปไว้ในไฟล์บันทึก

เมื่อบัฟเฟอร์ เต็ม รายการใหม่จะถูกปฏิเสธ จนกว่าจะมีที่ว่างในบัฟเฟอร์ เมื่อสถานการณ์นี้เกิดขึ้น รายการข้อผิดพลาดถูกสร้างเพื่อแจ้งคุณ ถึงปัญหา และคุณสามารถแก้ไขปัญหาโดยขยายขนาดบัฟเฟอร์

การกำหนดการจัดการข้อผิดพลาดที่ซ้ำกันเอง

โดยค่าเริ่มต้น เริ่มต้นด้วย AIX 5.1 error daemon ขจัดข้อผิดพลาดที่ซ้ำซ้อน โดยการดูที่แต่ละข้อผิดพลาดที่ถูกบันทึก ข้อผิดพลาดเกิดการซ้ำซ้อนถ้า เหมือนกับข้อผิดพลาดก่อนหน้า และถ้าเกิดขึ้นภายในช่วงเวลา โดยประมาณที่ระบุด้วย **/usr/lib/errdemon -t time-interval** ค่าเวลาเริ่มต้นคือ 10000 หรือ 10 วินาที ค่าอยู่ในแบบมิลลิวินาที

แฟล็ก **-m maxdups** ควบคุมจำนวนการซ้ำซ้อน ที่สะสมได้ ก่อนที่รายการซ้ำซ้อนจะถูกบันทึก ค่าดีฟอลต์คือ 1000 ถ้าข้อผิดพลาด มีข้อผิดพลาดที่เหมือนกันเกิดขึ้น 1000 ครั้งตามมา ถูกบันทึก ข้อผิดพลาดซ้ำซ้อนถูกบันทึกที่จุดนั้นแทนการรอช่วงเวลา ให้หมดอายุหรือเพื่อรอข้อผิดพลาดที่ไม่ซ้ำเกิดขึ้น

ตัวอย่างเช่น ถ้าตัวจัดการอุปกรณ์เริ่มบันทึกข้อผิดพลาดซ้ำกันจำนวนมากอย่างรวดเร็ว ข้อผิดพลาดส่วนใหญ่จะไม่แสดงในบันทึก แต่จะบันทึกเฉพาะข้อผิดพลาดที่เกิดครั้งแรก การเกิดขึ้นที่ตามมาจะไม่ถูกบันทึกในทันที แต่จะถูกนับไว้เท่านั้น เมื่อช่วงเวลาหมดอายุลง มีจำนวนถึงค่า **maxdups** หรือเมื่อข้อผิดพลาดอื่นถูกบันทึก รูปแบบอื่นของข้อผิดพลาด ถูกบันทึก ให้ข้อมูลเวลาของการซ้ำซ้อนแรกและครั้งสุดท้าย และจำนวน ของการซ้ำซ้อน

หมายเหตุ: ช่วงเวลาอ้างอิงถึงเวลาตั้งแต่ข้อผิดพลาดล่าสุด ไม่ใช่เวลาตั้งแต่การเกิดขึ้นครั้งแรกของข้อผิดพลาดนี้ นั้นคือ เวลาถูกรีเซ็ต แต่ละครั้งที่ข้อผิดพลาดถูกบันทึก นอกจากนี้ จะเป็นการซ้ำซ้อนได้ข้อผิดพลาดจะต้องตรง กับข้อผิดพลาดก่อนหน้า ถ้า ตัวอย่าง มีข้อมูลรายละเอียดใด ต่างไปจากข้อผิดพลาดก่อนหน้า จะถือว่าข้อผิดพลาดไม่ซ้ำ และบันทึกเป็นข้อผิดพลาดแยก

การลบรายการบันทึกข้อผิดพลาด

รายการถูกเอาออกจากบันทึกข้อผิดพลาดเมื่อผู้ใช้ root รันคำสั่ง **errclear** เมื่อคำสั่ง **errclear** ถูกเรียกโดยอัตโนมัติโดยงาน **cron** รายวัน หรือเมื่อไฟล์บันทึกข้อผิดพลาดเวียนรอบเนื่องจากถึง ขนาดสูงสุด เมื่อขนาดไฟล์บันทึกข้อผิดพลาดถึงขนาดสูงสุดที่ระบุใน ฐานข้อมูลการกำหนดค่า error-log รายการเก่าที่สุดจะถูกเขียนทับโดย รายการใหม่ล่าสุด

การลบโดยอัตโนมัติ

ไฟล์ **crontab** ได้จัดเตรียม ระบบลบข้อผิดพลาดฮาร์ดแวร์ที่เก่ากว่า 90 วันและข้อผิดพลาดอื่นที่เก่า กว่า 30 วัน เมื่อต้องการแสดงรายการ **crontab** สำหรับ ระบบของคุณ พิมพ์:

```
crontab -l Command
```

เมื่อต้องการเปลี่ยนรายการเหล่านี้ พิมพ์:

```
crontab -e Command
```

คำสั่ง errclear

คำสั่ง **errclear** สามารถถูกใช้ เพื่อเอารายการออกโดยการเลือกจากบันทึกข้อผิดพลาด เกณฑ์การเลือกที่ คุณอาจระบุได้รวมถึง error ID number, sequence number, error label, resource name, resource class, error class และ error type คุณยังต้อง ระบุอายุของรายการที่จะถูกลบออก รายการที่ตรงกับเกณฑ์การเลือก ที่คุณระบุ และเก่ากว่าจำนวนวันที่คุณระบุ จะถูกลบ

การเปิดใช้งานและการปิดใช้งานการบันทึกเหตุการณ์

คุณสามารถปิดใช้งานการบันทึกหรือการรายงานเหตุการณ์เฉพาะ โดยการปรับเปลี่ยนฟิลด์ **Log** หรือ **Report** ของแม่แบบข้อผิดพลาดสำหรับเหตุการณ์ คุณสามารถใช้คำสั่ง **errupdate** เพื่อเปลี่ยนการตั้งค่าปัจจุบันสำหรับ เหตุการณ์

การแสดงเหตุการณ์ที่การบันทึกถูกปิดใช้งาน

เมื่อต้องการแสดงเหตุการณ์ทั้งหมดซึ่งการบันทึก ถูกปิดใช้งานในขณะนี้ พิมพ์:

```
errpt -t -F Log=0
```

เหตุการณ์ซึ่งการบันทึกถูกปิดใช้งานจะไม่ถูกบันทึก ในไฟล์บันทึกข้อผิดพลาด

การแสดงเหตุการณ์ที่การรายงานถูกปิดใช้งาน

เมื่อต้องการแสดงเหตุการณ์ทั้งหมดซึ่งการรายงาน ถูกปิดใช้งานในขณะนี้ พิมพ์:

```
errpt -t -F Report=0
```

เหตุการณ์ซึ่งการรายงานถูกปิดใช้งานถูกบันทึกใน ไฟล์บันทึกข้อผิดพลาดเมื่อเหตุการณ์เกิดขึ้น แต่จะไม่ถูกแสดงโดยคำสั่ง **errpt**

การเปลี่ยนค่าติดตั้งปัจจุบันสำหรับเหตุการณ์

เมื่อต้องการเปลี่ยนการตั้งค่าปัจจุบันสำหรับเหตุการณ์ คุณสามารถใช้คำสั่ง **errupdate** อินพุตที่จำเป็นกับคำสั่ง **errupdate** สามารถอยู่ในไฟล์หรือมาจากอินพุต มาตรฐาน

ตัวอย่างดังต่อไปนี้ใช้อินพุตมาตรฐาน เมื่อต้องการปิดใช้งาน การรายงานของเหตุการณ์ **ERRLOG_OFF** (error ID 192AC071) ให้พิมพ์ดังต่อไปนี้เพื่อรันคำสั่ง **errupdate**:

```
errupdate <Enter>
=192AC071: <Enter>
Report=False <Enter>
<Ctrl-D>
<Ctrl-D>
```

การบันทึกกิจกรรมการซ่อมบำรุง

คำสั่ง **errlogger** อนุญาต ให้ผู้ดูแลระบบบันทึกข้อความลงในบันทึกข้อผิดพลาด เมื่อใดก็ตามที่คุณ ดำเนินกิจกรรมการดูแลรักษา เช่นการล้างรายการออกจากบันทึกข้อผิดพลาด เปลี่ยนฮาร์ดแวร์ หรือนำการแก้ไขซอฟต์แวร์มาใช้ เป็นความคิดที่ดีที่จะบันทึก กิจกรรมนี้ไว้ในบันทึกข้อผิดพลาดระบบ

คำสั่ง **ras_logger** จัดเตรียมวิธีในการบันทึก ข้อผิดพลาดจากบรรทัดคำสั่ง สามารถถูกใช้เพื่อทดสอบแม่แบบข้อผิดพลาดที่สร้าง ขึ้นใหม่ และจัดเตรียมวิธีในการติดตามข้อผิดพลาดจากเชลล์สคริปต์

การเปลี่ยนทิศทางข้อความ syslog ไปยังบันทึกข้อผิดพลาด

บางแอ็พพลิเคชันใช้ syslog สำหรับบันทึกข้อผิดพลาด และเหตุการณ์อื่น เพื่อแสดงข้อความบันทึกข้อผิดพลาดและข้อความ syslog ในรายงานเดียว ให้เปลี่ยนทิศทางข้อความ syslog ไปที่ error log คุณสามารถทำได้โดยระบุ *errlog* เป็นปลายทางในไฟล์การกำหนดค่า syslog (**/etc/syslog.conf**) โปรดดูที่ **syslogd** daemon สำหรับข้อมูลเพิ่มเติม

การกำหนดทิศทางข้อความบันทึกข้อผิดพลาดไปยัง syslog

คุณสามารถบันทึกเหตุการณ์บันทึกข้อผิดพลาดในไฟล์ **syslog** โดยใช้คำสั่ง **logger** ที่มี ความสามารถในการแจ้งเตือนข้อผิดพลาดที่เกิดขึ้น ตัวอย่างเช่น เมื่อต้องการบันทึก ข้อความระบบ (syslog) ให้เพิ่มอ็อบเจ็กต์ errnotify ด้วยเนื้อหาดังต่อไปนี้:

```
errnotify:
        en_name = "errdupxmp"
        en_persistenceflg = 1
        en_method = "logger Msg from Error Log: `errpt -l $1 | grep -v 'ERROR_ID TIMESTAMP'`"
```

ตัวอย่างเช่น สร้างไฟล์ชื่อ **/tmp/syslog.add** ด้วยเนื้อหาเหล่านี้ แล้วรันคำสั่ง **odmadd /tmp/syslog.add** (คุณต้องล็อกอินเป็นผู้ใช้ root)

สำหรับข้อมูลเพิ่มเติมเกี่ยวกับการแจ้งเตือนข้อผิดพลาดที่เกิดขึ้นพร้อมกัน โปรดดูที่ การแจ้งเตือนเกี่ยวกับข้อผิดพลาด

การแจ้งเตือนข้อผิดพลาด

คลาสอ็อบเจ็กต์ Error Notification ระบุเงื่อนไขและการดำเนินการ ที่จะกระทำ เมื่อข้อผิดพลาดถูกบันทึกในบันทึกข้อผิดพลาดของระบบ ผู้ใช้ระบุ เงื่อนไขและการดำเนินการเหล่านี้ในอ็อบเจ็กต์ Error Notification

แต่ละครั้งที่ข้อผิดพลาดถูกบันทึก **error notification** daemon ระบุว่ารายการบันทึกข้อผิดพลาดตรงกับ เกณฑ์การเลือกของอ็อบเจ็กต์ Error Notification ใดหรือไม่ ถ้าตรงกัน daemon รันการดำเนินการที่โปรแกรมไว้ และเรียก *เมธอด การแจ้งเตือน* สำหรับแต่ละอ็อบเจ็กต์ที่ตรงกัน

คลาสอ็อบเจ็กต์ Error Notification อยู่ในไฟล์ **/etc/objrepos/errnotify** อ็อบเจ็กต์ Error Notification ถูกเพิ่มให้กับคลาสอ็อบเจ็กต์โดยใช้คำสั่ง Object Data Manager (ODM) มีเพียงกระบวนการที่รันด้วยสิทธิ root user สามารถเพิ่มอ็อบเจ็กต์ให้กับคลาสอ็อบเจ็กต์ Error Notification อ็อบเจ็กต์ Error Notification มี descriptors ดังต่อไปนี้:

`en_alertflg`
ระบุว่าข้อผิดพลาดสามารถถูกแจ้งได้หรือไม่ descriptor นี้ถูกจัดเตรียม เพื่อใช้โดยเอเจนต์การแจ้งเตือนที่สัมพันธ์กับแอ็พพลิเคชันการจัดการเครือข่ายโดยใช้ SNA Alert Architecture ค่า descriptor การแจ้งเตือนที่ใช้ได้คือ:

TRUE สามารถถูกแจ้งเตือนได้

FALSE ไม่สามารถถูกแจ้งเตือนได้

`en_class`
ระบุคลาสของรายการบันทึกข้อผิดพลาดที่ต้องการจับคู่ ค่า descriptor **en_class** ที่ใช้ได้คือ:

H คลาส Hardware Error

S คลาส Software Error

O ข้อความจากคำสั่ง **errlogger**

U ไม่ได้กำหนด

`en_crcid`
ระบุตัวระบุข้อผิดพลาดที่สัมพันธ์กับข้อผิดพลาด ตัวระบุข้อผิดพลาดสามารถเป็นค่าตัวเลขทั้งหมดที่ใช้ได้กับค่าแอ็ตทริบิวต์ คลาสอ็อบเจ็กต์ Predefined Attribute (PdAt) คำสั่ง **errpt** แสดงตัวระบุข้อผิดพลาดเป็นเลขฐานสิบหก ตัวอย่างเช่น เมื่อต้องการเลือกรายการที่คำสั่ง **errpt** แสดงด้วย `IDENTIFIER: 67581038`, ให้ระบุ `en_crcid = 0x67581038`

en_dup

ถ้าตั้งค่าไว้ ระบุว่าข้อผิดพลาดซ้ำตามที่กำหนด โดยเคอร์เนลควรถูกจับคู่หรือไม่ ค่า descriptor **en_dup** ที่ใช้ได้คือ:

TRUE ข้อผิดพลาดซ้ำ

FALSE ข้อผิดพลาดไม่ซ้ำ

en_err64

ถ้าตั้งค่า ระบุว่าข้อผิดพลาดจากสภาพแวดล้อม 64-บิต หรือ 32-บิตที่ควรถูกจับคู่ ค่า descriptor **en_err64** ที่ใช้ได้คือ:

TRUE ข้อผิดพลาดมาจากสภาพแวดล้อม 64-บิต

FALSE ข้อผิดพลาดมาจากสภาพแวดล้อม 32-บิต

en_label

ระบุเลเบลที่สัมพันธ์กับตัวระบุข้อผิดพลาด ตามที่กำหนดในเอาต์พุตของคำสั่ง **errpt -t**

en_method

ระบุการดำเนินการ user-programmable เช่น เชลล์สคริปต์หรือสตริงคำสั่ง ที่จะถูกรันเมื่อข้อผิดพลาดตรงกับเกณฑ์การเลือกของอ็อบเจ็กต์ Error Notification ถูกบันทึก การแจ้งเตือนข้อผิดพลาด daemon ใช้คำสั่ง **sh -c** เพื่อดำเนินการเมธอดการแจ้งเตือน

คีย์เวิร์ดดังต่อไปนี้ถูกขยายโดยอัตโนมัติโดย **การแจ้งเตือนข้อผิดพลาด** daemon เป็นอาร์กิวเมนต์ไปที่เมธอดการแจ้งเตือน

$1 หมายเลขลำดับจากรายการบันทึกข้อผิดพลาด

$2 Error ID จากรายการบันทึกข้อผิดพลาด

$3 คลาสจากรายการบันทึกข้อผิดพลาด

$4 ชนิดจากรายการบันทึกข้อผิดพลาด

$5 ค่าแฟล็กการแจ้งเตือนจากรายการบันทึกข้อผิดพลาด

$6 ชื่อทรัพยากรจากรายการบันทึกข้อผิดพลาด

$7 ชนิดทรัพยากรจากรายการบันทึกข้อผิดพลาด

$8 คลาสทรัพยากรจากรายการบันทึกข้อผิดพลาด

$9 เลเบลข้อผิดพลาดจากรายการบันทึกข้อผิดพลาด

en_name

ระบุดอ็อบเจ็กต์เฉพาะ ชื่อเฉพาะนี้ถูกใช้เพมื่อเอา อ็อบเจ็กต์ออก

en_persistenceflg

กำหนดว่าอ็อบเจ็กต์ Error Notification ควรถูกเอาออก โดยอัตโนมัติหรือไม่ เมื่อเริ่มระบบใหม่ ตัวอย่างเช่น เพื่อหลีกเลี่ยงการส่งสัญญาณที่ผิดพลาด อ็อบเจ็กต์ Error Notification มีเมธอดที่ส่งสัญญาณไปที่กระบวนการอื่น ไม่ควรคงอยู่ผ่านการเริ่มระบบใหม่ กระบวนการที่รับข้อมูลและ process ID ไม่คงอยู่ผ่านการเริ่มระบบใหม่

ผู้สร้างอ็อบเจ็กต์ Error Notification มีหน้าที่เอาอ็อบเจ็กต์ Error Notification ออกในเวลาที่เหมาะสม ในเหตุการณ์ที่กระบวนการจบการทำงาน และล้มเหลวในการเอาอ็อบเจ็กต์ Error Notification ออก **en_persistenceflg** descriptor ประกันว่าอ็อบเจ็กต์ Error Notification ที่ไม่ใช้แล้วจะถูกเอาออก เมื่อเริ่มระบบใหม่

ค่า descriptor **en_persistenceflg** ที่ใช้ได้คือ:

0 ไม่คงอยู่ (ถูกเอาออกเมื่อบูตเครื่อง)

1 คงอยู่ (คงอยู่ผ่านการบูตเครื่อง)

`en_pid`

ระบุ process ID (PID) สำหรับใช้ในการระบุอ็อบเจ็กต์ Error Notification อ็อบเจ็กต์ที่มี PID ที่ระบุควรมี **en_persistenceflg** ตั้งค่าเป็น 0

`en_rclass`

ระบุคลาสของทรัพยากรที่ล้มเหลว สำหรับคลาสข้อผิดพลาด ฮาร์ดแวร์ คลาสทรัพยากรคือคลาสอุปกรณ์ คลาสข้อผิดพลาดของทรัพยากรใช้ไม่ได้กับคลาสข้อผิดพลาดซอฟต์แวร์

`en_resource`

ระบุชื่อของทรัพยากรที่ล้มเหลว สำหรับคลาสข้อผิดพลาด ฮาร์ดแวร์ ชื่อทรัพยากรคือชื่ออุปกรณ์

`en_rtype`

ระบุชนิดของทรัพยากรที่ล้มเหลว สำหรับคลาสข้อผิดพลาด ฮาร์ดแวร์ ชนิดทรัพยากรคือชนิดอุปกรณ์ซึ่งทรัพยากรเป็นที่รู้จักในคลาสอ็อบเจ็กต์อุปกรณ์

`en_symptom`

เปิดใช้งานการแจ้งเตือนของข้อผิดพลาดร่วมกับสตริงอาการ เมื่อตั้งค่าเป็น **TRUE**

`en_type`

ระบุความรุนแรงของรายการบันทึกข้อผิดพลาดที่จะจับคู่ ค่า descriptor**en_type** คือ:

INFO ใช้เป็นข้อมูล

PEND ใกล้จะสูญเสียความพร้อมใช้

PERM ถาวร

PERF การลดลงของประสิทธิภาพที่ยอมรับไม่ได้

TEMP ชั่วคราว

UNKN ไม่รู้จัก

ตัวอย่าง

1. เมื่อต้องการสร้างเมธอดการแจ้งเตือนที่ส่งเมล์รายการ ข้อผิดพลาดที่จัดรูปแบบไปที่ root แต่ละครั้งที่ข้อผิดพลาดดิสก์ชนิด PERM ถูกบันทึก สร้าง ไฟล์ชื่อ `/tmp/en_sample.add` มี อ็อบเจ็กต์ Error Notification ดังต่อไปนี้:

```
errnotify:
    en_name = "sample"
    en_persistenceflg = 0
    en_class = "H"
    en_type = "PERM"
    en_rclass = "disk"
    en_method = "errpt -a -l $1 | mail -s 'Disk Error' root"
```

เมื่อต้องการเพิ่ม อ็อบเจ็กต์ให้กับคลาสอ็อบเจ็กต์ Error Notification พิมพ์:

```
odmadd /tmp/en_sample.add
```

คำสั่ง **odmadd** เพิ่มอ็อบเจ็กต์ Error Notification ที่มีใน **/tmp/en_sample.add** ไปที่ไฟล์ **errnotify**

2. เมื่อต้องการตรวจสอบว่าอ็อบเจ็กต์ Error Notification ถูกเพิ่มให้กับคลาสอ็อบเจ็กต์ พิมพ์:

```
odmget -q"en_name='sample'" errnotify
```

คำสั่ง **odmget** ค้นหา อ็อบเจ็กต์ Error Notification ภายในไฟล์ **errnotify** ที่มีค่า **en_name** เป็น "sample" และแสดงอ็อบเจ็กต์ เอาต์พุตดังต่อไปนี้ถูกส่งกลับ:

```
errnotify:
    en_pid = 0
    en_name = "sample"
    en_persistenceflg = 0
    en_label = ""
    en_crcid = 0
    en_class = "H"
    en_type = "PERM"
    en_alertflg = ""
    en_resource = ""
    en_rtype = ""
    en_rclass = "disk"
    en_method = "errpt -a -l $1 | mail -s 'Disk Error' root"
```

3. เมื่อต้องการลบอ็อบเจ็กต์ Error Notification sample จากคลาสอ็อบเจ็กต์ Error Notification พิมพ์:

```
odmdelete -q"en_name='sample'" -o errnotify
```

คำสั่ง **odmdelete** ค้นหาอ็อบเจ็กต์ Error Notification ภายในไฟล์ **errnotify** มีค่า **en_name** เป็น "sample" และเอาออกจากคลาสอ็อบเจ็กต์ Error Notification

4. To send an email to root when a duplicate error occurs, create a file called **/tmp/en_sample.add** containing the following error notification stanza:

```
errnotify:
        en_name = "errdupxmp"
        en_persistenceflg = 1
        en_dup = "TRUE"
        en_method = "/usr/lib/dupmethod $1"
```

สร้างสคริปต์ **/usr/lib/dupmethod** ดังนี้:

```
#!/bin/sh
# email root when a duplicate error is logged.
# We currently don't clear the duplicate from the log.
#
# Input:
#   $1 contains the error log sequence number.
#
# Use errpt to generate the body of this email.
/usr/bin/errpt -al$1 | /usr/bin/mail -s "Duplicate Error Logged" root >/dev/null

# Now delete that error (currently not done)
#/usr/bin/errclear -l$1 0
exit $?
```

งานการบันทึกข้อผิดพลาด

ส่วนนี้อธิบายงาน error-logging และข้อมูล

งานการบันทึกข้อผิดพลาดและข้อมูลเพื่อช่วยคุณ ในการใช้หน่วยบริการการบันทึกข้อผิดพลาดรวมถึง:

- การอ่านรายงานความผิดพลาด
- ตัวอย่างของรายงานความผิดพลาดละเอียด
- ตัวอย่างของรายงานความผิดพลาดโดยสรุป
- การสร้างรายงานความผิดพลาด
- การหยุดบันทึกข้อผิดพลาด
- การล้างข้อมูลบันทึกข้อผิดพลาด
- การคัดลอกบันทึกข้อผิดพลาดไปที่ดิสเก็ตหรือเทป
- การใช้เซอร์วิส liberrlog

การอ่านรายงานข้อผิดพลาด

เพื่อรับรายงานของข้อผิดพลาด ทั้งหมดที่บันทึกใน 24 ชั่วโมงก่อนการล้มเหลว พิมพ์:

```
errpt -a -s mmddhhmmyy | pg
```

โดยที่ *mmddhhmmyy* หมายถึง เดือน วัน ชั่วโมง นาที และปี 24 ชั่วโมงก่อนการล้มเหลว

รายงาน error-log มีข้อมูลดังต่อไปนี้:

หมายเหตุ: ไม่ใช่ข้อผิดพลาดทั้งหมดที่สร้างข้อมูลสำหรับแต่ละประเภท following ดังต่อไปนี้

`LABEL`
ชื่อที่กำหนดไว้ล่วงหน้าสำหรับเหตุการณ์

`ID` ตัวระบุแบบตัวเลขสำหรับเหตุการณ์

`Date/Time`
วันที่และเวลาของเหตุการณ์

`Sequence Number`
หมายเลขเฉพาะสำหรับเหตุการณ์

`Machine ID`
หมายเลขระบุยูนิตโพรเซสเซอร์ระบบ

`Node ID`
ชื่อย่อยระบบของคุณ

`Class`
แหล่งข้อมูลทั่วไปของข้อผิดพลาด คลาสข้อผิดพลาดที่เป็นไปได้คือ:

H ฮาร์ดแวร์ (เมื่อคุณได้รับข้อผิดพลาดฮาร์ดแวร์ ให้อ้างอิงกับคู่มือปฏิบัติการ ระบบของคุณ เกี่ยวกับการวินิจฉัย

บนอุปกรณ์ที่มีปัญหาหรือ อุปกรณ์ชิ้นอื่น โปรแกรมวินิจฉัยทดสอบอุปกรณ์และวิเคราะห์ รายการบันทึกข้อผิดพลาดที่สัมพันธ์กับอุปกรณ์เพื่อกำหนดภาวะของอุปกรณ์)

S ซอฟต์แวร์

O ข้อความแสดงข้อมูล

U ไม่ได้กำหนด (ตัวอย่างเช่น เครือข่าย)

ชนิด

ความรุนแรงของข้อผิดพลาดที่เกิดขึ้น ประเภทข้อผิดพลาดดังต่อไปนี้ เกิดขึ้นได้:

PEND การสูญเสียความพร้อมใช้ของอุปกรณ์หรือส่วนประกอบใกล้จะเสียหาย

PERF ประสิทธิภาพของอุปกรณ์หรือส่วนประกอบลดลงต่ำกว่าระดับ ที่ยอมรับได้

PERM เงื่อนไขที่ไม่สามารถทำการกู้คืนได้ ประเภทข้อผิดพลาดที่มีค่านี้โดยปกติเป็นข้อผิดพลาดร้ายแรงที่สุด และมักจะหมายความว่าคุณมี อุปกรณ์ฮาร์ดแวร์หรือโมดูลซอฟต์แวร์ที่ชำรุด ประเภทข้อผิดพลาดที่ไม่ใช่ PERM ปกติไม่ได้บ่งชี้ถึงการชำรุด แต่ข้อมูล ถูกบันทึกเพื่อที่จะสามารถถูกวิเคราะห์โดยโปรแกรมวินิจฉัย

TEMP สภาพที่ถูกกู้คืนจากจำนวนความพยายามที่ไม่สำเร็จ ประเภทข้อผิดพลาดนี้ยังถูกใช้เพื่อบันทึกรายการเชิงข้อมูล เช่นสถิติ การถ่ายโอนข้อมูลสำหรับอุปกรณ์ DASD

UNKN ไม่สามารถที่จะกำหนดความรุนแรงของข้อผิดพลาดได้

INFO รายการบันทึกข้อผิดพลาดเป็นเพียงข้อมูลและไม่ใช่ผลจากข้อผิดพลาด

`Resource name`

ชื่อของทรัพยากรที่ตรวจพบข้อผิดพลาด สำหรับข้อผิดพลาดซอฟต์แวร์ นี่เป็นชื่อของคอมโพเนนต์ซอฟต์แวร์หรือโปรแกรมเรียกทำงาน สำหรับข้อผิดพลาดฮาร์ดแวร์ นี่เป็นชื่อของอุปกรณ์หรือคอมโพเนนต์ระบบ ซึ่งไม่ได้บ่งชี้ว่า คอมโพเนนต์มีข้อผิดพลาดหรือจำเป็นต้องเปลี่ยน แต่ ถูกใช้เพื่อ กำหนดโมดูลการวินิจฉัยที่เหมาะสมที่จะถูกใช้เพื่อวิเคราะห์ข้อผิดพลาด

คลาสรีซอร์ส

คลาสทั่วไปของทรัพยากรที่ตรวจพบความล้มเหลว (ตัวอย่างเช่น คลาสอุปกรณ์ `disk`)

ชนิดรีซอร์ส

ชนิดของทรัพยากรที่ตรวจพบความล้มเหลว (ตัวอย่างเช่น ชนิดอุปกรณ์ `355mb`)

`Location code`

พาธไปยังอุปกรณ์ อาจมีข้อมูลได้มากถึงสี่ฟิลด์ ซึ่งอ้างถึง drawer, slot, connector และ port ตามลำดับ

`VPD`

ข้อมูลผลิตภัณฑ์สำคัญ เนื้อหาของฟิลด์นี้ถ้ามีจะหลากหลาย รายการ บันทึกข้อผิดพลาดสำหรับอุปกรณ์โดยปกติส่งกลับข้อมูลเกี่ยวกับ ผู้ผลิต อุปกรณ์ หมายเลขรหัสอุปกรณ์ ระดับ Engineering Change และระดับ Read Only Storage

คำอธิบาย

ข้อมูลสรุปของข้อผิดพลาด

สาเหตุที่อาจเป็นได้

รายการของแหล่งข้อมูลที่เป็นไปได้ของข้อผิดพลาด

สาเหตุจากผู้ใช้

รายการของสาเหตุที่เป็นไปได้สำหรับข้อผิดพลาดเนื่องจากความผิดพลาดของผู้ใช้ การใส่ดิสก์ไม่ถูกต้องและอุปกรณ์ภายนอก (เช่นโมเด็มและเครื่องพิมพ์) ที่ไม่ได้เปิด เป็นตัวอย่างของข้อผิดพลาดที่เกิดจากผู้ใช้

การดำเนินการที่แนะนำ

คำอธิบายของการดำเนินการเพื่อแก้ไขข้อผิดพลาดที่เกิดจากผู้ใช้

`Install causes`

รายการของสาเหตุที่เป็นไปได้สำหรับข้อผิดพลาดเนื่องจากขั้นตอนการติดตั้ง หรือการกำหนดค่าที่ไม่ถูกต้อง ตัวอย่างของข้อผิดพลาดประเภทนี้ รวมถึงฮาร์ดแวร์และ ซอฟต์แวร์ไม่ตรงกัน การติดตั้งสายเคเบิลไม่ถูกต้องหรือการเชื่อมต่อสายเคเบิล หลวม และระบบที่กำหนดค่าไม่ถูกต้อง

การดำเนินการที่แนะนำ

คำอธิบายของการดำเนินการเพื่อแก้ไขข้อผิดพลาดที่เกิดจากการติดตั้ง

สาเหตุความล้มเหลว

รายการของความบกพร่องที่เป็นไปได้ในฮาร์ดแวร์หรือซอฟต์แวร์

หมายเหตุ: ส่วน failure causes ในบันทึกข้อผิดพลาดซอฟต์แวร์โดยปกติจะบ่งชี้ความบกพร่อง ของซอฟต์แวร์ บันทึกที่แสดงสาเหตุจาก ผู้ใช้หรือการติดตั้ง หรือทั้งสองอย่าง แต่ไม่ใช่ failure causes โดยปกติบ่งชี้ว่าปัญหาไม่ใช่ความบกพร่องของซอฟต์แวร์

ถ้าคุณสงสัยความบกพร่องของซอฟต์แวร์ หรือไม่สามารถสาเหตุจาก ผู้ใช้หรือการติดตั้ง ให้รายงานปัญหากับแผนกบริการซอฟต์แวร์ของคุณ

การดำเนินการที่แนะนำ

คำอธิบายของการดำเนินการสำหรับการแก้ไขความล้มเหลว สำหรับข้อผิดพลาดฮาร์ดแวร์ `PERFORM PROBLEM DETERMINATION PROCEDURES` เป็นหนึ่งใน การดำเนินการที่แนะนำ สำหรับข้อผิดพลาดฮาร์ดแวร์ นี่จะนำไปสู่การรันโปรแกรมวินิจฉัย

ข้อมูลรายละเอียด

- ข้อมูลความล้มเหลวที่เป็นค่าเฉพาะสำหรับแต่ละรายการบันทึกข้อผิดพลาด เช่นข้อมูล device sense
- ข้อมูลในไดเร็กทอรีทำงานปัจจุบันของกระบวนการ เช่น FILE SYSTEM SERIAL NUMBER และ INODE NUMBER เมื่อกระบวนการดัมพ์คอร์

เมื่อต้องการแสดงแบบย่อของรายงานละเอียดที่สร้างโดยแฟล็ก *–a* ให้ใช้แฟล็ก *–A* แฟล็ก *–A* ใช้ไม่ได้ กับแฟล็ก *–a*, *–g* หรือ *–t* รายการที่รายงาน เมื่อคุณใช้ *–A* เพื่อสร้างแบบย่อของรายงานคือ:

- Label
- Date and time
- ชนิด
- ชื่อรีซอร์ส
- คำอธิบาย
- Detail data

เอาต์พุตตัวอย่างของแฟล็กนี้อยู่ในรูปแบบดังต่อไปนี้:

```
LABEL:           STOK_RCVRY_EXIT
Date/Time:       Tue Dec 14 15:25:33
Type:            TEMP Resource Name:   tok0
Description PROBLEM RESOLVED
Detail Data FILE NAME line: 273 file: stok_wdt.c
SENSE DATA
0000 0000 0000 0000 0000 0000 DEVICE ADDRESS 0004 AC62 25F1
```

การรายงานสามารถถูกปิดสำหรับบางข้อผิดพลาด เมื่อต้องการแสดง ข้อผิดพลาดที่มีการปิดการรายงาน ให้พิมพ์:

```
errpt -t -F report=0 | pg
```

ถ้าการรายงานถูกปิดสำหรับข้อผิดพลาดทั้งหมด เปิดใช้งาน การรายงานของข้อผิดพลาดทั้งหมดโดยใช้คำสั่ง **errupdate**

การบันทึกสามารถถูกปิดได้สำหรับบางข้อผิดพลาด เมื่อต้องการแสดงข้อผิดพลาดที่มีการปิดการบันทึก ให้พิมพ์:

```
errpt -t -F log=0 | pg
```

ถ้าการบันทึกถูกปิดสำหรับข้อผิดพลาดทั้งหมด เปิดใช้งาน การบันทึกสำหรับข้อผิดพลาดทั้งหมดโดยใช้คำสั่ง **errupdate** การบันทึกข้อผิดพลาดทั้งหมดมีประโยชน์ ถ้าหากจำเป็นต้อง สร้างข้อผิดพลาดของระบบใหม่

ตัวอย่างของรายงานข้อผิดพลาดแบบละเอียด

ต่อไปนี้ เป็นรายการรายงานความผิดพลาดตัวอย่างที่ถูกสร้างโดยเรียกคำสั่ง **errpt -a**

ค่าของ error-class ของ **H** และ ค่า error-type ของ **PERM** บ่งชี้ว่าระบบ พบปัญหาฮาร์ดแวร์ (ตัวอย่างเช่น กับไดรเวอร์อุปกรณ์อะแด็ปเตอร์ SCSI) และไม่สามารถกู้คืนได้ ข้อมูลวินิจฉัยอาจถูกนำมาเชื่อมโยงกับ ประเภทข้อผิดพลาดนี้ ถ้าเป็นดังนั้น ข้อมูลจะแสดงที่จุดสิ้นสุดของรายการข้อผิดพลาด ตามที่แสดงในตั้งอย่างของปัญหาที่พบกับไดรเวอร์อุปกรณ์ ดังต่อไปนี้:

```
LABEL:      SCSI_ERR1
ID:         0502F666

Date/Time:        Jun 19 22:29:51
Sequence Number:  95
Machine ID:       123456789012
Node ID:          host1
Class:            H
Type:             PERM
Resource Name:    scsi0
Resource Class:   adapter
Resource Type:    hscsi
Location:         00-08
VPD:
     Device Driver Level.........00
     Diagnostic Level............00
     Displayable Message.........SCSI
     EC Level....................C25928
     FRU Number..................30F8834
     Manufacturer................IBM97F
     Part Number.................59F4566
     Serial Number...............00002849
     ROS Level and ID............24
     Read/Write Register Ptr.....0120
```

```
Description
ADAPTER ERROR

Probable Causes
ADAPTER HARDWARE CABLE
CABLE TERMINATOR DEVICE

Failure Causes
ADAPTER
CABLE LOOSE OR DEFECTIVE

          Recommended Actions
          PERFORM PROBLEM DETERMINATION PROCEDURES
          CHECK CABLE AND ITS CONNECTIONS

Detail Data
SENSE DATA
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

Diagnostic Log sequence number:  153
Resource Tested:        scsi0
Resource Description:   SCSI I/O Controller
Location:               00-08
SRN:                    889-191
Description:            Error log analysis indicates hardware failure.
Probable FRUs:
    SCSI Bus        FRU: n/a            00-08
                    Fan Assembly
    SCSI2           FRU: 30F8834        00-08
                    SCSI I/O Controller
```

ค่า error-class **ของ H และ ค่า** error-type **ของ PEND บ่งชี้ว่าชิ้นส่วนของ ฮาร์ดแวร์** (Token Ring) **อาจไม่พร้อมใช้งานเนื่องจากตรวจพบข้อผิดพลาด จำนวนมากโดยระบบ**

```
LABEL:    TOK_ESERR
ID:       AF1621E8

Date/Time:       Jun 20 11:28:11
Sequence Number: 17262
Machine Id:      123456789012
Node Id:         host1
Class:           H
Type:            PEND
Resource Name:   TokenRing
Resource Class:  tok0
Resource Type:   Adapter
Location:        TokenRing

รายละเอียด
EXCESSIVE TOKEN-RING ERRORS

Probable Causes
TOKEN-RING FAULT DOMAIN

Failure Causes
```

```
TOKEN-RING FAULT DOMAIN

        Recommended Actions
        REVIEW LINK CONFIGURATION DETAIL DATA
        CONTACT TOKEN-RING ADMINISTRATOR RESPONSIBLE FOR THIS LAN

Detail Data
SENSE DATA
0ACA 0032 A440 0001 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 2080 0000 0000 0010 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 78CC 0000 0000 0005 C88F 0304 F4E0 0000 1000 5A4F 5685
1000 5A4F 5685 3030 3030 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000
```

ค่า error-class **ของ S และ ค่า** error-type **ของ PERM บ่งชี้ว่าระบบพบ ปัญหากับซอฟต์แวร์และไม่สามารถกู้คืนได้**

```
LABEL:    DSI_PROC
ID:       20FAED7F

Date/Time:       Jun 28 23:40:14
Sequence Number: 20136
Machine Id:      123456789012
Node Id:         123456789012
Class:           S
Type:            PERM
Resource Name:   SYSVMM

รายละเอียด
Data Storage Interrupt, Processor

Probable Causes
SOFTWARE PROGRAM

Failure Causes
SOFTWARE PROGRAM

        Recommended Actions
        IF PROBLEM PERSISTS THEN DO THE FOLLOWING
        CONTACT APPROPRIATE SERVICE REPRESENTATIVE

Detail Data
Data Storage Interrupt Status Register
4000 0000
Data Storage Interrupt Address Register
0000 9112
Segment Register, SEGREG
D000 1018
EXVAL
0000 0005
```

ค่า error-class **ของ S และ ค่า** error-type **ของ TEMP บ่งชี้ว่าระบบพบ ปัญหากับซอฟต์แวร์ หลังจากความพยายามหลายครั้ง ระบบสามารถ ทำการแก้ปัญหาได้**

```
LABEL:          SCSI_ERR6
ID:             52DB7218

Date/Time:       Jun 28 23:21:11
Sequence Number: 20114
Machine Id:      123456789012
Node Id:         host1
Class:           S
Type:            INFO
Resource Name:   scsi0

รายละเอียด
SOFTWARE PROGRAM ERROR

Probable Causes
SOFTWARE PROGRAM

Failure Causes
SOFTWARE PROGRAM

        Recommended Actions
        IF PROBLEM PERSISTS THEN DO THE FOLLOWING
        CONTACT APPROPRIATE SERVICE REPRESENTATIVE

Detail Data
SENSE DATA
0000 0000 0000 0000 0000 0011 0000 0008 000E 0900 0000 0000 FFFF
FFFE 4000 1C1F 01A9 09C4 0000 000F 0000 0000 0000 0000 FFFF FFFF
0325 0018 0040 1500 0000 0000 0000 0000 0000 0000 0000 0000 0800
0000 0100 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000
```

ค่าคลาสข้อผิดพลาด O หมายถึงว่า ข้อความแจ้งให้ทราบได้ถูกบันทึก

```
LABEL:     OPMSG
ID:        AA8AB241

Date/Time:       Jul 16 03:02:02
Sequence Number: 26042
Machine Id:      123456789012
Node Id:         host1
Class:           O
Type:            INFO
Resource Name:   OPERATOR

รายละเอียด
OPERATOR NOTIFICATION

User Causes
errlogger COMMAND

        Recommended Actions
        REVIEW DETAILED DATA
```

```
Detail Data
MESSAGE FROM errlogger COMMAND
hdisk1 : Error log analysis indicates a hardware failure.
```

ตัวอย่างรายงานข้อผิดพลาดแบบสรุป

ต่อไปนี้เป็น ตัวอย่างของรายงานความผิดพลาดโดยสรุปที่สร้างโดยใช้คำสั่ง **errpt** หนึ่งบรรทัดของข้อมูลถูกส่งกลับสำหรับแต่ละรายการข้อผิดพลาด

```
ERROR_
IDENTIFIER TIMESTAMP  T CL RESOURCE_NAME ERROR_DESCRIPTION
192AC071   0101000070 I O  errdemon      Error logging turned off
0E017ED1   0405131090 P H  mem2          Memory failure
9DBCFDEE   0101000070 I O  errdemon      Error logging turned on
038F2580   0405131090 U H  scdisk0       UNDETERMINED ERROR
AA8AB241   0405130990 I O  OPERATOR      OPERATOR NOTIFICATION
```

การสร้างรายงานข้อผิดพลาด

เมื่อต้องการสร้างรายงานความผิดพลาดของซอฟต์แวร์หรือ ฮาร์ดแวร์ให้ทำดังต่อไปนี้:

1. ระบุว่าการบันทึกข้อผิดพลาดเปิดหรือปิดทำงานอยู่โดยดูว่าบันทึกข้อผิดพลาด มีรายการ:

   ```
   errpt -a
   ```

   คำสั่ง **errpt** สร้างรายงานความผิดพลาดจากรายการ ในบันทึกข้อผิดพลาดของระบบ
   ถ้าบันทึกข้อผิดพลาดไม่มี รายการ, การบันทึกข้อผิดพลาดถูกปิด เปิดใช้งานหน่วยบริการ โดยพิมพ์:

   ```
   /usr/lib/errdemon
   ```

   หมายเหตุ: คุณต้องมีการเข้าถึง root user เพื่อรันคำสั่งนี้

   errdemon daemon เริ่ม การบันทึกข้อผิดพลาดและเขียนรายการบันทึกข้อผิดพลาดในบันทึกข้อผิดพลาดของระบบ ถ้า daemon ไม่ได้รันอยู่ ข้อผิดพลาดจะไม่ถูกบันทึก
2. สร้างรายงานบันทึกข้อผิดพลาดโดยใช้คำสั่ง **errpt** ตัวอย่างเช่น เพื่อดูข้อผิดพลาดทั้งหมดสำหรับดิสก์ไดร์ฟ `hdisk1` พิมพ์:

   ```
   errpt -N hdisk1
   ```
3. สร้างรายงานบันทึกข้อผิดพลาดโดยใช้ SMIT ตัวอย่างเช่นใช้คำสั่ง **smit errpt**:

   ```
   smit errpt
   ```

   a. เลือก **1** เพื่อส่งรายงานความผิดพลาดไปที่เอาต์พุตมาตรฐาน หรือเลือก **2** เพื่อส่งรายงานไปที่เครื่องพิมพ์

   b. เลือก **yes** เพื่อแสดงหรือพิมพ์รายการบันทึกข้อผิดพลาด ตามที่เกิดขึ้น หรือเลือก **no**

   c. ระบะชื่ออุปกรณ์ที่เหมาะสมในตัวเลือก **Select resource names** (เช่น `hdisk1`)

   d. เลือก **Do**

การหยุดบันทึกข้อผิดพลาด

ขั้นตอนนี้ อธิบายวิธีหยุดหน่วยบริการ error-logging

เมื่อต้องการปิดการบันทึกข้อผิดพลาด ใช้คำสั่ง **errstop** คุณต้องมีสิทธิ root user เพื่อใช้คำสั่งนี้

โดยปกติ คุณจะไม่ต้องการปิดหน่วยบริการ error-logging แต่ คุณควรล้างข้อมูลบันทึกข้อผิดพลาดที่มีรายการเก่าแล้วหรือรายการที่ไม่จำเป็น

ปิดหน่วยบริการ error-logging เมื่อคุณกำลังติดตั้ง หรือทดลองซอฟต์แวร์หรือฮาร์ดแวร์ใหม่ วิธีนี้ daemon การบันทึกข้อผิดพลาด ไม่ใช้เวลา CPU เพื่อบันทึกปัญหาที่คุณทำให้เกิดขึ้น

การล้างบันทึกข้อผิดพลาด

การล้างข้อมูล Error-log โดยปกติ กระทำให้กับคุณเป็นส่วนหนึ่งของคำสั่ง **cron** รายวัน ถ้าไม่ทำโดยอัตโนมัติ ล้างข้อมูลบันทึกข้อผิดพลาดด้วยตัว คุณเองทุกสองวัน หลังจากคุณได้ตรวจสอบเนื้อหาเพื่อให้แน่ใจว่า ไม่มีข้อผิดพลาดที่สำคัญ

คุณยังสามารถล้างข้อมูลข้อผิดพลาดจำเพาะได้ด้วย ตัวอย่างเช่น ถ้าคุณได้รับดิสก์ใหม่ และคุณไม่ต้องการให้ข้อผิดพลาดของดิสก์เก่าอยู่ในบันทึก คุณสามารถล้างข้อมูลของข้อผิดพลาดของดิสก์เก่าเท่านั้น

ลบรายการทั้งหมดในบันทึกข้อผิดพลาดของคุณโดยทำ อย่างใดอย่างหนึ่งดังต่อไปนี้:

- ใช้คำสั่ง **errclear -d** ตัวอย่างเช่น เพื่อลบ ข้อผิดพลาดซอฟต์แวร์ทั้งหมด พิมพ์:

  ```
  errclear -d S 0
  ```

  คำสั่ง **errclear** ลบรายการจากบันทึกข้อผิดพลาดที่ เก่ากว่าจำนวนวันที่ระบุ 0 ใน ตัวอย่างก่อนหน้าบ่งชี้ว่าคุณต้องการลบรายการสำหรับทุกวัน
- ใช้คำสั่ง **smit errclear**:

  ```
  smit errclear
  ```

การคัดลอกบันทึกข้อผิดพลาดลงในดิสเก็ตหรือเทป

คัดลอกบันทึกข้อผิดพลาดโดยทำ ตามหนึ่งในวิธีดังต่อไปนี้:

- เมื่อต้องการคัดลอกบันทึกข้อผิดพลาด ไปที่ดิสเก็ต ใช้คำสั่ง **ls** และ **backup** แทรกดิสเก็ต ที่ฟอร์แมตแล้วในดิสก์ไดร์ฟแล้วพิมพ์:

  ```
  ls /var/adm/ras/errlog | backup -ivp
  ```
- เมื่อต้องการคัดลอกบันทึกข้อผิดพลาดไปที่เทป ให้ใส่เทปในไดร์ฟแล้วพิมพ์:

  ```
  ls /var/adm/ras/errlog | backup -ivpf/dev/rmt0
  ```
- เพื่อรวบรวมข้อมูลการกำหนดค่าระบบในไฟล์ **tar** และคัดลอกไปที่ดิสเก็ต ใช้คำสั่ง **snap** แทรกดิสเก็ต ที่ฟอร์แมตแล้วในดิสก์ไดร์ฟแล้วพิมพ์:

  ```
  snap -a -o /dev/rfd0
  ```

  หมายเหตุ: เมื่อต้องการใช้คำสั่ง **snap** คุณต้องมีสิทธิ root user

  คำสั่ง **snap** ในตัวอย่างนี้ใช้แฟล็ก **-a** เพื่อรวบรวมข้อมูลทั้งหมดเกี่ยวกับการกำหนดค่าระบบของคุณ แฟล็ก **-o** คัดลอกไฟล์ **tar** ที่บีบอัดลงในอุปกรณ์ที่คุณกำหนดชื่อ /dev/rfd0 กำหนดชื่อดิสก์ไดร์ฟของคุณ

  เพื่อรวบรวมข้อมูลการกำหนดค่า ทั้งหมดในไฟล์ **tar** และคัดลอกไปที่เทป พิมพ์:

  ```
  snap -a -o /dev/rmt0
  ```

  /dev/rmt0 กำหนดชื่อเทปไดร์ฟของคุณ

การใช้เซอร์วิส liberrlog

เซอร์วิส **liberrlog** อนุญาตให้คุณอ่านรายการ จากบันทึกข้อผิดพลาด และจัดเตรียมความสามารถในการอัพเดตที่จำกัด ซึ่งมีประโยชน์มาก จากเมธอดการแจ้งเตือนข้อผิดพลาดที่เขียนในภาษาโปรแกรม C แทนการใช้เชลล์สคริปต์ การเข้าถึงบันทึกข้อผิดพลาดโดยใช้ฟังก์ชัน **liberrlog** มีประสิทธิภาพมากกว่าการใช้คำสั่ง **errpt**

ข้อมูลที่เกี่ยวข้อง:

error_open

errorlog_close

errlog_find, errlog_error_sequence,errlog_find_next

errlog_set_direction

errlog_write

การบันทึกข้อผิดพลาดและการเตือน

ส่วนนี้อธิบายกระบวนการของการบันทึกข้อผิดพลาดและการรับ การเตือน

ถ้าฟิลด์ **Alert** ของ แม่แบบการบันทึกข้อผิดพลาดถูกตั้งค่าเป็น `True` โปรแกรม ที่ประมวลผลการแจ้งเตือนใช้ฟิลด์ดังต่อไปนี้ในบันทึกข้อผิดพลาดเพื่อสร้าง การแจ้งเตือน:

- **Class**
- **ชนิด**
- **คำอธิบาย**
- **Probable Cause**
- **สาเหตุจากผู้ใช้**
- **สาเหตุจากการติดตั้ง**
- **สาเหตุจากความล้มเหลว**
- **การดำเนินการที่แนะนำ**
- **ข้อมูลรายละเอียด**

ฟิลด์แม่แบบเหล่านี้ต้องถูกตั้งค่าตาม SNA Generic Alert Architecture ที่อธิบายใน *SNA Formats*, หมายเลขออร์เดอร์ GA27-3136 คุณสามารถดูหนังสือได้ที่ http://publib.boulder.ibm.com/cgi-bin/bookmgr/BOOKS/D50A5007 การแจ้งเตือนที่ไม่ได้ถูกตั้งค่าตามสถาปัตยกรรมไม่สามารถถูก ประมวลผลอย่างถูกต้องโดยโปรแกรมที่ได้รับ เช่น NetView®

ข้อความที่เพิ่มให้กับชุดข้อความ error-logging ต้องไม่ขัดแย้งกับ SNA Generic Alert Architecture เมื่อคำสั่ง **errmsg** ถูกใช้เพื่อเพิ่มข้อความ คำสั่งเลือก หมายเลขข้อความที่ไม่ขัดแย้งกับสถาปัตยกรรม

ถ้าฟิลด์ **Alert** ของแม่แบบการบันทึก ข้อผิดพลาดถูกตั้งค่าเป็น `False` คุณสามารถใช้ ฟิลด์ใดๆ ในแค็ตตาล็อกข้อความ error-logging

การควบคุมการบันทึกข้อผิดพลาด

เพื่อควบคุมหน่วยบริการ error-logging คุณสามารถใช้คำสั่ง error-logging รูทีนย่อยและเซอร์วิสเคอร์เนล เช่นเดียวกับไฟล์

คำสั่ง Error-logging

`errclear`
ลบรายการจากบันทึกข้อผิดพลาด คำสั่งนี้สามารถลบรายการ บันทึกข้อผิดพลาด ลบรายการที่มีหมายเลข error ID คลาส หรือชนิด ที่ระบุ

`errdead`
แยกข้อผิดพลาดที่มีในบัฟเฟอร์ **/dev/error** ที่ดักจับในดัมพ์ระบบ ดัมพ์ระบบจะมีบันทึกข้อผิดพลาดถ้า **errdemon** daemon ไม่แอ็คทีฟก่อนดัมพ์

`errdemon`
อ่านบันทึกข้อผิดพลาดจากไฟล์ **/dev/error** และเขียนรายการบันทึกข้อผิดพลาดลงในบันทึกข้อผิดพลาดระบบ **errdemon** ยังดำเนินการแจ้งเตือนข้อผิดพลาดตามที่ระบุในอ็อบเจ็กต์การแจ้งเตือน ข้อผิดพลาดใน Object Data Manager (ODM) daemon นี้เริ่มต้นโดยอัตโนมัติ ระหว่างการกำหนดค่าเริ่มต้นระบบ

`errinstall`
สามารถถูกใช้เพื่อเพิ่มหรือแทนที่ข้อความในแค็ตตาล็อกข้อความแสดงข้อผิดพลาด ที่จัดเตรียมสำหรับใช้โดยโพรซีเดอร์การติดตั้งซอฟต์แวร์ ระบบสร้างไฟล์สำรองข้อมูล ชื่อ *File*.**undo** ไฟล์ **undo** อนุญาตให้คุณยกเลิกการเปลี่ยนแปลงที่ได้ทำโดยเรียกคำสั่ง **errinstall**

`errlogger`
เขียนรายการข้อความตัวดำเนินการลงในบันทึกข้อผิดพลาด

`errmsg`
นำการบันทึกข้อผิดพลาดไปใช้ในแอ็พพลิเคชัน in-house รายการคำสั่ง **errmsg** เพิ่ม หรือลบข้อความที่เก็บในแค็ตตาล็อก ข้อความแสดงข้อผิดพลาด การใช้คำสั่งนี้ ข้อความสามารถถูกเพิ่มให้กับชุดข้อความ `Error Description`, `Probable Cause`, `User Cause`, `Install Cause`, `Failure Cause`, `Recommended Action` และ `Detailed Data`

`errpt`
สร้างรายงานความผิดพลาดจากรายการในบันทึกข้อผิดพลาดระบบ รายงาน สามารถถูกจัดรูปแบบเป็นบรรทัดเดียวของข้อมูลสำหรับแต่ละรายการ หรือรายงานสามารถเป็นรายการละเอียดของข้อมูลที่สัมพันธ์กับ แต่ละรายการในบันทึกข้อผิดพลาด รายการของคลาสและชนิดต่างๆ สามารถ ถูกละเว้นหรือรวมไว้ในรายงาน

`errstop`
หยุด **errdemon** daemon ซึ่งถูกกำหนดค่า ระหว่างการกำหนดค่าเริ่มต้นของระบบ การรันคำสั่ง **errstop** ปิดการใช้งานบางฟังก์ชันการวินิจฉัยและกู้คืนของระบบเช่นกัน

`errupdate`
เพิ่มหรือลบแม่แบบใน Error Record Template Repository ปรับเปลี่ยนแอ็ตทริบิวต์ Alert, Log และ Report ของแม่แบบข้อผิดพลาด ที่จัดเตรียมสำหรับใช้โดยโพรซีเดอร์การติดตั้งซอฟต์แวร์

รูทีนย่อยการบันทึกข้อผิดพลาดและเคอร์เนลเซอร์วิส

`errlog`
เขียนข้อผิดพลาดไปที่ไดรเวอร์อุปกรณ์บันทึกข้อผิดพลาด

`errsave และ errlast`
อนุญาตให้เคอร์เนลและส่วนขยายเคอร์เนลเขียนข้อมูลลงในบันทึกข้อผิดพลาด

`errlog_open`
เปิดบันทึกข้อผิดพลาด

`errlog_close`
ปิดบันทึกข้อผิดพลาด

`errlog_find_first`
ค้นหาการเกิดขึ้นครั้งแรกของรายการบันทึกข้อผิดพลาด

`errlog_find_next`
ค้นหาการเกิดขึ้นครั้งถัดไปของรายการบันทึกข้อผิดพลาด

`errlog_find_sequence`
ค้นหารายการบันทึกข้อผิดพลาดตามหมายเลขลำดับที่ ระบุ

`errlog_set_direction`
ตั้งค่าทิศทางสำหรับฟังก์ชันการค้นหาบันทึกข้อผิดพลาด

`errlog_write`
อัพเดตรายการบันทึกข้อผิดพลาด

`errresume`
ดำเนินบันทึกข้อผิดพลาดต่อหลังจากคำสั่ง **errlast** ถูกส่ง

ไฟล์การบันทึกข้อผิดพลาด

`/dev/error`
จัดเตรียมอินเตอร์เฟสไดรเวอร์อุปกรณ์มาตรฐานที่ต้องการโดยคอมโพเนนต์ บันทึกข้อผิดพลาด

`/dev/errorctl`
จัดเตรียมอินเตอร์เฟสไดรเวอร์อุปกรณ์ที่ไม่เป็นแบบมาตรฐานสำหรับการควบคุมระบบ การบันทึกข้อผิดพลาด

`/usr/include/sys/err_rec.h`
มีโครงสร้างที่กำหนดเป็นอาร์กิวเมนต์ไปที่ **เซอร์วิสเคอร์เนล errsave** และรูทีนย่อย **errlog**

`/usr/include/sys/errlog.h`
กำหนดอินเตอร์เฟสให้กับรูทีนย่อย **liberrlog**

`/var/adm/ras/errlog`
เก็บตัวอย่างของข้อผิดพลาดและความล้มเหลวที่พบโดยระบบ

`/var/adm/ras/errtmplt`
มี Error Record Template Repository

ระบบไฟล์และโลจิคัลวอลุ่ม

ไฟล์เป็นอาร์เรย์ไบต์หนึ่งมิติที่สามารถบรรจุ ข้อมูล ASCII หรือไบนารี

ไฟล์ AIX สามารถบรรจุข้อมูล เชลล์สคริปต์ และโปรแกรม ชื่อไฟล์ยังถูกใช้เพื่อแสดงอ็อบเจ็กต์ abstract เช่นซ็อกเก็ตหรือไดรเวอร์อุปกรณ์

ไฟล์ถูกแสดงภายในโดยโหนดดัชนี (*i-nodes*) ภายใน journaled file system (JFS), i-node เป็นโครงสร้างที่มีข้อมูล การเข้าถึง การประทับเวลา ความเป็นเจ้าของ และข้อมูลตำแหน่ง ทั้งหมดสำหรับแต่ละไฟล์ i-node มีขนาด 128-ไบต์ ใน JFS และ 512-ไบต์ใน enhanced journaled file system (JFS2) ตัวชี้ภายในโครงสร้าง i-node กำหนดแอดเดรสดิสก์จริงของบล็อกข้อมูลที่สัมพันธ์กับไฟล์ i-node ถูกระบุโดยหมายเลขออฟเซ็ต (*i-number*) และไม่มีข้อมูลชื่อไฟล์ การเชื่อมต่อของ i-numbers และชื่อไฟล์ เรียกว่า *link*

ชื่อไฟล์มีอยู่เฉพาะในไดเร็กทอรี ไดเร็กทอรีเป็น ชนิดเฉพาะของไฟล์ที่ให้โครงสร้างลำดับชั้นกับระบบไฟล์ ไดเร็กทอรีมีรายการรายการ แต่ละรายการไดเร็กทอรีมีชื่อไฟล์ และ i-number

JFS และ JFS2 ได้รับการสนับสนุนโดยระบบปฏิบัติการนี้ ระบบไฟล์เชื่อมโยงข้อมูลไฟล์และไดเร็กทอรีกับโครงสร้างที่ใช้โดยพื้นที่จัดเก็บข้อมูลและกลไกการเรียกข้อมูล

ข้อมูลที่เกี่ยวข้อง:

ls

mkfs

pr

fullstat.h

stat

statfs

ชนิดไฟล์

ไฟล์คืออาร์เรย์ของไบต์แบบ หนึ่งมิติที่มีฮาร์ดลิงก์อย่างน้อยหนึ่งลิงก์ (ชื่อไฟล์) ไฟล์สามารถมีข้อมูล ASCII หรือไบนารี

ไฟล์เก็บข้อมูล เชลล์สคริปต์ หรือโปรแกรม ชื่อไฟล์ ยังถูกใช้แทนอ็อบเจ็กต์เชิงนามธรรม เช่น ซ็อกเก็ต ไพพ์ และไดรเวอร์อุปกรณ์

เคอร์เนลไม่ได้จำแนกความแตกต่างของขอบเขตเร็กคอร์ดใน ไฟล์ทั่วไป ดังนั้นโปรแกรมสามารถสร้างตัวทำเครื่องหมายขอบเขตของตนเอง

ไฟล์ถูกแสดงใน journaled file system (JFS และ JFS2) โดยโหนดดัชนีดิสก์ (i-node) ข้อมูล เกี่ยวกับไฟล์ (เช่น ความเป็นเจ้าของ โหมดการเข้าถึง เวลาที่เข้าถึง แอดเดรสข้อมูล และเวลาที่ทำการแก้ไข) ถูกเก็บใน i-node

journaled file system สนับสนุนประเภทไฟล์ ต่อไปนี้:

ชนิดไฟล์ที่สนับสนุนระบบไฟล์เจอร์นัล

ชนิดของไฟล์	ชื่อแมโครที่ใช้ใน mode.h	คำอธิบาย
ทั่วไป	**S_ISREG**	ลำดับของไบต์ที่มีอย่างน้อยหนึ่งชื่อ ไฟล์ทั่วไปสามารถมี ข้อมูล ASCII หรือไบนารี ไฟล์เหล่านี้สามารถเข้าถึงแบบสุ่ม (อ่าน หรือเขียน) จากไบต์ใดๆ ในไฟล์
ไดเร็กทอรี	**S_ISDIR**	มีรายการไดเร็กทอรี (คู่ชื่อไฟล์ และ i-number) รูปแบบ ไดเร็กทอรีถูกกำหนดโดยระบบไฟล์ กระบวนการจะอ่านไดเร็กทอรีเหมือนกับที่ อ่านไฟล์ปกติ แต่เคอร์เนลสงวนสิทธิในการเขียนลงในไดเร็กทอรี ชุดของรูทีนย่อยพิเศษจะควบคุมรายการไดเร็กทอรี
บล็อกพิเศษ	**S_ISBLK**	เชื่อมโยงไดรเวอร์อุปกรณ์ที่จัดโครงสร้างกับชื่อไฟล์
อักขระพิเศษ	**S_ISCHR**	เชื่อมโยงไดรเวอร์อุปกรณ์ที่ไม่จัดโครงสร้างกับชื่อไฟล์
ไพพ์	**S_ISFIFO**	กำหนดช่องสัญญาณ interprocess communication (IPC) รูทีนย่อย **mkfifo** สร้างไพพ์ที่กำหนดชื่อ รูทีนย่อย **pipe** สร้างไพพ์ที่ไม่ได้กำหนดชื่อ
ลิงก์สัญลักษณ์	**S_ISLNK**	ไฟล์ที่มีชื่อพาธสัมบูรณ์หรือสัมพัทธ์ไปยัง ชื่อไฟล์อีกชื่อหนึ่ง
ซ็อกเก็ต	**S_ISSOCK**	วิธี IPC ที่อนุญาตให้แอ็พพลิเคชันแลกเปลี่ยนข้อมูล รูทีนย่อย **socket** สร้างซ็อกเก็ต และรูทีนย่อย **bind** อนุญาตให้มีการกำหนดชื่อซ็อกเก็ต

ขนาดสูงสุดของไฟล์ทั่วไปในระบบไฟล์ JFS ถูกเปิดให้ใช้สำหรับ ไฟล์ขนาดใหญ่ที่มีขนาดน้อยกว่า 64 กิกะไบต์ (68589453312) ในระบบไฟล์อื่น ถูกเปิดให้ใช้ไฟล์ขนาดใหญ่ และประเภทระบบไฟล์ JFS อื่น ไฟล์ทั้งหมด ไม่ถูกแสดงเป็นไฟล์ทั่วไปในตารางก่อนหน้าที่มีขนาดไฟล์สูงสุด 2 กิกะไบต์ลบ 1 (2147483647) ขนาดสูงสุดของไฟล์ใน JFS2 ถูกจำกัดโดยขนาดของระบบไฟล์เอง

ข้อจำกัดด้านสถาปัตยกรรมเกี่ยวกับขนาดของระบบไฟล์ JFS2 คือ 2^{52} ไบต์ หรือ 4 เพตะไบต์ ขนาดไฟล์สูงสุด ที่สนับสนุนโดยเคอร์เนล 64 บิตคือ 2^{44} – 4096 ไบต์ หรือน้อยกว่า 16 เทระไบต์

ความยาวสูงสุดของชื่อไฟล์คือ 255 อักขระ และความยาวสูงสุด ของชื่อพาธคือ 1023 ไบต์

การทำงานกับไฟล์

ระบบปฏิบัติการจัดให้มีรูทีนย่อยมากมายที่ ใช้ดำเนินการไฟล์ สำหรับคำอธิบายโดยย่อของรูทีนย่อยที่ควบคุมไฟล์ที่ใช้ส่วนใหญ่ โปรดดูที่ต่อไปนี้:

การสร้างไฟล์

รูทีนย่อยต่อไปนี้ถูกใช้เมื่อสร้างไฟล์:

creat สร้างไฟล์ปกติใหม่ที่ว่างเปล่า

link สร้างชื่อเพิ่ม (รายการไดเร็กทอรี) สำหรับไฟล์ที่มีอยู่

mkdir สร้างไดเร็กทอรี

mkfifo สร้างไพพ์ที่กำหนดชื่อ

mknod สร้างไฟล์ที่กำหนดอุปกรณ์

open สร้างไฟล์ว่างเปล่าใหม่ถ้าแฟล็ก **O_CREAT** ถูกตั้งค่า

pipe สร้าง IPC

socket สร้างซ็อกเก็ต

การจัดการกับไฟล์ (การเขียนโปรแกรม)

รูทีนย่อยต่อไปนี้สามารถใช้ดำเนินการไฟล์:

access พิจารณาความสามารถในการเข้าใช้งานของไฟล์

chmod เปลี่ยนโหมดการเข้าถึงของไฟล์

chown เปลี่ยนค่าความเป็นเจ้าของไฟล์

close เปิด file descriptors ที่เปิด (รวมถึงซ็อกเก็ต)

fclear สร้างพื้นที่ในไฟล์

fcntl, dup หรือ dup2
ควบคุม file descriptors ที่เปิด

fsync บันทึกการเปลี่ยนแปลงในไฟล์ที่เป็นสื่อบันทึกถาวร

ioctl ควบคุมฟังก์ชันที่สัมพันธ์กับ file descriptors ที่เปิด รวมถึง ไฟล์พิเศษ ซ็อกเก็ต และการสนับสนุนอุปกรณ์ทั่วไป เช่น อินเตอร์เฟสเทอร์มินัล ทั่วไปของ termio

lockf หรือ flock
ควบคุม file descriptors ที่เปิด

lseek หรือ llseek
ย้ายตำแหน่งตัวชี้ I/O ในไฟล์ที่เปิด

open ส่งกลับ file descriptor ที่ใช้โดยรูทีนย่อยอื่นเพื่ออ้างอิงไปยัง ไฟล์ที่เปิด การดำเนินการ **เปิด** รับชื่อไฟล์ทั่วไป และโหมดสิทธิที่บ่งชี้ว่าไฟล์จะถูกอ่าน เขียน หรือทั้งสอง

read รับข้อมูลจากไฟล์ที่เปิดถ้ามีสิทธิเหมาะสม (**O_RDONLY** หรือ **O_RDWR**) ถูกตั้งค่าโดยรูทีนย่อยที่ **เปิด**

rename เปลี่ยนชื่อไฟล์

rmdir ลบไดเร็กทอรีออกจากระบบไฟล์

stat รายงานสถานะของไฟล์ รวมถึงเจ้าของและโหมดการเข้าถึง

truncate
เปลี่ยนความยาวไฟล์

เขียน วางข้อมูลลงในไฟล์ที่เปิดถ้ามีสิทธิเหมาะสม (**O_WRONLY** หรือ **O_RDWR**) ถูกตั้งค่าโดยรูทีนย่อยที่ **เปิด**

สำหรับข้อมูลเพิ่มเติมเกี่ยวกับประเภทและคุณสมบัติของ ระบบไฟล์โปรดดูที่ ระบบไฟล์ใน *Operating system and device management*

การทำงานกับไดเร็กทอรี JFS

ไดเร็กทอรีจัดให้มีโครงสร้างลำดับชั้นกับ ระบบไฟล์ ไฟล์การเชื่อมโยง และชื่อไดเร็กทอรีย่อย i-node ไม่มีการจำกัด ความลึกของไดเร็กทอรีที่ซ้อนกัน พื้นที่ดิสก์ถูกจัดสรรสำหรับไดเร็กทอรี ในบล็อก 4096 ไบต์ แต่ระบบปฏิบัติการจัดสรรพื้นที่ไดเร็กทอรีใน เร็กคอร์ด 512 ไบต์

กระบวนการสามารถอ่านไดเร็กทอรีเหมือนไฟล์ทั่วไป อย่างไรก็ตาม เคอร์เนลเขียนไดเร็กทอรี สำหรับเหตุผลนี้ ไดเร็กทอรีถูกสร้างและ ดูแลโดยชุดของรูทีนย่อยเฉพาะ

โครงสร้างไดเร็กทอรี JFS

ไดเร็กทอรีมีลำดับของรายการไดเร็กทอรี แต่ละรายการไดเร็กทอรีมีฟิลด์ความยาวคงที่สามฟิลด์ (หมายเลขดัชนี ที่สัมพันธ์กับ i-node ของไฟล์ ความยาวของชื่อไฟล์ และจำนวน ไบต์สำหรับรายการ) และหนึ่งฟิลด์ความยาวตัวแปรสำหรับชื่อไฟล์ ฟิลด์ชื่อไฟล์ถูกปิดท้ายด้วย null และเสริมถึง 4 ไบต์ ชื่อไฟล์ยาว ได้ถึง 255 ไบต์

รายการไดเร็กทอรีมีความยาวผันแปรได้เพื่อให้ชื่อไฟล์มีความยืดหยุ่นมากที่สุด อย่างไรก็ตาม พื้นที่ไดเร็กทอรีทั้งหมด ถูกจัดสรรตลอดเวลา

รายการไดเร็กทอรีไม่สามารถขายได้เกินเซ็คชัน 512 ไบต์ของ ไดเร็กทอรี เมื่อไดเร็กทอรีต้องการมากกว่า 512 ไบต์ เร็กคอร์ด 512 ไบต์ อื่นจะถูกผนวกกับเร็กคอร์ดดั้งเดิม ถ้าเร็กคอร์ด 512 ไบต์ทั้งหมด ในบล็อกข้อมูลที่จัดสรรถูกกำหนดค่า บล็อกข้อมูลเพิ่มเติม (4096 ไบต์) จะถูกแบ่ง

เมื่อไฟล์ถูกเอาออก พื้นที่ซึ่งไฟล์ได้ใช้ใน โครงสร้างไดเร็กทอรีจะถูกเพิ่มให้กับรายการไดเร็กทอรีก่อนหน้า ข้อมูล เกี่ยวกับไดเร็กทอรีที่เอาออกยังอยู่จนกว่ารายการใหม่พอดีกับ พื้นที่ที่ว่าง

ทุกไดเร็กทอรีมีรายการ . (จุด) และ .. (จุด, จุด) . (จุด) รายการไดเร็กทอรีชี้ไปที่ i-node สำหรับตัว ไดเร็กทอรีเอง .. (จุด, จุด) รายการไดเร็กทอรีชี้ไปที่ i-node สำหรับพาเรนต์ ไดเร็กทอรี โปรแกรม **mkfs** กำหนดค่าเริ่มต้นระบบไฟล์ ดังนั้นรายการ . (จุด) และ .. (จุด, จุด) ในไดเร็กทอรี root ใหม่ชี้ไปที่ root i-node ของระบบไฟล์

ไดเร็กทอรีมีโหมดการเข้าถึงดังต่อไปนี้:

โหมด	คำอธิบาย
อ่าน	อนุญาตให้กระบวนการอ่านรายการไดเร็กทอรี
เขียน	อนุญาตให้กระบวนการสร้างรายการไดเรกทอรีใหม่หรือเอารายการเก่าออก โดยใช้รูทีนย่อย **creat, mknod, link** และ **unlink**
execute	อนุญาตให้กระบวนการใช้ไดเร็กทอรีเป็นไดเร็กทอรีการทำงานปัจจุบัน หรือเพื่อค้นหาข้อมูลภายในไดเร็กทอรีในแผนผังไฟล์

การทำงานกับไดเร็กทอรี JFS (การเขียนโปรแกรม)

ต่อไปนี้เป็นรายการของรูทีนย่อย ที่พร้อมใช้งานสำหรับการทำงานกับไดเร็กทอรี:

closedir
ปิดสตรีมไดเร็กทอรีและปลดโครงสร้างที่สัมพันธ์กับพารามิเตอร์ *DirectoryPointer*

mkdir สร้างไดเร็กทอรี

opendir
เปิดไดเร็กทอรีที่กำหนดโดยพารามิเตอร์ *DirectoryName* และเชื่อมโยงสตรีมไดเร็กทอรีกับไดเร็กทอรี

readdir ส่งกลับตัวชี้ไปที่รายการไดเร็กทอรีถัดไป

rewinddir
รีเซ็ตตำแหน่งของสตรีมไดเร็กทอรีที่ระบุไปที่จุดเริ่มต้น ของไดเร็กทอรี

rmdir เอาไดเร็กทอรีออก

seekdir ตั้งค่าตำแหน่งของการดำเนินการของรูทีนย่อย **readdir** ถัดไปบนสตรีมไดเร็กทอรี

telldir ส่งกลับตำแหน่งปัจจุบันพร้อมกับสตรีมไดเร็กทอรี ที่ระบุ

การเปลี่ยนไดเร็กทอรีปัจจุบันของกระบวนการ

เมื่อ ระบบถูกบูต กระบวนการแรกใช้ไดเร็กทอรี root ของระบบไฟล์ root เป็นไดเร็กทอรีปัจจุบัน กระบวนการใหม่ที่สร้างด้วยรูทีนย่อย **fork** สืบทอดไดเร็กทอรีปัจจุบันที่ใช้โดยกระบวนการพาเรนต์ รูทีนย่อย **chdir** เปลี่ยนไดเร็กทอรีปัจจุบันของกระบวนการ

รูทีนย่อย **chdir** วิเคราะห์หา ชื่อพาธเพื่อประกันว่าไฟล์เป้าหมายเป็นไดเร็กทอรีและเจ้าของกระบวนการ มีสิทธิในไดเร็กทอรี หลังจากรูทีนย่อย **chdir** ถูกรัน กระบวนการใช้ไดเร็กทอรีปัจจุบันใหม่เพื่อค้นหาชื่อพาธทั้งหมด ที่ไม่ได้เริ่มต้นด้วย / (สแลช)

การเปลี่ยนไดเร็กทอรี root ของกระบวนการ

คุณสามารถทำให้ไดเร็กทอรีที่ตั้งชื่อโดยพารามิเตอร์ *Path* ของกระบวนการเป็นไดเร็กทอรี root มีประสิทธิผลโดยใช้รูทีนย่อย **chroot** กระบวนการ Child ของกระบวนการที่เรียกถือว่าไดเร็กทอรีที่ระบุโดยรูทีนย่อย **chroot** เป็นไดเร็กทอรี root โลจิคัลของระบบ ไฟล์

กระบวนการใช้ไดเร็กทอรี root ระบบไฟล์โกลบอล สำหรับชื่อพาธทั้งหมดที่เริ่มต้นด้วย / (สแลช) การค้นหาชื่อพาธทั้งหมดเริ่มต้น ที่ / (สแลช) เริ่มที่ไดเร็กทอรี root ใหม่

รูทีนย่อยที่ควบคุมไดเร็กทอรี JFS

เนื่องจากลักษณะที่เป็นเอกลักษณ์ของไฟล์ไดเร็กทอรี ไดเร็กทอรี ถูกควบคุมโดยชุดของรูทีนย่อยพิเศษ รูทีนย่อยดังต่อไปนี้ถูกกำหนดให้กับไดเร็กทอรีการควบคุม:

chdir เปลี่ยนไดเรกทอรีทำงานปัจจุบัน

chroot เปลี่ยนไดเร็กทอรี root ที่มีประสิทธิผล

getcwd หรือ getwd
รับพาธไปที่ไดเร็กทอรีปัจจุบัน

mkdir สร้างไดเร็กทอรี

opendir, readdir, telldir, seekdir, rewinddir, หรือ closedir
ทำการดำเนินการหลากหลายกับไดเร็กทอรี

rename เปลี่ยนชื่อไดเร็กทอรี

rmdir เอาไดเรกทอรีออก

การทำงานกับไดเร็กทอรี JFS2

ไดเรกทอรีจัดให้มีโครงสร้างลำดับชั้นกับ ระบบไฟล์ ไฟล์การเชื่อมโยง และชื่อไดเร็กทอรีย่อย i-node ไม่มีการจำกัด ความลึกของไดเรกทอรีที่ซ้อนกัน

พื้นที่ดิสก์ถูกจัดสรรสำหรับไดเร็กทอรีในบล็อก ระบบไฟล์

กระบวนการสามารถอ่านไดเรกทอรีเหมือนไฟล์ทั่วไป อย่างไรก็ตาม เคอร์เนลเขียนไดเรกทอรี สำหรับเหตุผลนี้ ไดเรกทอรีถูกสร้างและ ดูแลโดยชุดของรูทีนย่อยเฉพาะ

โครงสร้างไดเร็กทอรี JFS2

ไดเร็กทอรีมีรายการที่อธิบายอ็อบเจ็กต์ที่มีใน ไดเร็กทอรี รายการไดเร็กทอรีมีความยาวคงที่และมีข้อมูลดังต่อไปนี้:

- หมายเลข i-node
- ชื่อ (ยาวได้ถึง 22 ไบต์)
- ฟิลด์ความยาวชื่อ
- ฟิลด์ที่จะดำเนินรายการถ้าชื่อยาวกว่า 22 ไบต์

รายการไดเร็กทอรีถูกเก็บใน B+ ทรีที่เรียงตามชื่อ ตัวข้อมูลเอง (.) หรือข้อมูลพาเรนต์ (..) มีอยู่ใน i-node แทนที่จะอยู่ในรายการไดเร็กทอรี

ไดเรกทอรีมีโหมดการเข้าถึงดังต่อไปนี้:

โหมด	คำอธิบาย
อ่าน	อนุญาตให้กระบวนการอ่านรายการไดเรกทอรี
เขียน	อนุญาตให้กระบวนการสร้างรายการไดเร็กทอรีใหม่หรือเอารายการเก่าออก โดยใช้รูทีนย่อย **creat, mknod, link** และ **unlink**
execute	อนุญาตให้กระบวนการใช้ไดเรกทอรีเป็นไดเรกทอรีการทำงานปัจจุบัน หรือเพื่อค้นหาข้อมูลภายในไดเรกทอรีในแผนผังไฟล์

การทำงานกับไดเร็กทอรี JFS2 (การเขียนโปรแกรม)

ต่อไปนี้เป็นรายการของรูทีนย่อย ที่พร้อมใช้งานสำหรับการทำงานกับไดเร็กทอรี:

closedir
ปิดสตรีมไดเรกทอรีและปลดโครงสร้างที่สัมพันธ์กับพารามิเตอร์ *DirectoryPointer*

mkdir สร้างไดเรกทอรี

opendir
ส่งกลับตัวชี้โครงสร้างที่ถูกใช้โดยรูทีนย่อย **readdir** เพื่อรับรายการไดเร็กทอรีถัดไป โดย **rewinddir** เพื่อรีเซ็ตตำแหน่งอ่านไปที่จุดเริ่มต้น และโดย **closedir** เพื่อปิดไดเร็กทอรี

readdir ส่งกลับตัวชี้ไปที่รายการไดเรกทอรีถัดไป

rewinddir
รีเซ็ตตำแหน่งของสตรีมไดเรกทอรีที่ระบุไปที่จุดเริ่มต้น ของไดเรกทอรี

rmdir เอาไดเรกทอรีออก

seekdir ส่งกลับไปที่ตำแหน่งก่อนหน้าที่ได้รับด้วยรูทีนย่อย **telldir**

telldir ส่งกลับตำแหน่งปัจจุบันพร้อมกับสตรีมไดเรกทอรี ที่ระบุ

อย่าใช้รูทีนย่อย **open, read, lseek** และ **close** เพื่อเข้าถึงไดเร็กทอรี

การเปลี่ยนไดเร็กทอรีปัจจุบันของกระบวนการ

เมื่อ ระบบถูกบูต กระบวนการแรกใช้ไดเร็กทอรี root ของระบบไฟล์ root เป็นไดเร็กทอรีปัจจุบัน กระบวนการใหม่ที่สร้างด้วยรูทีนย่อย **fork** สืบทอดไดเร็กทอรีปัจจุบันที่ใช้โดยกระบวนการพาเรนต์ รูทีนย่อย **chdir** เปลี่ยนไดเร็กทอรีปัจจุบันของกระบวนการ

รูทีนย่อย **chdir** วิเคราะห์หา ชื่อพาธเพื่อประกันว่าไฟล์เป้าหมายเป็นไดเรกทอรีและเจ้าของกระบวนการ มีสิทธิในไดเรกทอรี หลังจากรูทีนย่อย **chdir** ถูกรัน กระบวนการใช้ไดเร็กทอรีปัจจุบันใหม่เพื่อค้นหาชื่อพาธทั้งหมด ที่ไม่ได้เริ่มต้นด้วย / (สแลช)

การเปลี่ยนไดเร็กทอรี root ของกระบวนการ

กระบวนการสามารถเปลี่ยนความเข้าใจที่มีกับไดเร็กทอรี root ผ่านรูทีนย่อย **chroot** กระบวนการ Child ของกระบวนการที่เรียกถือว่า ไดเร็กทอรีที่ระบุโดยรูทีนย่อย **chroot** เป็นไดเร็กทอรี root โลจิคัลของระบบไฟล์

กระบวนการใช้ไดเร็กทอรี root ระบบไฟล์โกลบอลสำหรับ ชื่อพาธทั้งหมดที่เริ่มต้นด้วย / (สแลช) การค้นหาชื่อพาธทั้งหมดเริ่มต้นด้วย / (สแลช) เริ่มที่ไดเร็กทอรี root ใหม่

รูทีนย่อยที่ควบคุมไดเร็กทอรี JFS2

เนื่องจากลักษณะที่เป็นเอกลักษณ์ของไฟล์ไดเรกทอรี, ไดเรกทอรี ถูกควบคุมโดยชุดของรูทีนย่อยพิเศษ รูทีนย่อยดังต่อไปนี้ถูกกำหนดให้กับไดเรกทอรีการควบคุม:

chdir เปลี่ยนไดเรกทอรีทำงานปัจจุบัน

chroot เปลี่ยนไดเร็กทอรีรากที่มีประสิทธิผล

opendir, readdir, telldir, seekdir, rewinddir, หรือ closedir
ทำการดำเนินการหลากหลายกับไดเรกทอรี

getcwd หรือ getwd
รับพาธไปที่ไดเร็กทอรีปัจจุบัน

mkdir สร้างไดเร็กทอรี

rename เปลี่ยนชื่อไดเรกทอรี

rmdir เอาไดเรกทอรีออก

การทำงานกับ JFS i-nodes

ไฟล์ใน journaled file system (JFS) ถูกแสดง เป็นการภายในเป็นโหนดดัชนี (i-nodes) JFS i-nodes มีอยู่ในรูปเชิงสถิติบนดิสก์และได้รับข้อมูลการเข้าถึงสำหรับไฟล์ รวมถึงตัวชี้ไปยัง ดิสก์แอดเดรสที่แท้จริงของบล็อกข้อมูลของไฟล์

จำนวนของ i-nodes ดิสก์ที่มีอยู่ในระบบไฟล์นั้นขึ้นอยู่กับขนาดของระบบไฟล์ ขนาดกลุ่มการจัดสรร (ค่าดีฟอลต์คือ 8 MB) และจำนวนไบต์ต่ออัตรา i-node (ค่าดีฟอลต์คือ 4096) พารามิเตอร์เหล่านี้ถูกกำหนดให้แก่คำสั่ง **mkfs** ในตอนสร้างระบบไฟล์ เมื่อสร้างไฟล์เพียงพอ สำหรับใช้ i-nodes ที่มีอยู่แล้ว จะไม่สามารถสร้างไฟล์ใดๆ ได้อีก แม้ว่าระบบไฟล์จะมีพื้นที่ว่างก็ตาม

ในการพิจารณาจำนวนของ i-nodes ที่มี ให้ใช้คำสั่ง **df -v** i-nodes ดิสก์ถูก กำหนดในไฟล์ **/usr/include/jfs/ino.h**

โครงสร้างดิสก์ i-node สำหรับ JFS

แต่ละ i-node ดิสก์ใน JFS มีโครงสร้าง 128 ไบต์ ออฟเซ็ต ของ i-node เฉพาะภายในรายการ i-node ของระบบไฟล์จะสร้างหมายเลขเฉพาะ (i-number) ที่ระบบปฏิบัติการใช้บระบุ i-node แม็พแสดงบิต ที่รู้จักในชื่อ *แม็พ i-node* ซึ่งติดตาม ความพร้อมใช้ของ i-nodes ดิสก์ที่ว่างสำหรับระบบไฟล์

i-nodes ดิสก์ประกอบด้วยข้อมูลต่อไปนี้:

ฟิลด์	เนื้อหา
i_mode	ประเภทของไฟล์และบิตโหมดสิทธิการเข้าถึง
i_size	ขนาดของไฟล์เป็นไบต์
i_uid	สิทธิการเข้าถึงสำหรับ ID ผู้ใช้
i_gid	สิทธิการเข้าถึงสำหรับ ID กลุ่ม
i_nblocks	จำนวนบล็อกที่จัดสรรให้แก่ไฟล์
i_mtime	เวลาล่าสุดที่ไฟล์ถูกแก้ไข
i_atime	เวลาล่าสุดที่ไฟล์ถูกเข้าถึง
i_ctime	เวลาล่าสุดที่ i-node ถูกแก้ไข
i_nlink	จำนวนฮาร์ดลิงก์ไปยังไฟล์
i_rdaddr[8]	ดิสก์แอดเดรสจริงของข้อมูล
i_rindirect	ดิสก์แอดเดรสจริงของบล็อกทางอ้อม ถ้ามี

คุณไม่สามารถเปลี่ยนแปลงข้อมูลไฟล์โดยไม่เปลี่ยน i-node แต่เป็นไปได้ที่จะเปลี่ยน i-node โดยไม่ทำการเปลี่ยนแปลงเนื้อหาของ ไฟล์ ตัวอย่าง เมื่อสิทธิถูกเปลี่ยนแปลง ข้อมูลภายใน i-node (`i_mode`) จะถูกแก้ไข แต่ข้อมูลใน ไฟล์ยังคงเหมือนเดิม

ฟิลด์ **i_rdaddr** ภายใน i-node ดิสก์มี 8 ดิสก์แอดเดรส แอดเดรสเหล่านี้ชี้ไปยัง 8 บล็อกข้อมูลแรก ที่กำหนดไปยังไฟล์ ฟิลด์แอดเดรส **i_rindirect** ชี้ไปยังบล็อกทางอ้อม บล็อกทางอ้อมเป็นแบบทางอ้อมเดียว หรือ ทางอ้อมคู่ ดังนั้น รูปแบบของการจัดสรรบล็อกที่เป็นไปได้มีสามแบบ สำหรับไฟล์: ทางตรง ทางอ้อม หรือทั้งสองทาง

i-nodes ดิสก์ไม่มีข้อมูลไฟล์หรือชื่อพาธ รายการไดเร็กทอรีถูกนำมาใช้เพื่อลิงก์ชื่อไฟล์กับ i-nodes i-node ใดๆ สามารถลิงก์กับชื่อไฟล์หลายๆ ไฟล์ได้โดยการสร้างรายการไดเร็กทอรีเพิ่มด้วยรูทีนย่อย **link** หรือ **symlink** ในการพิจารณาหมายเลข i-node ที่กำหนด ให้แก่ไฟล์ ให้ใช้คำสั่ง **ls -i**

i-nodes ที่แทนไฟล์ที่กำหนดอุปกรณ์ จะมีข้อมูลแตกต่างจาก i-nodes สำหรับไฟล์ปกติเล็กน้อย ไฟล์ ที่เชื่อมโยงกับอุปกรณ์เรียกว่า *ไฟล์พิเศษ* ซึ่งไม่มีแอดเดรสบล็อกข้อมูลอยู่ในไฟล์อุปกรณ์พิเศษ แต่หมายเลขอุปกรณ์หลัก และรองจะถูกรวมในฟิลด์ **i_rdev**

i-node ดิสก์ถูกรีลีสเมื่อจำนวนนับลิงก์ (`i_nlink`) ไปยัง i-node เท่ากับ 0 ลิงก์จะแทน ชื่อไฟล์ที่เชื่อมโยงกับ i-node เมื่อจำนวนนับลิงก์ไปยัง i-node ดิสก์ เท่ากับ 0 บล็อกข้อมูลทั้งหมดที่เชื่อมโยงกับ i-node จะถูกรีลีสไปยังแม็พ แสดงบิตของบล็อกข้อมูลว่างสำหรับระบบไฟล์ จากนั้น i-node จะถูกนำไปไว้บน แม็พ i-node ว่าง

โครงสร้าง JFS In-core i-node

เมื่อไฟล์ถูกเปิดใช้งาน ระบบปฏิบัติการจะสร้าง in-core i-node *in-core i-node* มีสำเนา ของฟิลด์ทั้งหมดที่กำหนดใน i-node ดิสก์ บวกกับฟิลด์อื่นๆ เพิ่มเติมสำหรับการติดตาม และการจัดการการเข้าถึง in-core i-node เมื่อไฟล์ถูกเปิดใช้งาน ข้อมูล ใน i-node ดิสกีจะถูกทำสำเนาไปไว้ใน in-core i-node เพื่อให้เข้าถึงได้ง่าย In-core i-nodes ถูกกำหนดในไฟล์ **/usr/include/jfs/inode.h** ข้อมูลเพิ่มเติมบางอย่างที่ติดตามโดย in-core i-node มีดังนี้:

- สถานะของ in-core i-node รวมถึงแฟล็กที่ระบุ:
  - การล็อก i-node
  - การประมวลผลที่รอ i-node ปลดล็อก
  - การเปลี่ยนแปลงไปยังข้อมูล i-node ของไฟล์
  - การเปลี่ยนแปลงไปยังข้อมูลของไฟล์
- หมายเลขอุปกรณ์โลจิคัลของระบบไฟล์ที่มีไฟล์
- i-number ที่ใช้ระบุ i-node
- จำนวนนับการอ้างอิง เมื่อฟิลด์จำนวนนับการอ้างอิงเท่ากับ 0 in-core i-node จะถูกรีลีส

เมื่อ in-core i-node ถูกรีลีส (ตัวอย่างเช่น กับรูทีนย่อย **close**) จำนวนนับการอ้างอิง in-core i-node จะถูกลดลง 1 ถ้าการลดนี้ส่งผล ให้จำนวนนับการอ้างอิง in-core i-node เป็น 0 i-node จะถูกรีลีส จากตาราง in-core i-node และเนื้อหาของ in-core i-node ถูก เขียนลงในสำเนาดิสก์ของ i-node (ถ้าสองเวอร์ชันต่างกัน)

การทำงานกับ JFS2 i-nodes

ไฟล์ใน JFS2 ถูกแสดงเป็นการภายในเป็นโหนดดัชนี (i-nodes)

i-nodes ดิสก์ JFS2 มีอยู่ในรูปแบบสแตติกบนดิสก์และมีข้อมูลการเข้าถึงสำหรับไฟล์ รวมถึงตัวชี้ไปยังดิสก์แอดเดรสจริงของบล็อกข้อมูลของไฟล์ i-nodes ถูกจัดสรรพื้นที่แบบไดนามิกโดย JFS2 i-nodes ดิสก์ถูกกำหนดใน ไฟล์ **/usr/include/j2/j2_dinode.h**

เมื่อไฟล์ถูกเปิดใช้งาน ระบบปฏิบัติการจะสร้าง in-core i-node in-core i-node มีสำเนาของฟิลด์ทั้งหมดที่กำหนดใน i-node ดิสก์ บวกกับฟิลด์อื่นๆ เพิ่มเติมสำหรับกการติดตาม in-core i-node In-core i-nodes ถูกกำหนดในไฟล์ **/usr/include/j2/j2_inode.h**

โครงสร้าง Disk i-node สำหรับ JFS2

แต่ละ i-node ดิสก์ใน JFS2 มีโครงสร้าง 512 ไบต์ ดัชนีของแม็พการจัดสรร i-node เฉพาะของระบบไฟล์จะสร้างหมายเลขเฉพาะ (i-number) ที่ระบบปฏิบัติการใช้ระบุ i-node แม็พการจัดสรร i-node จะคอยติดตามตำแหน่งของ i-nodes บนดิสก์ รวมถึงความพร้อมใช้

i-nodes ดิสก์ประกอบด้วยข้อมูลต่อไปนี้:

ฟิลด์	เนื้อหา
di_mode	ประเภทของไฟล์และบิตโหมดสิทธิการเข้าถึง
di_size	ขนาดของไฟล์เป็นไบต์
di_uid	สิทธิการเข้าถึงสำหรับ ID ผู้ใช้
di_gid	สิทธิการเข้าถึงสำหรับ ID กลุ่ม
di_nblocks	จำนวนบล็อกที่จัดสรรให้แก่ไฟล์
di_mtime	เวลาล่าสุดที่ไฟล์ถูกแก้ไข
di_atime	เวลาล่าสุดที่ไฟล์ถูกเข้าถึง
di_ctime	เวลาล่าสุดที่ i-node ถูกแก้ไข
di_nlink	จำนวนฮาร์ดลิงก์ไปยังไฟล์
di_btroot	Root ของ B+ tree ที่ใช้อธิบายดิสก์แอดเดรสของข้อมูล

คุณไม่สามารถเปลี่ยนแปลงข้อมูลไฟล์โดยไม่เปลี่ยน i-node แต่เป็นไปได้ที่จะเปลี่ยน i-node โดยไม่ทำการเปลี่ยนแปลงเนื้อหาของ ไฟล์ ตัวอย่าง เมื่อสิทธิถูกเปลี่ยนแปลง ข้อมูลภายใน i-node (**di_mode**) จะถูกแก้ไข แต่ข้อมูลใน ไฟล์ยังคงเหมือนเดิม

di_btroot อธิบาย root ของ B+ tree โดยอธิบายข้อมูลสำหรับ i-node di_btroot มีฟิลด์ที่ระบุ จำนวนรายการใน i-node ที่กำลังถูกใช้งาน และอีกฟิลด์อธิบาย ว่าเป็นโหนด leaf หรือโหนดภายในสำหรับ B+ tree

i-nodes ดิสก์ไม่มีข้อมูลไฟล์หรือชื่อพาธ รายการไดเร็กทอรี ถูกนำมาใช้เพื่อลิงก์ชื่อไฟล์กับ i-nodes i-node ใดๆ สามารถลิงก์กับชื่อไฟล์หลายๆ ไฟล์ได้โดยการสร้างรายการไดเร็กทอรีเพิ่มด้วยรูทีนย่อย **link** หรือ **symlink** ในการพิจารณาหมายเลข i-node ที่กำหนด ให้แก่ไฟล์ ให้ใช้คำสั่ง **ls -i**

i-nodes ที่แทนไฟล์ที่กำหนดอุปกรณ์ จะมีข้อมูลแตกต่างจาก i-nodes สำหรับไฟล์ปกติเล็กน้อย ไฟล์ ที่เชื่อมโยงกับอุปกรณ์เรียกว่า *ไฟล์พิเศษ* ซึ่งไม่มี แอดเดรสบล็อกข้อมูลอยู่ในไฟล์อุปกรณ์พิเศษ แต่หมายเลขอุปกรณ์หลัก และรองจะถูกรวมในฟิลด์ **di_rdev**

i-node ดิสก์ถูกรีลีสเมื่อจำนวนนับลิงก์ (**di_nlink**) ไปยัง i-node เท่ากับ 0 ลิงก์จะแทน ชื่อไฟล์ที่เชื่อมโยงกับ i-node When the link count to the disk i-node is 0, all the data blocks associated with the i-node are released to the bitmap of free data blocks for the file system. จากนั้น i-node จะถูกนำไปไว้บน แม็พ i-node ว่าง

โครงสร้าง JFS2 in-core i-node

เมื่อไฟล์ถูกเปิดใช้งาน ข้อมูล ใน i-node ดิสก์จะถูกทำสำเนาไปไว้ใน in-core i-node เพื่อให้เข้าถึงได้ง่าย โครงสร้าง in-core i-node มีฟิลด์เพิ่มเติมที่จัดการการเข้าถึงข้อมูลสำคัญของ i-node ดิสก์ ฟิลด์สำหรับ in-core i-node ถูกกำหนดไว้ในไฟล์ **j2_inode.h** ข้อมูลเพิ่มเติมบางอย่างที่ติดตามโดย in-core i-node มีดังนี้:

- สถานะของ in-core i-node รวมถึงแฟล็กที่ระบุ:
  - การล็อก i-node
  - การประมวลผลที่รอ i-node ปลดล็อก
  - การเปลี่ยนแปลงไปยังข้อมูล i-node ของไฟล์
  - การเปลี่ยนแปลงไปยังข้อมูลของไฟล์
- หมายเลขอุปกรณ์โลจิคัลของระบบไฟล์ที่มีไฟล์
- i-number ที่ใช้ระบุ i-node
- จำนวนนับการอ้างอิง เมื่อฟิลด์จำนวนนับการอ้างอิงเท่ากับ 0 in-core i-node จะถูกรีลีส

เมื่อ in-core i-node ถูกรีลีส (ตัวอย่างเช่น กับรูทีนย่อย **close**) จำนวนนับการอ้างอิง in-core i-node จะถูกลดลง 1 ถ้าการลดนี้ส่งผล ให้จำนวนนับการอ้างอิง in-core i-node เป็น 0 i-node จะถูกรีลีส จากตาราง in-core i-node และเนื้อหาของ in-core i-node ถูก เขียนลงในสำเนาดิสก์ของ i-node (ถ้าสองเวอร์ชันต่างกัน)

การจัดสรรพื้นที่ไฟล์ JFS

การจัดสรรพื้นที่ไฟล์คือ วิธีการที่ข้อมูลถูกแบ่งส่วนพื้นที่เก็บข้อมูลฟิสิคัลใน ระบบปฏิบัติการ

เคอร์เนลจัดสรรพื้นที่ดิสก์ให้แก่ไฟล์หรือไดเร็กทอรีในรูปของ บล็อกโลจิคัล *บล็อกโลจิคัล* สำหรับ JFS หมายถึง การแบ่งเนื้อหาของไฟล์หรือไดเร็กทอรีออกเป็นหน่วยขนาด 4096 ไบต์ อย่างไรก็ตาม บล็อกโลจิคัลไม่สิ่งที่มีอยู่จริง ข้อมูลในบล็อกโลจิคัลจะใช้ พื้นที่เก็บข้อมูลฟิสิคัลบนดิสก์ แต่ละไฟล์หรือไดเร็กทอรีมีบล็อกโลจิคัลอย่างน้อย 0 หรือมากกว่า 0 บล็อก แฟรกเมนต์ที่ใช้แทนบล็อกโลจิคัล คือหน่วยพื้นฐาน สำหรับพื้นที่ดิสก์ที่จัดสรรใน JFS

บล็อกโลจิคัลเต็ม หรือบางส่วน

ไฟล์หรือไดเร็กทอรีอาจมีบล็อกโลจิคัลเต็มหรือ บางส่วน บล็อกโลจิคัลเต็มจะมีข้อมูล 4096 ไบต์ บล็อกโลจิคัล บางส่วนเกิดขึ้นเมื่อบล็อกโลจิคัลสุดท้ายของไฟล์หรือไดเร็กทอรีมี ข้อมูลน้อยกว่า 4096 ไบต์

ตัวอย่าง ไฟล์ขนาด 8192 ไบต์ใช้บล็อกโลจิคัลสองบล็อก 4096 ไบต์แรกอยู่ในบล็อกโลจิคัลแรก และ 4096 ไบต์ที่เหลืออยู่ในบล็อกโลจิคัลที่สอง ในทำนองเดียวกัน ไฟล์ขนาด 4608 ไบต์ใช้บล็อก โลจิคัลสองบล็อก อย่างไรก็ตาม บล็อกโลจิคัลสุดท้ายเป็นบล็อกโลจิคัลบางส่วน ที่มีเฉพาะ 512 ไบต์สุดท้ายของข้อมูลของไฟล์ เฉพาะบล็อกโลจิคัลสุดท้าย ของไฟล์เท่านั้นที่สามารถเป็นบล็อกโลจิคัลบางส่วนได้

การจัดสรรในระบบไฟล์แบบแบ่งแฟรกเมนต์

ขนาดแฟรกเมนต์ดีฟอลต์คือ 4096 ไบต์ คุณสามารถระบุ ขนาดแฟรกเมนต์เล็กกว่านี้ได้ด้วยคำสั่ง **mkfs** ระหว่าง การสร้างของระบบไฟล์ ขนาดแฟรกเมนต์ที่ใช้ได้คือ: 512, 1024, 2048 และ 4096 ไบต์ คุณสามารถใช้ขนาดแฟรกเมนต์ได้ขนาดเดียวเท่านั้นในระบบไฟล์

ในการดูแลรักษาประสิทธิภาพของการดำเนินการระบบไฟล์ JFS จะจัดสรร 4096 ไบต์ของพื้นที่แฟรกเมนต์ให้แก่ไฟล์และไดเร็กทอรีที่ มีขนาด 32 KB ขึ้นไป แฟรกเมนต์ที่ใช้พื้นที่ดิสก์ 4096 ไบต์จะถูก จัดสรรให้แก่บล็อกโลจิคัลเต็ม เมื่อข้อมูลถูกเพิ่มในไฟล์หรือไดเร็กทอรี เคอร์เนลจะจัดสรรดิสก์แฟรกเมนต์เพื่อใช้เก็บบล็อกโลจิคัล ดังนั้น ถ้า ขนาดแฟรกเมนต์ของระบบไฟล์คือ 512 ไบต์ บล็อกโลจิคัลเต็มจะเป็น การจัดสรรของแปดแฟรกเมนต์

เคอร์เนลจัดสรรพื้นที่ดิสก์เพื่อให้ไบต์สุดท้ายของข้อมูลได้รับการ การจัดสรรบล็อกบางส่วนเท่านั้น ขณะที่บล็อกบางส่วนที่ขนาดใหญ่ขึ้นเกินขีดจำกัด ของการจัดสรรปัจจุบัน จะมีการจัดสรรแฟรกเมนต์เพิ่ม ถ้าบล็อก บางส่วนเพิ่มขนาดเป็น 4096 ไบต์ ข้อมูลที่เก็บในแฟรกเมนต์ที่ถูกจัดสรรใหม่ เป็นการจัดสรรบล็อกระบบไฟล์ 4096 ไบต์ บล็อกโลจิคัลบางส่วนที่ที่มี ข้อมูลน้อยกว่า 4096 ไบต์ จะถูกจัดสรรตามจำนวนแฟรกเมนต์ที่เหมาะสมที่สุด กับความต้องการพื้นที่จัดเก็บ

การจัดสรรบล็อกใหม่ยังเกิดขึ้นถ้าข้อมูลถูกเพิ่มในบล็อกโลจิคัลที่ แทนช่องว่างไฟล์ *ช่องว่างไฟล์* คือบล็อกโลจิคัล ว่างที่อยู่ก่อนหน้าบล็อกโลจิคัลสุดท้ายที่เก็บข้อมูล (ช่องว่างไฟล์ไม่เกิดขึ้นภายในไดเร็กทอรี) บล็อกโลจิคัลว่างเหล่านี้ไม่ใช่แฟรกเมนต์ ที่จัดสรร อย่างไรก็ตาม เมื่อข้อมูลถูกเพิ่มในช่องว่างไฟล์ จะเกิดการจัดสรร แต่ละ บล็อกโลจิคัลที่ไม่ใช่พื้นที่ดิสก์ที่จัดสรรก่อนหน้านี้จะถูกจัดสรรเป็นจำนวน 4096 ไบต์ของพื้นที่แฟรกเมนต์

การจัดสรรบล็อกเพิ่มไม่จำเป็นถ้าข้อมูลที่มีตรงกลาง ของไฟล์หรือไดเร็กทอรีถูกเขียนทับ บล็อกโลจิคัลที่เก็บข้อมูลที่มีอยู่มีแฟรกเมนต์ที่จัดสรรแล้ว

JFS พยายามคงการจัดสรรให้ต่อเนื่องสำหรับบล็อกโลจิคัลของไฟล์ หรือไดเร็กทอรี บนดิสก์ การคงการจัดสรรไว้ให้ต่อเนื่องกันจะช่วยลดเวลาการค้นหาเนื่องจาก ข้อมูลสำหรับไฟล์หรือไดเร็กทอรีสามารถเข้าถึงได้แบบเรียงลำดับต่อกัน และพบ ได้บนพื้นที่เดียวกันในดิสก์ อย่างไรก็ตาม ดิสก์แฟรกเมนต์สำหรับบล็อกโลจิคัลหนึ่ง อาจไม่ต่อกับดิสก์แฟรกเมนต์ของอีกบล็อกโลจิคัลเสมอไป พื้นที่ดิสก์ ที่ต้องการสำหรับการจัดสรรแบบต่อเนื่องอาจไม่พร้อมใช้งาน ถ้ามี พื้นที่ที่ถูกเขียนไว้แล้วโดยไฟล์หรือไดเร็กทอรีอื่น อย่างไรก็ตาม การจัดสรร สำหรับบล็อกโลจิคัลเดียว จะมีแฟรกเมนต์ต่อเนื่องเสมอ

ระบบไฟล์ใช้บิตแม็พชื่อ *แม็พการจัดสรร บล็อก* เพื่อบันทึกสถานะของทุกบล็อกในระบบไฟล์ เมื่อ ระบบไฟล์ต้องการจัดสรรแฟรกเมนต์ใหม่ ระบบจะอ้างถึงแม็พการจัดสรร แฟรกเมนต์เพื่อระบุว่าแฟรกเมนต์ใดที่สามารถใช้ได้ แฟรกเมนต์สามารถถูกจัดสรร ให้แก่ไฟล์หรือไดเร็กทอรีเดี่ยวเท่านั้นในหนึ่งครั้ง

การจัดสรรในระบบไฟล์ JFS แบบบีบอัด

ในระบบไฟล์ ที่สนับสนุนการบีบอัดข้อมูล ไดเร็กทอรีจะถูกจัดสรรพื้นที่ดิสก์ การบีบอัดข้อมูลยังนำใช้กับไฟล์ทั่วไปและลิงก์สัญลักษณ์ที่มีขนาด ใหญ่กว่าของ i-nodes

การจัดสรรพื้นที่ดิสก์สำหรับระบบไฟล์บีบอัดจะเหมือนกับ ของแฟรกเมนต์ในระบบไฟล์แฟรกเมนต์ บล็อกโลจิคัลถูกจัดสรร 4096 ไบต์เมื่อถูกแก้ไข การจัดสรรนี้รับรองว่ามีพื้นที่ เก็บบล็อกโลจิคัลถ้าข้อมูลไม่บีบอัด ระบบ ต้องการให้การดำเนินการเขียนหรือจัดเก็บรายงานสภาวะที่ไม่มีพื้นที่ดิสก์ ในไฟล์ที่แม็พหน่วยความจำที่การแก้ไขเริ่มต้นของบล็อกโลจิคัล หลังจากการแก้ไขเสร็จสมบูรณ์ บล็อกโลจิคัลจะถูกบีบอัดก่อนถูกเขียน ลงดิสก์ จากนั้นบล็อกโลจิคัลที่บีบอัดจะถูกจัดสรรเท่าจำนวนแฟรกเมนต์ ที่จำเป็นสำหรับเป็นพื้นที่จัดเก็บเท่านั้น

ในระบบไฟล์แฟรกเมนต์ บล็อกโลจิคัลสุดท้ายของไฟล์ (ไม่ เกิน 32 KB) เท่านั้นที่สามารถถูกจัดสรรน้อยกว่า 4096 ไบต์ บล็อกโลจิคัล จะเป็นบล็อกโลจิคัลบางส่วน ในระบบไฟล์บีบอัด ทุกบล็อกโลจิคัล สามารถถูกจัดสรรให้มีขนาดน้อยกว่าบล็อกเต็ม

บล็อกโลจิคัลถูกพิจารณาว่าไม่มีการแก้ไขอีกต่อไปหลังถูกเขียนลง ดิสก์ แต่ละครั้งที่บล็อกโลจิคัลถูกแก้ไข บล็อกดิสก์เต็มจะถูกจัดสรร อีกครั้ง ตามความต้องการของระบบ การจัดสรรใหม่ของบล็อกเต็มเริ่มต้น จะเกิดขึ้นเมื่อบล็อกโลจิคัลของข้อมูลบีบอัดถูกเขียนลงดิสก์ได้สำเร็จ

การจัดสรรในระบบไฟล์ JFS จะเปิดใช้สำหรับไฟล์ขนาดใหญ่

ในระบบไฟล์ ที่เปิดใช้สำหรับไฟล์ขนาดใหญ่ JFS จัดสรรขนาดแฟรกเมนต์สองขนาดสำหรับ ไฟล์ปกติ แฟรกเมนต์ *"ขนาดใหญ่"* (32 X 4096) ถูกจัดสรร สำหรับบล็อกโลจิคัลที่มีพื้นที่เกิน 4 MB และแฟรกเมนต์ 4096 ไบต์ถูกจัดสรร สำหรับบล็อกโลจิคัลที่มีพื้นที่น้อยกว่า 4 MB ไฟล์ที่ไม่ใช่ไฟล์ทั่วไปทั้งหมดจัดสรร แฟรกเมนต์ 4096 ไบต์ การคำนวณนี้ให้ขนาดไฟล์ใหญ่สุด ขนาดเล็กกว่า 64 กิกะไบต์เล็กน้อย (68589453312)

แฟรกเมนต์A *ขนาดใหญ่* ประกอบด้วย แฟรกเมนต์ 4096 ไบต์ต่อเนื่องจาก 32 แฟรกเมนต์ เนื่องจากความต้องการนี้ ขอแนะนำ ให้เปิดใช้ระบบไฟล์สำหรับไฟล์ขนาดใหญ่ที่มีไฟล์ขนาดใหญ่มาก อยู่ภายใน การเก็บไฟล์ขนาดเล็กหลายไฟล์ (ไฟล์ขนาดน้อยกว่า 4 MB) อาจทำให้เกิดปัญหา การแตกแฟรกเมนต์ของพื้นที่ว่าง นี้อาจทำให้การจัดสรรพื้นที่ขนาดใหญ่ล้มเหลวโดยมีเงื่อนไข ข้อผิดพลาด ENOSPC เนื่องจากระบบไฟล์ไม่มีดิสก์แอดเดรสที่ต่อเนื่องกัน 32 แอดเดรส

รูปแบบแอดเดรสดิสก์

การสนับสนุนแฟรกเมนต์ JFS ต้องมีความสามารถในการระบุแอดเดรส ระดับแฟรกเมนต์ อันนั้น ดิสก์แอดเดรสจึงมีรูปแบบพิเศษสำหรับการแม็พ ตำแหน่งที่แฟรกเมนต์ของบล็อกโลจิคัลอยู่บนดิสก์ ระบบไฟล์ที่แฟรกเมนต์ และบีบอัดจะใช้วิธีการเดียวกันในการแสดงดิสก์แอดเดรส ดิสก์แอดเดรสมีอยู่ในฟิลด์ **i_rdaddr** ของ i-nodes หรือในบล็อกทางอ้อม แฟรกเมนต์ทั้งหมดที่อ้างอิงในแอดเดรส เดียวต้องอยู่ต่อเนื่องกันบนดิสก์

รูปแบบดิสก์แอดเดรสประกอบด้วยฟิลด์ **nfrags** และ **addr** ฟิลด์เหล่านี้อธิบายพื้นที่ของดิสก์ที่ครอบคลุม โดยแอดเดรส:

`addr`
ระบุว่าแฟรกเมนต์ใดบนดิสก์ที่เป็นแฟรกเมนต์เริ่มต้น

`nfrags`
ระบุจำนวนแฟรกเมนต์ต่อเนื้องทั้งหมดที่ไม่ถูกใช้โดย แอดเดรส

ตัวอย่าง ถ้าขนาดแฟรกเมนต์สำหรับระบบไฟล์เป็น 512 ไบต์และ ขนาดโลจิคัลถูกแบ่งออกเป็นแปดแฟรกเมนต์ ค่า **nfrags** จะเป็น 3 ระบุว่าห้าแฟรกเมนต์ถูกรวมในแอดเดรส

ตัวอย่างต่อไปนี้แสดงค่าที่เป็นได้สำหรับฟิลด์ **addr** และ **nfrags** สำหรับดิสก์แอดเดรสที่ต่างกัน ค่าเหล่านี้สมมุติว่าขนาดแฟรกเมนต์เป็น 512 ระบุว่าบล็อกโลจิคัล ถูกแบ่งออกเป็นแปดแฟรกเมนต์

แอดเดรสสำหรับแฟรกเมนต์เดียว:

```
addr:   143
nfrags: 7
```

แอดเดรสนี้ระบุว่าตำแหน่งเริ่มต้นของข้อมูลคือแฟรกเมนต์ 143 บนดิสก์ ค่า **nfrags** ระบุว่า จำนวนแฟรกเมนต์ทั้งหมดที่รวมในแอดเดรสคือหนึ่ง ค่า **nfrags** เปลี่ยนแปลงในระบบไฟล์ที่มีขนาดแฟรกเมนต์ไม่ใช่ 512 ไบต์ เพื่อให้อ่านค่า **nfrags** ได้อย่างถูกต้อง ระบบ หรือผู้ใช้ที่ตรวจสอบแอดเดรส ต้องทราบขนาดแฟรกเมนต์ ของระบบไฟล์

แอดเดรสสำหรับห้าแฟรกเมนต์:

```
addr:   1117
nfrags: 3
```

ในกรณีนี้ แอดเดรสเริ่มต้นที่หมายเลขแฟรกเมนต์ 1117 บนดิสก์และ ต่อเนื่องกันห้าแฟรกเมนต์ (รวมแฟรกเมนต์เริ่มต้น) สามแฟรกเมนต์ ยังคงเหลืออยู่ ดังแสดงโดยค่า **nfrags**

ดิสก์แอดเดรสมีขนาด 32 บิต บิตจะเป็นตัวเลขตั้งแต่ 0 ถึง 31 บิต 0 จะถูกสำรองไว้เสมอ บิต 1 ถึง 3 มีฟิลด์ **nfrags** บิต 4 ถึง 31 มีฟิลด์ **addr**

JFS indirect blocks

JFS ใช้บล็อกทางอ้อม เพื่อแอดเดรสพื้นที่ดิสก์ที่จัดสรรให้แก้ไฟล์ที่มีขนาดใหญ่กว่า บล็อกทางอ้อมมี ความยืดหยุ่นอย่างมากสำหรับขนาดไฟล์ และมีการดึงข้อมูลออกที่เร็วที่สุด บล็อก ทางอ้อมถูกกำหนดโดยใช้ฟิลด์ **i_rindirect** ของ i-node ดิสก์ ฟิลด์นี้ใช้สำหรับการคำนวณหรือวิธีการกำหนด แอดเดรสพื้นที่ดิสก์ต่อไปนี้:

- Direct
- Single indirect
- Double indirect

แต่ละวิธีเหล่านี้ใช้รูปแบบดิสก์แอดเดรสเหมือนกันเป็นระบบไฟล์บีบอัด และแฟรกเมนต์ เนื่องจากไฟล์ขนาดใหญ่กว่า 32 KB ถูกจัดสรรด้วยแฟรกเมนต์ ขนาด 4096 ไบต์ ฟิลด์ **nfrags** สำหรับการแอดเดรสที่ใช้ วิธี single indirect หรือ double indirect จะมีค่าเป็น 0

Direct method

เมื่อใช้วิธี direct ของการกำหนดแอดเดรสดิสก์ แต่ละค่าของแอดแอดเดรส ที่แสดงรายการในฟิลด์ **i_rdaddr** ของ i-node ดิสก์ชี้โดยตรงไปยังการจัดสรรเดียวของดิสก์แฟรกเมนต์ ขนาดใหญ่สุดของไฟล์ ที่ใช้การคำนวณโดยตรงคือ 32,768 ไบต์ (32KB) หรือ 8 x 4096 ไบต์ เมื่อ ไฟล์ต้องการมากกว่า 32 KB จะใช้บล็อกทางอ้อมเพื่อแอดเดรสพื้นที่ดิสก์ ของไฟล์

Single indirect method

ฟิลด์ **i_rindirect** มีแอดเดรสที่ชี้ ไปยังบล็อกทางอ้อมแบบเดี่ยว หรือบล็อกทางอ้อมแบบคู่ เมื่อใช้วิธีการกำหนดแอดเดรสดิสก์ทางอ้อมเดี่ยว ฟิลด์ **i_rindirect** มีแอดเดรสของบล็อกทางอ้อมที่มี 1024 แอดเดรส แอดเดรสเหล่านี้ชี้ไปที่ดิสก์แฟรกเมนต์สำหรับแต่ละการจัดสรร การใช้ การคำนวณบล็อกทางอ้อมแบบเดี่ยว ไฟล์สามารถมีขนาดได้ถึง 4,194,304 ไบต์ (4 MB) หรือ 1024 x 4096 ไบต์

Double indirect method

วิธีการกำหนดแอดเดรสดิสก์ทางอ้อมแบบคู่ใช้ฟิลด์ **i_rindirect** เพื่อชี้ไปยังบล็อกทางอ้อมแบบคู่ บล็อกทางอ้อมแบบคู่ มี 512 แอดเดรสที่ชี้ไปยังบล็อกทางอ้อม ซึ่งมีตัวชี้ไปยัง การจัดสรรแฟรกเมนต์ ขนาดไฟล์ใหญ่สุดที่สามารถใช้กับการคำนวณ ทางอ้อมแบบคู่ในระบบไฟล์ที่ไม่เปิดใช้สำหรับไฟล์ขนาดใหญ่คือ 2,147,483,648 ไบต์ (2 GB) หรือ 512(1024 x 4096) ไบต์

หมายเหตุ: ขนาดไฟล์ใหญ่สุดที่การเรียกใช้ระบบ **read** และ **write** จะอนุญาตคือ 2 GB ลบ 1 ($2^{31}-1$) เมื่อใช้อินเตอร์เฟสการแม็พหน่วยความจำ สามารถกำหนดแอดเดรสได้ 2 GB

ระบบไฟล์ที่เปิดใช้ไฟล์ขนาดใหญ่อนุญาตให้มีขนาดไฟล์ใหญ่สุด เล็กว่า 64 กิกะไบต์เล็กน้อย (68589453312) บล็อกทางอ้อมแบบเดี่ยว บล็อกแรกมีแฟรกเมนต์ 4096 ไบต์ และบล็อกทางอ้อมแบบเดี่ยวต่อมาทั้งหมด มีแฟรกเมนต์ (32 X 4096) ไบต์ ต่อไปนี้จะมีขนาดไฟล์ใหญ่สุด สำหรับระบบไฟล์ที่เปิดใช้ไฟล์ขนาดใหญ่:

```
(1 * (1024 * 4096)) + (511 * (1024 * 131072))
```

การจัดสรรแฟรกเมนต์ที่กำหนดให้แก่ไดเร็กทอรีถูกแบ่งออกเป็นเร็กคอร์ดละ 512 ไบต์ และขยายขึ้นตามการจัดสรรของเร็กคอร์ดเหล่านี้

โควต้า

ดิสก์โควต้าจำกัด จำนวนพื้นที่ระบบไฟล์ที่ผู้ใช้คนหนึ่งหรือกลุ่มหนึ่งจะสามารถครอบครองได้

รูทีนย่อย **quotactl** ตั้งค่าขีดจำกัดทั้งจำนวนไฟล์และจำนวนดิสก์บล็อกที่จัดสรร ให้แก่ผู้ใช้หรือกลุ่มแต่ละรายบนระบบไฟล์ โควต้าบังคับใช้กับขีดจำกัด ประเภทต่อไปนี้:

hard ขีดจำกัดสูงสุดที่อนุญาต เมื่อการประมวลผลถึงขีดจำกัด hard การร้องขอ พื้นที่เพิ่มขึ้นจะล้มเหลว

soft ขีดจำกัดในทางปฏิบัติ ถ้าการประมวลผลถึงขีดจำกัด soft จะมีคำเตือนพิมพ์ ออกที่เทอร์มินัลของผู้ใช้ คำเตือนนี้มักถูกแสดงตอนล็อกอิน ถ้าผู้ใช้ ไม่สามารถแก้ไขปัญหาได้หลังเซสชันล็อกอินผ่านไปหลายครั้ง ขีดจำกัด soft จะเปลี่ยนเป็นขีดจำกัด hard

คำเตือนระบบถูกออกแบบให้กระตุ้นให้ผู้ใช้ให้ความเอาใจใส่ต่อขีดจำกัด soft อย่างไรก็ตาม ระบบโควต้าอนุญาตให้การประมวลผลเข้าถึง ขีดจำกัด hard ที่สูงขึ้นไปได้เมื่อจำเป็นต้องใช้รีซอร์สมากขึ้นชั่วคราว

การจัดสรรพื้นที่ไฟล์ JFS2

การจัดสรรพื้นที่ไฟล์คือ วิธีการที่ข้อมูลถูกแบ่งส่วนพื้นที่เก็บข้อมูลฟิสิคัลใน ระบบปฏิบัติการ

เคอร์เนลจัดสรรพื้นที่ดิสก์ให้แก่ ไฟล์หรือไดเร็กทอรีในรูปของ *บล็อกโลจิคัล* บล็อกโลจิคัลหมายถึงการแบ่งเนื้อหาของไฟล์หรือไดเร็กทอรีออกเป็นหน่วยขนาด 512, 1024, 2048 หรือ 4096 ไบต์ เมื่อสร้างระบบไฟล์ JFS2 ขนาด บล็อกโลจิคัลถูกระบุให้เป็นค่าใดค่าหนึ่งใน 512, 1024, 2048 หรือ 4096 ไบต์ อย่างไรก็ตาม บล็อกโลจิคัลไม่สิ่งที่มีอยู่จริง ข้อมูลในบล็อกโลจิคัลจะใช้ พื้นที่เก็บข้อมูลฟิสิคัลบนดิสก์ แต่ละไฟล์หรือไดเร็กทอรีมีบล็อกโลจิคัลอย่างน้อย 0 หรือมากกว่า 0 บล็อก

บล็อกโลจิคัลแบบเต็มและบางส่วน

ไฟล์หรือไดเร็กทอรีอาจมีบล็อกโลจิคัลเต็มหรือ บางส่วน บล็อก โลจิคัลเต็มมีข้อมูล 512, 1024, 2048 หรือ 4096 ไบต์ ขึ้นอยู่กับ ขนาดบล็อกของระบบไฟล์ที่ระบุเมื่อสร้างระบบไฟล์ JFS2 บล็อกโลจิคัลบางส่วนเกิดขึ้นเมื่อบล็อกโลจิคัลสุดท้ายของไฟล์หรือไดเร็กทอรี มีน้อยกว่าขนาดบล็อกระบบไฟล์ของข้อมูล

ตัวอย่าง ระบบไฟล์ JFS2 ที่มีขนาดบล็อกโลจิคัลเป็น 4096 กับไฟล์ขนาด 8192 ไบต์คือสองบล็อกโลจิคัล 4096 ไบต์แรกอยู่ในบล็อกโลจิคัลแรก และ 4096 ไบต์ที่เหลืออยู่ใน บล็อกโลจิคัลที่สอง ในทำนองเดียวกัน ไฟล์ขนาด 4608 ไบต์ใช้บล็อก โลจิคัลสองบล็อก อย่างไรก็ตาม บล็อกโลจิคัลสุดท้ายเป็นบล็อกโลจิคัลบางส่วน ที่มีเฉพาะ 512 ไบต์สุดท้ายของข้อมูลของไฟล์

การจัดสรรพื้นที่ไฟล์ JFS2

ขนาดบล็อกดีฟอลต์คือ 4096 ไบต์ คุณสามารถระบุ ขนาดบล็อกเล็กกว่านี้ได้ด้วยคำสั่ง **mkfs** ระหว่าง การสร้างของระบบไฟล์ ขนาดบล็อกที่ใช้ได้คือ 512, 1024, 2048 และ 4096 ไบต์ คุณสามารถใช้ ขนาดบล็อกได้ขนาดเดียวเท่านั้นในระบบไฟล์

เคอร์เนลจัดสรรพื้นที่ดิสก์ดังนั้นบล็อกของข้อมูลระบบไฟล์บล็อกสุดท้าย เท่านั้นที่รับการจัดสรรบล็อกบางส่วน ขณะที่บล็อกบางส่วนที่ขนาดใหญ่ขึ้นเกินขีดจำกัด ของการจัดสรรปัจจุบัน จะมีการจัดสรรบล็อกเพิ่ม

การจัดสรรบล็อกใหม่ยังเกิดขึ้นถ้าข้อมูลถูกเพิ่มในบล็อกโลจิคัลที่ แทนช่องว่างไฟล์ *ช่องว่างไฟล์* คือบล็อกโลจิคัล ว่างที่อยู่ก่อนหน้าบล็อกโลจิคัลสุดท้ายที่เก็บข้อมูล (ช่องว่างไฟล์ ไม่เกิดขึ้นภายในไดเร็กทอรี) บล็อกโลจิคัลว่างเหล่านี้ไม่ใช่บล็อกที่จัดสรร อย่างไรก็ตาม เมื่อข้อมูลถูกเพิ่มในช่องว่างไฟล์ จะเกิดการจัดสรร แต่ละ บล็อกโลจิคัลที่ไม่ใช่พื้นที่ดิสก์ที่จัดสรรก่อนหน้านี้จะถูกจัดสรรเป็นจำนวนพื้นที่ ของบล็อกระบบไฟล์

การจัดสรรบล็อกเพิ่มไม่จำเป็นถ้าข้อมูลที่มีตรงกลาง ของไฟล์หรือไดเร็กทอรีถูกเขียนทับ บล็อกโลจิคัลที่เก็บข้อมูลที่มีอยู่มีบล็อกที่จัดสรรแล้ว

JFS2 พยายามคงการจัดสรรให้ต่อเนื่องสำหรับบล็อกโลจิคัลของไฟล์ หรือไดเร็กทอรี บนดิสก์ การคงการจัดสรรไว้ให้ต่อเนื่องกันจะช่วยลดเวลาการค้นหาเนื่องจาก ข้อมูลสำหรับไฟล์หรือไดเร็กทอรีสามารถเข้าถึงได้แบบเรียงลำดับต่อกัน และพบ ได้บนพื้นที่เดียวกันในดิสก์ พื้นที่ดิสก์ที่ต้องการสำหรับการจัดสรรแบบต่อเนื่อง อาจไม่พร้อมใช้งาน ถ้ามีไฟล์หรือไดเร็กทอรีถูกเขียน อยู่แล้ว

ระบบไฟล์ใช้บิตแมพชื่อ *แม็พการจัดสรร บล็อก* เพื่อบันทึกสถานะของทุกบล็อกในระบบไฟล์ เมื่อ ระบบไฟล์ต้องการจัดสรร บล็อกใหม่ ระบบจะอ้างถึงแม็พการจัดสรร บล็อกเพื่อระบุว่าบล็อกใดที่สามารถใช้ได้ บล็อกสามารถถูกจัดสรร ให้แก่ไฟล์หรือไดเร็กทอรีเดี่ยวเท่านั้นในหนึ่งครั้ง

Extents

extent คือลำดับของบล็อกระบบไฟล์ที่มีความยาวผันแปรได้ที่ต่อเนื่องกัน ที่จัดสรรให้แก่อ็อบเจ็กต์ JFS2 เป็นหนึ่งหน่วย extents ขนาดใหญ่อาจแตกออกเป็นกลุ่มการจัดสรร หลายกลุ่ม

i-node แทนทุกอ็อบเจ็กต์ JFS2 I-nodes มีข้อมูลเฉพาะอ็อบเจ็กที่ควรมี เช่นการประทับเวลา หรือประเภทไฟล์ (ปกติ หรือไดเร็กทอรี และ อื่นๆ) รวมทั้งยังมี B+ เพื่อบันทึกการจัดสรรของ extents

ความยาวและค่าแอดเดรสจำเป็นสำหรับกำหนด extent ความยาว ถูกวัดเป็นหน่วยของขนาดบล็อกระบบไฟล์ ค่า 24 บิตแสดงความยาวของ extent ดังนั้นหนึ่ง extent สามารถมีขนาดได้ตั้งแต่ 1 ถึง 2^{24} -1 บล็อกระบบไฟล์ ดังนั้น ขนาดของ extent ใหญ่สุดขึ้นอยู่กับ ขนาดบล็อกระบบไฟล์ แอดเดรสคือแอดเดรสของบล็อกแรกของ extent แอดเดรสยังอยู่ในหน่วยของบล็อกระบบไฟล์ เป็น บล็อกออฟเซ็ตจากจุดเริ่มต้นของระบบไฟล์

ระบบไฟล์ที่เป็นแบบ extent ประกอบด้วยขนาดบล็อกระบบไฟล์ที่ผู้ใช้ระบุ ที่อนุญาตให้ JFS2 ไม่ต้องมีการสนับสนุนสำหรับการแตกแฟรกเมนต์ภายในแยก คุณสามารถตั้งค่าระบบไฟล์ด้วยขนาดบล็อกระบบไฟล์ขนาดเล็ก เช่น 512 ไบต์ เพื่อลดขนาดการแตกแฟรกเมนต์ภายในให้เล็กสุดสำหรับระบบไฟล์ที่มี จำนวนของไฟล์ขนาดเล็กจำนวนมาก

โดยทั่วไป นโยบายการจัดสรรสำหรับ JFS2 พยายามให้มีขนาดการจัดสรร ต่อเนื่องจากใหญ่สุดโดยการอนุญาตให้มีจำนวน extents ต่ำสุด โดยที่แต่ละ extent มีขนาดใหญ่ และต่อเนื่องได้เท่าที่จำเป็น นี้อนุญาตสำหรับการถ่ายโอน I/O ขนาดใหญ่ ซึ่งจะส่งผล ต่อผลการทำงานที่ปรับปรุง อย่างไรก็ตาม ในบางกรณี การทำนี้อาจไม่สามารถทำได้

B+ trees

โครงสร้างข้อมูล B+ tree ถูกใช้สำหรับไฟล์เลย์เอาต์ การดำเนินการส่วนใหญ่ ที่ JFS2 ดำเนินการคือการอ่านและการเขียน extents B+ trees ถูกใช้เพื่อช่วย ให้เกิดผลการทำงานของการดำเนินการเหล่านี้

descriptor การจัดสรร extent (โครงสร้าง **xad_t**) อธิบาย extent และเพิ่มสองฟิลด์ที่จำเป็นสำหรับการแสดง ไฟล์: ฟิลด์ออฟเซ็ตซึ่งอธิบายโลจิคัลไบต์แอดเดรสที่ extent แสดง และฟิลด์แฟล็ก โครงสร้าง **xad_t** ถูกกำหนดใน ไฟล์ **/usr/include/j2/j2_xtree.h**

โครงสร้าง **xad** อธิบายช่วงเชิงหลักการสองช่วง:

- ช่วงฟิสิคัลของบล็อกดิสก์ ค่านี้เริ่มที่แอดเดรส addressXAD(xadp) หมายเลขบล็อก ของระบบไฟล์ และ extends สำหรับบล็อกระบบไฟล์ lengthXAD(xadp)
- ช่วงโลจิคัลของไบต์ภายในไฟล์ ค่านี้เริ่มที่จำนวนไบต์ offsetXAD(xadp)*(file system block size) และ extends ที่ lengthXAD(xadp)*(file system block size)

ช่วงฟิสิคัลและช่องโลจิคัลมีความยาวจำนวนไบต์ เท่านั้น โปรดทราบว่าออฟเซ็ตถูกเก็บเป็นหน่วยของขนาดบล็อกระบบไฟล์ (ตัวอย่าง ค่า 3) ในออฟเซ็ตหมายถึง 3 บล็อกระบบไฟล์ ไม่ใช่ 3 ไบต์ Extents ภายในไฟล์ถูกจัดให้อยู่ที่ขอบของขนาดบล็อกระบบไฟล์เสมอ

ข้อจำกัด JFS2

JFS2 ต้องการพื้นที่ว่างยาวต่อเนื่องกันอย่างน้อยหนึ่งหน้า หรือ 4 KB เมื่อไฟล์ถูกขยายเพิ่ม ถ้าคุณไม่มีพื้นที่ว่างต่อเนื่องกันอย่างน้อย 4 KB ระบบไฟล์จะไม่อนุญาตให้มีส่วนขยายของไฟล์ แม้ว่าจะมีพื้นที่เก็บข้อมูลเหลือเพียงพอในขนาดที่เล็กลง

โครงร่างระบบไฟล์ JFS

ระบบไฟล์คือชุดของไฟล์ ไดเร็กทอรี และโครงสร้างอื่นๆ

ระบบไฟล์เก็บรักษาข้อมูลและระบุตำแหน่งของข้อมูล ของไฟล์หรือไดเร็กทอรีบนดิสก์ นอกเหนือจากไฟล์ และไดเร็กทอรีแล้ว ระบบไฟล์ JFS ยังมีบล็อกการบูต superblock บิตแม็พ และกลุ่มการจัดสรรอย่างน้อยหนึ่งกลุ่ม แต่ละระบบไฟล์จะครอบครองพื้นที่หนึ่งโลจิคัลวอลุ่ม

บล็อกการบูต JFS

บล็อกการบูตครอบครอง 4096 ไบต์แรก ของระบบไฟล์ เริ่มที่ไบต์ออฟเซ็ต 0 บนดิสก์ บล็อกการบูต พร้อมใช้งานตอนเริ่มทำงานระบบปฏิบัติการ

JFS superblock

superblock คือบล็อกขนาด 4096 ไบต์และ เริ่มต้นที่ไบต์ออฟเซ็ต 4096 บนดิสก์ superblock เก็บรักษาข้อมูล เกี่ยวกับระบบไฟล์ทั้งระบบและรวมฟิลด์ต่อไปนี้:

- ขนาดของระบบไฟล์
- จำนวนบล็อกข้อมูลในระบบไฟล์
- แฟล็กที่ระบุสถานะของระบบไฟล์
- ขนาดกลุ่มการจัดสรร

บิตแม็พการจัดสรร JFS

ระบบไฟล์มี บิตแม็พการจัดสรรต่อไปนี้:

- แม็พการจัดสรรแฟรกเมนต์บันทึกสถานะการจัดสรรของแต่ละแฟรกเมนต์
- แม็พการจัดสรร i-node ดิสก์บันทึกสถานะของแต่ละ i-node

แฟรกเมนต์ JFS

ระบบไฟล์ หลายระบบมีบล็อกดิสก์ หรือบล็อกข้อมูล บล็อกเหล่านี้แบ่งดิสก์ออกเป็น ส่วนที่มีขนาดเท่าๆ กันเพื่อเก็บข้อมูลในโลจิคัลบล็อกของไฟล์หรือไดเร็กทอรี บล็อกดิสก์ยังอาจถูกแบ่งออกเป็นหน่วยการจัดสรรที่มีขนาดคงที่เรียกว่า *แฟรกเมนต์* บางระบบไม่อนุญาตให้มีการจัดสรรแบบแฟรกเมนต์ ที่จะแตกขอบเขตของบล็อกดิสก์ หรืออีกนัยหนึ่ง โลจิคัลบล็อกไม่สามารถ ถูกจัดสรรเป็นแฟรกเมนต์จากดิสก์บล็อกที่ต่างกัน

อย่างไรก็ตาม journaled file system (JFS) ได้จัดให้มีมุมมองของระบบไฟล์ เป็นชุดต่อเนื่องของแฟรกเมนต์ แฟรกเมนต์ JFS คือหน่วยการจัดสรร พื้นฐานและดิสก์ถูกกำหนดแอดเดรสที่ระดับแฟรกเมนต์ ดังนั้นการจัดสรรแฟรกเมนต์ สามารถแตกขอบเขตของสิ่งใดๆ ที่มิใช่บล็อกดิสก์ ขนาด แฟรกเมนต์ JFS ดีฟอลต์คือ 4096 ไบต์ คุณสามารถระบุขนาดเล็กลงได้ นอกเหนือจาก เก็บข้อมูลสำหรับไฟล์และไดเร็กทอรีแล้ว แฟรกเมนต์ยังเก็บ ดิสก์แอดเดรสและข้อมูลสำหรับบล็อกโดยอ้อม

กลุ่มการจัดสรร JFS

ชุดของแฟรกเมนต์ที่ประกอบเป็นระบบไฟล์จะถูกแบ่งออกเป็น หน่วยของแฟรกเมนต์ที่ต่อกันที่มีขนาดคงที่หนึ่งหรือหลาย หน่วย แต่ละหน่วยเป็นหนึ่ง *หน่วยการจัดสรร* กลุ่มแรกของกลุ่มเหล่านี้เริ่มระบบไฟล์และ มีพื้นที่ที่สำรองไว้ที่ครอบครองพื้นที่ 32 x 4096 ไบต์แรกของกลุ่ม 4096 ไบต์แรกของพื้นที่นี้เก็บค่าบล็อกการบูต 4096 ไบต์ที่สองเก็บค่า superblock ระบบไฟล์

แต่ละกลุ่มการจัดสรรมีจำนวน i-nodes ดิสก์ที่ต่อเนื่องกันเป็นจำนวนแบบสแตติก ที่ครอบครองแฟรกเมนต์ของกลุ่มบางส่วน แฟรกเมนต์เหล่านี้ถูกสำรองไว้สำหรับ i-nodes เมื่อทำการสร้างระบบไฟล์และเวลาการขยาย สำหรับกลุ่มการจัดสรร แรก i-nodes ดิสก์ครอบครองแฟรกเมนต์ที่ต่อจากพื้นที่บล็อก ที่ถูกสำรองไว้ สำหรับกลุ่มต่อๆ มา i-nodes ดิสก์จะถูกพบที่ตำแหน่ง เริ่มต้น ของแต่ละกลุ่ม i-nodes ดิสก์มีขนาด 128 ไบต์และถูกระบุโดย หมายเลข i-node ดิสด์หรือ i-number ค่าเฉพาะ i-number จะแม็พ i-node ดิสก์กับ ตำแหน่งบนดิสก์ หรือไปยัง i-node ภายในกลุ่มการจัดสรร

ขนาดกลุ่มการจัดสรรแฟรกเมนต์ของระบบไฟล์และขนาดกลุ่มการจัดสรร i-node ดิสก์จะถูกระบุเป็นจำนวนแฟรกเมนต์ และ i-nodes ดิสก์ที่ มีอยู่ในแต่ละกลุ่มการจัดสรร ขนาดกลุ่มการจัดสรรดีฟอลต์คือ 8 MB แต่สามาถขนาดขนาดได้ถึง 64 MB ค่าเหล่านี้ถูกเก็บใน superblock ระบบไฟล์ และถูกตั้งค่าในตอนเริ่มการสร้างระบบไฟล์

กลุ่มการจัดสรรอนุญาตให้นโยบายการจัดสรรรีซอร์ส JFS ใช้วิธีการ อันมีประสิทธิภาพเพื่อให้เกิดผลการทำงาน I/O ของระบบไฟล์สูงสุด นโยบายการจัดสรร เหล่านี้พยายามแบ่งส่วนบล็อกดิสก์และ i-nodes ดิสก์สำหรับข้อมูลที่เกี่ยวข้องเพื่อให้ เกิดการใช้งานตำแหน่งสำหรับดิสด์ดีที่สุด ส่วนใหญ่ไฟล์ถูกอ่านและเขียนแบบเรียงลำดับ และไฟล์ภายในไดเร็กทอรีส่วนใหญ่ถูกเข้าถึงพร้อมกัน รวมทั้งนโยบายการจัดสรร เหล่านี้พยายามที่จะกระจายข้อมูลที่ไม่เกี่ยวข้องกันออกไปทั่วระบบไฟล์เพื่อ พยายามลดการแตกแฟรกเมนต์ของพื้นที่ว่างให้น้อยที่สุด

JFS disk i-nodes

โลจิคัลบล็อกข้อมูลของไฟล์หรือ ไดเร็กทอรีเป็นจำนวนหน่วยขนาด 4096 ไบต์ แต่ละโลจิคัลบล็อกถูกจัดสรรเป็นแฟรกเมนต์สำหรับเป็นที่เก็บข้อมูล แต่ละไฟล์และไดเร็กทอรีมี i-node ที่มี ข้อมูลการเข้าถึงเช่นประเภทไฟล์ สิทธิในการเข้าถึง ID ของเจ้าของ และ จำนวนลิงก์ไปยังไฟล์นั้น i-nodes เหล่านี้ยังมี "แอดเดรส" สำหรับค้นหา ตำแหน่งบนดิสก์ที่ข้อมูลสำหรับโลจิคัลบล็อกถูกเก็บ

แต่ละ i-node มีอาร์เรย์ของส่วนที่กำหนดหมายเลข แต่ละส่วนมี แอดเดรสสำหรับหนึ่งในโลจิคัลบล็อกของไฟล์หรือไดเร็กทอรี แอดเดรสเหล่านี้ บ่งชี้แฟรกเมนต์เริ่มต้นและจำนวนแฟรกเมนต์ทั้งหมดที่รวม ในการจัดสรรเดียว ตัวอย่าง ไฟล์ที่มีขนาด 4096 ไบต์มี แอดเดรสเดียวบนอาร์เรย์ของ i-node ข้อมูลของ 4096 ไบต์นั้นถูกเก็บ ในโลจิคัลบล็อกเดียว ไฟล์ขนาดใหญ่ขึ้นที่มีขนาด 6144 ไบต์มี สองแอดเดรส หนึ่งแอดเดรสเก็บ 4096 ไบต์แรก และแอดเดรสที่สอง เก็บ 2048 ที่เหลือ (โลจิคัลบล็อกบางส่วน) ถ้าไฟล์มีโลจิคัลบล็อกจำนวนมาก i-node ไม่เก็บดิสก์แอดเดรส แต่ i-node จะชี้ไปยังบล็อกโดยอ้อมที่เก็บแอดเดรสเพิ่มเติม แทน

โครงร่างระบบไฟล์ JFS2

ระบบไฟล์คือชุดของไฟล์, ไดเร็กทอรี และโครงสร้างอื่นๆ

ระบบไฟล์ เก็บรักษาข้อมูลและระบุตำแหน่งของข้อมูล บนดิสก์สำหรับไฟล์และไดเร็กทอรี นอกเหนือจากไฟล์และไดเร็กทอรีแล้ว ระบบไฟล์ JFS2 ยังมี superblock, แม็พการจัดสรร และกลุ่มการจัดสรรอย่างน้อยหนึ่ง กลุ่ม แต่ละระบบไฟล์จะครอบครองพื้นที่หนึ่งโลจิคัลวอลุ่ม

JFS2 superblock

superblock คือบล็อกขนาด 4096 ไบต์และเริ่มต้นที่ไบต์ออฟเซ็ต 32768 บน ดิสก์ superblock เก็บรักษาข้อมูล เกี่ยวกับระบบไฟล์ทั้งระบบและรวมฟิลด์ต่อไปนี้:

- ขนาดของระบบไฟล์
- จำนวนบล็อกข้อมูลในระบบไฟล์
- แฟล็กที่ระบุสถานะของระบบไฟล์
- ขนาดกลุ่มการจัดสรร
- ขนาดบล็อกระบบไฟล์

แม็พการจัดสรร JFS2

ระบบไฟล์มีแม็พการจัดสรรต่อไปนี้:

- แม็พการจัดสรร i-node บันทึกตำแหน่งและการจัดสรรของ i-nodes ทั้งหมด ในระบบไฟล์
- แม็พการจัดสรรบล็อกบันทึกสถานะการจัดสรรของแต่ละบล็อก ระบบไฟล์

JFS2 disk i-nodes

โลจิคัลบล็อกมีข้อมูลของไฟล์หรือไดเร็กทอรีเป็นจำนวนหน่วยของบล็อก ระบบไฟล์ แต่ละโลจิคัลบล็อกถูกจัดสรรบล็อกระบบไฟล์สำหรับเป็นพื้นที่จัดเก็บ ข้อมูล แต่ละไฟล์และไดเร็กทอรีมี i-node ที่มี ข้อมูลการเข้าถึงเช่นประเภทไฟล์, สิทธิในการเข้าถึง, ID ของเจ้าของ และ จำนวนลิงก์ไปยังไฟล์นั้น i-nodes เหล่านี้ยังมี B+ tree สำหรับค้นหาตำแหน่งบน ดิสก์ที่ข้อมูลสำหรับโลจิคัลบล็อกถูกเก็บ

กลุ่มการจัดสรร JFS2

กลุ่มการจัดสรรแบ่งพื้นที่บนระบบไฟล์ออกเป็น chunks กลุ่ม การจัดสรรถูกเป็นเทคนิคในการแก้ไขปัญหาเท่านั้นซึ่งเป็นวิธีแก้ปัญหาที่เหมาะสมที่สุด พบโดยการพยายามด้วยวิธีการอื่น และถูกเลือกในลำดับขั้น ต่อมาของโปรแกรมสำหรับการใช้ในขั้นตอนต่อไปของโปรแกรม กลุ่ม การจัดสรรอนุญาตให้นโยบายการจัดสรรรีซอร์ส JFS2 ใช้วิธีการที่เป็นที่รู้จักดีเพื่อ ให้เกิดผลการทำงาน I/O สูงสุด อันดับแรก นโยบายการจัดสรร เหล่านี้พยายามแบ่งส่วนบล็อกดิสก์และ i-nodes ดิสก์สำหรับข้อมูลที่เกี่ยวข้องเพื่อให้ เกิดการใช้งานตำแหน่งสำหรับดิสด์ดีที่สุด ส่วนใหญ่ไฟล์ถูกอ่านและเขียนแบบเรียงลำดับ และไฟล์ภายในไดเร็กทอรีส่วนใหญ่ถูกเข้าถึงพร้อมกัน อันดับที่สอง นโยบายการจัดสรรพยายาม กระจายข้อมูลที่ไม่เกี่ยวข้องออกไปทั่วระบบไฟล์เพื่อสะดวก ต่อการจัดตำแหน่งดิสก์

กลุ่มการจัดสรรภายในระบบไฟล์ถูกระบุโดยดัชนีกลุ่มการจัดสรรที่เริ่มที่ ตำแหน่งศูนย์ เป็นหมายเลขกลุ่มการจัดสรร

ขนาดกลุ่มการจัดสรรต้องถูกเลือกยอมรับกลุ่มการจัดสรรที่ มีขนาดใหญ่เพียงพอที่จะจัดให้มีการจัดสรรรีซอร์สแบบต่อเนื่องกันได้ในช่วงเวลาหนึ่ง กลุ่มการจัดสรรถูกจำกัดให้มีจำนวนกลุ่มสูงสุด 128 กลุ่ม ขนาด กลุ่มการจัดสรรสูงสุดคือ 8192 บล็อกระบบไฟล์

กลุ่มการจัดสรร Partial

ระบบไฟล์ที่ไม่ได้มีขนาดเป็นจำนวนผลคูณของขนาดกลุ่มการจัดสรร จะมีกลุ่มการจัดสรรบางส่วน กลุ่มการจัดสรรล่าสุดของระบบไฟล์ที่ไม่ได้ครอบครองดิสก์บล็อกเต็มบล็อก กลุ่มการจัดสรรบางส่วนนี้ จะถูกถือเป็นกลุ่มการจัดสรรที่สมบูรณ์ ยกเว้นว่าดิสก์บล็อกที่ไม่มีอยู่ ถูกทำเครื่องหมายเป็นถูกจัดสรรอยู่ในแม็พการจัดสรรบล็อก

การเขียนโปรแกรมที่เข้าถึงไฟล์ขนาดใหญ่

AIX สนับสนุนไฟล์ที่มีขนาดใหญ่กว่า 2 กิกะไบต์ (2 GB) ส่วนนี้ จะช่วยโปรแกรมเมอร์ให้ทำความเข้าใจในความหมายของไฟล์ขนาดใหญ่สำหรับแอ็พพลิเคชัน และยังช่วยโปรแกรมเมอร์ในการแก้ไขแอ็พพลิเคชันของตนเอง แอ็พพลิเคชันโปรแกรมสามารถแก้ไขได้ผ่านโปรแกรมมิงอินเตอร์เฟส เพื่อให้รับรู้ไฟล์ที่มีขนาดใหญ่ โปรแกรมมิงอินเตอร์เฟสของระบบไฟล์จะอ้างอิงตามชนิดข้อมูล **off_t**

ความหมายสำหรับโปรแกรมที่มีอยู่

สภาวะแวดล้อมสำหรับแอ็พพลิเคชันแบบ 32 ที่แอ็พพลิเคชันทั้งหมดใช้ก่อนหน้า AIX 4.2 ยังคงไม่เกิดการเปลี่ยนแปลง อย่างไรก็ตาม แอ็พพลิเคชันโปรแกรมที่มีอยู่ไม่สามารถจัดการไฟล์ที่มีขนาดใหญ่ได้

ตัวอย่างเช่น ฟิลด์ **st_size** ที่อยู่ในโครงสร้าง **stat** ซึ่งจะถูกใช้เพื่อส่งคืนขนาดของไฟล์ จะถูกลงนามด้วยความยาวแบบ 32 บิต ดังนั้น โครงสร้าง **stat** นั้นจึงไม่สามารถใช้เพื่อส่งคืนขนาดของไฟล์ที่ใหญ่กว่า LONG_MAX ได้ ถ้าแอ็พพลิเคชันพยายามใช้รูทีนย่อย **stat** กับไฟล์ที่มีขนาดใหญ่กว่า LONG_MAX รูทีนย่อย **stat** จะเกิดความล้มเหลว และ errno จะมีค่าเป็น EOVERFLOW ซึ่งบ่งชี้ว่า ขนาดของไฟล์โอเวอร์โฟลว์ฟิลด์ขนาดของโครงสร้างที่จะถูกใช้โดยโปรแกรม

ลักษณะการทำงานนี้จะมีลักษณะสำคัญ เนื่องจากโปรแกรมที่มีอยู่อาจไม่ได้ปรากฏว่ามีผลกระทบใดๆ ที่เป็นผลมาจากไฟล์ที่มีขนาดใหญ่ ซึ่งเกิดความล้มเหลวในไฟล์ขนาดใหญ่ที่มีอยู่ แม้ว่า ขนาดของไฟล์นั้นจะไม่สัมพันธ์กันก็ตาม

errno EOVERFLOW ยังสามารถส่งคืนได้โดยตัวชี้ **lseek** และโดยรูทีนย่อย **fcntl** ถ้าค่าที่ต้องการส่งคืนนั้นมีขนาดใหญ่กว่าชนิดข้อมูล หรือโครงสร้างที่โปรแกรมกำลังใช้อยู่ สำหรับ **lseek** ถ้าออฟเซ็ตผลลัพธ์มีขนาดใหญ่กว่า LONG_MAX แล้ว **lseek** จะเกิดความล้มเหลว และ errno จะมีค่า EOVERFLOW สำหรับรูทีนย่อย **fcntl** ถ้าตัวเรียกใช้ F_GETLK และบล็อกออฟเซ็ตเริ่มต้นของล็อก หรือความยาวมีขนาดใหญ่กว่า LONG_MAX แล้ว การเรียก **fcntl** จะล้มเหลว และ errno จะมีค่า EOVERFLOW

การป้องกันการเปิด

แอ็พพลิเคชันโปรแกรมจำนวนมากที่มีอยู่ อาจมีลักษณะการทำงานที่ไม่คาดไม่ถึง ซึ่งรวมถึงความเสียหายของข้อมูล หากอนุญาตให้ดำเนินการกับไฟล์ที่มีขนาดใหญ่ AIX ใช้ scheme เปิดการปกป้อง เพื่อเปิดการปกป้องแอ็พพลิเคชันจากคลาสที่เกิดความล้มเหลวนี้

นอกจากเปิดการปกป้องแล้ว จำนวนของรูทีนย่อยอื่นๆ นำเสนอการปกป้องด้วยการจัดเตรียมสภาวะแวดล้อมที่สามารถเรียกทำงานได้ ซึ่งมีลักษณะเฉพาะกับสภาวะแวดล้อมภายใต้การพัฒนาโปรแกรมเหล่านี้ ถ้าแอ็พพลิเคชันใช้รูทีนย่อยในตระกูล **write** และคำร้องขอ **write** ข้ามขอบเขตที่มีขนาด 2 GB รูทีนย่อย **write** จะถ่ายโอนข้อมูลมากที่สุด 2 GB ลบ 1 ถ้าแอ็พพลิเคชันพยายามเขียนลงขอบเขตที่มีขนาดใกล้ 2GB – 1 รูทีนย่อย **write** จะล้มเหลว และตั้งค่า errno ให้มีค่า EFBIG ลักษณะการทำงานของรูทีนย่อย **mmap, ftruncate** และ **fclear** จะคล้ายกัน

รูทีนย่อยในตระกูล **read** ยังมีส่วนร่วมใน scheme เปิดการปกป้อง ถ้าแอ็พพลิเคชันพยายามอ่านไฟล์ข้าม threshold ที่มีขนาด 2 GB เฉพาะข้อมูลที่มีขนาดมากที่สุด 2 GB ลบ 1 เท่านั้นที่จะถูกอ่าน การอ่านที่ไฟล์ที่มีขนาดใกล้ 2GB – 1 จะล้มเหลว และ errno จะมีค่า EOVERFLOW

เปิดการปกป้องจะถูกนำมาปฏิบัติด้วยแฟล็กที่เชื่อมโยงกับ file description ที่เปิดอยู่ สถานะปัจจุบันของแฟล็กสามารถเคียวรีด้วยรูทีนย่อย **fcntl** ได้โดยใช้คำสั่ง F_GETFL ซึ่งแฟล็กสามารถแก้ไขได้ด้วยรูทีนย่อย **fcntl** โดยใช้คำสั่ง F_SETFL

เนื่องจากการเปิด file description จะสืบทอดระหว่างรูทีนย่อยในตระกูล **exec** แอ็พพลิเคชันโปรแกรมที่ส่ง file descriptor ซึ่งถูกเปิดใช้งานให้ไฟล์ขนาดใหญ่สามารถเข้าถึงโปรแกรมอื่นๆ ได้ควรจะพิจารณาว่า โปรแกรมที่รับ สามารถเข้าถึงไฟล์ขนาดใหญ่ด้วยความปลอดภัย

การพอร์ตแอ็พพลิเคชันไปยังสภาวะแวดล้อมที่มีไฟล์ขนาดใหญ่

AIX ได้จัดเตรียมเมธอดสองวิธีสำหรับแอ็พพลิเคชันเพื่อเปิดใช้งานการเข้าถึงไฟล์ขนาดใหญ่ แอ็พพลิเคชันโปรแกรมเมอร์ต้องตัดสินใจเลือกวิธีที่เหมาะสมกับความต้องการมากที่สุ:

- นิยาม _LARGE_FILES ซึ่งจะนิยามชนิดข้อมูลที่เกี่ยวข้องทั้งหมด โครงสร้าง และชื่อรูทีนย่อยให้กับชุดสำเนาที่เปิดใช้งานไฟล์ที่มีขนาดใหญ่ การนิยาม _LARGE_FILES มีข้อดีในเรื่องของการขยายความสามารถในการพอร์ตแอ็พพลิเคชัน ให้กับแพล็ตฟอร์มอื่นๆ เนื่องจากแอ็พพลิเคชันยังคงเขียนลงในอินเตอร์เฟส POSIX และ XPG ที่ใช้กันตามปกติ ซึ่งจะมีข้อเสียในเรื่องของการสร้างความคลุมเครือในโค้ด เนื่องจากขนาดของหน่วยข้อมูลที่หลากหลาย ไม่สามารถนำมาพิจารณาได้จากการดูโค้ด
- บันทึกแอ็พพลิเคชันเพื่อเรียกรูทีนย่อยที่เปิดใช้งานไฟล์ขนาดใหญ่อย่างชัดเจน การบันทึกแอ็พพลิเคชันมีข้อเสียในเรื่องของความพยายามที่ต้องการมีมากขึ้น และลดความสามารถในการพอร์ตแอ็พพลิเคชันลง ซึ่งสามารถนำมาใช้เมื่อผลกระทบของการนิยามใหม่ของ _LARGE_FILES จะมีผลกระทบในด้านลบสำหรับโปรแกรม หรือ เมื่อมีความต้องการในการแปลงส่วนของโปรแกรมเพียงเล็กน้อย

ในกรณีอื่น แอ็พพลิเคชันโปรแกรม *ต้อง* ถูกตรวจสอบเพื่อให้มั่นใจว่ามีลักษณะการทำงานที่ถูกต้องในสภาวะแวดล้อมใหม่

การใช้ _LARGE_FILES

ในดีฟอลต์สภาวะแวดล้อมของการรวบรวม ชนิดข้อมูล **off_t** จะถูกกำหนดเป็นเครื่องหมายแบบ 32 บิต long ถ้าแอ็พพลิเคชันกำหนด _LARGE_FILES ก่อนที่จะรวมไฟล์ส่วนหัวใดๆ เข้าไว้ด้วยกัน สภาวะแวดล้อมโปรแกรมมิงไฟล์ขนาดใหญ่ จะถูกเปิดใช้งาน และ **off_t** จะถูกกำหนดให้เป็นเครื่องหมายแบบ 64 บิต long long นอกจากนี้ รูทีนย่อยทั้งหมดที่ทำงานกับขนาดไฟล์หรือออฟเซ็ตไฟล์จะถูกกำหนดขึ้นใหม่ให้เป็นสำเนาไฟล์ขนาดใหญ่ที่เปิดใช้งาน เช่นเดียวกัน โครงสร้างข้อมูลทั้งหมดที่มีขนาดไฟล์ที่ฝังตรึงไว้ หรือออฟเซ็ตจะถูกกำหนดขึ้นใหม่

ตารางต่อไปนี้แสดงนิยามที่กำหนดขึ้นใหม่ ซึ่งเกิดขึ้นในสภาวะแวดล้อม _LARGE_FILES:

เอนทิตี	ถูกอ้างอิงซ้ำเป็น	ไฟล์ส่วนหัว
off_t Object	long long	<sys/types.h>
fpos_t Object	long long	<sys/types.h>
โครงสร้าง struct stat	struct stat64	<sys/stat.h>
รูทีนย่อย stat	stat64()	<sys/stat.h>
รูทีนย่อย fstat	fstat64()	<sys/stat.h>
รูทีนย่อย lstat	lstat64()	<sys/stat.h>
รูทีนย่อย mmap	mmap64()	<sys/mman.h>
รูทีนย่อย lockf	lockf64()	<sys/lockf.h>
รูทีนย่อย struct flock	struct flock64	<sys/flock.h>
รูทีนย่อย open	open64()	<fcntl.h>

เอนทิตี	ถูกอ้างอิงซ้ำเป็น	ไฟล์ส่วนหัว
รูทีนย่อย creat	creat64()	<fcntl.h>
พารามิเตอร์คำสั่ง F_GETLK	F_GETLK64	<fcntl.h>
พารามิเตอร์คำสั่ง F_SETLK	F_SETLK64	<fcntl.h>
พารามิเตอร์คำสั่ง F_SETLKW	F_SETLKW64	<fcntl.h>
รูทีนย่อย ftw	ftw64()	<ftw.h>
รูทีนย่อย nftw	nftw64()	<ftw.h>
รูทีนย่อย fseeko	fseeko64()	<stdio.h>
รูทีนย่อย ftello	ftello64()	<stdio.h>
รูทีนย่อย fgetpos	fgetpos64()	<stdio.h>
รูทีนย่อย fsetpos	fsetpos64()	<stdio.h>
รูทีนย่อย fopen	fopen64()	<stdio.h>
รูทีนย่อย freopen	freopen64()	<stdio.h>
รูทีนย่อย lseek	lseek64()	<unistd.h>
รูทีนย่อย ftruncate	ftruncate64()	<unistd.h>
รูทีนย่อย truncate	truncate64()	<unistd.h>
รูทีนย่อย fclear	fclear64()	<unistd.h>
รูทีนย่อย pwrite	pwrite64()	<unistd.h>
รูทีนย่อย pread	pread64()	<unistd.h>
โครงสร้าง struct aiocb	struct aiocb64	<sys/aio.h>
รูทีนย่อย aio_read	aio_read64()	<sys/aio.h>
รูทีนย่อย aio_write	aio_write64()	<sys/aio.h>
รูทีนย่อย aio_cancel	aio_cancel64()	<sys/aio.h>
รูทีนย่อย aio_suspend	aio_suspend64()	<sys/aio.h>
รูทีนย่อย aio_return	aio_return64()	<sys/aio.h>
รูทีนย่อย aio_error	aio_error64()	<sys/aio.h>
โครงสร้าง liocb	liocb64	<sys/aio.h>
รูทีนย่อย lio_listio	lio_listio64()	<sys/aio.h>

การใช้รูทีนย่อยระบบไฟล์ 64 บิต

การใช้สภาวะแวดล้อม _LARGE_FILES อาจทำไม่ได้สำหรับบางแอ็พพลิเคชัน เนื่องจากห่างไกลจากความหมายโดยนัยของการเปลี่ยนขนาดของ off_t ไปเป็นแบบ 64 บิต หากจำนวนของการเปลี่ยนแปลงมีขนาดเล็ก อาจเป็นจริงมากกว่าในการแปลงส่วนของแอ็พพลิเคชันที่มีขนาดเล็ก ซึ่งต้องการเปิดใช้งานไฟล์ขนาดใหญ่ ชนิดข้อมูลของระบบไฟล์แบบ 64 บิต โครงสร้างและรูทีนย่อยจะถูกแสดงไว้ด้านล่าง:

```
<sys/types.h>
typedef long long off64_t;
typedef long long fpos64_t;

<fcntl.h>

extern int      open64(const char *, int, ...);
extern int      creat64(const char *, mode_t);

#define F_GETLK64
#define F_SETLK64
#define F_SETLKW64

<ftw.h>
extern int ftw64(const char *, int (*)(const char *,const struct stat64 *, int), int);
extern int nftw64(const char *, int (*)(const char *, const struct stat64 *, int,struct FTW *),int, int);

<stdio.h>

extern int      fgetpos64(FILE *, fpos64_t *);
extern FILE     *fopen64(const char *, const char *);
extern FILE     *freopen64(const char *, const char *, FILE *);
extern int      fseeko64(FILE *, off64_t, int);
extern int      fsetpos64(FILE *, fpos64_t *);
extern off64_t  ftello64(FILE *);

<unistd.h>

extern off64_t  lseek64(int, off64_t, int);
extern int      ftruncate64(int, off64_t);
extern int      truncate64(const char *, off64_t);
extern off64_t  fclear64(int, off64_t);
extern ssize_t  pread64(int, void *, size_t, off64_t);
extern ssize_t  pwrite64(int, const void *, size_t, off64_t);
extern int      fsync_range64(int, int, off64_t, off64_t);

<sys/flock.h>

struct flock64;

<sys/lockf.h>

extern int lockf64 (int, int, off64_t);

<sys/mman.h>

extern void     *mmap64(void *, size_t, int, int, int, off64_t);

<sys/stat.h>

struct stat64;

extern int      stat64(const char *, struct stat64 *);
extern int      fstat64(int, struct stat64 *);
extern int      lstat64(const char *, struct stat64 *);

<sys/aio.h>

struct aiocb64
```

```
int     aio_read64(int, struct aiocb64 *):
int     aio_write64(int, struct aiocb64 *);
int     aio_listio64(int, struct aiocb64 *[],
        int, struct      sigevent *);
int     aio_cancel64(int, struct aiocb64 *);
int     aio_suspend64(int, struct aiocb64 *[]);

struct liocb64
int     lio_listio64(int, struct liocb64 *[], int, void *);
```

กับดักทั่วไปในการใช้สภาวะแวดล้อมที่มีไฟล์ขนาดใหญ่

การพอร์ตของแอ็พพลิเคชันโปรแกรม ไปยังสภาวะแวดล้อมของไฟล์ขนาดใหญ่สามารถแสดงจำนวนของปัญหาต่างๆ ในแอ็พพลิเคชันได้ บ่อยครั้งที่ปัญหาเหล่านี้อาจเกิดจากฝึกเขียนโค้ดที่ไม่ดี ซึ่งเป็นอันตรายในสภาวะแวดล้อม **off_t** แบบ 32 บิต แต่สามารถแสดงให้เห็นได้ด้วยตนเอง เมื่อคอมไพล์ในสภาวะแวดล้อม **off_t** แบบ 64 บิต ปัญหาและโซลูชันทั่วไปบางอย่างจะถูกกล่าวถึงในส่วนนี้

หมายเหตุ: สำหรับตัวอย่างต่อไปนี้ **off_t** จะสมมติว่าเป็นอ็อฟเซ็ตไฟล์แบบ 64 บิต

การใช้ชนิดข้อมลที่ไม่ถูกต้อง

ซอร์สทั่วไปของปัญหาที่มีแอ็พพลิเคชันโปรแกรม คือความล้มเหลวในการชนิดข้อมูลที่ถูกต้อง ถ้าแอ็พพลิเคชันพยายามเก็บขนาดไฟล์ หรืออ็อฟเซ็ตไฟล์ในตัวแปรเลขจำนวนเต็ม ค่าผลลัพธ์ที่ได้จะถูกตัดปลาย และสูญเสียส่วนที่สำคัญ หากต้องการหลีกเลี่ยงปัญหานี้ ให้ใช้ชนิดข้อมูล **off_t** เพื่อเก็บขนาดไฟล์และอ็อฟเซ็ต

ไม่ถูกต้อง:

```
int file_size;
struct stat s;

file_size = s.st_size;
```

ดีกว่า:

```
off_t file_size;
struct stat s;
file_size = s.st_size;
```

เมื่อคุณกำลังส่งเลขจำนวนเต็มแบบ 64 บิตไปยังฟังก์ชันในฐานะเป็นอาร์กิวเมนต์ หรือเมื่อคุณกำลังส่งคืนเลขจำนวนเต็มแบบ 64 บิตจากฟังก์ชัน ตัวเรียกทั้งสองและฟังก์ชันที่เรียก *ต้อง* ยอมรับชนิดของอาร์กิวเมนต์ และส่งคืนค่า

การส่งเลขจำนวนเต็มแบบ 32 บิตไปยังฟังก์ชันที่ต้องการเลขจำนวนเต็มแบบ 64 บิตจะเป็นสาเหตุทำให้ฟังก์ชันที่เรียกตีความอาร์กิวเมนต์ของตัวเรียกผิดไป และนำไปสู่ลักษณะการทำงานที่ไม่ได้คาดการณ์ไว้ ปัญหาแบบนี้อาจมีความรุนแรง หากโปรแกรมส่งค่าสเกลาร์ไปยังฟังก์ชันที่ต้องการรับเลขจำนวนเต็มแบบ 64 บิต

คุณสามารถหลีกเลี่ยงปัญหานี้ได้โดยใช้ฟังก์ชันต้นแบบ ด้วยความระมัดระวัง ในการแตกโค้ดข้างต้น **fexample( )** คือฟังก์ชันที่ใช้อ็อฟเซ็ตไฟล์แบบ 64 บิตซึ่งเป็นพารามิเตอร์ สำหรับตัวอย่างแรก คอมไพลเลอร์จะสร้างการลิงก์ฟังก์ชันสำหรับเลขจำนวนเต็มแบบ 32 บิต ซึ่งจะไม่ถูกต้อง เนื่องจากฟังก์ชันที่รับต้องการลิงก์เลขจำนวนเต็มแบบ 64 บิต สำหรับตัวอย่างที่สอง ตัวระบุ

LL จะถูกเพิ่มไว้ ซึ่งจะบังคับให้คอมไพลเลอร์ใช้ลิงก์ที่ถูกต้อง สำหรับตัวอย่างสุดท้าย ฟังก์ชันต้นแบบ อาจเป็นสาเหตุทำให้คอมไพลเลอร์พัฒนาค่าสเกลาร์ไปเป็นเลขจำนวนเต็มแบบ 64 บิต นี่คือวิธีที่นำเสนอไว้ เนื่องจากซอร์สโค้ดยังคงสามารถพอร์ตระหว่างสภาวะแวดล้อมแบบ 32 บิต และ 64 บิต

ไม่ถูกต้อง:

```
fexample(0);
```

ดีกว่า:

```
fexample(0LL);
```

ดีที่สุด:

```
\est:
```

การลิงก์สำหรับโปรแกรมเมอร์

ลิงก์คือการเชื่อมโยงระหว่างชื่อไฟล์และ i-node (ฮาร์ดลิงก์) หรือระหว่างชื่อไฟล์ (ลิงก์สัญลักษณ์)

การลิงก์อนุญาตให้เข้าถึง i-node จากชื่อไฟล์หลายชื่อ รายการไดเร็กทอรีคู่กับชื่อไฟล์ด้วย i-nodes ชื่อไฟล์นั้นง่าย ต่อผู้ใช้ในการระบุ และ i-nodes มีแอดเดรสดิสก์จริงของ ข้อมูลของไฟล์ จำนวนนับการอ้างอิงของลิงก์ไปยัง i-node ได้รับการดูแลรักษาในฟิลด์ **i_nlink** ของ i-node รูทีนย่อยที่สร้าง และทำลายลิงก์ใช้ชื่อไฟล์ ไม่ใช่ file descriptors ดังนั้น ไม่จำเป็นที่จะต้องเปิดไฟล์เมื่อทำการสร้างลิงก์

การประมวลผลสามารถเข้าถึงและเปลี่ยนแปลงเนื้อหาของ i-node ได้โดยชื่อไฟล์ใดๆ ที่ถูกลิงก์ AIX สนับสนุนฮาร์ดลิงก์และลิงก์สัญลักษณ์

ฮาร์ดลิงก์

รูทีนย่อย	คำอธิบาย
link	รูทีนย่อยที่สร้างฮาร์ดลิงก์ การแสดงฮาร์ดลิงก์เป็นการประกัน ถึงการมีอยู่ของไฟล์เนื่องจากฮาร์ดลิงก์เพิ่มค่าจำนวนนับลิงก์ในฟิลด์ **i_nlink** ของ i-node
unlink	รูทีนย่อยที่รีลีสลิงก์ เมื่อฮาร์ดลิงก์ทั้งหมดไปยัง i-node ถูก รีลีส ไฟล์ไม่สามารถเข้าถึงได้อีกต่อไป

ฮาร์ดลิงก์ต้องลิงก์ชื่อไฟล์และ i-nodes ภายในระบบไฟล์เดียวกัน เนื่องจากหมายเลข i-node สัมพันธ์กับระบบไฟล์เดียว ฮาร์ดลิงก์ อ้างถึงไฟล์เฉพาะเสมอเนื่องจากรายการไดเร็กทอรีที่สร้างโดย ฮาร์ดลิงก์จะจับคู่ชื่อไฟล์ใหม่ไปยัง i-node ID ผู้ใช้ที่สร้างไฟล์ต้นฉบับเป็นเจ้าของไฟล์และกำหนดสิทธิโหมดการเข้าถึงสำหรับไฟล์ มิฉะนั้น ฮาร์ดลิงก์ทั้งหมดถูกปฏิบัติเท่าเทียมกันโดยระบบปฏิบัติการ

ตัวอย่าง: ถ้าไฟล์ **/u/tom/bob** ถูกลิงก์ไปยังไฟล์ **/u/jack/foo** จำนวนนับลิงก์ใน ฟิลด์ **i_nlink** ของไฟล์ **foo** คือ 2 ฮาร์ดลิงก์ทั้งสองมีค่าเท่านั้น ถ้า **/u/jack/foo** ถูกลบออก ไฟล์ยังคงมีอยู่ด้วยชื่อ **/u/tom/bob** และสามารถเข้าถึงได้โดยผู้ใช้ที่มีการเข้าถึงไดเร็กทอรี **tom** อย่างไรก็ตาม เจ้าของไฟล์คือ `jack` แม้ว่า **/u/jack/foo** ถูกลบออกแล้ว พื้นที่ ที่ครอบครองโดยไฟล์จะถูกเปลี่ยนเป็นบัญชีโควต้าของ jack ในการเปลี่ยนความเป็นเจ้าของ ให้ใช้รูทีนย่อย **chown**

ลิงก์สัญลักษณ์

ลิงก์ สัญลักษณ์ถูกนำไปใช้เป็นไฟล์ที่มีชื่อพาธโดยการใช้คำสั่ง **symlink** เมื่อการประมวลผล พบลิงก์สัญลักษณ์ พาธที่อยู่ในลิงก์สัญลักษณ์จะถูกเติมหน้า พาธที่การประมวลผลกำลังค้นหา ถ้าชื่อพาธในลิงก์สัญลักษณ์ เป็นชื่อพาธสัมบูรณ์ การประมวลผลจะค้นหาจากไดเร็กทอรี root เพื่อหาไฟล์ที่กำหนดชื่อ ถ้าชื่อพาธเป็นลิงก์สัญลักษณ์ไม่ได้เริ่มต้นด้วย / (slash) การประมวลผลจะแปลความหมายว่าส่วนที่เหลือของพาธสัมพันธ์กับตำแหน่ง ของชื่อพาธสัญลักษณ์ รูทีนย่อย **unlink** ยังลบ ลิงก์สัญลักษณ์ออก

ลิงก์สัญลักษณ์สามารถสำรวจในระบบไฟล์ได้ เนื่องจากถูกถือเสมือนเป็น ไฟล์ปกติโดยระบบปฏิบัติการมากกว่าจะเป็นส่วนหนึ่งของโครงสร้างระบบไฟล์ การมีลิงก์สัญลักษณ์อยู่ไม่ได้รับประกัน การมีอยู่ของไฟล์เป้าหมาย เนื่องจากลิงก์สัญลักษณ์ไม่มีผลต่อไฟล์ **i_nlink** ของ i-node

รูทีนย่อย	คำอธิบาย
readlink	รูทีนย่อยที่อ่านเนื้อหาจากลิงก์สัญลักษณ์ หลายรูทีนย่อย (รวมถึงรูทีนย่อย **open** และ **stat**) ต่อพาธ สัญลักษณ์
lstat	รูทีนย่อยที่สร้างเพื่อรายงานเกี่ยวกับสถานะของไฟล์ที่มี ลิงก์สัญลักษณ์และไม่ตามลิงก์ โปรดดูที่รูทีนยอย **symlink** เพื่อดูรายการรูทีนยอยที่สำรวจลิงก์สัญลักษณ์

ลิงก์สัญลักษณ์ยังถูกเรียกว่า *ซอฟต์ลิงก์* เนื่องจาก ลิงก์ไปยังไฟล์ด้วยชื่อพาธ ถ้าไฟล์เป้าหมายถูกเปลี่ยนชื่อหรือลบออก ลิงก์สัญลักษณ์จะไม่สามารถแก้ไขได้

ตัวอย่าง: ลิงก์สัญลักษณ์ไปยัง **/u/joe/foo** คือไฟล์ที่มีข้อมูลค่าคงที่ใดๆ **/u/joe/foo** เมื่อเจ้าของไฟล์ **foo** ลบไฟล์นี้ออก การเรียกใช้รูทีนย่อยที่ทำไปยังลิงก์สัญลักษณ์จะไม่สำเร็จ ถ้าเจ้าของไฟล์ สร้างไฟล์ใหม่ชื่อ **foo** ในเดียวกัน ลิงก์สัญลักษณ์จะนำไปยังไฟล์ใหม่ ดังนั้น ลิงก์ถูกพิจารณาเป็นแบบ ซอฟต์เนื่องจากถูกลิงก์ไปยัง i-nodes ที่เปลี่ยนได้

ในการแสดงรายการคำสั่ง **ls -l** l ในตำแหน่งแรก ระบุไฟล์ที่ถูกลิงก์ ในคอลัมน์สุดท้ายของการแสดงรายการดังกล่าว ลิงก์ระหว่างไฟล์จะถูกแทนเป็น `Path2 -> Path1` (หรือ `Newname -> Oldname`)

รูทีนย่อย	คำอธิบาย
unlink	รูทีนย่อยที่ลบรายการไดเร็กทอรีออก พารามิเตอร์ *Path* ในรูทีนย่อยระบุไฟล์ที่จะถูกยกเลิกการเชื่อมต่อ เมื่อเสร็จการเรียกใช้ **unlink** จำนวนนับลิงก์ ของ i-node ถูกลดค่าลง 1
remove	รูทีนย่อยที่ลบชื่อไฟล์ออกโดยการเรียกใช้รูทีนย่อย **unlink** หรือ **rmdir**

ลิงก์ไดเร็กทอรี

รูทีนย่อย	คำอธิบาย
mkdir	รูทีนย่อยที่สร้างรายการไดเร็กทอรีสำหรับไดเร็กทอรีใหม่ ซึ่ง จะสร้างฮาร์ดลิงก์ไปยัง i-node ที่แทนไดเร็กทอรี

ลิงก์สัญลักษณ์ได้รับการแนะนำให้ใช้สำหรับการสร้างลิงก์เพิ่มให้แก่ไดเร็กทอรี ลิงก์สัญลักษณ์จะไม่ยุ่งเกี่ยวกับรายการไดเร็กทอรี . และ .. และจะ คงรักษาสถานะไดเร็กทอรีที่ฟอร์มอย่างดีและว่าง ต่อไปนี้แสดง ตัวอย่างของไดเร็กทอรีที่ฟอร์มเป็นอย่างดีและว่าง **/u/joe/foo** และค่า **i_nlink**

```
/u
```

ค่า	empty values	empty values	ไดเร็กทอรี			
68			j	o	e	0

```
/u/joe
mkdir ("foo", 0666)
```

ค่า	empty values	empty values	ไดเร็กทอรี			
68			n	0	0	0
			n	n	0	0
235			f	o	o	0

```
/u/joe/foo
```

ค่า	empty values	empty values	ไดเร็กทอรี			
235			n	0	0	0
68			n	n	0	0

```
ค่า i_nlink

  i = 68

n_link 3

สำหรับ i = 68 ค่า n_link คือ 3 (/u; /u/joe; /u/joe/foo)

i = 235

n_link 2

สำหรับ i = 235 ค่า n_link คือ 2 (/u/joe; /u/joe/foo)
```

การใช้ file descriptors

ไฟล์ descriptor คือจำนวนเต็ม ไม่มีเครื่องหมายที่ใช้โดยกระบวนการเพื่อระบุไฟล์ที่เปิด

จำนวนของไฟล์ descriptors ที่พร้อมใช้งานกับกระบวนการถูกจำกัดโดยการควบคุม **/OPEN_MAX** ในไฟล์ **sys/limits.h** จำนวนของไฟล์ descriptors ยังถูกควบคุมโดยแฟล็ก **ulimit -n** รูทีนย่อย **open, pipe, creat** และ **fcntl** ทั้งหมดสร้าง ไฟล์ descriptors ไฟล์ descriptors โดยทั่วไปเป็นค่าเฉพาะกับแต่ละกระบวนการ แต่ สามารถถูกแบ่งใช้โดยกระบวนการ child ที่สร้างด้วยรูทีนย่อย **fork** หรือที่คัดลอกโดยรูทีนย่อย **fcntl, dup** และ **dup2**

ไฟล์ descriptors เป็นดัชนีของตารางไฟล์ descriptor ในพื้นที่ **u_block** ที่ดูแลโดยเคอร์เนลสำหรับ แต่ละกระบวนการ วิธีธรรมดาที่สุดสำหรับกระบวนการเพื่อรับไฟล์ descriptors คือผ่านการดำเนินการ **open** หรือ **creat** หรือผ่านการสืบทอดจากการ

กระบวนการ parent เมื่อการดำเนินการ **fork** เกิดขึ้น ตาราง descriptor ถูกคัดลอกสำหรับกระบวนการ child ซึ่งอนุญาตให้กระบวนการ child มีการเข้าถึงที่เท่าเทียมกับไฟล์ที่ใช้โดยกระบวนการ parent

ตาราง File descriptor และตาราง system open file

ไฟล์ descriptor และโครงสร้างตารางไฟล์เปิดติดตามแต่ละการเข้าถึง ของกระบวนการกับไฟล์และรับประกัน data integrity

ตาราง	คำอธิบาย

ตารางไฟล์ descriptor	แปลหมายเลขดัชนี (ไฟล์ descriptor) ในตารางเป็นไฟล์ เปิด ตารางไฟล์ descriptor ถูกสร้างสำหรับแต่ละกระบวนการและถูกเก็บใน พื้นที่ **u_block** ที่พักไว้สำหรับกระบวนการนั้น แต่ละ รายการในตารางไฟล์ descriptor มีฟิลด์ดังต่อไปนี้: พื้นที่แฟล็ก และตัวชี้ไฟล์ มีจำนวนได้มากถึง **OPEN_MAX** ไฟล์ descriptors. โครงสร้างของตารางไฟล์ descriptor เป็นดังนี้:
	`struct ufd`
	`{`
	`  struct file *fp;`
	`  int flags;`
	`} *u_ufd`

ตารางไฟล์เปิดของระบบ	มีรายการสำหรับแต่ละไฟล์ที่เปิด รายการตารางไฟล์ติดตาม ออฟเซ็ตปัจจุบันที่อ้างอิงโดยการดำเนินการอ่านหรือเขียนทั้งหมดที่มีกับไฟล์ และโหมดเปิด (**O_RDONLY**, **O_WRONLY**, หรือ **O_RDWR**) ของไฟล์ โครงสร้างตารางไฟล์ที่เปิดมีออฟเซ็ต I/O ปัจจุบันสำหรับไฟล์ ระบบ จะถือว่าแต่ละการดำเนินการ อ่าน/เขียนเป็นบ่งชี้ถึงการค้นหาออฟเซ็ตปัจจุบัน ดังนั้นถ้า *x* ไบต์ถูกอ่านหรือเขียน ตัวชี้จะชี้ไปข้างหน้า *x* ไบต์ รูทีนย่อย **lseek** สามารถถูกใช้เพื่อกำหนดออฟเซ็ตปัจจุบันใหม่ให้กับ ตำแหน่งที่ระบุในไฟล์ที่ถูกเข้าถึงแบบสุ่ม ไฟล์ประเภทสตรีม (เช่นไพพ์และซ็อกเก็ต) ไม่ใช่ออฟเซ็ตเนื่องจากข้อมูลใน ไฟล์ไม่ได้เป็นการเข้าถึงแบบสุ่ม

การจัดการ file descriptors

เนื่องจากไฟล์สามารถถูกแบ่งใช้โดยผู้ใช้หลายคน จึงจำเป็นต้องอนุญาตให้กระบวนการที่เกี่ยวข้อง แบ่งใช้ตัวชี้ออฟเซ็ตทั่วไป และ มีตัวชี้ออฟเซ็ตปัจจุบันแยกสำหรับกระบวนการอิสระที่เข้าถึง ไฟล์เดียวกัน รายการตารางไฟล์ที่เปิดรักษาจำนวนการอ้างอิงเพื่อ ติดตามจำนวนของไฟล์ descriptors ที่กำหนดให้กับไฟล์

การอ้างอิงหลายการอ้างอิงไปที่ไฟล์เดียวเกิดขึ้นได้โดยการดำเนินการดังต่อไปนี้:

- กระบวนการแยกกันเปิดไฟล์
- กระบวนการ Child ได้รับไฟล์ descriptors ที่กำหนดให้กับกระบวนการ parent
- รูทีนย่อย **fcntl** หรือ **dup** สร้าง สำเนาของไฟล์ descriptors

การแบ่งใช้ open files

แต่ละการดำเนินการเปิดสร้างรายการตารางไฟล์ที่เปิดของระบบ รายการตารางแยกกันประกันว่าแต่ละกระบวนการมีออฟเซ็ต I/O ปัจจุบันแยกกัน ออฟเซ็ตอิสระป้องกัน integrity ของข้อมูล

เมื่อไฟล์ descriptor ถูกทำสำเนา จากนั้นกระบวนการ สองกระบวนการแบ่งใช้ออฟเซ็ตเดียวกันและการแทรกเกิดขึ้นได้ ในไบต์ที่ไม่ได้ถูกอ่านหรือเขียนตามลำดับ

การทำซ้ำ file descriptors

ไฟล์ descriptors สามารถถูกทำซ้ำระหว่างกระบวนการ ในวิธีดังต่อไปนี้: รูทีนย่อย **dup** หรือ **dup2**, รูทีนย่อย **fork** และรูทีนย่อย **fcntl** (การควบคุมไฟล์ descriptor)

รูทีนย่อย dup และ dup2

รูทีนย่อย **dup** สร้างสำเนาของไฟล์ descriptor สำเนาถูกสร้างที่พื้นที่ว่างในตารางไฟล์ descriptor ผู้ใช้ ซึ่งมี descriptor ดั้งเดิม กระบวนการ **dup** เพิ่ม จำนวนการอ้างอิงในรายการตารางไฟล์ทีละ 1 และส่งกลับหมายเลข ดัชนีของไฟล์ descriptor ซึ่งสำเนาถูกนำไปไว้

รูทีนย่อย **dup2** สแกนหาการกำหนด descriptor ที่ร้องขอ และปิดไฟล์ descriptor ที่ร้องขอถ้าไฟล์เปิดอยู่ ซึ่งอนุญาตให้กระบวนการ กำหนดรายการ descriptor ใดที่สำเนาจะใช้ ถ้าจำเป็นต้องมีรายการ ตาราง descriptor จำเพาะ

รูทีนย่อย fork

รูทีนย่อย **fork** สร้างกระบวนการ child ที่ สืบทอดไฟล์ descriptors ที่กำหนดให้กับกระบวนการ parent จากนั้น กระบวนการ child execs กระบวนการใหม่ descriptors ที่สืบทอดที่มีแฟล็ก **close-on-exec** ตั้งค่าโดยรูทีนย่อย **fcntl** จะถูกปิด

รูทีนย่อย fcntl (ตัวควบคุม file descriptor)

รูทีนย่อย **fcntl** จัดการโครงสร้างไฟล์ และควบคุมไฟล์ descriptors ที่เปิด ซึ่งสามารถถูกใช้เพื่อทำการเปลี่ยนแปลง กับ descriptor ดังต่อไปนี้:

- ทำสำเนาไฟล์ descriptor (เหมือนกับรูทีนย่อย **dup**)
- รับหรือตั้งค่าแฟล็ก close-on-exec
- ตั้งค่าโหมด nonblocking สำหรับ descriptor
- ผนวกการเขียนข้อมูลในอนาคตกับตำแหน่งสิ้นสุดของไฟล์ (**O_APPEND**)
- เปิดใช้งานการสร้างสัญญาไปที่กระบวนการเมื่อ เป็นไปได้ที่จะดำเนินการ I/O
- ตั้งค่าหรือรับ process ID หรือ group process ID สำหรับการจัดการ **SIGIO**
- ปิดไฟล์ descriptors ทั้งหมด

ค่า file descriptor ที่กำหนดไว้ล่วงหน้า

เมื่อเชลล์รันโปรแกรม จะเปิดไฟล์สามไฟล์ พร้อมกับ descriptors 0, 1 และ 2 การกำหนดค่าเริ่มต้นสำหรับ descriptors เหล่านี้มีดังนี้:

Descriptor	คำอธิบาย
0	แสดงอินพุตมาตรฐาน
1	แสดงเอาต์พุตมาตรฐาน
2	แสดงข้อผิดพลาดมาตรฐาน

ไฟล์ descriptors เริ่มต้นถูกเชื่อมต่อกับเทอร์มินัล ดังนั้นถ้าโปรแกรมอ่านไฟล์ descriptor 0 และเขียนไฟล์ descriptors 1 และ 2, โปรแกรมเก็บค่าเก็บรวบรวมจากเทอร์มินัลและส่งเอาต์พุต ไปที่เทอร์มินัล ขณะที่โปรแกรมใช้ไฟล์อื่น ไฟล์ descriptors ถูกกำหนดใน ลำดับจากน้อยไปหามาก

ถ้า I/O ถูกเปลี่ยนทิศทางโดยใช้สัญลักษณ์ < (น้อยกว่า) หรือ > (มาก กว่า) การกำหนด file descriptor ดีฟอลต์ของเชลล์จะถูกเปลี่ยน ตัวอย่างเช่น ข้อมูลต่อไปนี้เปลี่ยนการกำหนดค่าเริ่มต้นสำหรับ ไฟล์ descriptors 0 และ 1 จากเทอร์มินัลไปที่ไฟล์ที่เหมาะสม:

```
prog < FileX > FileY
```

ในตัวอย่างนี้ ไฟล์ descriptor 0 ขณะนี้อ้างอิงถึง FileX และไฟล์ descriptor 1 อ้างอิงถึง FileY ไฟล์ descriptor 2 ไม่มีการเปลี่ยนแปลง โปรแกรมไม่จำเป็นต้องทราบ ว่าอินพุตมาจากไหน หรือจะถูกส่งไปไหน ตราบเท่าที่ไฟล์ descriptor 0 แสดงไฟล์อินพุต และไฟล์ descriptor 1 และ 2 แสดงไฟล์เอาต์พุต

โปรแกรมตัวอย่างดังต่อไปนี้แสดงการเปลี่ยนทิศทาง ของเอาต์พุตมาตรฐาน:

```
#include <fcntl.h>
#include <stdio.h>

void redirect_stdout(char *);

main()
{
       printf("Hello world\n");       /*this printf goes to
                                      * standard output*/
       fflush(stdout);
       redirect_stdout("foo");        /*redirect standard output*/
       printf("Hello to you too, foo\n");
                                      /*printf goes to file foo */
       fflush(stdout);
}

void
redirect_stdout(char *filename)
{
        int fd;
        if ((fd = open(filename,O_CREAT|O_WRONLY,0666)) < 0)
                                        /*open a new file */
        {
                perror(filename);
                exit(1);
        }
        close(1);                       /*close old */
                                        *standard output*/
        if (dup(fd) !=1)                /*dup new fd to
                                        *standard input*/
        {
                fprintf(stderr,"Unexpected dup failure\n");
                exit(1);
        }
        close(fd);                       /*close original, new fd,*/
                                         * no longer needed*/
}
```

ภายในตารางไฟล์ descriptor หมายเลขไฟล์ descriptor ถูกกำหนดหมายเลข descriptor ต่ำสุดที่มีเมื่อมีการร้องขอ descriptor อย่างไรก็ตาม ค่าใดๆ ก็ได้สามารถถูกกำหนดภายในตารางไฟล์ descriptor โดยใช้รูทีนย่อย **dup**

ข้อจำกัดรีซอร์สของ File descriptor

จำนวนของไฟล์ descriptors ที่สามารถถูกจัดสรรให้กับกระบวนการ ถูกดูแลโดยข้อจำกัดทรัพยากร ค่าดีฟอลต์ถูกตั้งค่าในไฟล์ **/etc/security/limits** โดยปกติตั้งค่าที่ 2000 ข้อจำกัดสามารถเปลี่ยนแปลงได้โดยคำสั่ง **ulimit** หรือรูทีนย่อย **setrlimit** ขนาดสูงสุดถูกกำหนด โดยค่าคงที่ **OPEN_MAX**

การสร้างและการลบไฟล์

ส่วนนี้อธิบายขั้นตอนภายในที่ดำเนินการ โดยระบบปฏิบัติการเมื่อมีการ สร้าง เปิด ปิด หรือลบไฟล์

การสร้างไฟล์

รูทีนย่อยที่แตกต่างกันสร้างชนิดของไฟล์จำเพาะ ดังนี้:

รูทีนย่อย	ชนิดของไฟล์ที่สร้าง
creat	ทั่วไป
open	ธรรมดา (เมื่อแฟล็ก **O_CREAT** ถูกตั้งค่า)
mknod	ธรรมดา เข้าก่อนออกก่อน (FIFO) หรือ พิเศษ
mkfifo	ไพพ์ที่มีชื่อ (FIFO)
pipe	ไพพ์ที่ไม่มีชื่อ
socket	ซ็อกเก็ต
mkdir	ไดเร็กทอรี
symlink	ลิงก์สัญลักษณ์

การสร้างไฟล์ธรรมดา (รูทีนย่อย creat, open หรือ mknod)

คุณใช้รูทีนย่อย **creat** เพื่อสร้างไฟล์ตามค่าที่ตั้งในพารามิเตอร์ *Pathname* และ *Mode* ถ้าไฟล์มีชื่อในพารามิเตอร์ *Pathname* มีอยู่ และกระบวนการมีสิทธิในการเขียนลงไฟล์รูทีนย่อย **creat** ตัดไฟล์ การตัดส่วนท้ายรีลีสบล็อกข้อมูลและ ตั้งค่าขนาดไฟล์เป็น 0 คุณยังสามารถสร้างไฟล์ธรรมดาได้ใหม่ โดยใช้รูทีนย่อย **open** ด้วยแฟล็ก **O_CREAT**

ไฟล์ที่สร้างด้วยรูทีนย่อย **creat, mkfifo** หรือ **mknod** ใช้สิทธิการเข้าถึงที่ตั้งค่าใน พารามิเตอร์ *Mode* ไฟล์ธรรมดาที่สร้างด้วยรูทีนย่อย **open** ใช้โหมดการเข้าถึงจากแฟล็ก **O_CREAT** พารามิเตอร์ *Mode* รูทีนย่อย **umask** ตั้งค่ามาสก์การสร้าง file-mode (การตั้งค่า โหมดการเข้าถึง) สำหรับไฟล์ที่สร้างใหม่โดยกระบวนการและส่งกลับ ค่าก่อนหน้าของมาสก์

บิตสิทธิในไฟล์ที่สร้างใหม่เป็นผลจากการ กลับบิต **umask** แล้ว AND กับบิตโหมดการสร้างไฟล์ ที่ตั้งค่าโดยกระบวนการสร้าง เมื่อไฟล์ใหม่ถูกสร้างโดยกระบวนการ ระบบปฏิบัติการดำเนินการดังต่อไปนี้:

- กำหนดสิทธิของกระบวนการสร้าง
- เรียกคืนค่า **umask** ที่เหมาะสม
- กลับค่า **umask**
- ใช้การดำเนินการ AND เพื่อรวมสิทธิของกระบวนการสร้าง กับการกลับค่า **umask**

การสร้างไฟล์พิเศษ (รูทีนย่อย mknod หรือ mkfifo)

คุณสามารถใช้รูทีนย่อย **mknod** และ **mkfifo** เพื่อสร้างไฟล์พิเศษใหม่ รูทีนย่อย **mknod** จัดการไฟล์ ไพพ์ที่มีชื่อ (FIFO) ปกติ และ อุปกรณ์ ซึ่งสร้าง i-node สำหรับไฟล์เหมือนกับที่สร้างโดยรูทีนย่อย **creat** เมื่อคุณใช้รูทีนย่อย **mknod** ฟิลด์ file-type ถูกตั้งค่าเพื่อระบุชนิดของไฟล์ ที่ถูกสร้าง ถ้าไฟล์เป็นบล็อกหรือไฟล์อุปกรณ์ชนิดอักขระ ชื่อของอุปกรณ์หลักและอุปกรณ์รองถูกเขียนลงใน i-node

รูทีนย่อย **mkfifo** เป็นอินเตอร์เฟส สำหรับรูทีนย่อย **mknod** และถูกใช้เพื่อสร้างไพพ์ ที่มีชื่อ

การเปิดไฟล์

รูทีนย่อย **open** เป็นขั้นตอนแรกที่จำเป็นสำหรับกระบวนการในการเข้าถึง ไฟล์ที่มีอยู่ รูทีนย่อย **open** ส่งกลับไฟล์ descriptor การอ่าน การเขียน การค้นหา การทำซ้ำ การตั้งค่าพารามิเตอร์ I/O การระบุ สถานะของไฟล์ และการปิดไฟล์ ทั้งหมดใช้ไฟล์ descriptor ที่ส่งกลับโดย การเรียก **open** รูทีนย่อย **open** สร้าง รายการสำหรับไฟล์ในตารางไฟล์ descriptor เมื่อกำหนดไฟล์ descriptors

รูทีนย่อย **open** กระทำ ดังนี้:

- ตรวจสอบสิทธิที่เหมาะสมซึ่งอนุญาตให้กระบวนการเข้าถึง ไฟล์
- กำหนดรายการในตารางไฟล์ descriptor สำหรับเปิดไฟล์ รูทีนย่อย **open** ตั้งค่าออฟเซ็ตไบต์ อ่าน/เขียน เป็น 0 จุดเริ่มต้นของไฟล์

รูทีนย่อย **ioctl** หรือ **ioctlx** ทำการควบคุมการดำเนินการ บนไฟล์อุปกรณ์พิเศษที่เปิดอยู่

การปิดไฟล์

เมื่อกระบวนการไม่จำเป็นต้องเข้าถึงไฟล์ที่เปิด รูทีนย่อย **close** เอา รายการสำหรับไฟล์ออกจากตาราง ถ้ามีไฟล์ descriptor มากกว่าหนึ่งไฟล์อ้างอิง รายการตารางไฟล์สำหรับไฟล์ การนับการอ้างอิงสำหรับไฟล์ถูกลดลง 1 และการปิดเสร็จสมบูรณ์ ถ้าไฟล์มีการอ้างอิงเพียง 1 ตารางไฟล์ จะถูกทำให้ว่าง ความพยายามของกระบวนการที่จะใช้ไฟล์ที่ไฟล์ descriptor ที่ยกเลิกการเชื่อมต่อไปแล้วมีผลให้เกิดข้อผิดพลาดจนกว่ารูทีนย่อย **open** อื่น จะกำหนดค่าใหม่สำหรับค่าไฟล์ descriptor นั้น เมื่อกระบวนการจบการทำงาน เคอร์เนล จะตรวจสอบไฟล์ descriptors ของผู้ใช้ที่ทำงานอยู่ และปิดแต่ละไฟล์แบบ ภายใน การดำเนินการนี้ประกันว่าไฟล์ทั้งหมดปิดก่อนที่กระบวนการจะสิ้นสุด

การลบไฟล์

เมื่อไม่จำเป็นต้องใช้ไฟล์ คุณสามารถใช้รูทีนย่อย **unlink** เพื่อเอาไฟล์ที่ระบุออกจากไดเร็กทอรีที่มี ไฟล์อยู่ ถ้ามีหลายฮาร์ดลิงก์ไปที่ไฟล์เดียวกันรูทีนย่อย **unlink** จะเอาลิงก์ที่ระบุออก ถ้ามีเพียงลิงก์เดียว รูทีนย่อย **unlink** จะเอาตัวไฟล์เองออก สำหรับข้อมูลเพิ่มเติม โปรดดูที่รูทีนย่อย **unlink**

การทำงานกับไฟล์ I/O

การดำเนินการอินพุตและเอาต์พุต (I/O) ทั้งหมดใช้ ข้อมูลออฟเซ็ตไฟล์ปัจจุบันที่เก็บในโครงสร้างไฟล์ระบบ

ออฟเซ็ต I/O ปัจจุบันกำหนดออฟเซ็ตไบต์ที่ถูกติดตามอย่างต่อเนื่องสำหรับทุกไฟล์ ที่เปิด ออฟเซ็ต I/O ปัจจุบันส่งสัญญาณการดำเนินการอ่านหรือเขียน ตำแหน่งที่เริ่ม การดำเนินการในไฟล์ รูทีนย่อย **open** รีเซ็ตเป็น 0 ตัวชี้สามารถถูกตั้งค่าหรือเปลี่ยนโดยใช้รูทีนย่อย **lseek**

การจัดการกับออฟเซ็ตปัจจุบัน

การดำเนินการอ่านและเขียนสามารถเข้าถึงไฟล์อย่างเป็นลำดับ เนื่องจากออฟเซ็ต I/O ปัจจุบันของไฟล์ติดตามและออฟเซ็ตไบต์ของแต่ละ การดำเนินการก่อนหน้า ออฟเซ็ตถูกเก็บในตารางไฟล์ระบบ

คุณสามารถปรับออฟเซ็ตของไฟล์ที่สามารถเข้าถึงได้แบบสุ่ม เช่นไฟล์ปกติและชนิดพิเศษ โดยใช้รูทีนย่อย **lseek**

รูทีนย่อย	คำอธิบาย
lseek	อนุญาตให้กระบวนการกำหนดตำแหน่งที่ไบต์ที่กำหนด รูทีนย่อย **lseek** กำหนดตำแหน่งตัวชี้ที่ไบต์ที่กำหนด โดยตัวแปร *Offset* ค่า *Offset* สามารถถูกคำนวณจากจุดต่อไปนี้ในไฟล์ (ที่กำหนดโดยค่าของตัวแปร *Whence*): **ออฟเซ็ตสัมบูรณ์** จุดเริ่มต้นของไฟล์ **ออฟเซ็ตสัมพันธ์** ตำแหน่งของตัวชี้ก่อนหน้า **ออฟเซ็ต end_relative** จุดสิ้นสุดของไฟล์

ค่าที่ส่งกลับสำหรับรูทีนย่อย **lseek** เป็นค่าปัจจุบันของตำแหน่งตัวชี้ในไฟล์ ตัวอย่างเช่น:

```
cur_off= lseek(fd, 0, SEEK_CUR);
```

รูทีนย่อย **lseek** ถูกสร้างใน ตารางไฟล์ การดำเนินการอ่านและเขียนตามลำดับใช้ตำแหน่งใหม่ ของออฟเซ็ตเป็นจุดเริ่มต้น

หมายเหตุ: ออฟเซ็ตไม่สามารถถูกเปลี่ยนแปลงบนไฟล์ไพพ์หรือชนิดซ็อกเก็ต

รูทีนย่อย	คำอธิบาย
fclear	รูทีนย่อยที่สร้างพื้นที่ว่างในไฟล์โดยตั้งค่าจำนวนไบต์ ที่กำหนดในตัวแปร *NumberOfBytes* เป็นศูนย์ เริ่มต้นที่ออฟเซ็ตปัจจุบัน รูทีนย่อย **fclear** ไม่สามารถ ถูกใช้ถ้าแฟล็ก **O_DEFER** ถูกตั้งค่าเมื่อ เปิดไฟล์

การอ่านไฟล์

ส่วนนี้อธิบายรูทีนย่อยการอ่าน

การอ่านไฟล์

รูทีนย่อย	คำอธิบาย
read	รูทีนย่อยที่คัดลอกจำนวนไบต์ที่ระบุจากไฟล์เปิด ไปที่บัฟเฟอร์ที่ระบุ การคัดลอกเริ่มที่จุดที่กำหนดโดยออฟเซ็ต ปัจจุบัน จำนวนไบต์และบัฟเฟอร์ถูกระบุโดยพารามิเตอร์ *NBytes* และ *Buffer*

รูทีนย่อย **read** ทำงาน ดังต่อไปนี้:

1. ตรวจสอบว่าพารามิเตอร์ *FileDescriptor* ถูกต้อง และกระบวนการนั้นมีสิทธิ **read** จากนั้นรูทีนย่อย รับรายการตารางไฟล์ที่ระบุโดยพารามิเตอร์ *FileDescriptor*
2. ตั้งค่าแฟล็กในไฟล์เพื่อระบุว่าการดำเนินการอ่านกำลังทำงานอยู่ การดำเนินการ นี้ล็อกกระบวนการอื่นออกจากไฟล์ระหว่างการดำเนินการ
3. แปลงค่าไบต์ออฟเซ็ตและค่าของตัวแปร *NBytes* ลงในบล็อกแอดเดรส
4. ถ่ายโอนเนื้อหาของบล็อกที่ระบุลงในบัฟเฟอร์พื้นที่จัดเก็บข้อมูล
5. คัดลอกเนื้อหาของบัฟเฟอร์พื้นที่จัดเก็บข้อมูลลงในพื้นที่ที่กำหนดโดย ตัวแปร *Buffer*
6. อัพเดตออฟเซ็ตปัจจุบันตามจำนวนไบต์ที่อ่านจริง รีเซ็ตออฟเซ็ตที่ข้อมูลถูกอ่านตามลำดับโดย กระบวนการอ่านถัดไป
7. ลดจำนวนไบต์ที่อ่านจากผลรวมที่ระบุในตัวแปร *NByte*
8. ลูปจนกว่าจำนวนไบต์ที่จะถูกอ่านเป็นไปตามต้องการ

9. ส่งกลับผลรวมจำนวนไบต์ที่อ่าน

วงจรสมบูรณ์เมื่อไฟล์ที่จะอ่านว่าง จำนวนไบต์ที่ร้องขอเป็นไปตามที่ต้องการ หรือพบข้อผิดพลาดในการอ่านระหว่าง กระบวนการ

เพื่อหลีกเลี่ยงการวนซ้ำเพิ่ม ในลูปการอ่าน เริ่ม การร้องขอการอ่านที่จุดเริ่มต้อนของขอบเขตบล็อกข้อมูลและเป็นผลคูณ ของขนาดบล็อกข้อมูล ถ้ากระบวนการอ่านบล็อกตามลำดับ ระบบปฏิบัติการ จะถือว่าการอ่านข้อมูลที่ตามมาจะเป็นไปตามลำดับด้วย

ระหว่างการดำเนินการอ่าน i-node ถูกล็อก ไม่ อนุญาตให้กระบวนการอื่นปรับเปลี่ยนเนื้อหาของไฟล์ขณะการอ่าน กำลังดำเนินการ อย่างไรก็ตามไฟล์ถูกปลดล็อกทันที่ที่การดำเนินการอ่าน สมบูรณ์ ถ้ากระบวนการอื่นเปลี่ยนแปลงไฟล์ระหว่างการอ่านสองการดำเนินการ ข้อมูลผลลัพธ์จะต่างกัน แต่ความสมบูรณ์ของโครงสร้างข้อมูลจะถูก รักษาไว้

ตัวอย่างดังต่อไปนี้แสดงวิธีใช้รูทีนย่อย read เพื่อนับจำนวน null ไบต์ในไฟล์ **foo**:

```
#include <fcntl.h>
#include <sys/param.h>

main()
{
        int fd;
        int nbytes;
        int nnulls;
        int i;
        char buf[PAGESIZE];      /*A convenient buffer size*/
        nnulls=0;
        if ((fd = open("foo",O_RDONLY)) < 0)
                exit();
        while ((nbytes = read(fd,buf,sizeof(buf))) > 0)
                for (i = 0; i < nbytes; i++)
                        if (buf[i] == '\0';
                                nnulls++;
        printf("%d nulls found\n", nnulls);
}
```

การเขียนไฟล์

ส่วนนี้อธิบายรูทีนย่อยการเขียน

รูทีนย่อย	คำอธิบาย
เขียน	รูทีนย่อยที่เพิ่มจำนวนของข้อมูลที่ระบุในตัวแปร *NBytes* จากพื้นที่ที่กำหนดโดยตัวแปร *Buffer* ไปที่ไฟล์ที่อธิบายโดยตัวแปร *FileDescriptor* ซึ่งทำงานเหมือนกับรูทีนย่อย **read** ออฟเซ็ตไบต์สำหรับการเขียนพบได้ในออฟเซ็ตปัจจุบันของ ตารางไฟล์ระบบ

ถ้าคุณเขียนไปที่ไฟล์ที่ไม่มีบล็อก ที่ตรงกับออฟเซ็ตไบต์ที่เป็นผลมาจากกระบวนการเขียนรูทีนย่อย **write** จะจัดสรรบล็อกใหม่ บล็อกใหม่นี้ถูก เพิ่มให้กับข้อมูล i-node ที่กำหนดไฟล์ การเพิ่มบล็อกใหม่ อาจจัดสรรมากกว่าหนึ่งบล็อกถ้าระบบไฟล์ภายในจำเป็น ต้องเพิ่มบล็อกสำหรับการกำหนดแอดเดรสบล็อกไฟล์

ระหว่างการดำเนินการเขียน i-node ถูกล็อก ไม่ อนุญาตให้กระบวนการอื่นปรับเปลี่ยนเนื้อหาของไฟล์ขณะการเขียน กำลังดำเนินการ อย่างไรก็ตามไฟล์ถูกปลดล็อกทันที่ที่การดำเนินการเขียน สมบูรณ์ ถ้ากระบวนการอื่นเปลี่ยนแปลงไฟล์ระหว่างการเขียนสองการดำเนินการ ข้อมูลผลลัพธ์จะต่างกัน แต่ความสมบูรณ์ของโครงสร้างข้อมูลจะถูก รักษาไว้

รูทีนย่อย **write** ลูปเหมือนกับ รูทีนย่อย **read** โดยเขียน หนึ่งบล็อกแบบโลจิคัลไปที่ดิสก์สำหรับแต่ละการวนซ้ำ ในแต่ละการวนซ้ำ กระบวนการเขียน ทั้งบล็อกหรือเฉพาะบางส่วน ถ้าเฉพาะส่วนของบล็อกข้อมูลจำเป็น เพื่อให้การดำเนินการสำเร็จ รูทีนย่อย **write** อ่าน บล็อกจากดิสก์เพื่อหลีกเลี่ยงการเขียนทับข้อมูลที่มีอยู่ ถ้าต้องการ บล็อกทั้งหมด รูทีนย่อยจะไม่อ่านบล็อกเนื่องจากทั้งบล็อก ถูกเขียนทับ การดำเนินการอ่านดำเนินไปทีละบล็อกจนกว่าจำนวนไบต์ ที่กำหนดในพารามิเตอร์ *NBytes* ได้ถูกเขียน

การเขียนที่หน่วงเวลา

คุณสามารถกำหนดกระบวนการเขียนที่หน่วงเวลาด้วยแฟล็ก **O_DEFER** จากนั้นข้อมูลถูกถ่ายโอนไปที่ดิสก์เป็น ไฟล์ชั่วคราว คุณลักษณะการเขียนที่หน่วงเวลาแคชข้อมูลในกรณีที่กระบวนการ อื่นอ่านหรือเขียนข้อมูลเร็วกว่ากำหนด การเขียนที่หน่วงเวลาลดการทำงานของดิสก์ โปรแกรมจำนวนมาก เช่นเมล์และตัวแก้ไข สร้างไฟล์ชั่วคราวในไดเร็กทอรี **/tmp** และลบไฟล์ออกอย่างรวดเร็ว

เมื่อไฟล์ถูกเปิดด้วยแฟล็ก deferred update (**O_DEFER**) ข้อมูลไม่ถูกเขียนไปที่พื้นที่จัดเก็บข้อมูลถาวร จำกว่ากระบวนการทำการเรียกรูทีนย่อย **fsync** หรือกระบวนทำการซิงโครนัส **write** ไปที่ไฟล์ (ที่เปิดด้วยแฟล็ก **O_SYNC**) รูทีนย่อย **fsync** บันทึกการเปลี่ยนแปลงทั้งหมดในไฟล์เปิดไปที่ดิสก์ โปรดดูที่รูทีนย่อย **open** สำหรับคำอธิบายของแฟล็ก **O_DEFER** และ **O_SYNC**

การตัดไฟล์

รูทีนย่อย **truncate** หรือ **ftruncate** เปลี่ยนความยาว ของไฟล์ปกติ กระบวนการตัดต้องมีสิทธิ write กับไฟล์ ค่าตัวแปร *Length* ระบุขนาดของ ไฟล์หลังจากการตัดสมบูรณ์ การวัดทั้งหมด สัมพันธ์กับไบต์แรกของไฟล์ ไม่ใช่ออฟเซ็ตปัจจุบัน ถ้า ความยาวใหม่ (ที่กำหนดในตัวแปร *Length*) น้อย กว่าความยาวก่อนหน้า ข้อมูลระหว่างทั้งสองจุดจะถูกลบ ถ้าความยาว ใหม่มากกว่าความยาวที่มีอยู่ ศูนย์จะถูกเพิ่มเพื่อขยาย ขนาดไฟล์ เมื่อการตัดสมบูรณ์ บล็อกเต็มถูกส่งกลับไปที่ ไฟล์ระบบ และขนาดไฟล์ถูกอัพเดต

Direct I/O vs. normal cached I/O

โดยทั่วไป เพจไฟล์แคช JFS หรือ JFS2 ในหน่วยความจำ เคอร์เนล เมื่อแอ็พพลิเคชันทำการร้องขอเพื่ออ่านไฟล์ ถ้าเพจไฟล์ไม่อยู่ใน หน่วยความจำ JFS หรือ JFS2 จะอ่านข้อมูลจากดิสก์ลงใน แคชไฟล์ จากนั้นคัดลอกข้อมูลจากแคชไฟล์ไปยัง บัฟเฟอร์ของผู้ใช้..

สำหรับ การเขียนแอ็พพลิเคชัน ข้อมูลจะถูกคัดลอกจากบัฟเฟอร์ของผู้ใช้เท่านั้น ลงในแคช การเขียนข้อมูลจริงไปที่ดิสก์กระทำภายหลัง

ประเภทของนโยบายการแคชนี้ มีประสิทธิภาพอย่างมาก ที่สุดเมื่อ อัตราการเข้าถึงแคชมีค่าสูง และยังเปิดใช้งานนโยบาย read-ahead และ write-behind สุดท้าย มีการเขียนไฟล์อะซิงโครนัส อนญาตให้แอ็พพลิเคชัน ดำเนินการประมวลผลต่อไปแทนการรอการร้องขอ I/O ให้เสร็จสิ้นก่อน

Direct I/O เป็นนโยบายการแคชทางเลือก ที่ทำให้ ข้อมูลไฟล์ถูกถ่ายโอนโดยตรงระหว่างดิสก์และบัฟเฟอร์ของผู้ใช้ Direct I/O สำหรับไฟล์ทำงานเหมือนกับ raw I/O สำหรับอุปกรณ์ แอ็พพลิเคชัน สามารถใช้ direct I/O บนไฟล์ JFS หรือ JFS2

ประโยชน์ของ direct I/O

ประโยชน์หลักของ direct I/O คือเพื่อลดการใช้งาน CPU สำหรับการอ่านและการเขียนไฟล์โดยขจัดการคัดลอกจากแคชไปที่บัฟเฟอร์ ของผู้ใช้

และยังมีประโยชน์สำหรับข้อมูลไฟล์ซึ่งซึ่งมีอัตราการเข้าถึง แคชไม่ดี ถ้าอัตราการเข้าถึงแคชต่ำ แล้วการร้องขอการอ่านส่วนใหญ่จำเป็น ต้องไปที่ดิสก์ Direct I/O ยังสามารถให้ประโยชน์กับแอ็พพลิเคชันที่ต้องใช้การเขียนซิงโครนัส เนื่องจากการเขียนเหล่านี้ต้องไปที่ดิสก์ ในทั้งสองกรณี การใช้งาน CPU ลดลงเนื่องจากการคัดลอกข้อมูลถูกขจัดไป

ประโยชน์ข้อที่สองของ direct I/O คือทำให้แอ็พพลิเคชัน หลีกเลี่ยงการลดประสิทธิภาพของการแคชไฟล์อื่น เมื่อใดก็ตามที่ไฟล์ ถูกอ่านหรือเขียน ไฟล์นั้นมีการชิงพื้นที่ในแคช สถานการณ์ นี้อาจทำให้ข้อมูลไฟล์อื่นถูกพลักดันออกไปจากแคช ถ้าข้อมูลที่แคชใหม่มีคุณลักษณะการนำมาใช้ใหม่ที่แย่มาก ประสิทธิผลของ แคชสามารถถูกทำให้ลดลงได้ Direct I/O ให้ความสามารถแก่แอ็พพลิเคชันในการระบุ ไฟล์ซึ่งนโยบายการแคชปกติไม่มีประสิทธิภาพ ดังนั้นรีลีสพื้นที่ แคชมากขึ้นสำหรับไฟล์ที่นโยบายมีประสิทธิภาพ

ต้นทุนทางประสิทธิภาพของ direct I/O

แม้ว่า direct I/O สามารถช่วยลดการใช้ CPU โดยปกติการใช้งาน direct I/O มีผลให้กระบวนการใช้เวลามากขึ้นในการทำให้สมบูรณ์โดยเฉพาะสำหรับ การร้องขอขนาดเล็ก สิ่งชดเชยนี้เกิดจากความแตกต่างพื้นฐานระหว่าง I/O ที่แคชปกติและ direct I/O

การอ่าน direct I/O

ทุกการอ่าน direct I/O ทำให้มีการอ่านซิงโครนัสจาก ดิสก์ไม่เหมือนกับนโยบาย I/O ที่แคชปกติซึ่งการจากแคช ก็เพียงพอ นี่สามารถทำให้ประสิทธิภาพลดลงมาก ถ้าข้อมูล อยู่ในหน่วยความจำภายใต้นโยบายการแคชปกติ

Direct I/O ยังข้ามอัลกอริธึม JFS หรือ JFS2 read-ahead ปกติ อัลกอริธึมเหล่านี้สามารถมีประสิทธิภาพได้สูงสดสำหรับการเข้าถึง ไฟล์ตามลำดับโดยการส่งการร้องขอการอ่านที่ขนาดใหญ่มาและโดย การซ้อนทับการอ่านของบล็อกอนาคตที่มีการประมวลผลแอ็พพลิเคชัน

แอ็พพลิเคชันสามารถชดเชยการสูญเสีย JFS หรือ JFS2 read-ahead โดยส่งการร้องขอการอ่านที่ใหญ่ขึ้น ที่ขั้นต่ำสุด ตัวอ่าน direct I/O ควรส่งการร้องของการอ่านอย่างน้อย 128k เพื่อให้ตรงกับคุณลักษณะ JFS หรือ JFS2 read-ahead

แอ็พพลิเคชันยังสามารถจำลอง JFS หรือ JFS2 read-ahead โดยการส่งอะซิงโครนัส direct I/O read-ahead โดยใช้หลายเธรด หรือโดยใช้รูทีนย่อย **aio_read**

การเขียน Direct I/O

ทุกการเขียน direct I/O ทำให้มีการเขียนซิงโครนัสไปที่ ดิสก์ไม่เหมือนกับนโยบาย I/O ที่แคชปกติ ซึ่งข้อมูลถูกคัดลอกเท่านั้น แล้วถูกเขียนไปที่ดิสก์ภายหลัง ความแตกต่างพื้นฐานนี้สามารถทำให้เกิด ผลเสียต่อประสิทธิภาพอย่างเห็นได้ชัดสำหรับแอ็พพลิเคชันที่ถูกแปลงเพื่อใช้ direct I/O

การขัดกันของโหมดการเข้าถึงไฟล์

เพื่อหลีกเลี่ยงปัญหาเรื่องความสอดคล้องกันระหว่างโปรแกรมที่ใช้ direct I/O และโปรแกรมที่ใช้ I/O ที่แคชปกติ direct I/O เป็นโหมดใช้งาน เฉพาะ ถ้ามีไฟล์เปิดอยู่หลายไฟล์ และบางไฟล์เป็นแบบ direct ขณะนี้ไฟล์อื่นไม่ได้เป็น ไฟล์จะอยู่ในโหมดการเข้าถึงที่แคชปกติ เฉพาะเมื่อไฟล์ถูกเปิดแบบเฉพาะ โดยโปรแกรม I/O ไฟล์จะถูก นำไปไว้ในโหมด direct I/O

เช่นเดียวกัน ถ้าไฟล์ถูกแม็พลงในหน่วยความจำเสมือน ผ่านการเรียกระบบ **shmat** หรือ **mmap** ไฟล์จะอยู่ในโหมดที่แคชปกติ

JFS หรือ JFS2 จะพยายามย้ายไฟล์ไปไว้ในโหมด direct I/O ทุกครั้งที่ความขัดแย้งล่าสุดหรือการเข้าถึงแบบ non-direct ถูกขจัดไป (โดยรูทีนย่อย **close, munmap** หรือ **shmdt**) การเปลี่ยนไฟล์จากโหมดปกติไปเป็นโหมด I/O อาจต้องจ่ายราคาสูง เนื่องจากต้องการการเขียนเพจที่มีการเปลี่ยนแปลงทั้งหมด ลงในดิสก์และเอาเพจของไฟล์ทั้งหมดออกจากหน่วยความจำ

การเปิดใช้งานแอ็พพลิเคชันเพ่อใช้ direct I/O

แอ็พพลิเคชันเปิดใช้งานการเข้าถึง direct I/O กับไฟล์โดยผ่าน แฟล็ก O_DIRECT ไปที่รูทีนย่อย open แฟล็ก นี้ถูกกำหนดในไฟล์ **fcntl.h** แอ็พพลิเคชันต้องถูกคอมไพล์โดยมีการเปิดใช้งาน _ALL_SOURCE เพื่อดูข้อกำหนดของ O_DIRECT

ข้อกำหนดการจัด Offset/Length/Address ของบัฟเฟอร์เป้าหมาย

เพื่อให้ direct I/O ทำงานได้อย่างมีประสิทธิภาพ การร้องขอควร กำหนดเงื่อนไขอย่างเหมาะสม แอ็พพลิเคชันสามารถเคียวรีข้อกำหนดการจัด ออฟเซ็ต ความยาว และแอดเดรส โดยใช้รูทีนย่อย **finfo** และ **ffinfo** เมื่อคำสั่ง FI_DIOCAP ถูกใช้ รูทีนย่อย **finfo** และ **ffinfo** ส่งกลับข้อมูล ในโครงสร้าง **diocapbuf** ตามที่อธิบายในไฟล์ **sys/finfo.h** โครงสร้างนี้จะมีฟิลด์ต่อไปนี้:

`dio_offset`
การจัดแนวออฟเซ็ตที่แนะนำสำหรับ direct I/O เขียนข้อมูลไปที่ไฟล์ ในระบบไฟล์นี้

`dio_max`
ความยาวการเขียนสูงสุดที่แนะนำสำหรับ direct I/O เขียนข้อมูลไปที่ไฟล์ ในระบบนี้

`dio_min`
ความยาวการเขียนต่ำสุดที่แนะนำสำหรับ direct I/O เขียนข้อมูลไปที่ไฟล์ ในระบบไฟล์นี้

`dio_align`
การจัดแนวบัฟเฟอร์ที่แนะนำสำหรับ direct I/O เขียนข้อมูลไปที่ไฟล์ ในระบบไฟล์นี้

ความล้มเหลวจากการไม่เป็นไปตามข้อกำหนดเหล่านี้อาจทำให้การอ่าน และการเขียนไฟล์ใช้โมเดลที่แคชปกติ และอาจทำให้ direct I/O ถูก ปิดใช้งานสำหรับไฟล์ ระบบไฟล์ที่ต่างกันอาจมีข้อกำหนดที่ต่างกัน ตามที่แสดงในตารางดังต่อไปนี้

รูปแบบ ระบบไฟล์	dio_offset	dio_max	dio_min	dio_align
JFS fixed, 4 K blk	4 K	2 MB	4 K	4 K
JFS fragmented	4 K	2 MB	4 K	4 K
JFS compressed	n/a	n/a	n/a	n/a
JFS big file	128 K	2 MB	128 K	4 K
JFS2	4 K	4 GB	4 K	4 K

ข้อจำกัดของ Direct I/O

Direct I/O ไม่สนับสนุนไฟล์ในระบบไฟล์ ที่มีการบีบอัดข้อมูล ความพยายามเปิดไฟล์เหล่านี้ด้วย O_DIRECT จะถูกละเว้น และไฟล์จะถูกเข้าถึงด้วยเมธอด I/O ที่แคชปกติ

การทำ Direct I/O และ data I/O integrity ให้สมบูรณ์

แม้ว่าการเขียน direct I/O ถูกทำแบบซิงโครไนซ์ แต่การเขียนข้อมูลนี้ ไม่ได้จัดเตรียมการทำ I/O data integrity ให้สมบูรณ์ที่ซิงโครไนซ์ ตามที่กำหนดโดย POSIX แอ็พพลิเคชันที่ต้องการคุณลักษณะนี้ควรใช้ O_DSYNC นอกเหนือจาก O_DIRECT O_DSYNC การันตีว่าข้อมูลทั้งหมดและ metadata ที่เพียงพอ (ตัวอย่างเช่น บล็อกทางอ้อม) ได้ถูกเขียนไปที่ stable store เพื่อให้สามารถเรียกคืนข้อมูล หลังจากระบบขัดข้อง O_DIRECT เขียนเฉพาะข้อมูลเท่านั้น ไม่เขียน metadata

การทำงานกับไพพ์

ไพพ์คืออ็อบเจ็กต์ไม่มีชื่อที่สร้างเพื่ออนุญาตให้สองกระบวนการ สื่อสารกันได้

กระบวนการหนึ่งอ่านข้อมูลและอีกกระบวนการเขียนข้อมูลไปที่ไฟล์ ไพพ์ ชนิดเฉพาะของไฟล์นี้เรียกอีกอย่างว่าไฟล์ first-in-first-out (FIFO) บล็อกข้อมูลของ FIFO ถูกจัดการในคิวแบบวงกลม รักษาตัวชี้ การอ่านและเขียนไว้ภายในเพื่อคงลำดับ FIFO ของข้อมูล ตัวแปรระบบ **PIPE_BUF** ที่กำหนดในไฟล์ **limits.h** กำหนดจำนวนไบต์สูงสุดที่การันตี ความเป็น atomic เมื่อเขียนไปที่ไพพ์

เชลล์ใช้ไพพ์ที่ไม่มีชื่อเพื่อสร้างการทำไพพ์ไลน์คำสั่ง ไพพ์ไม่มีชื่อส่วนใหญ่ถูกสร้างโดยเชลล์ สัญลักษณ์ | (แนวดิ่ง) แสดง ไพพ์ระหว่างกระบวนการ ในตัวอย่างดังต่อไปนี้ เอาต์พุตของคำสั่ง **ls** ถูกพิมพ์ไปที่หน้าจอ:

```
ls | pr
```

ไพพ์ถูกปฏิบัติเหมือนไฟล์ปกติเท่าที่เป็นไปได้ โดยปกติ ข้อมูลออฟเซ็ตปัจจุบันถูกเก็บในตารางไฟล์ระบบ อย่างไรก็ตาม เนื่องจากไพพ์ถูกแบ่งใช้โดยกระบวนการ ตัวชี้ อ่าน/เขียน ต้อง จำเพาะกับไฟล์ไม่ใช่กับกระบวนการ รายการตารางไฟล์ถูกสร้างโดยรูทีนย่อย **open** และเฉพาะกับกระบวนการ open ไม่ใช่กับไฟล์ กระบวนการที่มีการเข้าถึงไพพ์แบ่งใช้การเข้าถึงผ่าน รายการตารางไฟล์ระบบ

การใช้รูทีนย่อย pipe

รูทีนย่อย **pipe** สร้างแชนเนล ระหว่างกระบวนการและส่งกับสอง file descriptors File descriptor 0 ถูกเปิด เพื่อการอ่าน File descriptor 1 ถูกเปิดเพื่อการเขียน การดำเนินการอ่าน เข้าถึงข้อมูลในแบบ FIFO file descriptors เหล่านี้ถูกใช้กับรูทีนย่อย **read, write** และ **close**

ในตัวอย่างดังต่อไปนี้กระบวนการ child ถูกสร้าง และส่ง process ID ของตัวเองกลับไปผ่านไพพ์:

```
#include <sys/types.h>
main()
{
        int p[2];
        char buf[80];
        pid_t pid;

        if (pipe(p))
        {
                perror("pipe failed");
            exit(1)'
        }
        if ((pid=fork()) == 0)
        {
                                    /* in child process */
            close(p[0]);            /*close unused read */
                                    *side of the pipe */
            sprintf(buf,"%d",getpid());
                                    /*construct data */
```

```
                                /*to send */
            write(p[1],buf,strlen(buf)+1);
                    /*write it out, including
                    /*null byte */
            exit(0);
    }
                                 /*in parent process*/
    close(p[1]);                 /*close unused write side of pipe */
    read(p[0],buf,sizeof(buf));  /*read the pipe*/
    printf("Child process said: %s/n", buf);
                                /*display the result */
    exit(0);
}
```

ถ้ากระบวนการอ่านไพพ์ว่าง กระบวนการจะรอ จนกว่าข้อมูลมาถึง ถ้ากระบวนการเขียนข้อมูลไปที่ไพพ์ซึ่งเต็ม (**PIPE_BUF**) กระบวนการรอจนกว่าพื้นที่พร้อมใช้งาน ถ้าด้านเขียนข้อมูล ของไพพ์ถูกปิด การดำเนินการอ่านตามลำดับไปที่ไพพ์ส่งกลับ end-of-file

รูทีนย่อยอื่นที่ควบคุมไพพ์คือรูทีนย่อย **popen** และ **pclose**:

popen สร้างไพพ์ (โดยใช้รูทีนย่อย **pipe**) จากนั้น แยกเพื่อสร้างสำเนาของตัวเรียก กระบวนการ child เลือกว่าควรจะ อ่าน หรือเขียน ปิดอีกด้านหนึ่งของไพพ์ แล้วเรียก เชลล์ (โดยใช้รูทีนย่อย **execl**) เพื่อรันกระบวนการที่ต้องการ

parent ปิดปลายอีกด้านของไพพ์ที่ ไม่ได้ใช้ การปิดนี่จำเป็นในการทำให้การทดสอบ end-of-file ทำงานถูกต้อง ตัวอย่างเช่น ถ้ากระบวนการ child ที่พยายามอ่านไพพ์ไม่ได้ปิด ปลายด้านที่เขียนของไพพ์ กระบวนการจะไม่ได้พบสิ้นสุดไฟล์จุดสิ้นสุดของไฟล์บน ไพพ์ เนื่องจากมีหนึ่งกระบวนการเขียนทำงานอยู่

วิธีปกติในการเชื่อมโยงไพพ์ descriptor กับอินพุตมาตรฐาน ของกระบวนการคือ:

```
close(p[1]);
close(0);
dup(p[0]);
close(p[0]);
```

รูทีนย่อย **close** ยกเลิกการเชื่อมต่อ file descriptor 0 อินพุตมาตรฐาน รูทีนย่อย **dup** ส่งกลับสำเนาของไฟล์ descriptor ที่เปิดอยู่ ไฟล์ descriptors ถูกกำหนดตามลำดับจากน้อยไปหามากและ ค่าแรกที่พร้อมใช้งานจะถูกส่งกลับ ผลของรูทีนย่อย **dup** คือเพื่อคัดลอกไฟล์ descriptor สำหรับไพพ์ (ด้านอ่าน) ไปที่ file descriptor 0 ดังนั้นอินพุตมาตรฐาน กลายเป็นด้านอ่านของไพพ์ ท้ายที่สุด ด้านการอ่าน ก่อนหน้าถูกปิด กระบวนการเหมือนกันสำหรับกระบวนการ child ในการเขียนข้อมูล จาก parent

pclose ปิดไพพ์ระหว่างโปรแกรมที่เรียกและคำสั่งเชลล์ที่จะถูก ดำเนินการ ใช้รูทีนย่อย **pclose** เพื่อปิดสตรีมที่เปิดอยู่ ด้วยรูทีนย่อย **popen**

รูทีนย่อย **pclose** รอกระบวนการที่สัมพันธ์กันจบการทำงาน จากนั้นปิด และส่งกลับสถานะจบการทำงานของคำสั่ง รูทีนย่อยนี้เหมาะกับ รูทีนย่อย close เนื่องจาก **pclose** รอกระบวนการ child เสร็จสิ้นการทำงานก่อนการปิดไพพ์ สิ่งสำคัญเท่าเทียมกัน เมื่อกระบวนการ สร้างหลาย children เฉพาะจำนวนที่จำกัดของกระบวนการ child ที่ยังไม่จบการทำงาน ที่ยังทำงานอยู่ได้ แม้ว่าบางกระบวนการได้เสร็จสิ้นงานแล้ว การดำเนินการรอ อนุญาตให้กระบวนการ child ทำงานให้สมบูรณ์

Synchronous I/O

โดยค่าเริ่มต้น การเขียนไปที่ไฟล์ในระบบไฟล์ JFS หรือ JFS2 เป็นแบบอะซิงโครนัส

อย่างไรก็ตามระบบไฟล์ JFS และ JFS2 สนับสนุนชนิดของ synchronous I/O ดังต่อไปนี้:

- ที่ระบุโดยแฟล็กเปิด O_DSYNC เมื่อไฟล์ถูกเปิดโดยใช้โหมดเปิด O_DSYNC การเรียก write () ของระบบจะไม่ส่งกลับจนกว่าข้อมูลไฟล์แล meta-data ระบบไฟล์ทั้งหมดที่จำเป็นในการเรียกคืนข้อมูลไฟล์ ทั้งคู่ได้ถูกเขียนข้อมูลไปที่ตำแหน่งพื้นที่จัดเก็บข้อมูลถาวร
- ที่ระบุโดยแฟล็กเปิด O_SYNC นอกจากรายการที่ระบุโดย O_DSYNC, O_SYNC ระบุว่าการเรียก write () ของระบบจะไม่ส่งกลับจนกว่า ไฟล์แอ็ตทริบิวต์ทั้งหมดที่สัมพันธ์กับ I/O จะถูกเขียนลงในตำแหน่งพื้นที่จัดเก็บข้อมูล ถาวร แม้ว่าแอ็ตทริบิวต์จะไม่จำเป็นในการเรียกข้อมูลไฟล์

  ก่อนที่โหมดเปิด O_DSYNC จะมีขึ้น AIX ใช้ซีแมนทิกส์ O_DSYNC กับ O_SYNC ด้วยเหตุผลความเข้ากันได้ของไบนารี ลักษณะการทำงานนี้จึงยังคงมีอยู่ ถ้าจำเป็นต้องมีลักษณะการทำงาน O_SYNC จริง ดังนั้นทั้งแฟล็กเปิด O_DSYNC และ O_SYNC ต้องถูกระบุ การส่งออกตัวแปรสภาวะแวดล้อม **XPG_SUS_ENV=ON** ยังเปิดใช้งานลักษณะการทำงาน O_SYNC จริงเช่นกัน
- ที่ระบุโดยแฟล็กเปิด O_RSYNC และมีการใช้ลักษณะการทำงาน ที่สัมพันธ์กับ O_SYNC หรือ _DSYNC ในการอ่านข้อมูลสำหรับไฟล์ในระบบไฟล์ JFS และ JFS2 เฉพาะการรวมกันของ O_RSYNC | O_SYNC ที่มีความหมาย ระบุว่า การเรียกระบบอ่านจะไม่ส่งกลับจนกว่าเวลาเข้าถึงของไฟล์ จะถูกเขียนลงที่ตำแหน่งพื้นที่จัดเก็บข้อมูลถาวรของไฟล์

สถานะไฟล์

ข้อมูลสถานะไฟล์อยู่ใน i-node

รูทีนย่อย **stat** ถูกใช้เพื่อ ส่งกลับข้อมูลเกี่ยวกับไฟล์ รูทีนย่อย **stat** รายงานประเภทไฟล์ เจ้าของไฟล์ โหมดการเข้าถึง ขนาดไฟล์ จำนวนลิงก์ จำนวน i-node และจำนวนครั้งการเข้าถึงไฟล์ รูทีนย่อยเหล่านี้เขียนข้อมูลลงใน โครงสร้างข้อมูลที่ออกแบบโดยตัวแปร *Buffer* กระบวนการต้องมีสิทธิในการค้นหาไดเร็กทอรีในพาธไปยัง ไฟล์ที่ออกแบบ

รูทีนย่อย	คำอธิบาย
stat	รูทีนย่อยที่ส่งคืนข้อมูลเกี่ยวกับที่กำหนดชื่อโดยพารามิเตอร์ *Path* ถ้าขนาดของไฟล์ไม่สามารถแสดงใน โครงสร้างที่ออกแบบโดยตัวแปร *Buffer* ได้ **stat** จะล้มเหลวโดย errno ตั้งค่าเป็น EOVERFLOW
lstat	รูทีนย่อยที่ให้ข้อมูลเกี่ยวกับลิงก์สัญลักษณ์ และรูทีนย่อย **stat** ส่งกลับข้อมูลเกี่ยวกับไฟล์ที่ถูกอ้างถึง โดยลิงก์
fstat	ส่งคืนข้อมูลจากไฟล์ที่เปิดโดยใช้ file descriptor

รูทีนย่อย **statfs, fstafs** และ **ustat** ส่งกลับข้อมูลสถานะเกี่ยวกับระบบไฟล์

รูทีนย่อย	คำอธิบาย
fstatfs	ส่งกลับข้อมูลเกี่ยวกับระบบไฟล์ที่มีไฟล์ ที่สัมพันธ์กับ file descriptor ที่กำหนด โครงสร้างของข้อมูลที่ส่งกลับ ถูกอธิบายในไฟล์ **/usr/include/sys/statfs.h** สำหรับ รูทีนย่อย **statfs** and **fstatfs** และในไฟล์ **ustat.h** สำหรับรูทีนย่อย **ustat**
statfs	ส่งกลับข้อมูลเกี่ยวกับระบบไฟล์ที่มีไฟล์ที่ระบุโดยพารามิเตอร์ *Path*
ustat	ส่งกลับข้อมูลเกี่ยวกับระบบไฟล์ที่เม้าท์ที่กำหนดโดยตัวแปร *Device* ตัวระบุอุปกรณ์นี้สำหรับไฟล์ที่ถูกกำหนดใดๆ และสามารถถูกกำหนดโดยการพิจารณาฟิลด์ `st_dev` ของโครงสร้าง **stat** ที่กำหนดในไฟล์ **/usr/include/sys/stat.h** รูทีนย่อย **ustat** จะไม่ถูกใช้โดยรูทีนย่อย **statfs** และ **fstatfs** อีกแล้ว
utimes และ **utime**	รวมทั้งมีผลต่อข้อมูลสถานะไฟล์โดยการเปลี่ยนการเข้าถึงไฟล์และ เวลาที่ทำการเปลี่ยนใน i-node

ความสามารถเข้าถึงไฟล์

ทุกไฟล์ถูกสร้างด้วย โหมด access แต่ละโหมด access ให้สิทธิ read, write หรือ execute แก่ผู้ใช้ กลุ่มผู้ใช้ และผู้ใช้อื่น

บิต access ในไฟล์ที่สร้างใหม่เป็นผลจากการ กลับบิต **umask** แล้ว AND กับบิตโหมดการสร้างไฟล์ ที่ตั้งค่าโดยกระบวนการสร้าง เมื่อไฟล์ใหม่ถูกสร้างโดยกระบวนการ ระบบปฏิบัติการดำเนินการดังต่อไปนี้:

- กำหนดสิทธิของกระบวนการสร้าง
- เรียกคืนค่า **umask** ที่เหมาะสม
- กลับค่า **umask**
- ใช้การดำเนินการ AND เพื่อรวมสิทธิของกระบวนการสร้าง กับการกลับค่า **umask**

ตัวอย่างเช่น ถ้าไฟล์ที่มีอยู่มีการตั้งค่าบิตสิทธิเป็น 027 ผู้ใช้จะไม่ได้รับสิทธิใด สิทธิ Write ให้แก่ กลุ่ม การเข้าถึง Read, write และ execute ถูกกำหนดให้กับส่วนอื่นทั้งหมด ค่า **umask** โหมดสิทธิ 027 จะเป็น 750 (ค่าตรงกันข้าม กับสิทธิดั้งเดิม) เมื่อ 750 ถูก AND กับ 666 (บิต โหมดการสร้างไฟล์ตั้งค่าโดยการเรียกระบบที่สร้างไฟล์) สิทธิแท้จริง สำหรับไฟล์คือ 640 อีกรูปแบบหนึ่งของตัวอย่างนี้:

```
027 = _ _ _  _ W _  R W X       Existing file access mode
750 = R W X  R _ X  _ _ _       Reverse (umask) of original
                                permissions
666 = R W _  R W _  R W _       File creation access mode
ANDED TO
750 = R W X  R _ X  _ _ _       The umask value
640 = R W _  R _ _  _ _ _       Resulting file access mode
```

รูทีนย่อย	**คำอธิบาย**
รูทีนย่อย **access**	ตรวจสอบและรายงานเกี่ยวกับโหมดการเข้าถึงของไฟล์ใน พารามิเตอร์ *Pathname* รูทีนย่อยนี้ใช้ user ID และ group ID จริงแทน effective user และ group ID การใช้ real user และ group ID อนุญาตให้โปรแกรมที่มีโหมดการเข้าถึง set-user-ID และ set-group-ID ในการจำกัดการเข้าถึงเฉพาะกับผู้ใช้ที่มีการอนุญาตที่ถูกต้อง
รูทีนย่อย **chmod** และ **fchmod**	การเปลี่ยนสิทธิการเข้าถึงไฟล์
รูทีนย่อย **chown**	รีเซ็ตฟิลด์ความเป็นเจ้าของของ i-node สำหรับไฟล์ และล้างค่าเจ้าของก่อนหน้า ข้อมูลใหม่ถูกเขียนกับ i-node และ process เสร็จสิ้น รูทีนย่อย **chmod** ทำงานในแบบเดียวกัน แต่แฟล็กโหมดสิทธิ ถูกเปลี่ยนแทนการเป็นเจ้าของไฟล์
umask	ตั้งและรับค่าของมาสก์การสร้างไฟล์

ในตัวอย่างดังต่อไปนี้ ผู้ใช้ไม่มีสิทธิการเข้าถึง ไฟล์ `secrets` อย่างไรก็ตาม เมื่อโปรแกรม `special` ถูกรันและโหมดการเข้าถึงสำหรับโปรแกรม คือ set-uID `root`, โปรแกรมสามารถเข้าถึงไฟล์โปรแกรมต้องใช้รูทีนย่อย **access** เพื่อป้องกัน เวอร์ชันย่อยของความปลอดภัยระบบ

```
$ ls -l
total 0
-r-s--x--x      1 root  system   8290 Jun 09 17:07 special
-rw-------      1 root  system   1833 Jun 09 17:07 secrets
$ cat secrets
cat: cannot open secrets
```

รูทีนย่อย **access** ต้องถูกใช้โดยโปรแกรม set-uID หรือ set-gID เพื่อป้องกันการบุกรุกประเภทนี้ การเปลี่ยน ความเป็นเจ้าของไฟล์และโหมดการเข้าถึงเป็นการดำเนินการที่มีผลกับ i-node, ไม่ใช่ ข้อมูลในไฟล์ เพื่อทำการเปลี่ยนแปลงนี้ เจ้าของกระบวนการต้องมีสิทธิ root user หรือเป็นเจ้าของไฟล์

การสร้างระบบไฟล์ชนิดใหม่

ถ้ามีความจำเป็น ในการสร้างชนิดระบบไฟล์ใหม่ ตัวช่วยระบบไฟล์และตัวช่วยเชื่อมต่อ ต้องถูกสร้าง

ส่วนนี้ให้ข้อมูลเกี่ยวกับการสร้าง ไวยากรณ์ จำเพาะและการกระทำการ ของตัวช่วยระบบไฟล์และตัวช่วยเชื่อมต่อ

ตัวช่วยระบบไฟล์

เพื่อเปิดใช้งานการสนับสนุนหลายชนิดระบบไฟล์ คำสั่งระบบไฟล์ส่วนใหญ่ไม่มีโค้ดที่สื่อสารกับแต่ละ ระบบไฟล์ แต่คำสั่งรวบรวม พารามิเตอร์ ชื่อระบบไฟล์ และข้อมูลอื่น ไม่จำเพาะกับชนิดระบบไฟล์หนึ่ง แล้วส่งข้อมูล นี้ไปที่โปรแกรม back-end โปรแกรม backend เข้าใจข้อมูลจำเพาะ เกี่ยวกับชนิดระบบไฟล์ที่สัมพันธ์และดำเนินงานในรายละเอียดของ การสื่อสารกับระบบไฟล์ โปรแกรม Back-end ที่ใช้คำสั่งระบบไฟล์รู้จัก กันในชื่อ *ตัวช่วยระบบไฟล์* และ *ตัวช่วย เชื่อมต่อ*

เมื่อต้องการกำหนดตัวช่วยระบบไฟล์ที่เหมาะสม คำสั่ง front-end ค้นหาตัวช่วยภายใต้ไฟล์ **/sbin/helpers/*vfstype*/*command*** ซึ่ง *vfstype* ตรงกับชนิดระบบไฟล์ที่พบในไฟล์ **/etc/vfs** และ *คำสั่ง* ตรงกับ ชื่อของคำสั่งที่กำลังถูกดำเนินการ แฟล็กที่ส่งไปที่คำสั่ง front-end ถูกส่งไปที่ตัวช่วยระบบไฟล์

หนึ่งตัวช่วยระบบไฟล์ที่จำเป็นซึ่งต้องถูกจัดเตรียม, ชื่อ fstype, ไม่ตรงกับชื่อคำสั่ง ตัวช่วยนี้ถูกใช้เพื่อระบุว่าวอลุ่มโลจิคัล ที่ระบุมีระบบไฟล์ของ vfstype ของตัวช่วยหรือไม่

- ตัวช่วยส่งกลับ 0 ถ้าวอลุ่มโลจิคัลไม่มีระบบไฟล์ ที่เป็นชนิดเดียวกับตัวช่วย ค่าส่งกลับ 0 บ่งชี้ว่าวอลุ่มโลจิคัลไม่มี รายการบันทึก
- ตัวช่วยส่งกลับ 1 ถ้าวอลุ่มโลจิคัลมีระบบไฟล์ที่เป็นชนิดเดียวกับตัวช่วย และระบบไฟล์ไม่ต้องการอุปกรณ์แยกสำหรับบันทึก ค่าส่งกลับ 1 บ่งชี้ว่าวอลุ่มโลจิคัลมีรายการบันทึก
- ตัวช่วยส่งกลับ 2 ถ้าวอลุ่มโลจิคัลมีระบบไฟล์ที่เป็นชนิดเดียวกับตัวช่วย และระบบไฟล์ไม่ต้องการอุปกรณ์แยกสำหรับบันทึก ถ้า มีการระบุแฟล็ก **-l** ตัวช่วย fstype ควรตรวจสอล บันทึกของชนิดระบบไฟล์ของตัวช่วย บนวอลุ่มโลจิคัลที่ระบุ

กลไกตัวช่วยระบบไฟล์แบบสัมบูรณ์

ส่วนนี้อธิบายกลไกช่วยระบบไฟล์แบบเก่าที่ถูกใช้ บนเวอร์ชันก่อนหน้าของ AIX กลไกนี้ยังคงมีอยู่ แต่ไม่ควรนำมาใช้อีกต่อไป

การดำเนินการตัวช่วยระบบไฟล์

ตารางดังต่อไปนี้แสดงการดำเนินการที่เป็นไปได้ ที่ร้องขอของตัวช่วยในไฟล์ **/usr/include/fshelp.h**:

การดำเนินการตัวช่วย	ค่า
#define FSHOP_NULL	0
#define FSHOP_CHECK	1
#define FSHOP_CHGSIZ	2
#define FSHOP_FINDATA	3
#define FSHOP_FREE	4
#define FSHOP_MAKE	5
#define FSHOP_REBUILD	6
#define FSHOP_STATFS	7
#define FSHOP_STAT	8
#define FSHOP_USAGE	9
#define FSHOP_NAMEI	10
#define FSHOP_DEBUG	11

อย่างไรก็ตาม ระบบไฟล์ JFS สนับสนุนเฉพาะการดำเนินการ ดังต่อไปนี้:

คำสั่งตรงกับค่าการดำเนินการ

```
#define FSHOP_CHECK     1    fsck

#define FSHOP_CHGSIZ    2    chfs

#define FSHOP_MAKE      5    mkfs

#define FSHOP_STATFS    7    df

#define FSHOP_NAMEI     10   ff
```

ไวยากรณ์การทำการตัวช่วยระบบไฟล์

ไวยากรณ์กระทำการของตัวช่วยระบบ ไฟล์เป็นดังนี้:

```
OpName OpKey FilsysFileDescriptor PipeFileDescriptor Modeflags
DebugLevel OpFlags
```

OpName

ระบุพารามิเตอร์ *arg0* เมื่อโปรแกรม เรียกตัวช่วย ค่าของฟิลด์ **OpName** แสดง ในรายการของกระบวนการ (โปรดดูที่คำสั่ง **ps**)

OpKey

ตรงกับการดำเนินการตัวช่วยที่มี ดังนั้นถ้าค่า **OpKey** คือ 1 การดำเนินการ **fsck** (file system check) จะถูกร้องขอ

FilsysFileDescriptor

ระบุไฟล์ descriptor ซึ่งระบบไฟล์ได้ถูกเปิด โดยโปรแกรม

PipeFileDescriptor

ระบุไฟล์ descriptor ของไพพ์ (โปรดดูที่รูทีนย่อย **pipe**) ที่ถูกเปิดระหว่างโปรแกรมดั้งเดิม และโปรแกรมตัวช่วย แชนเนลนี้อนุญาตให้ตัวช่วยสื่อสารกับ โปรแกรม front-end

ตัวอย่าง: ตัวช่วยส่ง การแจ้งข้อมูลการทำงานสำเร็จหรือล้มเหลวของตัวช่วยผ่านไพพ์ ซึ่ง สามารถมีผลการประมวลผล front-end ต่อไป นอกจากนี้ถ้าระดับการดีบักสูงพอ ตัวช่วยสามารถมีข้อมูลเพื่อส่งไปที่โปรแกรมดั้งเดิม

Modeflags

จัดเตรียมการบ่งชี้ถึงวิธีที่ตัวช่วยถูกเรียกและสามารถ มีผลกับการทำงานของตัวช่วย โดยเฉพาะที่เกี่ยวกับการรายงานความผิดพลาด แฟล็กโหมดถูกกำหนดในไฟล์ **/usr/include/fshelp.h**:

```
     Flags                              Indicator
#define FSHMOD_INTERACT_FLAG              "i"
#define FSHMOD_FORCE_FLAG                 "f"
#define FSHMOD_NONBLOCK_FLAG              "n"
#define FSHMOD_PERROR_FLAG                "p"
#define FSHMOD_ERRDUMP_FLAG               "e"
#define FSHMOD_STANDALONE_FLAG            "s"
#define FSHMOD_IGNDEVTYPE_FLAG            "I"
```

ตัวอย่าง: แฟล็ก **FSHMOD_INTERACT** บ่งชี้ ว่าคำสั่งถูกรันแบบโต้ตอบหรือไม่ (กำหนดโดยการทดสอบรูทีนย่อย **isatty** บนอินพุตมาตรฐาน) ไม่ใช่ทุก การดำเนินการที่ใช้โหมดเหล่านี้ (หรือบางส่วน) ทั้งหมด

`DebugLevel`
กำหนดจำนวนของข้อมูลการดีบักที่จำเป็น: ระดับ การดีบักยิ่งสูงแค่ไหน ข้อมูลที่ส่งกลับก็ละเอียดเพิ่มขึ้นเท่านั้น

`OpFlags`
รวมอุปกรณ์จริง ที่การดำเนินการจะ ปฏิบัติการและตัวเลือกอื่นที่ระบุโดย front end

ตัวอย่างการเรียกใช้ตัวช่วย

การร้องขอจริงของตัวช่วยระบบไฟล์ สำหรับ `fsck -fp /user` เป็นดังนี้:

```
execl("/etc/helpers/v3fshelpers","fshop_check","1","3","5","ifp",
        "0","devices=/dev/lv02,fast,preen,mounted")
```

ในตัวอย่างนี้:

- คำสั่งที่รันได้ **execd** คือ `/etc/helper/v3fshelpers`
- ชื่อแสดงในรายการของกระบวนการ (คำสั่ง **ps**) คือ `fshop_check`
- การดำเนินการที่ร้องขอคือ **FSHOP_CHECK** ดังที่แสดงโดยค่า "1"
- ระบบไฟล์ถูกเปิดบนไฟล์ descriptor "3"
- ไพพ์ที่ตัวช่วยสามารถส่งข้อมูลไปที่ โปรแกรม front-end ถูกเปิดบนไฟล์ descriptor "5"
- สตริง `ModeFlags` คือ "`-ifp`" บ่งชี้โหมด `interactive` mode plus `force` และ `perror`
- `DebugLevel` เป็น `0` ดังนั้นไม่มีเอาต์พุตดีบักพิเศษถูกส่งกลับไปที่คำสั่ง **fsck**
- สตริง `OpFlags` แจ้งแก่ โปรแกรม back-end ว่าอุปกรณ์ใดที่กำลังถูกดำเนินการอยู่ (`/dev/lv02`), ตัวเลือกที่ร้องขอ (`fast` และ `preen`) และกรุณาสังเกตว่าอุปกรณ์ถูกเชื่อมต่อ สำหรับ คำสั่ง **fsck** ไม่มีการทำการเปลี่ยนแปลงเนื่องจากคำสั่ง **fsck** ไม่ได้ทำงานบนระบบไฟล์ที่เชื่อมต่อ

อีกตัวอย่างของวิธีที่ตัวช่วยระบบไฟล์ถูกเรียก ใช้คำสั่ง **mkfs** เมื่อต้องการสร้างระบบไฟล์ JFS บน วอลุ่มโลจิคัลที่มีอยู่ชื่อ **/dev/lv02** ให้พิมพ์ คำสั่งต่อไปนี้เพื่อสร้างจุดเชื่อมต่อ:

```
mkfs /junk
```

ถ้าคุณต้องการสร้างระบบไฟล์และทราบเพียง อุปกรณ์ที่คุณต้องการเชื่อมต่อ ให้พิมพ์:

```
mkfs /dev/lv02
```

ในทั้งสองกรณี ตัวช่วยระบบไฟล์ดังต่อไปนี้ ถูกเรียก:

```
execl ("/etc/helpers/v3fshelpers", "fshop_make", "5", "3", "5", "-ip",\
  "0", "name=/junk,label=/junk,dev=/dev/lv02")
```

การดำเนินการที่ร้องขอขณะนี้คือ **FSHOP_MAKE** โหมดคือ `interactive` และ `perror` สตริง `OpFlags` รวมทั้ง จุดเชื่อมต่อ และอุปกรณ์

ตัวช่วยเชื่อมต่อ

คำสั่ง **mount** เป็นโปรแกรม front-end ที่ใช้ตัวช่วย ในการสื่อสารกับระบบไฟล์จำเพาะ โปรแกรมตัวช่วยสำหรับคำสั่ง **mount** และ **umount** (หรือ **unmount**) ถูกเรียก ว่า *ตัวช่วยเชื่อมต่อ*

เหมือนกับคำสั่งจำเพาะระบบไฟล์อื่น คำสั่ง **mount** รวบรวมพารามิเตอร์และตัวเลือกที่กำหนดให้ที่บรรทัดคำสั่ง และแปลข้อมูลนั้นภายในบริบทของข้อมูลการกำหนดค่าระบบไฟล์ ที่พบในไฟล์ **/etc/filesystems** การใช้ข้อมูลในไฟล์ **/etc/filesystems** คำสั่งร้องขอตัวช่วยเชื่อมต่อที่เหมาะสม สำหรับชนิดของระบบไฟล์ที่เกี่ยวข้อง ตัวอย่างเช่น ถ้าผู้ใช้พิมพ์ดังต่อไปนี้ คำสั่ง **mount** จะตรวจสอบไฟล์ **/etc/filesystems** สำหรับส่วนข้อมูล (stanza) ที่อธิบายระบบไฟล์ **/test**

```
mount /test
```

จากไฟล์ **/etc/filesystems**, คำสั่ง **mount** ระบุว่าระบบไฟล์ **/test** เป็นการเชื่อมต่อ NFS แบบรีโมตจากโหนดชื่อ `host1` คำสั่ง **mount** ยังบันทึกตัวเลือก ที่สัมพันธ์กับการเชื่อมต่อ

ตัวอย่างสแตนซาไฟล์ **/etc/filesystems** เป็นดังนี้:

```
/test:
        dev             = /export
        vfs             = nfs
        nodename        = host1
        options  = ro,fg,hard,intr
```

ชนิดระบบไฟล์ (`nfs` ใน ตัวอย่างของเรา) ระบุตัวช่วยเชื่อมต่อที่จะเรียก คำสั่งเปรียบเทียบ ชนิดระบบไฟล์กับฟิลด์แรกในไฟล์ **/etc/vfs** ฟิลด์ที่ตรงกันจะมี ตัวช่วยเชื่อมต่อเป็นฟิลด์ที่สาม

ไวยากรณ์การกระทำตัวช่วยเชื่อมต่อ

ต่อไปนี้เป็นตัวอย่างของไวยากรณ์กระทำการ ของตัวช่วยเชื่อมต่อ:

```
/etc/helpers/nfsmnthelp M 0 host1 /export /test ro,fg,hard,intr
```

ทั้งคำสั่ง **mount** และ **unmount** มีหกพารามิเตอร์ สี่พารามิเตอร์แรกเหมือนกัน ทั้งสองคำสั่ง:

`operation (การดำเนินการ)`
: บ่งชี้การดำเนินการที่ร้องขอของตัวช่วย ค่าเป็น M (การดำเนินการ mount) Q (การดำเนินการ query) หรือ U (การดำเนินการ unmount) การดำเนินการ เคียวรีเลิกใช้แล้ว

`debuglevel`
: กำหนดพารามิเตอร์ตัวเลขสำหรับแฟล็ก **-D** คำสั่ง **mount** และ **unmount** ไม่สนับสนุนแฟล็ก **-D** ดังนั้นค่าเป็น 0

`nodename`
: ชื่อของโหนดสำหรับการเชื่อมต่อรีโมตหรือสตริง null สำหรับการเชื่อมต่อโลคัล คำสั่ง **mount** หรือ **unmount** ไม่เรียกตัวช่วยเชื่อมต่อถ้าพารามิเตอร์ `nodename` เป็น null

`อ็อบเจ็กต์`
: ชื่ออุปกรณ์โลคัลหรือรีโมต ไดเร็กทอรี หรือไฟล์ที่ถูก เชื่อมต่อหรือยกเลิกการเชื่อมต่อ ไม่ใช่ทุกระบบไฟล์ที่สนับสนุนการรวมข้อมูลทั้งหมด ตัว อย่างเช่น ระบบไฟล์รีโมตส่วนใหญ่ไม่สนับสนุนการเชื่อมต่ออุปกรณ์ ขณะที่ระบบไฟล์โลคัลส่วนใหญ่ไม่สนับสนุนเลย

พารามิเตอร์ที่เหลือสำหรับคำสั่ง **mount** มีดังนี้:

`mount point`
ชื่อไดเร็กทอรีหรือไฟล์โลคัลซึ่งอ็อบเจ็กต์จะถูกเชื่อมต่อ

`ตัวเลือก`
แสดงรายการตัวเลือกจำเพาะระบบไฟล์ แยกโดยเครื่องหมายจุลภาค ข้อมูล สำหรับพารามิเตอร์นี้มาจากฟิลด์ **options** ของ stanza ที่เกี่ยวข้องในไฟล์ **/etc/filesystems** หรือจาก แฟล็ก **-o** *Options* บนบรรทัด คำสั่ง (**mount -o** *Options*) คำสั่ง **mount** สามารถใช้แฟล็ก **-r** (read-only) และแปลงเป็นสตริง `ro` ในฟิลด์ นี้

พารามิเตอร์ที่เหลือสำหรับคำสั่ง **unmount** มีดังนี้:

`vfsNumber`
กำหนดหมายเลขเฉพาะที่ระบุการเชื่อมต่อที่จะถูกยกเลิก หมาย เลขเฉพาะถูกส่งกลับโดยการเรียก **vmount** และสามารถเรียกคืนได้โดยการเรียกรูทีนย่อย **mntctl** หรือ **stat** สำหรับตัวช่วยเชื่อมต่อ พารามิเตอร์นี้ถูกใช้เป็น พารามิเตอร์แรกกับการเรียกรูทีนย่อย **uvmount** ที่มีการยกเลิกการเชื่อมต่อจริง

`แฟล็ก`
กำหนดค่าของพารามิเตอร์ที่สองให้กับพารามิเตอร์ **uvmount** ค่าเป็น 1 ถ้าการดำเนินการ unmount ถูกบังคับ โดยใช้แฟล็ก **-f** (**umount -f**) มิฉะนั้น ค่าจะเป็น 0 ไม่ใช่ระบบไฟล์ทั้งหมดที่สนับสนุนการบังคับ unmounts

โลจิคัลวอลุ่มโปรแกรมมิง

Logical Volume Manager (LVM) ประกอบด้วยไลบรารี ของรูทีนย่อย LVM และไดรเวอร์อุปกรณ์โลจิคัลวอลุ่ม ดังอธิบายดังนี้:

- ไลบรารีของรูทีนย่อย LVM รูทีนย่อยเหล่านี้กำหนดกลุ่มวอลุ่มและ เก็บรักษาโลจิคัลและฟิสิคัลวอลุ่มของกลุ่มวอลุ่ม
- ไดรเวอร์อุปกรณ์โลจิคัลวอลุ่ม ไดรเวอร์อุปกรณ์โลจิคัลวอลุ่มเป็นไดรเวอร์อุปกรณ์แฝง ที่ประมวลผลโลจิคัล I/O ทั้งหมดโดยเป็นเลเยอร์รหว่างระบบไฟล์และ ไดรเวอร์อุปกรณ์ดิสก์ ไดรเวอร์อุปกรณ์โลจิคัลวอลุ่มแปลง โลจิคัลแอดเดรสเป็นฟิสิคัลแอดเดรส จัดการการทำมิร์เรอร์และการย้ายตำแหน่งบล็อกที่เสีย จากนั้นส่งการร้องขอ I/O ไปยังไดรเวอร์อุปกรณ์ดิสก์ที่ระบุ อินเตอร์เฟส ไปยังไดรเวอร์อุปกรณ์โลจิคัลวอลุ่มจัดให้มีโดยรูทีนย่อย **open, close, read, write** และ **ioctl**

สำหรับรายละเอียดของพารามิเตอร์ส่วนขยาย **readx** และ **writex** และการดำเนินการ **ioctl** ที่เจาะจงสำหรับไดรเวอร์อุปกรณ์โลจิคัลวอลุ่ม โปรดดูที่ *Kernel Extensions and Device Support Programming Concepts*

สำหรับข้อมูลเพิ่มเติมเกี่ยวกับโลจิคัลวอลุ่ม โปรดดูที่ *Operating system and device management*

ไลบรารีของรูทีนย่อยโลจิคัลวอลุ่ม

รูทีนย่อย LVM กำหนดและ ดูแลรักษาโลจิคัลและฟิสิคัลวอลุ่มของกลุ่มวอลุ่ม

คำสั่งการจัดการ ระบบใช้รูทีนย่อยเหล่านี้เพื่อดำเนินงานการจัดการระบบสำหรับโลจิคัลและ ฟิสิคัลวอลุ่มของระบบ โปรแกรมมิงอินเตอร์เฟสสำหรับไลบรารี ของรูทีนย่อย LVM มีเพื่อให้ใช้เป็นทางเลือก หรือขยายฟังก์ชัน ของคำสั่งการจัดการระบบสำหรับโลจิคัลวอลุ่ม

หมายเหตุ: รูทีนย่อย LVM ใช้การเรียกใช้ระบบ **sysconfig** ซึ่งจำเป็นต้องมีสิทธิผู้ใช้ระดับ root เพื่อเคียวรีและอัพเดตโครงสร้างข้อมูลเคอร์เนล ที่ใช้อธิบายกลุ่มวอลุ่ม คุณต้องมีสิทธิผู้ใช้ระดับ root เพื่อ ใช้เซอร์วิสของไลบรารีรูทีนย่อย LVM

เซอร์วิสต่อไปนี้ที่สามารถใช้ได้:

เซอร์วิส	คำอธิบาย
lvm_querylv	เคียวรีโลจิคัลวอลุ่มและส่งกลับข้อมูลที่เกี่ยวข้องทั้งหมด
lvm_querypv	เคียวรีฟิสิคัลวอลุ่มและส่งกลับข้อมูลที่เกี่ยวข้องทั้งหมด
lvm_queryvg	เคียวรีกลุ่มวอลุ่มและส่งกลับข้อมูลที่เกี่ยวของ
lvm_queryvgs	เคียวรีกลุ่มวอลุ่มของระบบและส่งกลับข้อมูลสำหรับ กลุ่มที่เปลี่ยนแปลงออนไลน์

การดำเนินการ J2_CFG_ASSIST ioctl

การดำเนินการ **J2_CFG_ASSIST** ioctl ส่งกลับค่าสถิติ ผลการทำงานของระบบไฟล์ JFS2

การดำเนินการ **J2_CFG_ASSIST** ioctl ส่งกลับโครงสร้าง **cfg_assist** ดังที่กำหนด ในไฟล์ **/usr/include/sys/lvdd.h** โครงสร้างประกอบด้วย ฟิลด์ต่อไปนี้:

ฟิลด์	คำอธิบาย
throughput	ปริมาณงานเฉลี่ยของดิสก์ภายใต้ระบบไฟล์เป็น MB/sec สำหรับอุปกรณ์หน่วยเก็บที่สนับสนุน ปริมาณงานได้จากอุปกรณ์ มิฉะนั้นจะส่งกลับค่าปริมาณงานตอนรันไทม์ของระบบไฟล์แทน
latency	เวลาแฝงเฉลี่ยของดิสก์ทั้งหมดภายใต้ระบบไฟล์เป็น มิลลิวินาที สำหรับอุปกรณ์หน่วยเก็บที่สนับสนุน ปริมาณงานได้จากอุปกรณ์ มิฉะนั้นจะส่งกลับค่าเวลาแฝงตอนรันไทม์ของระบบไฟล์แทน
แฟล็ก	แฟล็กที่จะใช้ สำหรับรายการแฟล็กที่ใช้ได้ โปรดดูที่ไฟล์ **/usr/include/sys/lvdd.h**
vg_max_transfer	ขนาดการถ่ายโอนสูงสุดของกลุ่มวอลุ่ม (VG) เป็น KB ค่าฟิลด์ *vg_max_transfer* คือจำนวนข้อมูล สูงสุดที่สามารถถูกถ่ายโอนในหนึ่งการร้องขอ I/O ไปยังดิสก์ของ กลุ่มวอลุ่ม
write_atomicity	เขียนแบบเป็นหนึ่งเดียวเป็นจำนวนไบต์ ค่าฟิลด์ *write_atomicity* คือจำนวนไบต์มากสุดที่ไม่ถูกแบ่งย่อยเมื่อ ถูกเขียนลงบนขอบเขตที่มีการจัดตำแหน่ง

การดำเนินการ **J2_CFG_ASSIST** ioctl ส่งกลับพารามิเตอร์ ต่อไปนี้ให้แก่อุปกรณ์หน่วยเก็บที่สนับสนุนเท่านั้น มิฉะนั้นจะส่งกลับค่า null

พารามิเตอร์	คำอธิบาย
atomicWriteAlignment	การจัดตำแหน่งที่ต้องการสำหรับการเขียนแบบเป็นหนึ่งเดียวเป็นจำนวนไบต์
ideal_sequential_read_size	ขนาดการอ่านเรียงลำดับเชิงนามธรรมของดิสก์ภายใต้ ระบบไฟล์เป็น KB
ideal_sequential_write_size	ขนาดการเขียนเรียงลำดับเชิงนามธรรมของดิสก์ภายใต้ ระบบไฟล์เป็น KB
ideal_random_read_size	ขนาดการอ่านแบบสุ่มเชิงนามธรรมของดิสก์ภายใต้ ระบบไฟล์เป็น KB
ideal_random_write_size	ขนาดการเขียนแบบสุ่มเชิงนามธรรมของดิสก์ภายใต้ ระบบไฟล์เป็น KB
stripsize	ขนาด strip ของดิสก์ภายใต้ระบบไฟล์เป็น KB นี้คือจำนวนข้อมูลที่ต่อเนื่องกันบนแกนหมุนเดียวใน อาร์เรย์ RAID
stripesize	ค่าของพารามิเตอร์ *Stripesize* เป็น KB (*Stripesize* = *stripsize* × จำนวนแกนหมุนในอาร์เรย์ RAID – parity)
parallelism	จำนวนแกนหมุนที่ประกอบด้วยอุปกรณ์ RAID ที่สามารถถูกอ่านแบบ หรือเขียนพร้อมกันแบบขนาน

ค่าที่ส่งคืน

เมื่อคุณการดำเนินการนี้เสร็จสมบูรณ์ ค่า 0 จะถูกส่งกลับ ถ้าการดำเนินการ ล้มเหลว ค่า –1 จะถูกส่งกลับและตัวแปรโกลบอล **errno** ถูกตั้งค่าเป็นค่าใดค่าหนึ่งต่อไปนี้:

ค่า	คำอธิบาย
EFAULT	ระบุว่าการทำสำเนาของพารามิเตอร์ล้มเหลว
ENOMEM	ระบุว่าการจัดสรรหน่วยความจำล้มเหลว
EAGAIN	ระบุว่าสถิติรันไทม์ไม่พร้อมใช้งาน สำหรับฟิสิคัลวอลุ่มใดๆ ภายใต้ระบบไฟล์ ลองอีกครั้งหลังจาก ออก I/O เพิ่มเติมไปที่ระบบไฟล์

ข้อยกเว้น Floating-point

หัวข้อนี้ให้ข้อมูลเกี่ยวกับข้อยกเว้นการอิงดัชนี และวิธีที่โปรแกรมของคุณสามารถตรวจหา และจัดการข้อยกเว้น

Institute of Electrical and Electronics Engineers (IEEE) กำหนดมาตรฐานสำหรับข้อยกเว้น floating-point เรียกว่า IEEE Standard for Binary Floating-Point Arithmetic (IEEE 754) มาตรฐานนี้กำหนดข้อยกเว้น floating-point ห้าประเภทที่ต้องถูกส่งสัญญาณแจ้งเมื่อตรวจพบ:

- การดำเนินการไม่ถูกต้อง
- การหารด้วยศูนย์
- โอเวอร์โฟลว์
- อันเดอร์โฟลว์
- การคำนวณคาดเคลื่อน

เมื่อหนึ่งในข้อยกเว้นเหล่านี้เกิดขึ้นในกระบวนการผู้ใช้ จะมีการส่งสัญญาณโดยการตั้งค่าแฟล็กหรือใช้การดักจับ โดยค่าเริ่มต้น ระบบ ตั้งค่าแฟล็กสถานะใน Floating-Point Status and Control registers (FPSCR) บ่งชี้ว่าข้อยกเว้นเกิดขึ้น เมื่อแฟล็กสถานะถูกตั้งค่า โดยข้อยกเว้น, แฟล็กจะถูกล้างข้อมูลเฉพาะเมื่อกระบวนการล้างข้อมูลแฟล็ก หรือเมื่อกระบวนการสิ้นสุด ระบบปฏิบัติการจัดเตรียมรูทีนย่อยเพื่อเคียวรี ตั้งค่า หรือล้างแฟล็กเหล่านี้

ระบบยังสามารถทำให้ floating-point exception signal (**SIGFPE**) ถูกส่งออกมาถ้าข้อยกเว้น floating-point เกิดขึ้น เนื่องจากนี่ไม่ใช่ลักษณะการทำงานเริ่มต้น, ระบบปฏิบัติการจัดเตรียม รูทีนย่อยเพื่อเปลี่ยนภาวะของกระบวนการเพื่อให้สัญญาณถูกเปิดใช้งาน เมื่อ ข้อยกเว้น floating-point ส่งสัญญาณ **SIGFPE** กระบวนการหยุดการทำงานและสร้างไฟล์คอร์ ถ้าไม่มีรูทีนย่อย signal-handler อยู่ในกระบวนการ มิฉะนั้น กระบวนการจะเรียกรูทีนย่อย signal-handler

รูทีนย่อยข้อยกเว้น Floating-point

รูทีนย่อยข้อยกเว้น Floating-point สามารถถูกใช้เพื่อ:

- เปลี่ยนภาวะกระทำการของกระบวนการ
- เปิดใช้งานการส่งสัญญาณของข้อยกเว้น
- ปิดใช้งานข้อยกเว้นหรือล้างแฟล็ก
- ระบุข้อยกเว้นที่ทำให้เกิดสัญญาณ
- ทดสอบแฟล็ก exception sticky

รูทีนย่อยดังต่อไปนี้ถูกจัดเตรียมเพื่อบรรลุงานเหล่านี้:

รูทีนย่อย	งาน
fp_any_xcp หรือ **fp_divbyzero**	ทดสอบแฟล็ก exception sticky
fp_enable หรือ **fp_enable_all**	เปิดใช้งานการส่งสัญญาณของข้อยกเว้น
fp_inexact, fp_invalid_op, fp_iop_convert, fp_iop_infdinf, fp_iop_infmzr, fp_iop_infsinf, fp_iop_invcmp, fp_iop_snan, fp_iop_sqrt, fp_iop_vxsoft, fp_iop_zrdzr, หรือ **fp_overflow**	ทดสอบแฟล็ก exception sticky
fp_sh_info	ระบุข้อยกเว้นที่ทำให้เกิดสัญญาณ
fp_sh_set_stat	ปิดใช้งานข้อยกเว้นหรือล้างแฟล็ก
fp_trap	เปลี่ยนภาวะกระทำการของกระบวนการ
fp_underflow	ทดลองแฟล็ก exception sticky
sigaction	ติดตั้งตัวจัดการสัญญาณ

การดำเนินการตัวจัดการ Floating-point trap

เมื่อต้องการสร้างกับดัก โปรแกรมต้องเปลี่ยนภาวะกระทำการ ของกระบวนการโดยใช้รูทีนย่อย **fp_trap** และ เปิดใช้งานข้อยกเว้นที่จะถูกตรวจจับโดยใช้รูทีนย่อย **fp_enable** หรือ **fp_enable_all**

การเปลี่ยนภาวะกระทำการของโปรแกรมอาจทำให้ประสิทธิภาพ ลดลง เนื่องจากการตรวจจับ floating-point ทำให้กระบวนการทำงาน ในโหมดอนุกรม

เมื่อการตรวจจับ floating-point เกิดขึ้น สัญญาณ **SIGFPE** ถูกส่ง โดยค่าเริ่มต้น สัญญาณ **SIGFPE** ทำให้กระบวนการหยุดทำงานและสร้างไฟล์คอร์ เมื่อต้องการเปลี่ยนลักษณะการทำงาน นี้ โปรแกรมต้องสร้างตัวจัดการสัญญาณสำหรับสัญญาณนี้ โปรดดูที่รูทีนย่อย **sigaction, sigvec,** หรือ **signal** สำหรับข้อมูลเพิ่มเติมเกี่ยวกับตัวจัดการสัญญาณ

ข้อยกเว้น: การปิดใช้งานและการเปิดใช้งานการเปรียบเทียบ

อ้างอิงถึงรายการดังต่อไปนี้เพื่อแสดงความแตกต่าง ระว่างภาวะการประมวลผลปิดใช้งานและเปิดใช้งานและ รูทีนย่อยที่ถูกใช้

โมเดล Exceptions-disabled

รูทีนย่อยดังต่อไปนี้ทดสอบแฟล็กข้อยกเว้นใน ภาวะการประมวลผลปิดใช้งาน:

- **fp_any_xcp**
- **fp_clr_flag**
- **fp_divbyzero**
- **fp_inexact**
- **fp_invalid_op**
- **fp_iop_convert**
- **fp_iop_infdinf**
- **fp_iop_infmzr**
- **fp_iop_infsi**
- **fp_iop_invcmp**
- **fp_iop_snan**
- **fp_iop_sqrt**
- **fp_iop_vxsoft**

- **fp_iop_zrdzr**
- **fp_overflow**
- **fp_underflow**

โมเดล Exceptions-enabled

รูทีนย่อยดังต่อไปนี้ทำงานในภาวะ การประมวลผลเปิดใช้งาน:

รูทีนย่อย	สภาวะการประมวลผล
fp_enable หรือ **fp_enable_all**	เปิดใช้งานการส่งสัญญาณของข้อยกเว้น
fp_sh_info	ระบุข้อยกเว้นที่ทำให้เกิดสัญญาณ
fp_sh_set_stat	ปิดใช้งานข้อยกเว้นหรือล้างแฟล็ก
fp_trap	เปลี่ยนภาวะกระทำการของกระบวนการ
sigaction	ติดตั้งตัวจัดการสัญญาณ

โหมดการตรวจจับที่ไม่แม่นยำ

บางระบบมี *โหมดการตรวจจับ ที่ไม่แม่นยำ* ซึ่งหมายความว่าฮาร์ดแวร์สามารถตรวจพบข้อยกเว้น floating-point และส่งอินเตอร์รัปต์ แต่การประมวลผลอาจดำเนินต่อ ขณะที่สัญญาณ ถูกส่ง ดังนั้น instruction address register (IAR) อยู่ที่คำสั่ง ที่ต่างกันเมื่ออินเตอร์รัปต์ถูกส่ง

โหมดการตรวจจับที่ไม่แม่นยำทำให้ประสิทธิภาพลดลง น้อยกว่า *โหมดการตรวจจับที่แม่นยำ* อย่างไรก็ตาม การดำเนินการกู้คืน บางประเภทไม่สามารถทำได้ เนื่องจากการดำเนินการทำให้เกิดข้อยกเว้น ที่ไม่สามารถระบุได้หรือเนื่องจากคำสั่งที่ตามมาได้แก้ไข อาร์กิวเมนต์ที่ทำให้เกิดข้อยกเว้น

เมื่อต้องการใช้ข้อยกเว้นที่ไม่แม่นยำ ตัวจัดการสัญญาณต้อง สามารถระบุว่าการตรวจจับเป็นแบบแม่นยำหรือไม่แม่นยำ

การตรวจจับที่แม่นยำ

ในการตรวจจับที่แม่นยำ instruction address register (IAR) ชี้ไปที่คำสั่งที่ทำให้เกิดการตรวจจับ โปรแกรมสามารถปรับเปลี่ยน อาร์กิวเมนต์ของคำสั่งและทำการรีสตาร์ท หรือแก้ไขผลของการดำเนินการ และดำเนินต่อด้วยคำสั่งถัดไป เพื่อดำเนินการต่อ IAR ต้อง ถูกเพิ่มค่าเพื่อชี้ไปที่คำสั่งถัดไป

การตรวจจับที่ไม่แม่นยำ

ในการตรวจจับที่ไม่แม่นยำ IAR ชี้ไปที่คำสั่ง หลังคำสั่งที่ทำให้เกิดข้อยกเว้น คำสั่งที่ IAR ชี้ไม่ได้ถูกเริ่มทำงาน เพื่อเริ่มกระทำการต่อ ตัวจัดการสัญญาณไม่ เพิ่มค่า IAR

เพื่อขจัดความคลุมเครือ ฟิลด์ `trap_mode` ถูกจัดเตรียมในโครงสร้าง **fp_sh_info** ฟิลด์นี้ระบุโหมดการตรวจจับที่มีผล ในกระบวนการผู้ใช้ เมื่อตัวจัดการสัญญาณถูกป้อน ข้อมูลนี้ยังสามารถ ถูกกำหนดโดยการตรวจสอบ Machine Status register (MSR) ในโครงสร้าง **mstsave**

รูทีนย่อย **fp_sh_info** อนุญาตตัวจัดการสัญญาณ floating-point เพื่อกำหนดว่า ข้อยกเว้น floating-point เป็นแบบแม่นยำหรือไม่แม่นยำ

หมายเหตุ: แม้เมื่อโหมดการตรวจจับความแม่นยำถูกเปิดใช้งานบางข้อยกเว้น floating-point อาจเป็นแบบไม่แม่นยำ (เช่นการดำเนินการที่นำไปใช้ในซอฟต์แวร์) เช่นเดียวกัน ในโหมดการตรวจจับ แบบไม่แม่นยำบางข้อยกเว้นอาจแม่นยำ

เมื่อใช้ข้อยกเว้นแบบไม่แม่นยำ บางส่วนของโค้ดของคุณ อาจต้องการให้ข้อยกเว้น floating-point ทั้งหมดถูกรายงานก่อนการดำเนินการต่อ รูทีนย่อย **fp_flush_imprecise** ถูกจัดเตรียมเพื่อบรรลุเป้าหมายนี้ และขอแนะนำให้ใช้รูทีนย่อย **atexit** เพื่อรีจิสเตอร์รูทีนย่อย **fp_flush_imprecise** เพื่อรันเมื่อโปรแกรมจบการทำงาน การรันเมื่อจบการทำงาน ประกันว่าโปรแกรมไม่ได้จบการทำงานด้วยข้อยกเว้นแบบไม่แม่นยำที่ไม่มีการรายงาน

รูทีนย่อยที่ระบุฮาร์แวร์

บางระบบมีคำสั่งฮาร์ดแวร์เพื่อคำนวณ สแควร์รูทของตัวเลขอิงดัชนีและเพื่อแปลงตัวเลขอิงดัชนี ไปเป็นจำนวนเต็ม โมเดลที่ไม่มีคำสั่งฮาร์ดแวร์เหล่านี้ใช้รูทีนย่อยซอฟต์แวร์ เพื่อดำเนินการดังกล่าว ทั้งสองเมธอดสามารถทำให้เกิดการตรวจจับ ถ้าข้อยกเว้นการดำเนินการ ไม่ถูกต้องถูกเปิดใช้งาน รายงานรูทีนย่อยซอฟต์แวร์ ผ่านรูทีนย่อย **fp_sh_info** ที่ข้อยกเว้นแบบไม่แม่นยำเกิดขึ้น เนื่องจาก IAR ไม่ชี้ไปที่คำสั่งเดี่ยวที่สามารถถูกรีสตาร์ทเพื่อทำการดำเนินการ อีกครั้ง

ตัวอย่างของตัวจัดการ floating-point trap

```
/*
 * This code demonstates a working floating-point exception
 * trap handler. The handler simply identifies which
 * floating-point exceptions caused the trap and return.
 * The handler will return the default signal return
 * mechanism longjmp().
 */

#include <signal.h>
#include <setjmp.h>
#include <fpxcp.h>
#include <fptrap.h>
#include <stdlib.h>
#include <stdio.h>

#define EXIT_BAD  -1
#define EXIT_GOOD  0

/*
 * Handshaking variable with the signal handler. If zero,
 * then the signal hander returns via  the default signal
 * return mechanism; if non-zero, then the signal handler
 * returns via longjmp.
 */
static int fpsigexit;
#define SIGRETURN_EXIT 0
#define LONGJUMP_EXIT  1

static jmp_buf jump_buffer;      /* jump buffer */
#define JMP_DEFINED 0            /* setjmp rc on initial call */
#define JMP_FPE     2            /* setjmp rc on return from */
                                 /* signal handler */
```

```
/*
 * The fp_list structure allows text descriptions
 * of each possible trap type to be tied to the mask
 * that identifies it.
 */

typedef struct
  {
  fpflag_t mask;
  char     *text;
  } fp_list_t;

/* IEEE required trap types */

fp_list_t
trap_list[] =
  {
      { FP_INVALID,      "FP_INVALID"},
      { FP_OVERFLOW,     "FP_OVERFLOW"},
      { FP_UNDERFLOW,    "FP_UNDERFLOW"},
      { FP_DIV_BY_ZERO,  "FP_DIV_BY_ZERO"},
      { FP_INEXACT,      "FP_INEXACT"}
  };

/* INEXACT detail list -- this is an system extension */

fp_list_t
detail_list[] =
  {
      { FP_INV_SNAN,   "FP_INV_SNAN" } ,
      { FP_INV_ISI,    "FP_INV_ISI" } ,
      { FP_INV_IDI,    "FP_INV_IDI" } ,
      { FP_INV_ZDZ,    "FP_INV_ZDZ" } ,
      { FP_INV_IMZ,    "FP_INV_IMZ" } ,
      { FP_INV_CMP,    "FP_INV_CMP" } ,
      { FP_INV_SQRT,   "FP_INV_SQRT" } ,
      { FP_INV_CVI,    "FP_INV_CVI" } ,
      { FP_INV_VXSOFT, "FP_INV_VXSOFT" }
  };

/*
 * the TEST_IT macro is used in main() to raise
 * an exception.
 */

#define TEST_IT(WHAT, RAISE_ARG)        \
  {                                     \
  puts(strcat("testing: ", WHAT));      \
  fp_clr_flag(FP_ALL_XCP);              \
  fp_raise_xcp(RAISE_ARG);              \
  }

/*
 * NAME: my_div
 *
 * FUNCTION:  Perform floating-point division.
 *
 */
```

```
double
my_div(double x, double y)
  {
  return x / y;
  }

/*
 * NAME: sigfpe_handler
 *
 * FUNCTION: A trap handler that is entered when
 *           a floating-point exception occurs.  *           function determines what exceptions caused
 *           the trap, prints this to stdout, and returns
 *           to the process which caused the trap.
 *
 * NOTES:    This trap handler can return either via the
 *           default return mechanism or via longjmp().
 *           The global variable fpsigexit determines which.
 *
 *           When entered, all floating-point traps are
 *           disabled.
 *
 *           This sample uses printf(). This should be used
 *           with caution since printf() of a floating-
 *           point number can cause a trap, and then
 *           another printf() of a floating-point number
 *           in the signal handler will corrupt the static
 *           buffer used for the conversion.
 *
 * OUTPUTS:  The type of exception that caused
 *           the trap.
 */

static void
sigfpe_handler(int sig,
               int code,
               struct sigcontext *SCP)
  {
  struct mstsave * state = &SCP->sc_jmpbuf.jmp_context;
  fp_sh_info_t flt_context;     /* structure for fp_sh_info()
                                /* call */
  int i;                        /* loop counter */
  extern int fpsigexit;         /* global handshaking variable */
  extern jmp_buf jump_buffer    /*  */

  /*
   * Determine which floating-point exceptions caused
   *           the trap. fp_sh_info() is used to build the floating signal
   * handler info  structure, then the member
   * flt_context.trap can be examined. First the trap type is
   * compared for the IEEE required traps, and if the trap type
   * is an invalid operation, the detail bits are examined.
   */

  fp_sh_info(SCP, &flt_context, FP_SH_INFO_SIZE);
```

```
static void
sigfpe_handler(int sig,
               int code,
               struct sigcontext *SCP)
  {
  struct mstsave * state = &SCP->sc_jmpbuf.jmp_context;
  fp_sh_info_t flt_context;     /* structure for fp_sh_info()
                                /* call */
  int i;                        /* loop counter */
  extern int fpsigexit;         /* global handshaking variable */
  extern jmp_buf jump_buffer;   /*  */

  /*
   * Determine which floating-point exceptions caused
   *           the trap. fp_sh_info() is used to build the floating signal
   * handler info  structure, then the member
   * flt_context.trap can be examined. First the trap type is
   * compared for the IEEE required traps, and if the trap type
   * is an invalid operation, the detail bits are examined.
   */

  fp_sh_info(SCP, &flt_context, FP_SH_INFO_SIZE);

  for (i = 0; i < (sizeof(trap_list) / sizeof(fp_list_t)); i++)
      {
      if (flt_context.trap & trap_list[i].mask)
        (void) printf("Trap caused by %s error\n", trap_list[i].text);
      }

  if (flt_context.trap & FP_INVALID)
      {
      for (i = 0; i < (sizeof(detail_list) / sizeof(fp_list_t)); i++)
          {
          if (flt_context.trap & detail_list[i].mask)
            (void) printf("Type of invalid op is %s\n", detail_list[i].text);
          }
      }

  /* report which trap mode was in effect */

  switch (flt_context.trap_mode)
      {
    case FP_TRAP_OFF:
      puts("Trapping Mode is OFF");
      break;

    case FP_TRAP_SYNC:
      puts("Trapping Mode is SYNC");
      break;

    case FP_TRAP_IMP:
      puts("Trapping Mode is IMP");
      break;

    case FP_TRAP_IMP_REC:
      puts("Trapping Mode is IMP_REC");
      break;
```

```
    default:
      puts("ERROR:  Invalid trap mode");
      }
  if (fpsigexit == LONGJUMP_EXIT)
      {
      /*
       * Return via longjmp. In this instance there is no need to
       * clear any exceptions or disable traps to prevent
       * recurrence of the exception, because on return the
       * process will have the signal handler's floating-point
       * state.
       */
      longjmp(jump_buffer, JMP_FPE);
      }
  else
      {
      /*
       * Return via default signal handler return mechanism.
       * In this case you must take some action to prevent
       * recurrence of the trap, either by clearing the
       * exception bit in the fpscr or by disabling the trap.
       * In this case, clear the exception bit.
       * The fp_sh_set_stat routine is used to clear
       * the exception bit.
       */
      fp_sh_set_stat(SCP, (flt_context.fpscr & ((fpstat_t) ~flt_context.trap)));
      /*
       * Increment the iar of the process that caused the trap,
       * to prevent re-execution of the instruction.
       * The FP_IAR_STAT bit in flt_context.flags indicates if
       * state->iar points to an instruction that has logically
       * started. If this bit is true, state->iar points to
       * an operation that has started and will cause another
       * exception if it runs again. In this case you want to
       * continue execution and ignore the results of that
       * operation, so the iar is advanced to point to the
       * next instruction. If the bit is false, the iar already
       * points to the next instruction that must run.
       */

      if ( flt_context.flags & FP_IAR_STAT )
          {
          puts("Increment IAR");
          state->iar += 4;
          }
      }
  return;
  }

/*
 * NAME: main
```

```
 *
 * FUNCTION: Demonstrate the sigfpe_handler trap handler.
 *
 */

int
main(void)
  {
  struct sigaction response;
  struct sigaction old_response;
  extern int fpsigexit;
  extern jmp_buf jump_buffer;
  int jump_rc;
  int trap_mode;
  double arg1, arg2, r;

  /*
   * Set up for floating-point trapping. ปฏิบัติดังต่อไปนี้:
   *  1.  Clear any existing floating-point exception flags.
   *  2.  Set up a SIGFPE signal handler.
   *  3.  Place the process in synchronous execution mode.
   *  4.  Enable all floating-point traps.
   */

  fp_clr_flag(FP_ALL_XCP);
  (void) sigaction(SIGFPE, NULL, &old_response);
  (void) sigemptyset(&response.sa_mask);
  response.sa_flags = FALSE;
  response.sa_handler = (void (*)(int)) sigfpe_handler;
  (void) sigaction(SIGFPE, &response, NULL);
  fp_enable_all();

  /*
   * Demonstate trap handler return via default signal handler
   * return. The TEST_IT macro will raise the floating-point
   * exception type given in its second argument. Testing
   * is done in this case with precise trapping, because
   * it is supported on all platforms to date.
   */

  trap_mode = fp_trap(FP_TRAP_SYNC);
  if ((trap_mode == FP_TRAP_ERROR) ||
      (trap_mode == FP_TRAP_UNIMPL))
      {
      printf("ERROR: rc from fp_trap is %d\n",
             trap_mode);
      exit(-1);
      }

  (void) printf("Default signal handler return: \n");
  fpsigexit = SIGRETURN_EXIT;

  TEST_IT("div by zero", FP_DIV_BY_ZERO);
  TEST_IT("overflow",    FP_OVERFLOW);
  TEST_IT("underflow",   FP_UNDERFLOW);
  TEST_IT("inexact",     FP_INEXACT);
```

```
TEST_IT("signaling nan",      FP_INV_SNAN);
TEST_IT("INF - INF",          FP_INV_ISI);
TEST_IT("INF / INF",          FP_INV_IDI);
TEST_IT("ZERO / ZERO",        FP_INV_ZDZ);
TEST_IT("INF * ZERO",         FP_INV_IMZ);
TEST_IT("invalid compare",    FP_INV_CMP);
TEST_IT("invalid sqrt",       FP_INV_SQRT);
TEST_IT("invalid coversion",  FP_INV_CVI);
TEST_IT("software request",   FP_INV_VXSOFT);

/*
 * Next, use fp_trap() to determine what the
 * the fastest trapmode available is on
 * this platform.
 */

trap_mode = fp_trap(FP_TRAP_FASTMODE);
switch (trap_mode)
    {
  case FP_TRAP_SYNC:
    puts("Fast mode for this platform is PRECISE");
    break;

  case FP_TRAP_OFF:
    puts("This platform dosn't support trapping");
    break;

  case FP_TRAP_IMP:
    puts("Fast mode for this platform is IMPRECISE");
    break;

  case FP_TRAP_IMP_REC:
    puts("Fast mode for this platform is IMPRECISE RECOVERABLE");
    break;

  default:
    printf("Unexpected return code from fp_trap(FP_TRAP_FASTMODE): %d\n",
           trap_mode);
    exit(-2);
    }

/*
 * if this platform supports imprecise trapping, demonstate this.
 */

trap_mode = fp_trap(FP_TRAP_IMP);
if (trap_mode != FP_TRAP_UNIMPL)
    {
    puts("Demonsrate imprecise FP execeptions");
    arg1 = 1.2;
    arg2 = 0.0;
    r = my_div(arg1, arg2);
    fp_flush_imprecise();
    }

/* demonstate trap handler return via longjmp().
 */

(void) printf("longjmp return: \n");
```

```
  fpsigexit = LONGJUMP_EXIT;
  jump_rc = setjmp(jump_buffer);

  switch (jump_rc)
      {
    case JMP_DEFINED:
      (void) printf("setjmp has been set up; testing ...\n");
      TEST_IT("div by zero", FP_DIV_BY_ZERO);
      break;

    case JMP_FPE:
      (void) printf("back from signal handler\n");
      /*
       * Note that at this point the process has the floating-
       * point status inherited from the trap handler. If the
       * trap hander did not enable trapping (as the example
       * did not) then this process at this point has no traps
       * enabled.  We create a floating-point exception to
       * demonstrate that a trap does not occur, then re-enable
       * traps.
       */
      (void) printf("Creating overflow; should not trap\n");
      TEST_IT("Overflow", FP_OVERFLOW);
      fp_enable_all();
      break;

    default:
      (void) printf("unexpected rc from setjmp: %d\n", jump_rc);
      exit(EXIT_BAD);
      }
  exit(EXIT_GOOD);
  }
```

ข้อมูลที่เกี่ยวข้อง:

fp_clr_flag, fp_set_flag, fp_read_flag, fp_swag

fp_raise_xcp

sigaction, sigvec, snap Command

การจัดการกับอินพุตและเอาต์พุต

หัวข้อนี้ให้คำแนะนำเบื้องต้นสำหรับข้อควรพิจารณาการโปรแกรมมิง สำหรับรูทีนย่อยการจัดการอินพุตและเอาต์พุต และการจัดการอินพุตและเอาต์พุต (I/O)

รูทีนย่อยไลบรารี I/O สามารถส่งข้อมูลไปยัง หรือจาก อุปกรณ์ หรือไฟล์ ระบบปฏิบัติต่ออุปกรณ์เสมือนเป็นไฟล์ I/O ตัวอย่าง คุณต้องเปิดและปิดอุปกรณ์เช่นกันเหมือนกับที่กระทำกับไฟล์

บางรูทีนย่อยใช้อินพุตและเอาต์พุตมาตรฐาน เป็นช่องสัญญาณ I/O อย่างไรก็ตาม สำหรับรูทีนย่อยส่วนใหญ่ คุณสามารถ ระบุไฟล์ที่แตกต่างออกไปสำหรับต้นทางหรือปลายทางของการถ่ายโอนข้อมูล สำหรับบางรูทีนย่อย คุณสามารถใช้ตัวชี้ไฟล์ไปยังโครงสร้างที่มี ชื่อของไฟล์ สำหรับรูทีนย่อยอื่นๆ คุณสามารถใช้ file descriptor (คือ เลขจำนวนเต็มบวกที่กำหนดให้แก่ไฟล์เมื่อถูกเปิด)

รูทีนย่อย I/O ที่เบใน C Library (**libc.a**) จัดให้มี I/O สตรีม ในการเข้าถึงรูทีนย่อย I/O สตรีมเหล่านี้ คุณ ต้องรวมไฟล์ **stdio.h** โดยใช้คำสั่ง ต่อไปนี้:

```
#include <stdio.h>
```

บางส่วนของรูทีนย่อยไลบรารี I/O เป็นแมโครที่กำหนด ในไฟล์ส่วนหัว และบางส่วนเป็นอ็อบเจ็กต์โมดูลของฟังก์ชัน หลายกรณี ไลบรารีจะมีแมโครและฟังก์ชันที่ทำมีการดำเนินการประเภทเดียวกัน พิจารณาสิ่งต่อไปนี้เมื่อต้องตัดสินใจว่าจะใช้แมโคร หรือฟังก์ชัน:

- คุณไม่สามารถตั้งค่าจุดหยุดสำหรับแมโครโดยใช้โปรแกรม **dbx**
- โดยปกติแมโครจะทำงานเร็วกว่าฟังก์ชันที่เทียบเท่ากัน เนื่องจาก ตัวประมวลผลก่อนแทนที่แมโครด้วยบรรทัดของโค้ดจริงในโปรแกรม
- แมโครส่งผลมีอ็อบเจ็กต์โค้ดขนาดใหญ่หลังจากถูกคอมไพล์
- ฟังก์ชันอาจก่อให้เกิดผลข้างเคียงที่ควรหลีกเลี่ยง

ไฟล์ คำสั่ง และรูทีนย่อยที่ใช้ในการจัดการ I/O มีอินเตอร์เฟสต่อไปนี้:

`Low-level`
: อินเตอร์เฟสระดับล่างมีฟังก์ชันการปิดและปิดเบื้องต้นสำหรับ ไฟล์และอุปกรณ์

`Stream`
: อินเตอร์เฟสสตรีมจัดให้มีการอ่านและเขียน I/O สำหรับไพพ์และ FIFOs

`Terminal`
: อินเตอร์เฟสเทอร์มินัลจัดให้มีเอาต์พุตที่จัดรูปแบบและการบัฟเฟอร์

อะซิงโครนัส
: อินเตอร์เฟสอะซิงโครนัสจัดให้มี I/O และการประมวลผลที่เกิดขึ้นพร้อมกัน

ภาษาในการอินพุต
: อินเตอร์เฟสภาษาในการอินพุตคำสั่งใช้คำสั่ง **lex** และ **yacc** เพื่อสร้างโปรแกรมวิเคราะห์คำและโปรแกรมวิเคราะห์คำสำหรับการแปล I/O

อินเตอร์เฟส Low-level I/O

อินเตอร์เฟส I/O ระดับล่างเป็น entry points เข้าสู่ เคอร์เนลโดยตรง ซึ่งมีฟังก์ชันเช่น การเปิดไฟล์ การอ่านและการเขียน ไฟล์ และการปิดไฟล์

คำสั่ง **line** มีอินเตอร์เฟสที่อนุญาตอ่านอินพุตมาตรฐานทีละหนึ่งบรรทัด และรูทีนย่อยต่อไปนี้มีฟังก์ชัน I/O ระดับล่างอื่นๆ:

`open, openx หรือ creat`
: จัดเตรียมไฟล์ หรือพาธอ็อบเจ็กต์อื่น สำหรับการอ่านและเขียน โดยวิธีใช้ file descriptor ที่กำหนด

`read, readx, readv หรือ readvx`
: อ่านจาก file descriptor ที่เปิด

`write, writex, writev หรือ writevx`
: เขียนลง file descriptor ที่เปิด

`close`
ยกเลิก file descriptor

รูทีนย่อย **open** และ **creat** ตั้งค่ารายการในตารางระบบสามตาราง file descriptor ทำดัชนีตารางแรก ซึ่งทำหน้าที่เป็นพื้นที่ข้อมูลต่อหนึ่งกระบวนการที่สามารถ เข้าถึงได้โดยรูทีนย่อยการไฟล์และเขียน แต่ละรายการในตารางนี้มี ตัวชี้ไปยังรายการที่สัมพันธ์ในตารางที่สอง

ตารางที่สองคือฐานข้อมูลหนึ่งต่อระบบ หรือตารางไฟล์ ที่อนุญาตให้ไฟล์ที่เปิดอยู่แบ่งใช้กับกระบวนการหลายๆ กระบวนการ รายการ ในตารางนี้บ่งชี้ว่าไฟล์ถูกเปิดเพื่ออ่าน เขียน หรือเป็นไฟล์ และเวลาที่ไฟล์ถูกปิด รวมทั้งมีออฟเซ็ตที่บ่งชี้ ตำแหน่งที่อ่าน หรือเขียนในครั้งถัดไปที่จะเกิดขึ้นและตัวชี้สุดท้ายที่บ่งชี้ รายการไปยังตารางที่สาม ซึ่งมีสำเนาของ i-node ของไฟล์

ตารางไฟล์มีรายการสำหรับทุกอินสแตนซ์ ของรูทีนย่อย **open** หรือ **create** บนไฟล์ แต่ตาราง i-node มีเพียงหนึ่งรายการสำหรับแต่ละไฟล์

หมายเหตุ: ขณะประมวลผลรูทีนย่อย **open** or **creat** สำหรับไฟล์พิเศษ ระบบเรียกใช้รูทีนย่อยที่ **เปิด** ของอุปกรณ์เสมอเพื่ออนุญาตให้มีการประมวลพิเศษใดๆ (เช่น การกรอเทปกลับ หรือการเปิดใช้โมเด็มลีด data-terminal-ready) อย่างไรก็ตาม ระบบใช้รูทีนย่อย **close** ต่อเมื่อกระบวนการสุดท้าย ปิดไฟล์ (คือ เมื่อรายการตาราง i-node ถูกยกเลิกการจัดสรร) นี้หมายความว่าอุปกรณ์ไม่สามารถรักษา หรือขึ้นกับจำนวนนับผู้ใช้ ยกเว้นว่าอุปกรณ์ที่ใช้เป็นการเฉพาะ (ซึ่งกันมิให้อุปกรณ์ถูกเปิดอีกครั้ง ก่อนที่จะปิด) ถูกนำไปปฏิบัติใช้

เมื่อมีการดำเนินการอ่าน หรือเขียน อาร์กิวเมนต์ ของผู้ใช้และรายการตารางไฟล์ถูกใช้เพื่อตั้งค่าตัวแปรต่อไปนี้:
- แอดเดรสผู้ใช้ของพื้นที่เป้าหมายของ I/O
- จำนวนไบต์สำหรับการถ่ายโอน
- ตำแหน่งปัจจุบันในไฟล์

ถ้าไฟล์ที่อ้างอิงเป็นไฟล์พิเศษที่เป็นประเภทอักขระ รูทีนย่อยการอ่านหรือเขียนที่เหมาะสมจะถูกเรียกใช้เพื่อถ่ายโอนข้อมูล รวมถึงอัพเดตจำนวนนับและตำแหน่งปัจจุบัน มิฉะนั้น ตำแหน่งปัจจุบัน จะถูกใช้เพื่อคำนวณจำนวนบล็อกโลจิคัลในไฟล์

ถ้าไฟล์เป็นไฟล์ธรรมดา หมายเลขบล็อก โลจิคัลต้องถูกแม็พกับหมายเลขบล็อกฟิสิคัล ไฟล์พิเศษประเภทบล็อก ไม่ต้องเป็นต้องแม็พ หมายเลขบล็อกฟิสิคัลที่ได้ถูกนำไปใช้เพื่ออ่านหรือ เขียนในอุปกรณ์ที่เหมาะสม

ไดรเวอร์อุปกรณ์แบบบล็อกช่วยให้มีความสามารถในการถ่ายโอน ข้อมูลโดยตรงระหว่างอิมเมจแกนของผู้ใช้กับอุปกรณ์ที่มีขนาดเป็นบล็อก ใหญ่เท่ากับการร้องขอโดยผู้เรียกใช้โดยปราศจากการใช้บัฟเฟอร์วิธีเกี่ยวข้อง กับการตั้งค่าไฟล์พิเศษประเภทอักขระที่สอดคล้องกับอุปกรณ์ raw และ การจัดให้มีรูทีนย่อยการอ่านและเขียนเพื่อสร้างส่วนหัวบัฟเฟอร์ไพรเวตที่ไม่แบ่งใช้ กับข้อมูลที่เหมาะสม สามารถจัดให้มีรูทีนย่อยเปิดและปิดแยกต่างหาก และสามารถเรียกใช้รูทีนย่อยฟังก์ชันพิเศษสำหรับ เทปแมเหล็ก

อินเตอร์เฟส I/O สตรีม

อินเตอร์เฟส I/O สตรีมจัดให้มีข้อมูลเป็นแบบสตรีมของไบต์ ที่ไม่ถูกแปลโดยระบบ ซึ่งช่วยให้เกิดการนำไปใช้งานที่มีประสิทธิภาพมากยิ่งขึ้น สำหรับเน็ตเวิร์กโปรโตคอลมากกว่าการประมวลผล I/O อักขระ ไม่มีขอบเขตเร็กคอร์ด เมื่ออ่านและเขียนโดยใช้ I/O สตรีม ตัวอย่าง กระบวนการการอ่าน 100 ไบต์จากไพพ์ไม่สามารถกำหนดได้ว่ากระบวนการที่เขียนข้อมูลลงในไพพ์จะทำการเขียนขนาด 100 ไบต์เพียงหนึ่งครั้ง หรือทำการเขียนขนาด 50 ไบต์สองครั้ง และแม้แต่ 100 ไบต์นั้นมาจากกระบวนการต่างกันสองกระบวนการหรือไม่

I/Os สตรีมอาจเป็นไพพ์ หรือ FIFOs (ไฟล์แบบเข้าก่อน ออกก่อน) FIFOs คล้ายกับไพพ์เนื่องจากอนุญาตให้ข้อมูลเข้าออกได้ทางเดียวเท่านั้น (ซ้ายไปขวา) อย่างไรก็ตาม FIFO สามารถถูกกำหนดชื่อและเข้าถึง โดยกระบวนการที่ไม่เกี่ยวข้อง ซึ่งต่างจากไพพ์ บางครั้ง FIFOs ถูกอ้างถึงเป็น *ไพพ์ที่กำหนดชื่อ* เนื่องจากมีชื่อ FIFO จึงสามารถถูกเปิด โดยใช้รูทีนย่อย I/O **fopen** มาตรฐาน ในการเปิดไพพ์ คุณต้องเรียกใช้รูทีนย่อย **pipe** ซึ่งส่งกลับ file descriptor และรูทีนย่อย I/O **fdopen** มาตรฐานเพื่อ เชื่อมโยงกับ file descriptor ที่เปิดด้วยสตรีม I/O มาตรฐาน

หมายเหตุ: อินเตอร์เฟส I/O สตรีมทำการบัฟเฟอร์ข้อมูลในระดับผู้ใช้ และไม่สามารถ เขียนข้อมูลได้จนกว่าจะมีการดำเนินการรูทีนย่อย **fclose** หรือ **fflush** ซึ่งอาจนำไปสู่ผลลัพธ์ที่ไม่คาดคิด เมื่อถูกรวมกับ I/O ของไฟล์เช่น read() หรือ write()

อินเตอร์เฟส I/O สตรีมถูกเข้าถึงผ่านรูทีนย่อยและแมโคร ต่อไปนี้:

`fclose`
ปิดสตรีม

`feof`, `ferror`, `clearerr` หรือ `fileno`
ตรวจสอบสถานะของสตรีม

`fflush`
เขียนอักขระที่ถูกบัฟเฟอร์ไว้ทั้งหมดจากสตรีม

`fopen`, `freopen` หรือ `fdopen`
เปิดสตรีม

`fread` หรือ `fwrite`
ดำเนินการอินพุตแบบไบนารี

`fseek`, `rewind`, `ftell`, `fgetpos` หรือ `fsetpos`
เปลี่ยนตำแหน่งตัวชี้ไฟล์ของสตรีม

`getc`, `fgetc`, `getchar` หรือ `getw`
รับค่าอักขระหรือคำจากอินพุตสตรีม

`gets` หรือ `fgets`
รับสตริงจากสตรีม

`getwc`, `fgetwc` หรือ `getwchar`
รับค่า wide character จากอินพุตสตรีม

`getws` หรือ `fgetws`
รับสตริงจากสตรีม

`printf`, `fprintf`, `sprintf`, `wsprintf`, `vprintf`, `vfprintf`, `vsprintf` หรือ `vwsprintf`
พิมพ์เอาต์พุตที่จัดรูปแบบ

`putc`, `putchar`, `fputc` หรือ `putw`
เขียนอักขระหรือคำลงในกระแส

`puts` หรือ `fputs`
เขียนสตริงลงสตรีม

`putwc, putwchar` หรือ `fputwc`
เขียนอักขระหรือคำลงในสตรีม

`putws` หรือ `fputws`
เขียนสตริง wide character ลงในสตรีม

`scanf, fscanf, sscanf` หรือ `wsscanf`
แปลงอินพุตที่จัดรูปแบบ

`setbuf, setvbuf, setbuffer` หรือ `setlinebuf`
กำหนดการบัฟเฟอร์ให้แก่สตรีม

`ungetc` หรือ `ungetwc`
ดันอักขระกลับไปในอินพุตสตรีม

อินเตอร์เฟส I/O เทอร์มินัล

อินเตอร์เฟส I/O เทอร์มินัลดำเนินงานระหว่างกระบวนการ และเคอร์เนล โดยจัดให้มีฟังก์ชันเช่นการบัฟเฟอร์และเอาต์พุตที่จัดรูปแบบ ทุกๆ เทอร์มินัลและเทอร์มินัลแบบ pseudo มีโครงสร้าง tty ที่มี ID กลุ่ม ของกระบวนการปัจจุบัน ฟิลด์นี้ระบุกลุ่มกระบวนการเพื่อรับสัญญาณ ที่สัมพันธ์กับเทอร์มินัล อินเตอร์เฟส I/O เทอร์มินัลถูกเข้าถึงผ่าน คำสั่ง **iostat** ซึ่ง มอนิเตอร์การโหลดอุปกรณ์ระบบ I/O และ **uprintfd** daemon ซึ่งอนุญาตให้ข้อความเคอร์เนลถูกเขียนลง คอนโซลระบบ

คุณสมบัติเทอร์มินัลสามารถถูกเปิดใช้งานหรือปิดใช้งาน ผ่านรูทีนย่อยต่อไปนี้:

cfgetospeed, cfsetospeed, cfgetispeed หรือ cfsetispeed
รับและตั้งค่าอัตราบอดของอินพุตและเอาต์พุต

ioctl ดำเนินการฟังก์ชันการควบคุมที่สัมพันธ์กับเทอร์มินัล

termdef
เคียวรีคุณสมบัติเทอร์มินัล

tcdrain รอเอาต์พุตให้เสร็จสมบูรณ์

tcflow ดำเนินการกังฟังก์ชันการควบคุมสายงาน

tcflush ละทิ้งข้อมูลจากคิวที่ระบุ

tcgetpgrp
ขอรับ ID กลุ่มของการประมวลผลส่วนหน้า

tcsendbreak
ส่งเส้นกั้นบนสายสื่อสารอนุกรมแบบอะซิงโครนัส

tcsetattr
ตั้งค่าสถานะของเทอร์มินัล

ttylock, ttywait, ttyunlock หรือ ttylocked
ควบคุมฟังก์ชันการล็อก tty

ttyname หรือ isatty
รับชื่อของเทอร์มินัล

ttyslot ค้นหาสล็อตในไฟล์ **utmp** สำหรับผู้ใช้ ปัจจุบัน

อินเตอร์เฟส I/O อะซิงโครนัส

รูทีนย่อย I/O อะซิงโครนัสอนุญาตให้กระบวนการเริ่มทำงาน เริ่มทำงาน I/O และให้รูทีนย่อยส่งค่ากลับทันทีหลังเริ่มหรือเข้าคิว การดำเนินการ อีกรูทีนย่อยหนึ่งจำเป็นต้องใช้เพื่อรอให้การดำเนินการ เสร็จเรียบร้อย (หรือส่งค่ากลับในทันทีถ้าการดำเนินการเสร็จเรียบร้อยแล้ว) นี้หมายความว่ากระบวนการสามารถเหลื่อมกับการทำงานกับ I/O หรือเหลื่อมกับ I/O ระหว่างอุปกรณ์ที่ต่างกัน แม้ว่า I/O อะซิงโครนัสไม่ได้ปรับปรุงผลการทำงาน ให้ดีขึ้นอย่างเห็นได้ชัดสำหรับกระบวนการที่กำลังอ่านจากดิสก์ไฟล์และเขียน ลงในอีกดิสก์ไฟล์หนึ่ง แต่ I/O อะซิงโครนัสก็ช่วยปรับปรุงผลการทำงานอย่างเด่นชัด สำหรับโปรแกรมการควบคุม I/O ประเภทอื่นๆ เช่นโปรแกรมที่ ทำหน้าที่ดัมพ์ดิสก์ลงในเทปแม่เหล็ก หรือแสดงอิมเมจบนจอแสดงผลอิมเมจ

แม้ไม่จำเป็น แต่กระบวนการที่กำลังทำงาน I/O อะซิงโครนัสก็สามารถ แจ้งต่อเคอร์เนลเพื่อแจ้งว่าเมื่อใด descriptor ที่ระบุจะพร้อมใช้งานสำหรับ I/O (หรือเรียก *I/O ที่ควบคุมด้วยสัญญาณ*) เมื่อใช้ LEGACY AIO เคอร์เนลจะแจ้งกระบวนการผู้ใช้ด้วยสัญญาณ SIGIO เมื่อใช้ POSIX AIO โครงสร้าง **sigevent** ถูกใช้โดยโปรแกรมเมอร์ เพื่อพิจารณาว่าสัญญาณสำหรับเคอร์เนลใดที่จะใช้เพื่อแจ้งกระบวนการแจ้ง สัญญาณประกอบด้วย **SIGIO**, **SIGUSR1** และ **SIGUSR2**

ในการใช้ I/O อะซิงโครนัส กระบวนการต้องดำเนินการ ขั้นตอนต่อไปนี้:

1. สร้าง handler สำหรับสัญญาณ **SIGIO** ขั้นตอนนี้ จำเป็นต่อเมื่อมีการร้องขอการแจ้งเตือนโดยสัญญาณเท่านั้น
2. ตั้งค่า ID กระบวนการหรือ ID กลุ่มกระบวนการเพื่อรับสัญญาณ **SIGIO** signals ขั้นตอนนี้ จำเป็นต่อเมื่อมีการร้องขอการแจ้งเตือนโดยสัญญาณเท่านั้น
3. เปิดใช้ I/O อะซิงโครนัส โดยปกติผู้ดูแลระบบจะพิจารณาว่าจะโหลด (เปิดใช้งาน) I/O อะซิงโครนัสหรือไม่ การเปิดใช้งานเกิดขึ้นในตอนเริ่มทำงานระบบ

รูทีนย่อย I/O อะซิงโครนัสต่อไปนี้ได้รับการจัดเตรียมไว้:

`aio_cancel`
ยกเลิกการร้องขอ I/O อะซิงโครนัสที่เหลืออยู่อย่างน้อยหนึ่งการร้องขอ

`aio_error`
เรียกข้อมูลสถานะข้อผิดพลาดของการร้องขอ I/O อะซิงโครนัส

`aio_fsync`
ซิงโครไนซ์ไฟล์อะซิงโครนัส

`aio_nwait`
หยุดทำงานชั่วคราวกระบวนการการเรียกใช้จนกว่าจะมีการร้องขอ I/O อะซิงโครนัสจำนวนหนึ่ง ดำเนินการเสร็จเรียบร้อย

`aio_read`
อ่านจาก file descriptor แบบอะซิงโครนัส

`aio_return`
เรียกข้อมูลสถานะการส่งกลับของการร้องขอ I/O อะซิงโครนัส

`aio_suspend`
หยุดทำงานชั่วคราวกระบวนการการเรียกใช้จนกว่าจะมีอย่างน้อยหนึ่งการร้องขอ I/O อะซิงโครนัส ดำเนินการเสร็จเรียบร้อย

`aio_write`
เขียนลงใน file descriptor แบบอะซิงโครนัส

`lio_listio`
เริ่มต้นรายการการร้องขอ I/O อะซิงโครนัสที่มีการเรียกใช้เดียว

`poll` หรือ `select`
ตรวจสอบสถานะ I/O ของหลายๆ file descriptors และคิวข้อความ

สำหรับใช้กับรูทีนย่อย **poll** จะมีไฟล์ส่วนหัวต่อไปนี้:

`poll.h`
กำหนดโครงสร้างและแฟล็กที่ใช้โดยรูทีนย่อย **poll**

`aio.h`
กำหนดโครงสร้างและแฟล็กที่ใช้โดยรูทีนย่อย **aio_read, aio_write** และ**aio_suspend**

คีย์ป้องกันหน่วยเก็บข้อมูล

หน่วยเก็บการป้องกันคีย์จะจัดเตรียมกลไกเพื่อให้คุณปรับปรุงโปรแกรมของคุณ ให้มีความเชื่อถือ

การป้องกันคีย์จะใช้กับหน่วยความจำเพจ และทำงานในระดับเพจ ซึ่งคล้ายคลึงกับรูทีนย่อย **mprotect** ที่สามารถใช้เพื่ออ่าน-หรือเขียนเพจที่ได้รับการป้องกันไว้ตั้งแต่หนึ่งเพจขึ้นไป อย่างไรก็ตาม ด้วยหน่วยเก็บคีย์นี้ คุณสามารถทำเครื่องหมายส่วนของข้อมูลของคุณ สำหรับการป้องกันในระดับของสิทธิการอ่านและสิทธิการเขียนโดยเฉพาะ การป้องกันหน่วยเก็บคีย์ คือฟังก์ชันที่ไม่ใช่สำหรับเพจข้อมูลเท่านั้น แต่ยังรวมถึงสิทธิเข้าถึงความพยายามของเธรด คุณสามารถเปิดใช้งานโค้ดพาธที่กำหนดไว้เป็นอย่างดี เพื่อเข้าถึงข้อมูลที่ไม่พร้อมใช้งานกับโปรแกรมขนาดใหญ่ของคุณ ครอบคลุมข้อมูลโปรแกรมที่สำคัญ และป้องกันข้อมูลจากความเสียหายโดยบังเอิญ

เนื่องจากการเข้าถึงเพจที่ป้องกันคีย์เป็นแอ็ตทริบิวต์ของเธรดที่รันอยู่ กลไลกนี้จะขยายแอ็พพลิเคชันแบบมัลติเธรด แต่ด้วยข้อจำกัดที่ใช้เหล่านี้ในแบบ 1:1 (หรือขอบเขตของระบบ) pthread รูทีนย่อย **mprotect** ไม่ใช่งานที่น่าเชื่อถือในสภาวะแวดล้อมแบบมัลติเธรด เนื่องจากคุณได้ถอนการป้องกันเธรดทั้งหมดออก เมื่อคุณต้องการให้สิทธิในการเข้าถึงเธรดใดๆ คุณสามารถใช้กลไกทั้งสองแบบพร้อมกันได้ และทั้งสองนี้จะถูกบังคับอย่างเต็มรูปแบบ ดังนั้น โปรแกรมของคุณไม่สามารถเขียนลงในเพจที่ป้องกันการบันทึก แม้ว่า การป้องกันคีย์จะอนุญาตให้กระทำก็ตาม

ป้องกันคีย์ที่ตัวอย่างใช้ประกอบด้วย:

- ครอบคลุมข้อมูลส่วนบุคคลของโปรแกรมของคุณโดยสมบูรณ์ จำกัดสิทธิในการเข้าถึงพาธของโค้ดที่เลือก
- ป้องกันข้อมูลส่วนตัวของโปรแกรมของคุณจากความเสียหายที่อาจเกิดขึ้นโดยบังเอิญด้วยการรันด้วยสิทธิการเขียนที่ได้รับ โดยการรันด้วยสิทธิการเขียนที่ได้รับ แต่ให้สิทธิการเขียนโดยเฉพาะ เมื่อคุณต้องการแก้ไขข้อมูลเท่านั้น ซึ่งการทำเช่นนี้จะมีประโยชน์ เมื่อโค้ดที่อยู่ในเอ็นจินหลักอนุญาตให้เรียกโค้ดที่เชื่อถือไม่ได้
- เมื่อคีย์ส่วนตัวจำนวนมากพร้อมใช้งานการป้องกันข้อมูลหลายๆ ส่วนเพิ่มเติมสามารถกระทำได้

คุณสามารถดีบักได้ง่ายขึ้นโดยการออกแบบแอ็พพลิเคชันของคุณด้วยการป้องกันคีย์เสมอ ค่าติดตั้งการป้องกันคีย์ของเพจและค่าติดตั้งชุดคีย์ของผู้ใช้ที่แอ็คทีฟ คือการเรียกของระบบ และเป็นการดำเนินการที่สิ้นเปลือง คุณควรออกแบบโปรแกรมของคุณ เพื่อให้ความถี่ของการดำเนินการเหล่านี้ ไม่ควรมีค่าเกินไป

การป้องกันคีย์ของผู้ใช้

คำแนะนำและข้อควรพิจารณาต่อไปนี้จะนำมาใช้เมื่อใช้การป้องกันคีย์:

- เพจที่ถูกเอ็กซ์พอร์ตแบบอ่านอย่างเดียวจากเคอร์เนล ต้องดำเนินการให้มองเห็นในโปรแกรมของคุณ เพจเหล่านี้มีการป้องกันคีย์ด้วย **UKEY_SYSTEM** การป้องกันคีย์นี้ไม่ใช้การป้องกันคีย์ที่อยู่ภายใต้การควบคุมของโปรแกรม แต่จะสามารถเข้าถึงได้เสมอด้วยโปรแกรมของคุณ
- หน่วยความจำเพจของโปรแกรมของคุณทั้งหมดจะมีพับลิกคีย์ของผู้ใช้ ที่กำหนดให้กับเพจ ตามที่ได้กล่าวไว้ข้างต้น การเข้าถึงหน่วยเก็บคีย์ 0 จะได้รับสิทธิในการเข้าถึงเสมอ การทำเช่นนี้จะสร้าง *พับลิกคีย์ของผู้ใช้*
- คุณสามารถตั้งค่าการป้องกันคีย์เฉพาะสำหรับข้อมูลปกติและข้อมูลแบ่งใช้ของคุณได้ คุณจะไม่สามารถป้องกันข้อมูลไลบรารี หน่วยความจำที่แบ่งใช้ซึ่งมีขนดาต่ำด้วยเคอร์เนล หรือข้อความของโปรแกรม
- ขึ้นอยู่กับตัวเลือกฮาร์แวร์และการดูแล เฉพาะจำนวนของคีย์ส่วนตัวของผู้ใช้ (เพียงหนึ่งชุด) ที่จำกัดเท่านั้นที่พร้อมใช้งาน เมื่อโปรแกรมของคุณกำหนดคีย์ส่วนตัว ให้กับเพจตั้งแต่หนึ่งเพจขึ้นไป ข้อมูลในเพจเหล่านั้นจะไม่พร้อมใช้งานอีกต่อไป ตามค่าดีฟอลต์ คุณต้องให้สิทธิการอ่าน หรือสิทธิการเขียนอย่างชัดเจนกับข้อมูลนี้โดยล้อมรอบด้วยพาธของโค้ดที่ต้องการสิทธิ พร้อมกับการเรียกเซอร์วิสใหม่เพื่อจัดการกับชุดคีย์ผู้ใช้ที่แอ็คทีฟของคุณ
- ฟิสิคัลฮาร์ดแวร์ที่สนับสนุนการป้องกันคีย์เพิ่มเติม ซึ่งไม่พร้อมใช้งานเป็นการป้องกันคีย์ของผู้ใช้
- ไม่มี privilege พิเศษที่ต้องกำหนดการป้องกันคีย์ให้กับเพจ เฉพาะข้อกำหนดเท่านั้น ที่มีสิทธิการอ่านปัจจุบันในเพจ
- ไม่มีการควบคุม execute authority ด้วยการป้องกันคีย์

ถ้าโปรแกรมของคุณเข้าถึงคีย์ที่ป้องกันข้อมูลไว้ในการละเมิดสิทธิในการเข้าถึง ที่แสดงอยู่ในชุดคีย์ของผู้ใช้ที่แอ็คทีฟ ซึ่งรับสัญญาณ **SIGSEGV** ตามที่อยู่ในกรณีของการละเมิดสิทธิการเขียน– หรือสิทธิการอ่าน– เพจที่ป้องกันไว้ ถ้าคุณเลือกที่จะจัดการกับสัญญาณนี้ ให้ระลึกว่า handler สัญญาณจะถูกเรียกใช้โดยไม่มีสิทธิในการเข้าถึงคีย์ส่วนตัว โค้ดการจัดการสัญญาณต้องเพิ่มสิทธิเข้าถึงที่จำเป็น ให้กับชุดคีย์ของผู้ใช้ที่แอ็คทีฟก่อนที่จะอ้างอิงข้อมูลการป้องกันคีย์

Child process ที่สร้างโดยการเรียกระบบ **fork** จะสืบทอดหน่วยความจำของการประมวลผลหลัก และสถานะการรัน ซึ่งรวมถึงการป้องกันคีย์ที่เชื่อมโยงกับแต่ละเพจ พร้อมกับชุดคีย์ของผู้ใช้ที่แอ็คทีฟของเธรดหลัก ในเวลาที่ fork

ขอบเขตการป้องกันโดยคีย์ผู้ใช้

การป้องกันคีย์โดยผู้ใช้สามารถป้องกันเพจในขอบเขตต่อไปนี้:

- ขอบเขตข้อมูล
- ขอบเขตสแต็กที่เป็นค่าดีฟอลต์
- **mmap**ed regions
- หน่วยความจำที่แบ่งใช้ซึ่งพ่วงต่อกับรูทีนย่อย **shmat( )** ยกเว้น
- หมวดหมู่ของเพจเหล่านี้ไม่สามารถใช้การป้องกันคีย์ได้:
  - ไฟล์ **shmat** และหน่วยความจำที่แบ่งใช้
  - เพจขนาดใหญ่ (ไม่สามารถเพจได้)
  - ข้อความโปรแกรม
  - สิทธิการอ่านอย่างเดียวสำหรับหน่วยความจำต่ำที่แบ่งใช้ด้วยเคอร์เนล

สิ่งที่จำเป็นต้องมีในระบบสำหรับการป้องกันคีย์

หน่วยเก็บการป้องกันคีย์คือ กลไกในการป้องกันเพจที่มี privilege ที่ระบุเฉพาะสำหรับฮาร์ดแวร์ ซึ่งทำให้พร้อมใช้งานโดยเคอร์เนล AIX สำหรับใช้ในแอ็พพลิเคชันโปรแกรม หากต้องการใช้คุณลักษณะนี้ ระบบของคุณต้อง:

- เริ่มต้นบนฟิสิคัลฮาร์ดแวร์ที่มีหน่วยเก็บการป้องกันคีย์
- รันด้วยเคอร์เนลแบบ 64 บิต
- เปิดใช้งานการใช้การป้องกันคีย์ของผู้ใช้

สิ่งที่จำเป็นต้องมีในโปรแกรมสำหรับการป้องกันคีย์

หากต้องการใช้คีย์ของผู้ใช้ โปรแกรมของคุณต้อง:

- ประกาศการรับรู้คีย์ของผู้ใช้ด้วยตนเอง และพิจารณาจำนวนของการป้องกันคีย์ผู้ใช้ที่พร้อมใช้งาน ถ้ามี ด้วยรูทีนย่อย **ukey_enable**
- จัดการกับข้อมูลที่ป้องกันไว้บนเขตของเพจ
- กำหนดคีย์ส่วนตัวให้กับแต่ละเพจที่คุณต้องการป้องกันด้วยรูทีนย่อย **ukey_protect**
- ถ้าคุณป้องกันข้อมูล malloc'd ให้ระลึกไว้ว่า ต้องถอนการป้องกัน ก่อนที่คุณจะปล่อยข้อมูลนี้ให้เป็นอิสระ
- จัสเตรียมชุดของคีย์ตั้งแต่หนึ่งชุดขึ้นไปด้วยรูทีนย่อย **ukeyset_init**
- Possibly add the required keys to your keyset with the **ukeyset_add_key** subroutine, to enable future read or write accesses as required.
- สร้างชุดคีย์ที่แอ็คทีฟด้วยรูทีนย่อย **ukeyset_activate** เพื่อให้สิทธิเข้าถึงตามที่กำหนดไว้ในชุดคีย์

โปรแกรมของคุณต้องไม่:

- สอดแทรก M:N (ขอบเขตของการประมวลผล) pthread ใดๆ ไว้
- สามารถมีจุดตรวจสอบที่ดำเนินการบนโปรแกรมได้ (ตัวอย่างเช่น มี CHECKPOINT=yes ใน สภาวะแวดล้อม)

หมายเหตุ: เมื่อโปรแกรมรับรู้คีย์ของผู้ใช้ โปรแกรมจะมีบริบทเพิ่มเติม ที่เชื่อมต่อกับบริบทปัจจุบันเพื่อแสดงชุดคีย์ของผู้ใช้ที่แอ็คทีฟ ซึ่งจะสามารถมองเห็นได้ใน:

- handler สัญญาณการรับโครงสร้าง **ucontext_t** ชุดคีย์ของผู้ใช้ที่แอ็คทีฟก่อนหน้านี้จะอยู่ใน **ucontext_t.__extctx.__ukeys** อาร์เรย์ของ ints สองตัวจะมีค่าชุดคีย์ของผู้ใช้แบบ 64 บิต
- ผู้ใช้โครงสร้างบริบทจะคอมไพล์พร้อมกับ **__EXTABI__** ที่กำหนดไว้ (โดยใช้ **setcontext, getcontext, makecontext, swapcontext**)

รูทีนย่อย

รูทีนย่อยเคอร์เนล AIX ใหม่จะถูกจัดเตรียมไว้ เพื่อใช้การป้องกันคีย์:

รูทีนย่อย	คำอธิบาย
sysconf	ใช้กับ **_SC_AIX_UKEYS** เพื่อกำหนดจำนวนของคีย์ผู้ใช้ที่สนับสนุน (สามารถเรียกได้ใน AIX เวอร์ชันที่เก่ากว่า)
ukey_enable	เปิดใช้งานสภาวะแวดล้อมการโปรแกรมมิงที่รับรู้คีย์ของผู้ใช้สำหรับการประมวลผลของคุณ และรายงานจำนวนคีย์ของผู้ใช้ที่พร้อมใช้งาน
ukeyset_init	กำหนดชุดคีย์ของผู้ใช้ ซึ่งจะแสดงถึงชุดของสิทธิเข้าถึงคีย์ส่วนตัว หรือคีย์

รูทีนย่อย	คำอธิบาย
ukeyset_add_key	เพิ่มสิทธิการอ่านหรือเขียน สิทธิทั้งสองสำหรับคีย์ที่ระบุ ในชุดคีย์
ukeyset_remove_key	ลบหรือให้สิทธิการเขียน หรือสิทธิทั้งสอง สำหรับคีย์ที่ระบุจากชุดคีย์
ukeyset_add_set	เพิ่มสิทธิในการเข้าถึงทั้งหมดในหนึ่งชุดคีย์ให้กับชุดคีย์อื่น
ukeyset_remove_set	ลบสิทธิในการเข้าถึงทั้งหมดในหนึ่งชุดคีย์ออกจากชุดคีย์อื่น
ukeyset_activate	ใช้สิทธิในการเข้าถึงชุดคีย์เพื่อรันเธรด
ukeyset_ismember	ทดสอบว่า สิทธิในการเข้าถึงทั้งหมดมีอยู่ในชุดคีย์แล้วหรือไม่
ukey_setjmp	รูปแบบของ **setjmp** ที่สงวนไว้สำหรับชุดคีย์ที่แอ็คทีฟ (ใช้โครงสร้าง **ukey_jmp_buf**)
pthread_attr_getukeyset_np	ขอรับแอ็ตทริบิวต์ชุดคีย์ของ pthread
pthread_attr_setukeyset_np	ตั้งค่าแอ็ตทริบิวต์ชุดคีย์สำหรับ pthread
ukey_protect	ตั้งค่าการป้องกันคีย์ผู้ใช้สำหรับช่วงของหน่วยความจำผู้ใช้ ที่จัดเรียงเพจแล้ว
ukey_getkey	ดึงข้อมูลการป้องกันคีย์ของผู้ใช้สำหรับแอดเดรสที่ระบุ

การดีบัก

คำสั่ง **dbx** จะเพิ่มส่วนสนับสนุนที่จำกัดสำหรับการป้องกันคีย์:

- ขณะที่ดีบักโปรแกรมที่รันอยู่:
  - คำสั่งย่อย **ukeyset** จะแสดงชุดคีย์ที่แอ็คทีฟ
  - คำสั่งย่อย **ukeyvalue** จะแสดงการป้องกันคีย์ ที่เชื่อมโยงกับตำแหน่งของหน่วยความจำที่กำหนดไว้
- ขณะที่ดีบักไฟล์หลัก คำสั่งย่อย **ukeyexcept** จะรายงานชุดคีย์ที่แอ็คทีฟ effective address ของข้อยกเว้นคีย์ และหน่วยเก็บคีย์ที่เกี่ยวข้อง

รายละเอียดของฮาร์ดแวร์

ชุดคีย์ของผู้ใช้ที่แอ็คทีฟในการรันบริบทของเธรดแบบรับรู้ด้วยคีย์ จะทำงานขยายกับ authority mask register (AMR) ของฮาร์ดแวร์ที่ใช้จริงในรูปแบบที่แสดงถึงโดย **ukeyset_t** abstract data type ข้อมูลนี้จะถูกจัดเตรียมไว้สำหรับวัตถุประสงค์ในการดีบักเท่านั้น ใช้โปรแกรมมิงเซอร์วิสที่กำหนดไว้ เพื่อตั้งค่าชุดคีย์ของผู้ใช้ที่แอ็คทีฟ

- AMR คือรีจิสเตอร์แบบ 64 บิตที่ประกอบด้วยการจับคู่แบบ 32 บิต หนึ่งคู่ต่อหนึ่งคีย์ มากสุด 32 คีย์ ตัวเลขตั้งแต่ 0 ถึง 31
  - บิตแรกของแต่ละคู่แสดงถึงสิทธิการเขียนไปยังคีย์หมายเลขที่สอดคล้องกัน
  - เช่นเดียวกัน บิตที่สองของแต่ละคู่แสดงถึงสิทธิการอ่าน ไปยังคีย์หมายเลขที่สอดคล้องกัน
- ค่าบิต 0 หมายถึงการให้สิทธิเข้าถึงที่สอดคล้องกัน และค่าบิต 1 จะปฏิเสธการให้สิทธิ
- การให้สิทธิในการเข้าถึงคู่ของบิตสำหรับคีย์ 0 ไม่ได้ควบคุมโดยโปรแกรม คีย์ของผู้ใช้ที่มีค่า 0 คือ *พับลิกคีย์ของผู้ใช้* และเธรดจะมีสิทธิเต็มรูปแบบในการเข้าถึงข้อมูลในคีย์นี้เสมอ โดยไม่พิจารณาถึงค่าติดตั้งของคุณ ในชุดคีย์ของผู้ใช้ที่แอ็คทีฟ
- การจับคู่บิตอื่นๆ ทั้งหมดจะแสดงถึง *คีย์ส่วนตัวของผู้ใช้* ซึ่งเกี่ยวกับสภาพพร้อมใช้งาน คุณสามารถใช้ เพื่อป้องกันข้อมูลของคุณตามที่คุณได้เห็น

โปรแกรมตัวอย่าง

ต่อไปนี้คือตัวอย่างโปรแกรมที่รับรู้การคีย์จากผู้ใช้:

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <sys/ukeys.h>
#include <sys/syspest.h>
#include <sys/signal.h>
#include <sys/vminfo.h>

#define ROUND_UP(size,psize)    ((size)+(psize)-1 & ~((psize)-1))

/*
 * This is an example skeleton for a user key aware program.
 *
 * The private_data_1 structure will map a malloc'd key protected area
 * which the main program can access freely, while the "untrusted"
 * subroutine will only have read access.
 */
struct private_data_1 {
        int some_data;
};
struct private_data_1 *p1;      /* pointer to structure for protected data */

ukeyset_t keyset1RW;            /* keyset to give signal handler access */

/*
 * The untrusted function here should successfully read protected data.
 *
 * When the count is 0, it will just return so the caller can write
 * the incremented value back to the protected field.
 *
 * When the count is 1, it will try to update the protected field itself.
 * This should result in a SIGSEGV.
 */
int untrusted(struct private_data_1 *p1) {
        int count = p1->some_data;      /* We can read protected data */
        if (count == 1)
                p1->some_data = count;  /* But should not be able to write it */
        return count + 1;
}

/*
 * Signal handler to catch the deliberate protection violation in the
 * untrusted function above when count == 1.
 * Note that the handler is entered with NO access to our private data.
 */
void handler(int signo, siginfo_t *sip, void *ucp) {
        printf("siginfo: signo %d code %d\n", sip->si_signo, sip->si_code);
        (void)ukeyset_activate(keyset1RW, UKA_REPLACE_KEYS);
        exit(1);
}

main() {
        int nkeys;
        int pagesize = 4096;            /* hardware data page size */
```

```
int padded_protsize_1;          /* page padded size of protected data */
struct vm_page_info page_info;

ukey_t key1 = UKEY_PRIVATE1;
ukeyset_t keyset1W, oldset;
int rc;
int count = 0;

struct sigaction sa;

/*
 * Attempt to become user key aware.
 */
nkeys = ukey_enable();
if (nkeys == -1) {
        perror("ukey_enable");
        exit(1);
}
assert(nkeys >= 2);

/*
 * Determine the data region page size.
 */
page_info.addr = (long)&p1;             /* address in data region */
rc = vmgetinfo(&page_info, VM_PAGE_INFO, sizeof(struct vm_page_info));

if (rc)
        perror("vmgetinfo");
else
        pagesize = page_info.pagesize;  /* pick up actual page size */

/*
 * We need to allocate page aligned, page padded storage
 * for any area that is going to be key protected.
 */
padded_protsize_1 = ROUND_UP(sizeof(struct private_data_1), pagesize);
rc = posix_memalign((void **)&p1, pagesize, padded_protsize_1);
if (rc) {
        perror("posix_memalign");
        exit(1);
}

/*
 * Initialize the private data.
 * We can do this before protecting it if we want.
 *
 * Note that the pointer to the private data is in public storage.
 * We only protect the data itself.
 */
p1->some_data = count;

/*
 * Construct keysets to use to access the protected structure.
 * Note that these keysets will be in public storage.
 */
```

```
rc = ukeyset_init(&keyset1W, 0);
if (rc) {
        perror("ukeyset_init");
        exit(1);
}

rc = ukeyset_add_key(&keyset1W, key1, UK_WRITE);        /* WRITE */
if (rc) {
        perror("ukeyset_add_key 1W");
        exit(1);
}

keyset1RW = keyset1W;
rc = ukeyset_add_key(&keyset1RW, key1, UK_READ);        /* R/W */
if (rc) {
        perror("ukeyset_add_key 1R");
        exit(1);
}

/*
 * Restrict access to the private data by applying a private key
 * to the page(s) containing it.
 */
rc = ukey_protect(p1, padded_protsize_1, key1);
if (rc) {
        perror("ukey_protect");
        exit(1);
}

/*
 * Allow our general code to reference the private data R/W.
 */
oldset = ukeyset_activate(keyset1RW, UKA_ADD_KEYS);
if (oldset == UKSET_INVALID) {
        printf("ukeyset_activate failed\n");
        exit(1);
}

/*
 * Set up a signal handler for SIGSEGV, to catch the deliberate
 * key violation in the untrusted code.
 */
sa.sa_sigaction = handler;
SIGINITSET(sa.sa_mask);
sa.sa_flags = SA_SIGINFO;
rc = sigaction(SIGSEGV, &sa, 0);
if (rc) {
        perror("sigaction");
        exit(1);
}

/*
 * Program's main processing loop.
 */
while (count < 2) {
```

```
                /*
                 * When we need to run "untrusted" code, change access
                 * to the private data to R/O by removing write access.
                 */
                (void)ukeyset_activate(keyset1W, UKA_REMOVE_KEYS);

                /*
                 * Call untrusted subroutine here.  It can only read
                 * the protected data passed to it.
                 */
                count = untrusted(p1);

                /*
                 * Restore our full access to private data.
                 */
                (void)ukeyset_activate(keyset1W, UKA_ADD_KEYS);

                p1->some_data = count;
        }
ukey_protect(p1, padded_protsize_1, UKEY_PUBLIC);
        free(p1);
        exit(0);
}
```

การสนับสนุนโปรแกรมขนาดใหญ่

หัวข้อนี้ให้ข้อมูลเกี่ยวกับการใช้โมเดลพื้นที่แอดเดรส ขนาดใหญ่และขนาดใหญ่มากให้เหมาะสมกับโปรแกรมที่ต้องการ พื้นที่ข้อมูลที่ใหญ่กว่าถูกเตรียมให้โดยโมเดลพื้นที่แอดเดรส ดีฟอลต์

โมเดลพื้นที่แอดเดรสขนาดใหญ่พร้อมให้ใช้ได้บน AIX 4.3 และใหม่กว่า โมเดลพื้นที่แอดเดรสขนาดใหญ่มากพร้อมให้ใช้ได้บน AIX 5.1 และใหม่กว่า

หมายเหตุ: การอภิปรายนี้ใช้กับกระบวนการ 32 บิตเท่านั้น

พื้นที่แอดเดรสเสมือนของการประมวลผล 32 บิตถูกแบ่งย่อยออกเป็นพื้นที่ 16 256 เมกะไบต์ (หรือ *เซ็กเมนต์*) แต่ละส่วนถูกแอดเดรสโดยรีจิสเตอร์ ฮาร์ดแวร์แยก ระบบปฏิบัติการอ้างอิงไปยังเซ็กเมนต์ 2 (แอดเดรสเสมือน 0x20000000-0x2FFFFFFF) เป็นเซ็กเมนต์ *การประมวลผลไพรเวต* โดยดีฟอลต์ เซ็กเมนต์นี้มีสแต็กผู้ใช้และข้อมูล รวมถึงผู้ฮีป เซ็กเมนต์การประมวลผลไพรเวตยังมี u-block ของการประมวลผล ซึ่ง ใช้โดยระบบปฏิบัติการ และแอ็พพลิเคชันไม่สามารถอ่านได้

เนื่องจากเซ็กเมนต์เดียวถูกใช้สำหรับทั้งข้อมูลผู้ใช้และสแต็ก ขนาดรวม สูงสุดจะน้อยกว่า 256 MB เล็กน้อย อย่างไรก็ตาม เฉพาะบางโปรแกรมต้องการใช้ พื้นที่ข้อมูลขนาดใหญ่ (ถูกเตรียมข้อมูลเบื้องต้นหรือยกเลิกการเตรียมข้อมูลเบื้องต้น) หรือโปรแกรม จำเป็นต้องจัดสรรพื้นที่หน่วยความจำขนาดใหญ่ที่มีรูทีนย่อย **malloc** หรือ **sbrk** โปรแกรมสามารถถูกสร้างขึ้นเพื่อใช้โมเดลพื้นที่แอดเดรสขนาดใหญ่และขนาดใหญ่มาก ที่อนุญาตให้โปรแกรมใช้เก็บข้อมูลได้สูงสุด 2 GB

เป็นไปได้ที่จะใช้โมเดลพื้นที่แอดเดรสขนาดใหญ่และขนาดใหญ่มากอย่างใดอย่างหนึ่งกับ โปรแกรมที่มีอยู่ โดยการให้ค่า *maxdata ที่ไม่เป็นศูนย์* ค่า **maxdata** ได้มาจาก ตัวแปรสภาวะแวดล้อม **LDR_CNTRL** หรือจากฟิลด์ใน ไฟล์เรียกทำงาน บางโปรแกรมมีการขึ้นต่อกันบนโมเดลพื้นที่แอดเดรสดีฟอลต์ และจะหยุดถ้าทำงานโดยใช้โมเดลพื้นที่แอดเดรสขนาดใหญ่

การทำความเข้าใจโมเดลพื้นที่แอดเดรสขนาดใหญ่

โมเดลพื้นที่แอดเดรสขนาดใหญ่อนุญาตให้โปรแกรมที่ข้อมูลระบุใช้ข้อมูล มากกว่า 256 MB โปรแกรมอื่นๆ ยังคงใช้โมเดลพื้นที่แอดเดรสดีฟอลต์ ในการอนุญาตให้โปรแกรมใช้โมเดลพื้นที่แอดเดรสขนาดใหญ่ ให้ระบุค่า **maxdata** ที่ไม่เป็นศูนย์ คุณสามารถระบุค่า **maxdata** ที่ไม่เป็นศูนย์โดยใช้คำสั่ง **ld** เมื่อคุณสร้างโปรแกรม หรือโดยการเอ็กซ์พอร์ตตัวแปรสภาวะแวดล้อม **LDR_CNTRL** ก่อนทำงานโปรแกรม

เมื่อโปรแกรมใช้โมเดลพื้นที่แอดเดรสขนาดใหญ่กำลังถูกเรียกใช้งาน ระบบปฏิบัติการ จะสำรองเซ็กเมนต์ขนาด 256 MB ให้ได้ตามจำนวนที่ต้องการเพื่อใช้เก็บข้อมูล ที่ระบุโดยค่า **maxdata** ดังนั้น การเริ่มต้นที่ เซ็กเมนต์ 3 ข้อมูลที่เตรียมเบื้องต้นของโปรแกรมจะถูกอ่านจากไฟล์เรียกทำงาน ไปไว้ในหน่วยความจำ การอ่านข้อมูลเริ่มต้นในเซ็กเมนต์ 3 แม้ว่าค่า **maxdata** จะน้อยกว่า 256 MB ด้วยการใช้โมเดลพื้นที่แอดเดรสขนาดใหญ่ โปรแกรมสามารถมีสูงสุด 8 เซ็กเมนต์ หรือ 2 GB หรือข้อมูล 3.25 GB ตามลำดับ

ในโมเดลพื้นที่แอดเดรสดีฟอลต์ มี 12 เซ็กเมนต์พร้อมให้ใช้ได้โดย รูทีนย่อย **shmat** หรือ **mmap** เมื่อใช้โมเดลพื้นที่แอดเดรสขนาดใหญ่ จำนวนเซ็กเมนต์ที่ถูกสำรองไว้สำหรับข้อมูลจะลดจำนวนเซ็กเมนต์ที่พร้อมใช้ได้ สำหรับรูทีนย่อย **shmat** และ **mmap** เนื่องจากขนาดสูงสุดของข้อมูลคือ 2 GB อย่างน้อยสองเซ็กเมนต์จะพร้อมใช้ได้ เสมอสำหรับรูทีนย่อย **shmat** และ **mmap**

สแต็กผู้ใช้ยังคงอยู่ในเซ็กเมนต์ 2 เมื่อใช้งานโมเดลพื้นที่แอดเดรส ขนาดใหญ่ เป็นผลให้ ขนาดของสแต็กถูกจำกัดให้น้อยกว่า 256 MB เล็กน้อย อย่างไรก็ตาม แอ็พพลิเคชันสามารถย้ายที่สแต็กผู้ใช้ไปไว้ในเซ็กเมนต์หน่วยความจำ ที่แบ่งใช้ หรือในหน่วยความจำที่จัดสรร

ขณะที่ขนาดของข้อมูลที่เตรียมเบื้องต้นในโปรแกรมสามารถมีขนาดใหญ่ แต่ยังคงมี ข้อจำกัดในส่วนของขนาดข้อความ ในไฟล์เรียกทำงานสำหรับโปรแกรม ขนาดของส่วนข้อความบวกขนาดของส่วนโหลดเดอร์ต้องน้อยกว่า 256 MB นี้เป็นข้อบังคับเพื่อให้ส่วนเหล่านี้พอดีกับขนาดเซ็กเมนต์เดียวที่ เป็นแบบอ่านอย่างเดียว (เซ็กเมนต์ 1 เซ็กเมนต์ TEXT) คุณสามารถใช้คำสั่ง **dump** เพื่อตรวจสอบขนาดส่วน

การทำความเข้าใจโมเดลพื้นที่แอดเดรสขนาดใหญ่มาก

โมเดลพื้นที่แอดเดรสขนาดใหญ่มากช่วยให้โปรแกรมข้อมูลขนาดใหญ่ ที่ทำงานได้เหมือนกับโมเดลพื้นที่แอดเดรสขนาดใหญ่ แม้จะมีความแตกต่าง บางอย่างระหว่างโมเดลทั้งสอง ในการอนุญาตให้โปรแกรมใช้โมเดลพื้นที่แอดเดรสขนาดใหญ่มาก คุณต้องระบุค่า **maxdata** และคุณสมบัติ dynamic segment allocation (dsa) ใช้คำสั่ง **ld** หรือตัวแปรสภาวะแวดล้อม **LDR_CNTRL** เพื่อระบุค่า **maxdata** และอ็อพชัน **DSA**

ถ้าค่า **maxdata** ถูกระบุ โมเดลพื้นที่แอดเดรส ขนาดใหญ่มากจะเหมือนกับโมเดลพื้นที่แอดเดรสขนาดใหญ่ตรงที่ข้อมูลของโปรแกรมถูกอ่าน มาไว้ในหน่วยความจำเริ่มต้นที่เซ็กเมนต์ 3 และครอบครองเซ็กเมนต์ได้เท่าที่ต้องการ อย่างไรก็ตาม เซ็กเมนต์ข้อมูลที่เหลือไม่ถูกสำรองไว้สำหรับพื้นที่ข้อมูลในเวลากระทำการ แต่จะถูกจัดหามาให้แบบไดนามิก จนกว่าเซ็กเมนต์จะจำเป็นสำหรับ พื้นที่ข้อมูลของโปรแกรม โดยสามารถถูกใช้โดยรูทีนย่อย **shmat** หรือ **mmap** ด้วยการใช้โมเดลพื้นที่แอดเดรสขนาดใหญ่มาก โปรแกรม สามารถมีได้สูงสุด 13 เซ็กเมนต์ หรือข้อมูล 3.25 GB โดยใน 13 เซ็กเมนต์เหล่านี้ 12 เซ็กเมนต์หรือ 3 GB จะพร้อมใช้งานสำหรับวัตถุประสงค์ของรูทีนย่อย **shmat** และ **mmap**

เมื่อการประมวลผลพยายามที่จะขยายพื้นที่ข้อมูลออกไปยังเซ็กเมนต์ใหม่ การดำเนินการ จะทำได้สำเร็จตราบใดที่เซ็กเมนต์ไม่ได้ถูกใช้งานโดยรูทีนย่อย **shmat** หรือ **mmap** โปรแกรมสามารถเรียกใช้ รูทีนย่อย **shmdt** หรือ **munmap** เพื่อหยุดการใช้เซ็กเมนต์เพื่อที่ สามารถนำเซ็กเมนต์ไปใช้สำหรับพื้นที่ข้อมูล อย่างไรก็ตาม หลังจากเซ็กเมนต์ได้ถูกใช้ สำหรับพื้นที่ข้อมูลแล้ว จะไม่สามารถใช้เพื่อวัตถุประสงค์อื่นอีกต่อไป แม้ว่า ขนาดของพื้นที่ข้อมูลจะลดลง

ถ้าค่า **maxdata** ไม่ถูกระบุ (maxdata = 0) ด้วยคุณสมบัติ dsa property การเปลี่ยนแปลงจากลักษณะการทำงานด้านบนเล็กน้อยจะทำได้สำเร็จ การประมวลผลจะมีข้อมูลและสแต็กของตนเองในเซ็กเมนต์ 2 คล้ายกับการประมวลผล ปกติ การประมวลผลจะไม่มีการเข้าถึงไลบรารีที่แบ่งใช้แบบโกลบอล ดังนั้นไลบรารีที่แบ่งใช้ทั้งหมดที่ใช้โดยการประมวลผลจะถูกโหลดแบบไพรเวต ข้อดี ของการทำงานวิธีนี้คือการประมวลผลจะมีทั้ง 13 เซ็กเมนต์ (3.25 GB) พร้อมสำหรับใช้งานได้โดยรูทีนย่อย **shmat** และ **mmap**

ในการลบโอกาสที่รูทีนย่อย **shmat** หรือ **mmap** จะใช้เซ็กเมนต์ที่อาจถูกใช้สำหรับพื้นที่ข้อมูล ระบบปฏิบัติการจะใช้กฎอื่นที่ต่างออกไปสำหรับการเลือกแอดเดรสที่จะส่งกลับ (ถ้าแอดเดรสที่ระบุไม่ถูกร้องขอ) โดยปกติ รูทีนย่อย **shmat** หรือ **mmap** ส่งกลับแอดเดรสใน เซ็กเมนต์ที่มีอยู่น้อยที่สุด เมื่อใช้โมเดลพื้นที่แอดเดรสขนาดใหญ่มาก รูทีนย่อยเหล่านี้จะส่งกลับแอดเดรสในเซ็กเมนต์ที่มีอยู่มากที่สุด การร้องขอแอดเดรสจำเพาะจะสำเร็จ ตราบใดที่แอดเดรสไม่อยู่ใน เซ็กเมนต์ที่ได้ถูกใช้สำหรับพื้นที่ข้อมูลแล้ว ลักษณะการทำงานนี้ ถูกปฏิบัติสำหรับการประมวลผลทั้งหมดที่ระบุคุณสมบัติ dsa

ด้วยการใช้โมเดลพื้นที่แอดเดรสขนาดใหญ่มาก ค่า **maxdata** เป็นศูนย์ หรือค่า 0xD0000000 สามารถระบุได้ ถ้าค่า **maxdata** มากกว่า 0xAFFFFFFF ถูกระบุ โปรแกรมจะไม่ใช้ไลบรารีที่แบ่งใช้ที่ถูกโหลดแบบโกลบอล แต่ไลบรารีที่แบ่งใช้ จะถูกโหลดแบบไพรเวตแทน ซึ่งสามารถส่งผลต่อผลการทำงานของโปรแกรม

การเปิดใช้งานโมเดลพื้นที่แอดเดรสขนาดใหญ่และขนาดใหญ่มาก

โมเดลพื้นที่แอดเดรสขนาดใหญ่จะถูกใช้ถ้าระค่าไม่เป็นศูนย์ค่าใดๆ แก่ค่า **maxdata** และคุณสมบัติ dynamic segment allocation (dsa) ไม่ถูกระบุ โมเดลพื้นที่แอดเดรสขนาดใหญ่มากถูกใช้ ถ้าค่า **maxdata** ใดๆ ถูกกำหนด และคุณสมบัติ dsa ถูกระบุ ใช้คำสั่ง **ld** ด้วยแฟล็ก **-bmaxdata** เพื่อระบุค่า **maxdata** และตั้งค่าคุณสมบัติ dsa

ใช้คำสั่งต่อไปนี้เพื่อลิงก์โปรแกรมที่จะมีสูงสุด 8 เซ็กเมนต์ถูกสำรองไว้สำหรับข้อมูล:

```
cc -bmaxdata:0x80000000 sample.o
```

ในการลิงก์โปรแกรมที่มีโมเดลพื้นที่แอดเดรสขนาดใหญ่มากที่เปิดใช้งานบน POWER processor-based platform ใช้คำสั่งต่อไปนี้:

```
cc -bmaxdata:0xD0000000/dsa sample.o
```

ในการลิงก์โปรแกรมที่มีโมเดลพื้นที่แอดเดรสขนาดใหญ่มากเปิดใช้งาน ใช้ คำสั่งต่อไปนี้:

```
cc -bmaxdata:0xD0000000/dsa sample.o
```

คุณสามารถทำให้โปรแกรมที่มีอยู่เพื่อใช้โมเดลพื้นที่แอดเดรสขนาดใหญ่หรือ ขนาดใหญ่มากได้โดยการระบุค่า **maxdata** ด้วยตัวแปรสภาวะแวดล้อม **LDR_CNTRL** ตัวอย่าง ใช้คำสั่งต่อไปนี้ เพื่อรันโปรแกรม **a.out** ที่มี 8 เซ็กเมนต์ถูกสำรองไว้ สำหรับพื้นที่ข้อมูล:

```
LDR_CNTRL=MAXDATA=0x80000000 a.out
```

คำสั่งต่อไปนี้รันโปรแกรม **a.out** โดยใช้โมเดลพื้นที่แอดเดรสขนาดใหญ่มาก ทำให้ขนาดข้อมูลของโปรแกรมสามารถใช้ได้สูงสุด 8 เซ็กเมนต์สำหรับข้อมูล:

```
LDR_CNTRL=MAXDATA=0x80000000@DSA a.out
```

คุณยังสามารถแก้ไขโปรแกรมที่มีอยู่เพื่อที่จะใช้โมเดลพื้นที่แอดเดรส ขนาดใหญ่หรือขนาดใหญ่มาก ในการตั้งค่า **maxdata** ในโปรแกรม XCOFF 32 บิตที่มีอยู่ **a.out** เป็นค่า 0x80000000 ใช้คำสั่งต่อไปนี้:

```
/usr/ccs/bin/ldedit -bmaxdata:0x80000000 a.out
```

ถ้าโปรแกรม XCOFF 32 บิตที่มีอยู่ **a.out** ที่มี ค่า **maxdata** เป็น 0x80000000 ยังไม่มีค่า คุณสมบัติ **DSA** คุณสามารถเพิ่ม คุณสมบัติได้ด้วย คำสั่งต่อไปนี้:

```
/usr/ccs/bin/ldedit -bmaxdata:0x80000000/dsa a.out
```

คุณสามารถใช้คำสั่ง **dump** เพื่อตรวจสอบค่า **maxdata** หรือพิจารณาว่า โปรแกรมมีคุณสมบัติ dsa หรือไม่

บางโปรแกรมมีการขึ้นต่อกันบนโมเดลพื้นที่แอดเดรสดีฟอลต์โปรแกรม เหล่านี้จะจบการทำงานถ้าค่า **maxdata** ที่ไม่เป็นศูนย์ ถูก ระบุ ไม่ว่าโดยไฟล์เรียกทำงานของโปรแกรมหรือโดยการตั้งค่า ตัวแปรสภาวะแวดล้อม **LDR_CNTRL**

การเรียกทำงานโปรแกรมที่มีพื้นที่ข้อมูลขนาดใหญ่

เมื่อคุณเรียกทำงานโปรแกรมที่ใช้โมเดลพื้นที่แอดเดรสขนาดใหญ่ ระบบปฏิบัติการจะพยายามแก้ไขขีดจำกัดแบบซอร์ฟบน ขนาดข้อมูล ถ้าจำเป็น เพื่อเพิ่มให้ตรงกับค่า **maxdata** ถ้าค่า **maxdata** มีขนาด *ใหญ่กว่า* ขีดจำกัด hard ปัจจุบันบนขนาดข้อมูล ประแกรมจะไม่ทำงานถ้าตัวแปรสภาวะแวดล้อม **XPG_SUS_ENV** ถูกตั้งค่าเป็น ON หรือขีดจำกัด soft จะถูกตั้งค่าเป็น ขีด จำกัด hard ปัจจุบัน

ถ้าค่า **maxdata** *น้อย* กว่าขนาดของ ข้อมูลสแตติกของโปรแกรม โปรแกรมจะไม่ทำงาน

หลังจากวางข้อมูลที่เตรียมเบื้องต้นและที่ยกเลิกการเตรียมเบื้องต้นของโปรแกรม ไว้ในเซ็กเมนต์ 3 และถัดไปแล้ว ค่าการหยุด จะถูกคำนวณ ค่าจุดหยุดกำหนดการสิ้นสุด ของข้อมูลสแตติกของการประมวลผล และการเริ่มต้นของข้อมูลที่สามารถจัดสรร แบบไดนามิก การใช้รูทีนย่อย **malloc, brk** หรือ **sbrk** การประมวลผลสามารถย้ายค่าจุดหยุดเพื่อเพิ่มขนาดของพื้นที่ข้อมูล

ตัวอย่าง ถ้าค่า **maxdata** ถูกระบุโดย โปรแกรมเป็น 0x68000000 ดังนั้นค่าจุดหยุดสูงสุดจะอยู่กลาง เซ็กเมนต์ 9 (0x98000000) รูทีนย่อย **brk** ขยายค่าจุดหยุดเต็มขอบเขตเซ็กเมนต์ แต่ขนาดของพื้นที่ข้อมูล ไม่สามารถเกินขีดจำกัดข้อมูล soft ปัจจุบัน

รูทีนย่อย **setrlimit** อนุญาตให้การประมวลผลตั้งค่าขีดจำกัดข้อมูล soft ของตนเป็นค่าใดๆ ที่ไม่เกิน ขีดจำกัดข้อมูล hard อย่าง ไรก็ตาม ขนาดสูงสุดของพื้นที่ข้อมูลจะถูกจำกัด เป็นค่า **maxdata** เริ่มต้น และปัดเศษให้เป็นจำนวนเท่าของ ผลคูณ 256 MB

ส่วนใหญ่ของรูทีนย่อยจะไม่ได้รับผลกระทบโดยโปรแกรมข้อมูลขนาดใหญ่ รูทีนย่อย **shmat** และ **mmap** จะได้รับผลมากที่สุด เนื่องจาก มีเซ็กเมนต์ที่พร้อมใช้งานน้อยกว่า ถ้าโปรแกรมที่ใช้โมเดลพื้นที่ข้อมูลขนาดใหญ่ forks กระบวนการชายน์จะสืบทอด ขีดจำกัดรีซอร์สข้อมูลปัจจุบันมาใช้

ข้อควรพิจารณาเป็นพิเศษ

โปรแกรมที่มีพื้นที่ข้อมูลขนาดใหญ่ต้องการใช้ พื้นที่การสลับหน้าจำนวนมาก ตัวอย่าง ถ้าโปรแกรมที่มีพื้นที่แอดเดรส 2 GB พยายาม เข้าถึงทุกหน้าในพื้นที่แอดเดรส ระบบต้องมีพื้นที่การสลับหน้าจำนวน 2 GB ระบบปฏิบัติการยุติกการประมวลผล เมื่อพื้นที่การสลับหน้าเหลือน้อย โปรแกรมที่มีพื้นที่ข้อมูลขนาดใหญ่จะถูกยุติการทำงานเป็นอันดับแรกเนื่องจากโดยปกติแล้ว จะใช้พื้นที่การสลับหน้าเป็นจำนวนมาก

การดีบักโปรแกรมโดยใช้โมเดลข้อมูลขนาดใหญ่ ไม่แตกต่างจากการดีบักโปรแกรมอื่นๆ คำสั่ง **dbx** สามารถดีบักโปรแกรม ขนาดใหญ่เหล่านี้ที่แอ็คทีฟ หรือจากคอร์ดัมพ์ คอร์ดัมพ์แบบเต็มจากโปรแกรมข้อมูลขนาดใหญ่อาจมีขนาด ค่อนข้างใหญ่ เพื่อหลีกเลี่ยงไฟล์คอร์ที่จะถูกตัดให้สั้นลง ขอให้แน่ใจว่าขีดจำกัดรีซอร์ส **coredump** มีขนาดใหญ่เพียงพอ และทำให้แน่ใจว่ามี พื้นที่ว่างเพียงพอ ในระบบไฟล์ที่โปรแกรมของคุณกำลังทำงาน

บางแอ็พพลิเคชันโปรแกรมอาจถูกเขียนด้วยวิธี ที่เป็นไปตามคุณสมบัติของโมเดลพื้นที่แอดเดรสดีฟอลต์ โปรแกรมเหล่านี้อาจไม่สามารถทำงานได้ถ้าโปรแกรมทำงานโดยใช้โมเดลพื้นที่แอดเดรส ขนาดใหญ่หรือขนาดใหญ่มาก อย่าตั้งค่าตัวแปรสภาวะแวดล้อม **LDR_CNTRL** เมื่อคุณใช้โปรแกรมเหล่านี้

การประมวลผลโดยใช้โมเดลพื้นที่แอดเดรสขนาดใหญ่มากต้องมีการเปลี่ยนแปลงโค้ด ในโปรแกรมเพื่อย้ายค่าจุดหยุดของพื้นที่แอดเดรสใน chunks ให้มีขนาดใหญ่กว่า 2 GB นี้เป็นขีดจำกัดของการเรียกใช้ระบบ **sbrk** ซึ่งใช้ค่าที่มีเครื่องหมายเป็นพารามิเตอร์ สำหรับเป็นวิธีแก้ไขเฉพาะหน้า โปรแกรมสามารถเรียกใช้ **sbrk** มากกว่าหนึ่งครั้งเพื่อย้าย ค่าจุดหยุดไปยังตำแหน่งที่ต้องการ

ข้อมูลที่เกี่ยวข้อง:

brk

exec

fork

malloc

setrlimit

คำสั่ง dd

คำสั่ง ld

XCOFF Object (a.out) File Format

การเขียนโปรแกรมบนระบบแบบมัลติโพรเซสเซอร์

บนระบบยูนิโพรเซสเซอร์ เธรดจะเรียกใช้งานหลังเธรดตัวอื่นในช่วงของการแบ่งเวลา ซึ่งจะต่างกับระบบมัลติโพรเซสเซอร์โดยที่หลายๆ เธรดจะเรียกใช้งานในเวลาเดียวกัน หนึ่งเธรดต่อหนึ่งโพรเซสเซอร์ที่พร้อมใช้งาน ผลการทำงานทั้งหมดสามารถปรับปรุงได้โดยรันเธรดกระบวนการอื่นบนโพรเซสเซอร์ตัวอื่น อย่างไรก็ตาม แต่ละโปรแกรมไม่สามารถใช้ประโยชน์จากการประมวลผลจำนวนมากได้ นอกจากโปรแกรมนั้นมีหลายเธรด

การประมวลผลจำนวนมากจะไม่แสดงให้ผู้ใช้ส่วนใหญ่ได้เห็น เนื่องจากการประมวลผลจำนวนมากจะถูกจัดการโดยระบบปฏิบัติการ และโปรแกรมที่รันการประมวลผลจำนวนมากนั้น ผู้ใช้สามารถเชื่อมโยงกระบวนการใดๆ ของผู้ใช้เองได้ (บังคับให้กระบวนการรันอยู่บนโพรเซสเซอร์บางตัว) อย่างไรก็ตาม สิ่งนี้ไม่ได้บังคับ หรือแนะนำให้ทำสำหรับการใช้ปกติ สำหรับโปรแกรมเมอร์ส่วนใหญ่ การใช้ประโยชน์จากการประมวลผลจำนวนมากจะรวมการใช้เธรดหลายเธรด หรืออีกนัยหนึ่งเคอร์เนลโปรแกรมเมอร์ต้องทำงานกับปัญหาต่างๆ ขณะที่แบกรับหรือสร้างโค้ดสำหรับระบบแบบมัลติโพรเซสเซอร์

การระบุตัวประมวลผล

เครื่อง Symmetric multiprocessor (SMP) มีแผง CPU อย่างน้อยหนึ่งแผง แต่ละแผงสามารถรองรับตัวประมวลผลได้สองตัว

ตัวอย่าง เครื่องที่มีตัวประมวลผลสี่ตัวจะมีแผง CPU สองแผง แต่ละแผงมีตัวประมวลผลสองตัว คำสั่ง รูทีนย่อย หรือข้อความที่อ้างถึงตัวประมวลผลต้องใช้ identification scheme ตัวประมวลผลถูกระบุโดยหมายเลขฟิสิคัลและโลจิคัล หรือโดยชื่อตัวประมวลผล Object Data Manager (ODM) และโค้ดตำแหน่ง

ชื่อตัวประมวลผล ODM

ODM คือระบบที่ใช้ระบุส่วนต่างๆ ของทั้งเครื่อง อันได้แก่ บัสอะแด็ปเตอร์ อุปกรณ์เสริม เช่นเครื่องพิมพ์หรือเทอร์มินัล ดิสก์ แผง หน่วยความจำ และแผงตัวประมวลผล

ODM กำหนดหมายเลขให้แก่แผงตัวประมวลผลและตัวประมวลผล ตามลำดับ เริ่มตั้งแต่ 0 (ศูนย์) และเพิ่มชื่อตามจำนวน เหล่านี้โดยการเพิ่มคำนำหน้า `cpucard` หรือ `proc` ดังนั้นแผงตัวประมวลผลแรกถูกเรียกว่า `cpucard0` และตัวประมวลผลที่สองเรียกว่า `proc1`

โค้ดตำแหน่ง ODM สำหรับตัวประมวลผลประกอบด้วยฟิลด์ 2 หลัก สี่ฟิลด์ ในรูปแบบ *AA-BB-CC-DD* ดังนี้:

โค้ด	คำอธิบาย
AA	เป็น `00` เสมอ ระบุยูนิตหลัก
BB	บ่งชี้หมายเลขแผงตัวประมวลผล อาจเป็น `0P`, `0Q`, `0R` หรือ `0S` การระบุจะเรียงลำดับการ์ดตัวประมวลผลลำดับที่หนึ่ง สอง สาม และสี่
CC	เป็น `00` เสมอ
DD	บ่งชี้ตำแหน่งตัวประมวลผลบนแผงตัวประมวลผล สามารถเป็น `00` หรือ `01`

จำนวนตัวประมวลผลโลจิคัล

ตัวประมวลผลยังสามารถถูกระบุโดยใช้หมายเลขโลจิคัล ที่เริ่มที่ 0 (ศูนย์) เฉพาะตัวประมวลผลที่เปิดใช้งานเท่านั้นที่มีหมาย เลขโลจิคัล

หมายเลขตัวประมวลผลโลจิคัล 0 (ศูนย์) ระบุว่าเป็นตัวประมวลผลฟิสิคัลตัวแรกที่มีสถานะถูกเปิดใช้งาน หมายเลขตัว ประมวลผลโลจิคัล 1 (หนึ่ง) ระบุ ตัวประมวลผลฟิสิคัลที่ถูกเปิดใช้งานตัวที่สอง และอื่นๆ โดยทั่วไป คำสั่ง ระบบปฏิบัติการทั้ง หมดและไลบรารีซับรูทีนใช้หมายเลขโลจิคัลเพื่อระบุตัวประมวลผล

สถานะตัวประมวลผล ODM

ถ้าตัวประมวลผลทำงานได้อย่างถูกต้อง จะสามารถเปิดใช้งาน หรือปิดใช้งานโดยใช้คำสั่งซอฟต์แวร์ ตัวประมวลผลถูกทำ เครื่องหมาย **faulty** ถ้าตรวจพบปัญหาด้านฮาร์ดแวร์ ODM จัดหมวดหมู่ตัวประมวลผล โดยใช้สถานะต่อไปนี้:

สถานะ	คำอธิบาย
enabled	ตัวประมวลผลทำงานและสามารถใช้โดย AIX
ปิดใช้งาน	ตัวประมวลผลทำงาน แต่ไม่สามารถใช้โดย AIX
faulty	ตัวประมวลผลไม่ทำงาน (ตรวจพบ ความผิดพลาดด้านฮาร์ดแวร์)

การควบคุมการใช้โพรเซสเซอร์

ส่วนนี้อธิบายวิธีควบคมการใช้โพรเซสเซอร์ บนระบบแบบมัลติโพรเซสเซอร์

บนระบบมัลติโพรเซสเซอร์ การใช้โพรเซสเซอร์สามารถ ถูกควบคุมได้ในวิธีดังต่อไปนี้:

- ผู้ใช้สามารถบังคับให้กระบวนการหรือเคอร์เนลเธรดรันบนโพรเซสเซอร์ที่กำหนด

การเชื่อมโยงกระบวนการและเคอร์เนลเธรด

ผู้ใช้อาจบังคับให้กระบวนการ รันบนโพรเซสเซอร์ที่กำหนด การดำเนินการนี้เรียกว่า *การเชื่อมโยง* ผู้ดูแลระบบอาจเชื่อมต่อ กระบวนการที่ต้องการ จากบรรทัดคำสั่ง การเชื่อมต่อถูกควบคุมด้วยคำสั่ง **bindprocessor**

ตัวกระบวนการเองไม่ได้ถูกเชื่อมโยง แต่เป็นเคอร์เนลเธรดของ กระบวนการที่ถูกเชื่อมโยง หลังจากเคอร์เนลเธรดถูกเชื่อมโยง เคอร์เนลเธรดถูก กำหนดตารางเวลาเสมอ ให้รันบนโพรเซสเซอร์ที่เลือก นอกจากถูกยกเลิกการเชื่อมโยงภายหลัง เมื่อเคอร์เนลเธรด ถูกสร้าง จะมีคุณสมบัติการเชื่อมโยงเหมือนกับต้นกำเนิด

สถานการณ์นี้ใช้กับ initial thread ในกระบวนการใหม่ที่สร้าง โดยรูทีนย่อย **fork** เธรดใหม่สืบทอดคุณสมบัติการเชื่อมโยงของเธรดที่เรียกรูทีนย่อย **fork** เมื่อรูทีนย่อย **exec** ถูกเรียกคุณสมบัติการเชื่อมโยงจะไม่เปลี่ยนแปลง หลังจาก กระบวนการถูกเชื่อมโยงกับโพรเซสเซอร์ ถ้าไม่มีการเชื่อมโยงอื่นหรือการยกเลิก การเชื่อมโยงเกิดขึ้น กระบวนการ child ทั้งหมดจะถูกเชื่อมต่อกับโพรเซสเซอร์เดิม

เป็นไปได้ที่จะเชื่อมโยงกระบวนการกับโพรเซสเซอร์ที่เปิดใช้งาน โดยใช้หมายเลขโพรเซสเซอร์โลจิคัลเท่านั้น เมื่อต้องการแสดงรายการหมายเลขโพรเซสเซอร์โลจิคัล ที่มีให้ใช้คำสั่ง **bindprocessor -q** สำหรับระบบที่มี สี่โพรเซสเซอร์ที่เปิดใช้งาน คำสั่งนี้สร้างเอาต์พุตเหมือนดังนี้:

```
The available processors are: 0 1 2 3
```

การเชื่อมโยงยังถูกควบคุมได้ภายในโปรแกรมโดยใช้ รูทีนย่อย **bindprocessor** ซึ่งอนุญาตให้โปรแกรมเมอร์เชื่อมโยงเคอร์เนลเธรดเดี่ยวหรือเคอร์เนลเธรดทั้งหมด ในกระบวนการ โปรแกรมเมอร์ยังสามารถยกเลิกการเชื่อมโยงเคอร์เนลเธรดเดี่ยว หรือเคอร์เนลเธรดทั้งหมดในกระบวนการ

การใช้ Dynamic Processor Deallocation

การสตาร์ทด้วยเครื่องชนิด 7044 รุ่น 270 ฮาร์ดแวร์ของระบบทั้งหมด ที่มีสองตัวประมวลผลสามารถตรวจพบข้อผิดพลาดที่สามารถแก้ไขได้ ซึ่งจะถูกแก้ไขโดยเฟิร์มแวร์ ข้อผิดพลาดเหล่านี้ไม่ใช่ข้อผิดพลาดที่รุนแรง แต่ข้อผิดพลาดนี้ยังคงเกิดขึ้นได้เสมอ ซึ่งสามารถละเว้นได้อย่างปลอดภัย อย่างไรก็ตาม เมื่อรูปแบบของความล้มเหลวถูกพัฒนาขึ้น สำหรับตัวประมวลผลที่ระบุเฉพาะ รูปแบบนี้อาจบ่งชี้ว่า คอมโพเนนต์นี้จะเป็นข้อห้ามสำหรับความล้มเหลวที่ไม่สามารถกู้คืนได้ในอนาคตอันใกล้ การคาดการณ์นี้จะถูกทำโดยเฟิร์มแวร์ที่อ้างอิงถึงอัตราความล้มเหลว และการวิเคราะห์ threshold

AIX นำการตรวจตราฮาร์ดแวร์แบบต่อเนื่องไปใช้ และสำรวจความคิดเห็นเฟิร์มแวร์สำหรับข้อผิดพลาดด้านฮาร์ดแวร์ เมื่อจำนวนของข้อผิดพลาดของตัวประมวลผล ตรงกับ threshold และเฟิร์มแวร์จะจดจำความน่าจะเป็นที่คอมโพเนนต์ของระบบนี้จะล้มเหลว เฟิร์มแวร์จะส่งคืนรายงานข้อผิดพลาดไปยัง AIX และบันทึกข้อผิดพลาดในบันทึกข้อผิดพลาดของระบบ นอกจากนี้ สำหรับระบบแบบมัลติโพรเซสเซอร์ขึ้นอยู่กับชนิดของความล้มเหลว AIX พยายามหยุดการใช้ตัวประมวลผลที่ไม่น่าไว้ใจ และจัดสรรคืน คุณลักษณะนี้เรียกว่า *การจัดสรรคืนตัวประมวลผลแบบไดนามิก*

ณ จุดนี้ เฟิร์มแวร์จะแฟล็กตัวประมวลผลสำหรับการจัดสรรคืนที่ยังคงอยู่ สำหรับการรีบูตตามลำดับ จนกว่าเซอร์วิสส่วนบุคคลจะแทนที่ตัวประมวลผล

ผลกระทบที่อาจเกิดกับแอ็พพลิเคชัน

การจัดสรรคืนตัวประมวลผลไม่ปรากฏให้เห็นชัดสำหรับแอ็พพลิเคชันหลักที่กว้างขวาง ซึ่งรวมถึงไดรเวอร์และส่วนขยายเคอร์เนล อย่างไรก็ตาม คุณสามารถใช้ AIX ที่เผยแพร่อินเตอร์เฟสเพื่อกำหนดแอ็พพลิเคชัน หรือส่วนขยายเคอร์เนลที่รันอยู่บนเครื่องแบบมัลติโพรเซสเซอร์ ค้นหาจำนวนของตัวประมวลผลที่มีอยู่ และเชื่อมโยงเธรดกับตัวประมวลผลที่ระบุเฉพาะ

อินเตอร์เฟส bindprocessor สำหรับการเชื่อมโยงการประมวลผลหรือเธรดกับตัวประมวลผล จะใช้การเชื่อมโยงจำนวนของ CPU การเชื่อมโยงจำนวนของ CPU จะอยู่ในช่วง [0..*N*–1] โดยที่ *N* คือจำนวนทั้งหมดของ CPU หากต้องการหลีกเลี่ยงการพักแอ็พพลิเคชัน หรือส่วนขยายเคอร์เนลที่ไม่มี "ช่องโหว่" ในการกำหนดหมายเลข CPU AIX จะทำให้ปรากฏขึ้นสำหรับแอ็พพลิเคชัน หาก CPU คือตัว "ล่าสุด" (หมายเลขสูงสุด) ที่เชื่อมโยง CPU ที่ต้องการจัดสรรคืน สำหรับกรณีนี้ SMP แบบ 8 ทิศทาง

หมายเลขการเชื่อมโยง CPU คือ [0..7] ถ้าจัดสรรคืนหนึ่งตัวประมวลผลแล้ว จำนวนทั้งหมดที่ CPU พร้อมใช้งานกลายเป็น 7 และจะมีหมายเลข [0..6] CPU 7 จะไม่ปรากฏขึ้น โดยไม่พิจารณาถึงความล้มเหลวของตัวประมวลผลแบบฟิสิคัล

หมายเหตุ: สำหรับส่วนที่เหลือของคำอธิบายนี้ คำว่า *CPU* จะถูกใช้สำหรับ entity แบบโลจิคัล และคำศัพท์ *ตัวประมวลผล* จะใช้สำหรับ entity แบบฟิสิคัล

แอ็พพลิเคชันหรือส่วนขยายเคอร์เนลที่ใช้การเชื่อมโยงการประมวลผล/เธรดสามารถขาดได้ หาก AIX ยกเลิกเธรดที่เชื่อมโยงไว้ หรือย้ายเธรดเหล่านั้นไปยัง CPU อื่น เมื่อหนึ่งในตัวประมวลผลจำเป็นต้องถูกจัดสรรคืน การจัดสรรคืนตัวประมวลผลแบบไดนามิกจะจัดเตรียมโปรแกรมมิงอินเตอร์เฟสไว้ ดังนั้น แอ็พพลิเคชันเหล่านั้นและส่วนขยายเคอร์เนลสามารถได้รับการแจ้งว่า การจัดสรรคืนตัวประมวลผลได้เกิดขึ้นแล้ว เมื่อแอ็พพลิเคชันเหล่านี้และส่วนขยายเคอร์เนลได้รับการแจ้งเตือน แอ็พพลิเคชันและส่วนขยายเคอร์เนลเหล่านี้จะรับผิดชอบต่อการย้ายเธรด และรีซอร์สที่เชื่อมโยงไว้ (เช่น ตัวจับเวลาคำร้องขอบล็อก) ออกจาก CPU ID ที่โยงไว้ล่าสุด และปรับเธรดและรีซอร์สเหล่านั้นให้เป็นคอนฟิกูเรชัน CPU ใหม่

หลังการแจ้งเตือนแอ็พพลิเคชันและส่วนขยายเคอร์เนลแล้ว หากเธรดบางส่วนยังคงโยงกับ CPU ID ล่าสุดที่โยงไว้ การจัดสรรคืนจะถูกยกเลิก ในกรณีนี้ AIX จะบันทึกการทำงานสำหรับการจัดสรรคืนที่ได้ถูกยกเลิก ลงในบันทึกข้อผิดพลาด และยังคงใช้ตัวประมวลผลที่มีข้อบกพร่อง ในที่สุด เมื่อตัวประมวลผลเกิดความล้มเหลว ตัวประมวลผลจะสร้างความล้มเหลวของระบบทั้งหมด ดังนั้น จึงมีความสำคัญสำหรับแอ็พพลิเคชันหรือส่วนขยายเคอร์เนลที่กำลังโยงเธรดกับ CPU เพื่อขอรับการแจ้งเตือนของการจัดสรรคืนตัวประมวลผลที่ใกล้เข้ามา และทำหน้าที่เกี่ยวกับการแจ้งนี้

แม้ในกรณีที่ยากที่จะเกิดขึ้น การจัดสรรคืนไม่สามารถเกิดขึ้นได้ การจัดสรรตัวประมวลผลแบบไดนามิกยังคงกำหนดการเตือนในระดับสูงให้กับผู้ดูแลระบบ ด้วยการบันทึกข้อผิดพลาดในบันทึกข้อผิดพลาด บันทึกข้อผิดพลาดนี้จะกำหนดโอกาสให้กับผู้ดูแลระบบ เพื่อจัดกำหนดตารางเวลาสำหรับการดำเนินการดูแลรักษาระบบ เพื่อแทนที่คอมโพเนนต์ที่มีข้อบกพร่อง ก่อนที่ความล้มเหลวระบบโกลบอลจะเกิดขึ้น

โฟลว์ของเหตุการณ์สำหรับการยกเลิกการจัดสรรตัวประมวลผล

การไหลของเหตุการณ์สำหรับการจัดสรรคืนตัวประมวลผลมีดังต่อไปนี้:

1. เฟิร์มแวร์ตรวจพบ threshold ข้อผิดพลาดที่สามารถกู้คืนได้ ซึ่งจะเข้าถึงด้วยหนึ่งในตัวประมวลผล
2. AIX จะบันทึกรายงานข้อผิดพลาดของเฟิร์มแวร์ในบันทึกข้อผิดพลาดของระบบ และเมื่อเรียกใช้งานเครื่องที่สนับสนุนการจัดสรรคืนของตัวประมวลผลที่สนับสนุน ให้เริ่มต้นกระบวนการจัดสรรคืน
3. AIX แจ้งเตือนการประมวลผลที่ไม่ใช่เคอร์เนลและเธรดที่โยงไว้กับ CPU ที่โยงไว้ล่าสุด
4. AIX จะรอจนกว่าเธรดที่มีข้อจำกัดทั้งหมดย้ายออกจาก CPU ที่โยงไว้ล่าสุด ถ้าเธรดยังคงมีข้อจำกัด AIX จะหมดเวลาใช้งาน (หลังจากสิบนาทีผ่านไป) และยกเลิกการจัดสรรคืน มิฉะนั้น AIX จะเรียกใช้งาน High Availability Event Handlers (HAEHs) ที่ลงทะเบียนไว้ก่อนหน้านี้ HAEH อาจส่งคืนข้อผิดพลาดที่จะยกเลิกการจัดสรรคืน มิฉะนั้น AIX จะดำเนินการต่อด้วยกระบวนการจัดสรรคืนและหยุดตัวประมวลผลที่กำลังล้มเหลว ในที่สุด

ในกรณีที่เกิดความล้มเหลวที่จุดใดๆ ของการจัดสรรคืน AIX จะบันทึกความล้มเหลวนั้น โดยบ่งชี้ถึงสาเหตุของการยกเลิกการจัดสรรคืน ผู้ดูแลระบบสามารถมองหาบันทึกข้อผิดพลาด ดำเนินการแก้ไข (เมื่อสามารถทำได้) และรีสตาร์ทการจัดสรรคืน ตัวอย่างเช่น ถ้าการจัดสรรคืนถูกยกเลิก เนื่องจากมีแอ็พพลิเคชันอย่างน้อยหนึ่งแอ็พพลิเคชันที่ไม่ได้โยงไว้กับเธรดที่มีข้อจำกัด ผู้ดูแลระบบสามารถหยุดแอ็พพลิเคชัน รีสตาร์ทการจัดสรรคืน (ซึ่งควรดำเนินการในเวลานี้) และรีสตาร์ทแอ็พพลิเคชัน

อินเตอร์เฟสการเขียนโปรแกรมที่จัดการกับตัวประมวลผลแต่ละตัว

ส่วนต่อไปนี้อธิบายถึงโปรแกรมมิงอินเตอร์เฟสที่พร้อมใช้งาน:

อินเตอร์เฟสเพื่อกำหนดจำนวนของ CPU บนระบบ

รูทีนย่อย sysconf

รูทีนย่อย **sysconf** ส่งคืนจำนวนของตัวประมวลผล โดยใช้พารามิเตอร์ต่อไปนี้:

- **_SC_NPROCESSORS_CONF**: จำนวนของตัวประมวลผลที่ปรับแต่ง
- **_SC_NPROCESSORS_ONLN**: จำนวนของตัวประมวลผลแบบออนไลน์

สำหรับข้อมูลเพิ่มเติม โปรดดู รูทีนย่อย **sysconf** ใน *Technical Reference: Base Operating System and Extensions, Volume 2*

ค่าที่ส่งคืนโดยรูทีนย่อย **sysconf** สำหรับ _SC_NPROCESSORS_CONF จะยังคงเป็นค่าคงที่ ในระหว่างที่รีบูต เครื่องแบบยูนิโพรเซสเซอร์ (UP) จะถูกบ่งชี้โดย 1 ค่าที่มากกว่า 1 จะบ่งชี้ว่าเป็นเครื่องแบบมัลติโพรเซสเซอร์ (MP) ค่าที่ส่งคืนสำหรับพารามิเตอร์ **_SC_NPROCESSORS_ONLN** จะเป็นจำนวนของ CPU ที่แอ็คทีฟ และจะถูกลดจำนวนลงทุกครั้งที่ตัวประมวลผลถูกจัดสรรคืน

ฟิลด์ **_system_configuration.ncpus** ระบุจำนวนของ CPU ที่แอ็คทีฟบนเครื่อง ฟิลด์นี้จะคล้ายคลึงกับพารามิเตอร์ **_SC_NPROCESSOR_ONLN** For more information, see **systemcfg.h** File in *Files Reference*.

สำหรับโค้ดที่ต้องจดจำจำนวนของตัวประมวลผลซึ่งพร้อมใช้งานในเวลาที่บูต ฟิลด์ **ncpus_cfg** จะถูกเพิ่มไปยังตาราง _system_configuration ซึ่งยังคงเป็นค่าคงที่ ระหว่างที่รีบูต

CPU จะถูกระบุโดย CPU ID ที่โยงไว้ในช่วง [0..(ncpus–1)] ตัวประมวลผล ยังมีจำนวนของ CPU แบบฟิสิคัลที่ขึ้นอยู่กับ CPU บอร์ดที่ CPU เหล่านั้นอยู่ โดยเรียงตามลำดับ และอื่นๆ คำสั่งและรูทีนย่อยที่ทำงานกับจำนวน CPU จะใช้จำนวน CPU ที่โยงไว้เพื่อให้ง่ายต่อการส่งผ่านต่อจำนวนที่ผกผันของ CPU จำนวน CPU ที่โยงไว้จะเป็นจำนวนที่ต่อเนื่องกันในช่วง [0..(ncpus–1) ผลกระทบต่อเหตุการณ์นี้จากมุมมองของผู้ใช้ เมื่อการจัดสรรคืนตัวประมวลผลเข้าแทนที่แล้ว การจัดสรรคืนนั้นจะดูคล้ายกับหมายเลขสูงสุด ("ล่าสุด") ที่โยงไว้กับ CPU จะไม่ปรากฏขึ้น โดยไม่พิจารณาถึงตัวประมวลผลแบบฟิสิคัลที่ล้มเหลว

หมายเหตุ: หากต้องการหลีกเลี่ยงปัญหา ให้ใช้ตัวแปร *ncpus_cfg* เพื่อกำหนดหมายเลขสูงสุดที่เป็นไปได้ของ CPU ที่โยงไว้สำหรับระบบโดยเฉพาะ

อินเตอร์เฟสเพื่อเชื่อมอินเตอร์เฟสกับตัวประมวลผลเฉพาะ

คำสั่ง **bindprocessor**และโปรแกรมมิงอินเตอร์เฟส **bindprocessor** อนุญาตให้คุณโยงเธรดหรือการประมวลผลกับ CPU ที่ระบุเฉพาะ ซึ่งกำหนดไว้โดยหมายเลข CPU ที่โยงไว้ อินเตอร์เฟสทั้งสองแบบจะอนุญาตให้คุณ โยงเธรดหรือการประมวลผลกับ CPU ที่แอ็คทีฟเท่านั้น โปรแกรมเหล่านั้นที่ใช้โปรแกรมมิงอินเตอร์เฟส **bindprocessor** โดยตรง หรือมีข้อจำกัดภายนอกด้วยคำสั่ง **bindprocessor** ต้องสามารถจัดการกับ การจัดสรรคืนตัวประมวลผล

ปัญหาหลักจะมองเห็นด้วยโปรแกรมที่โยงกับตัวประมวลผล เมื่อ CPU ที่ถูกจัดสรรคืนคือคำร้องขอให้โยงกับตัวประมวลผลที่ถูกจัดสรรคืนจะเกิดความล้มเหลว โค้ดที่ออกคำร้องขอ **bindprocessor** ควรตรวจสอบค่าส่งคืนจากคำร้องขอเหล่านี้

สำหรับข้อมูลเพิ่มเติมเกี่ยวกับอินเตอร์เฟสเหล่านี้ โปรดดู **bindprocessor** Command ใน *Commands Reference, Volume 1* หรือ **bindprocessor** Subroutine ใน *Technical Reference: Base Operating System and Extensions, Volume 1*

อินเตอร์เฟสสำหรับการแจ้งเตือนการยกเลิกการจัดสรรตัวประมวลผล

กลไลการแจ้งเตือนจะแตกต่างกันสำหรับแอ็พพลิเคชันในโหมดผู้ใช้ซึ่งมีเธรด ที่มีข้อจำกัดกับ CPU ที่โยงไว้ล่าสุดมากกว่าสำหรับส่วนขยายเคอร์เนล

การแจ้งเตือนในโหมดผู้ใช้

เธรดแต่ละตัวของแอ็พพลิเคชันในโหมดผู้ใช้ที่มีข้อจำกัดกับ CPU ที่โยงไว้ล่าสุดจะถูกส่งสัญญาณ **SIGCPUFAIL** และ **SIGRECONFIG** แอ็พพลิเคชันเหล่านี้ต้องการแก้ไขเพื่อจับสัญญาณเหล่านี้ และการควบคุมเธรดที่มีข้อจำกัดกับ CPU ที่โยงไว้ล่าสุด (ด้วยการยกเลิกการโยง หรือโยงไว้กับ CPU อื่นๆ อย่างใดอย่างหนึ่ง)

การแจ้งเตือนในโหมดเคอร์เนล

ไดรเวอร์และส่วนขยายเคอร์เนลที่ต้องถูกแจ้งเตือนการจัดสรรคืนตัวประมวลผลที่ใกล้เข้ามา ต้องลงทะเบียนรูทีน High-Availability Event Handler (HAEH) ด้วยเคอร์เนล รูทีนนี้จะถูกเรียกเมื่อการจัดสรรคืนตัวประมวลผล ซึ่งใกล้จะเกิดขึ้น อินเตอร์เฟสยังคงถูกจัดเตรียมไว้เพื่อยกเลิกการลงทะเบียน HAEH ก่อนที่ส่วนขยายเคอร์เนลจะถูกถอนการปรับแต่งหรือถอดออก

การลงทะเบียนตัวจัดการเหตุการณ์ที่มีความพร้อมใช้งานสูง

เคอร์เนลจะเอ็กซ์พอร์ตฟังก์ชันใหม่เพื่ออนุญาตให้การแจ้งเตือนของส่วนขยายเคอร์เนล ในกรณีที่เหตุการณ์นั้นมีผลกระทบกับสภาพพร้อมใช้งานของระบบ

การเรียกระบบคือ:

```
int register_HA_handler(ha_handler_ext_t *)
```

สำหรับข้อมูลเพิ่มเติมเกี่ยวกับการเรียกระบบนี้ โปรดดู **register_HA_handler** ใน *Operating system and device management*

ค่าส่งคืนจะมีค่าเท่ากับ 0 ในกรณีที่เป็นผลสำเร็จ ค่าที่ไม่ใช่ศูนย์บ่งชี้ถึงความล้มเหลว

อาร์กิวเมนต์การเรียกระบบคือตัวชี้ไปยังโครงสร้างที่กล่าวไว้ใน HAEH ของส่วนขยายเคอร์เนล โครงสร้างนี้จะถูกกำหนดไว้ในไฟล์ส่วนหัวที่ชื่อ **sys/high_avail.h** ดังต่อไปนี้:

```
typedef struct _ha_handler_ext_ {
    int (*_fun)();        /* Function to be invoked */
    long long _data;      /* Private data for (*_fun)() */
    char        _name[sizeof(long long) + 1];
} ha_handler_ext_t;
```

ฟิลด์ **_data** ส่วนบุคคลจะถูกจัดเตรียมไว้สำหรับการใช้ส่วนขยายเคอร์เนล หากมีความต้องการ ไม่ว่าค่าใดที่ถูกกำหนดไว้ในฟิลด์นี้ในเวลาของการลงทะเบียน จะส่งผ่านเป็นพารามิเตอร์ไปยังฟังก์ชันที่ลงทะเบียนแล้ว เมื่อฟิลด์ถูกเรียกเนื่องจากเหตุการณ์ความล้มเหลวของ CPU ที่คาดการณ์ไว้

ฟิลด์ **_name** คือสตริง null ที่ถูกยกเลิก ซึ่งมีความยาวอักขระสูงสุด 8 ตัวอักษร (ไม่รวมอักขระ null ของเทอร์มิเนเตอร์) ซึ่งถูกใช้เพื่อระบุส่วนขยายเคอร์เนลด้วยเคอร์เนล ชื่อนี้ต้องไม่ซ้ำกัน ระหว่างส่วนขยายเคอร์เนลที่ลงทะเบียนทั้งหมด ชื่อนี้ถูกแสดงไว้ในพื้นที่ข้อมูลโดยละเอียดของรายการบันทึกข้อผิดพลาด `CPU_DEALLOC_ABORTED` หากส่วนขยายเคอร์เนลส่งคืนข้อผิดพลาดเมื่อรูทีน HAEH ถูกเรียกโดยเคอร์เนล

ส่วนขยายเคอร์เนลควรลงทะเบียน HAEH เพียงครั้งเดียวเท่านั้น

การเรียกใช้ตัวจัดการเหตุการณ์ที่มีความพร้อมใช้งานสูง

พารามิเตอร์ต่อไปนี้จะเรียกรูทีน HAEH:

- ค่าของฟิลด์ **_data** ของโครงสร้าง **ha_handler_ext_t** จะส่งผ่านไปยัง **register_HA_handler**
- ตัวชี้ไปยังโครงสร้าง **ha_event_t** จะกำหนดไว้ในไฟล์ **sys/high_avail.h** ดังนี้:

```
typedef struct {                    /* High-availability related event */
    uint _magic;                    /* Identifies the kind of the event */
#define HA_CPU_FAIL 0x40505546      /* "CPUF" */
    union {
        struct {                   /* Predictive processor failure */
            cpu_t dealloc_cpu;     /* CPU bind ID of failing processor */
                       ushort domain;         /* future extension */
            ushort nodeid;         /* future extension */
            ushort reserved3;      /* future extension */
            uint reserved[4];      /* future extension */
        } _cpu;
        /* ... */                  /* Additional kind of events -- */
        /* future extension */
    } _u;
} haeh_event_t;
```

ฟังก์ชันจะส่งคืนหนึ่งในโค้ดต่อไปนี้ และยังกำหนดอยู่ในไฟล์ **sys/high_avail.h** :

```
#define HA_ACCEPTED 0     /* Positive acknowledgement */
#define HA_REFUSED -1     /* Negative acknowledgement */
```

ถ้าส่วนขยายของการลงทะเบียนไม่ได้ส่งคืน HA_ACCEPTED การจัดสรรคืนจะถูกยกเลิก รูทีน HAEH จะถูกเรียกในสภาวะแวดล้อมการประมวลผล และไม่ต้องการตรึงไว้

ถ้าส่วนขยายเคอร์เนลขึ้นอยู่กับคอนฟิกูเรชันของ CPU รูทีน HAEH ต้องตอบสนองกับการจัดสรรคืน CPU ที่กำลังจะเกิดขึ้น ปฏิกิริยาตอบสนองนี้คือการพึ่งพาแอ็พพลิเคชันสูง หากต้องการอนุญาตให้ AIX ดำเนินการกับการยกเลิกการตั้งค่าคอนฟิกูเรชัน คุณต้องย้ายเธรด ที่มีข้อจำกัดกับ CPU ที่โยงไว้ล่าสุด หากมี และ หากมีการใช้ตัวจับเวลาจากเธรดที่มีข้อจำกัด ตัวจับเวลาเหล่านั้นจะถูกย้ายไปยัง CPU อื่น ซึ่งเป็นส่วนของการยกเลิกการจัดสรรคืน CPU ถ้ามีการพึ่งพาใดๆ บนตัวจับเวลาเหล่านี้ที่กำลังส่งไปยัง CPU ที่ระบุเฉพาะ คุณต้องทำการดำเนินการ (เช่น หยุดทำงาน) และรีสตาร์ทคำร้องขอตัวจับเวลา เมื่อเธรดมีข้อจำกัดกับ CPU ใหม่

การยกเลิกการลงทะเบียนของตัวจัดการเหตุการณ์ที่มีความพร้อมใช้งานสูง

หากต้องการเก็บระบบที่มีความสอดคล้องและป้องกันระบบขัดข้อง ส่วนขยายเคอร์เนลที่ลงทะเบียน HAEH ต้องยกเลิกการลงทะเบียน เมื่อส่วนขยายเคอร์เนลเหล่านั้น ถูกยกเลิกการปรับแต่ และกำลังถูกถอดออก อินเตอร์เฟสมีดังต่อไปนี้:

```
int unregister_HA_handler(ha_handler_ext_t *)
```

อินเตอร์เฟสนี้จะส่งคืนอ 0 ในกรณีที่เป็นผลสำเร็จ ค่าส่งคืนที่ไม่ใช่ค่าศูนย์ใดๆ บ่งชี้ถึงข้อผิดพลาด

สำหรับข้อมูลเพิ่มเติมเกี่ยวกับการเรียกระบบ โปรดดู **unregister_HA_handler** ใน *Technical Reference: Kernel and Subsystems, Volume 1*

การยกเลิกการจัดสรรตัวประมวลผลในสภาวะแวดล้อมการทดสอบ

หากต้องการทดสอบการแก้ไขใดๆ ที่ทำในแอ็พพลิเคชัน หรือส่วนขยายเคอร์เนล เพื่อสนับสนุนการการจัดสรรคืนตัวประมวลผลนี้ ให้ใช้คำสั่งต่อไปนี้เพื่อทริกเกอร์การจัดสรรคืน CPU ที่ถูกออกแบบโดยหมายเลข CPU แบบโลจิคัล รูปแบบไวยากรณ์คือ:

```
cpu_deallocate cpunum
```

โดยที่:

cpunum คือหมายเลย CPU แบบโลจิคัล

คุณต้องรีบูตระบบเพื่อขอรับตัวประมวลผลเป็นหมายกลับคืนแบบออนไลน์ ดังนั้น คำสั่งนี้จะถูกจัดเตรียมไว้สำหรับวัตถุประสงค์ในการทดสอบเท่านั้น และ *ไม่* ทำตามเครื่องมือการดูแลระบบ

Dynamic memory guarding

ระบบ AIX ถูกออกแบบมาเพื่อแก้ไขเกี่ยวกับข้อผิดพลาดของหน่วยความจำ การแก้ไขข้อผิดพลาดของหน่วยความจำเป็นผลของการกู้คืนฮาร์ดแวร์และระดับ ระบบปฏิบัติการ

มีหลายวิธีในการจำแนกประเภทข้อผิดพลาดของหน่วยความจำ แต่สำหรับ วัตถุประสงค์ของการสนทนานี้ ข้อผิดพลาดของหน่วยความจำจะถูกแบ่งออกเป็นข้อผิดพลาดที่ สามารถกู้คืนได้และกู้คืนไม่ได้

ข้อผิดพลาดที่กู้คืนได้ทำให้ข้อมูลที่อยู่ในตำแหน่งที่ระบุสามารถเรียกค้นได้ และข้อผิดพลาดที่กู้คืนไม่ได้ทำให้สูญเสียข้อมูลจากตำแหน่งที่ระบุ ไป ข้อผิดพลาดที่กู้คืนไม่ได้โดยปกติจะถูกแก้ไขโดยใช้การลดความซ้ำซ้อนของฮาร์ดแวร์ ในระบบย่อยหน่วยความจำ หรือโดยการมาส์กพื้นที่ที่มีปัญหามิให้ใช้งาน ระหว่างการบูตของระบบปฏิบัติการ

AIX สนับสนุนการแก้ไขที่เป็นวิธีป้องกันข้อผิดพลาดหน่วยความจำที่กู้คืนได้ มิให้กลายเป็นข้อผิดพลาดที่กู้คืนไม่ได้โดยใช้เทคนิค Dynamic Memory Guarding Dynamic Memory Guarding ขึ้นอยู่กับการสนับสนุนที่ฮาร์ดแวร์ จัดให้มี ฮาร์ดแวร์จัดเตรียมกลไกสำหรับการตรวจหาและกู้คืนข้อผิดพลาด (เช่น การถูทำความสะอาดหน่วยความจำ และ error correcting circuits (ECC)) ฮาร์ดแวร์ ยังมีวิธีป้องกันการเกิดข้อผิดพลาดที่กู้คืนไม่ได้ในอนาคตด้วย รวมถึงการใช้บิตซ้ำซ้อน

ในฐานะเป็นส่วนเติมเต็มให้แก่กลไกฮาร์ดแวร์เหล่านี้ ฮาร์ดแวร์สามารถแจ้งให้ ระบบปฏิบัติการทราบเกี่ยวกับข้อผิดพลาดที่ได้รับการจัดการอย่างดีที่สุดผ่าน Dynamic Memory Guarding นี้ทำได้โดยการระบุพื้นที่ของหน่วยความจำที่จะถูกจัดสรรใหม่ ระบบปฏิบัติการ AIX ใช้ข้อมูลนี้เพื่อมาส์กพื้นที่หน่วยความจำที่มีปัญหา และเพื่อหยุดการใช้งาน ระบบปฏิบัติการจะย้ายข้อมูลใดๆ ที่ขณะนี้อยู่ใน พื้นที่หน่วยความจำที่มีข้อผิดพลาดไปยังพื้นที่หน่วยความจำอื่น และจากนั้นหยุดการใช้ เพจหน่วยความจำที่มีตำแหน่งข้อผิดพลาดที่เกิดข้อผิดพลาด การป้องกันหน่วยความจำนี้ ทำโดยระบบปฏิบัติการโดยไม่ต้องมีผู้ใช้เกี่ยวกับ และแสดงให้ ผู้ใช้และแอ็พพลิเคชันได้ทราบ

การสร้างเซอร์วิสการล็อก

โปรแกรมเมอร์บางคนอาจต้องการสร้างเซอร์วิส การล็อกระดับสูงของตัวเอง แทนการใช้เซอร์วิสการล็อกมาตรฐาน (mutexes) ที่จัดเตรียมในเธรดไลบรารี

ตัวอย่างเช่น ผลิตภัณฑ์ฐานข้อมูลอาจใช้ชุดของเซอร์วิสที่กำหนดไว้ภายในอยู่แล้ว เป็นเรื่องง่ายกว่า ที่จะปรับเซอร์วิสการล็อกเหล่านี้ให้เป็นระบบใหม่แทนการปรับโมดูล ภายในทั้งหมดที่ใช้เซอร์วิสเหล่านี้

สำหรับเหตุผลนี้ AIX จัดเตรียมการเซอร์วิสการล็อก atomic แบบ พื้นฐานที่สามารถถูกใช้เพื่อสร้างเซอร์วิสการล็อกระดับสูงกว่า เพื่อสร้างเซอร์วิสที่เป็นแบบ multiprocessor-safe (เหมือนเซอร์วิส mutex มาตรฐาน) โปรแกรมเมอร์ต้องใช้เซอร์วิสการล็อก atomic ตามที่อธิบายในส่วนนี้ ไม่ใช่เซอร์วิสที่ดำเนินการแบบไม่ใช่ atomic เช่นรูทีนย่อย **compare_and_swap**

เซอร์วิสการล็อก Multiprocessor-safe

เซอร์วิสการล็อกถูกใช้เพื่อ serialize การเข้าถึงทรัพยากร ที่อาจถูกใช้พร้อมกัน ตัวอย่างเช่น เซอร์วิสการล็อกสามารถถูกใช้สำหรับ การแทรกในรายการโยง ซึ่งต้องการการอัพเดตตัวชี้หลายตัว ถ้าการอัพเดต จัดลำดับโดยหนึ่งกระบวนการถูกอินเตอร์รัปต์โดยกระบวนการที่สองซึ่ง พยายามเข้าถึงรายการเดียวกัน จะมีข้อผิดพลาดเกิดขึ้นได้ ลำดับของการดำเนินการ ที่ไม่ควรถูกอินเตอร์รัปต์เรียกว่า *ส่วนวิกฤติ*

เซอร์วิสการล็อกใช้ lock word เพื่อระบุสถานะการล็อก : 0 (ศูนย์) สามารถถูกใช้เพื่อหมายถึง free และ 1 (หนึ่ง) หมายถึง busy ดังนั้น เซอร์วิสที่รับล็อกจะเป็นดังต่อไปนี้:

```
ทดสอบ lock word
ถ้าล็อกเป็น free
        ตั้งค่า lock word เป็น busy
        return SUCCESS
...
```

เนื่องจากลำดับการดำเนินการนี้ (read, test, set) โดยตัวมันเองเป็นส่วนวิกฤติ จำเป็นต้องมีการจัดการพิเศษ บนระบบ uniprocessor การปิดใช้งานอินเตอร์รัปต์ระหว่างส่วนวิกฤติป้องกันการอินเตอร์รัปต์โดย context switch แต่บนระบบ multiprocessor ฮาร์ดแวร์ต้องจัดเตรียม test-and-set พื้นฐาน โดยปกติด้วยคำสั่งเครื่องพิเศษ นอกจากนี้ คำสั่งการซิงโครไนซ์ processor-dependent พิเศษเรียกว่า *การปิดกั้นการอิมพอร์ตและเอ็กซ์พอร์ต* ถูกใช้เพื่อบล็อกการอ่านหรือ เขียนอื่นชั่วคราว ซึ่งป้องกันการเข้าถึงพร้อมกันโดยโพรเซสเซอร์หลายตัวและ ป้องกันการจัดลำดับการอ่านและเขียน ใหม่ที่กระทำโดยโพรเซสเซอร์สมัยใหม่ และถูกกำหนดดังนี้:

Import fences
: *import fence* เป็นคำสั่งเครื่องพิเศษ ที่หน่วงเวลาจนกว่าคำสั่งที่ออกมาก่อนหน้านี้เสร็จสมบูรณ์ เมื่อใช้ร่วม กับการล็อก จะช่วยป้องกันการกระทำการที่ไม่แน่นอนของ คำสั่งจนกว่าจะได้รับล็อก

Export fences
: *export fence* การันตีว่าข้อมูลที่ถูก ป้องกันจะมองเห็นได้โดยโพรเซสเซอร์อื่นก่อนล็อกถูกรีลีส

เพื่อลดความซับซ้อนและให้อิสระจาก คำสั่งที่ขึ้นกับเครื่องเหล่านี้ จึงมีการกำหนดรูทีนย่อยดังต่อไปนี้:

`_check_lock`
: อัพเดตตัวแปร single word แบบ atom ตามเงื่อนไข, เรียก ใช้ *import fence* สำหรับระบบ multiprocessor รูทีนย่อย **compare_and_swap** เหมือนกัน แต่ไม่มีการเรียก import fence ดังนั้นใช้ไม่ได้ในการสร้างล็อก

`_clear_lock`
: เขียนตัวแปร single word แบบ atom เรียกใช้ *export fence* สำหรับระบบ multiprocessor

เคอร์เนลโปรแกรมมิง

สำหรับรายละเอียดสมบูรณ์เกี่ยวกับเคอร์เนล โปรแกรมมิง โปรดดูที่ *Kernel Extensions and Device Support Programming Concepts* ส่วนนี้ไฮไลต์ความแตกต่างหลัก ที่จำเป็นสำหรับระบบ multiprocessor

บ่อยครั้งที่จำเป็นต้องใช้การ Serialize เมื่อเข้าถึง ทรัพยากรที่สำคัญมาก เซอร์วิสการล็อกสามารถถูกใช้เพื่อ serialize การเข้าถึงเธรด ในสภาพแวดล้อมกระบวนการ แต่เซอร์วิสจะไม่มีการป้องกันการเข้าถึงที่เกิดขึ้นใน สภาพแวดล้อมอินเตอร์รัปต์ โค้ดใหม่หรือที่พอร์ตควรใช้เคอร์เนลเซอร์วิส **disable_lock** และ **unlock_enable** ซึ่งใช้ล็อกธรรมดาเพื่ออินเตอร์รัปต์ การควบคุมแทนเซอร์วิสเคอร์เนล **i_disable** เซอร์วิสเคอร์เนลนี้ยังสามารถใช้กับระบบ uniprocessor ซึ่งใช้เซอร์วิสอินเตอร์รัปต์โดยไม่มีการล็อก สำหรับข้อมูล รายละเอียด โปรดดูที่ การล็อกเคอร์เนลเซอร์วิส ใน *Kernel Extensions and Device Support Programming Concepts*

ไดรเวอร์อุปกรณ์โดยค่าเริ่มต้นรันในสภาพแวดล้อมโลจิคัล uniprocessor เรียกว่าโหมด *funneled* ไดรเวอร์ ที่เขียนมาดีสำหรับระบบ uniprocessor จะทำงานโดยไม่มีการแก้ไข ในโหมดนี้ แต่คุณต้องตรวจสอบและปรับเปลี่ยนอย่างระมัดระวังเพื่อให้ได้รับประโยชน์ จาก multiprocessing สุดท้าย เคอร์เนลเซอร์วิสสำหรับตัวจับเวลาขณะนี้มีค่าส่งกลับ เนื่องจากค่าส่งกลับจะไม่สำเร็จเสมอไปในสภาพแวดล้อมมัลติโพรเซสเซอร์ ดังนั้น โค้ดใหม่หรือที่พอร์ตต้องตรวจสอบค่าส่งกลับเหล่านี้ สำหรับรายละเอียด โปรดดูที่ การใช้ Multiprocessor-Safe Timer Services ใน *Kernel Extensions and Device Support Programming Concepts*

ตัวอย่างของเซอร์วิสการล็อก

รูทีนย่อยการล็อก multiprocessor-safe สามารถถูกใช้ เพื่อสร้างรูทีนระดับสูงแบบกำหนดเองที่ไม่ขึ้นกับเธรดไลบรารี ตัวอย่างต่อไปนี้แสดงส่วนของการนำรูทีนย่อยไปปฏิบัติเหมือนกับ รูทีนย่อย **pthread_mutex_lock** และ**pthread_mutex_unlock** ในไลบรารีเธรด:

```
#include <sys/atomic_op.h>       /* for locking primitives */
#define SUCCESS          0
#define FAILURE          -1
#define LOCK_FREE        0
#define LOCK_TAKEN       1

typdef struct {
        atomic_p         lock;   /* lock word */
        tid_t            owner;  /* identifies the lock owner */
        ...              /* implementation dependent fields */
} my_mutex_t;

...

int my_mutex_lock(my_mutex_t *mutex)
{
tid_t   self;   /* caller's identifier */

        /*
        Perform various checks:
          is mutex a valid pointer?
          has the mutex been initialized?
        */
        ...

        /* test that the caller does not have the mutex */
        self = thread_self();
        if (mutex->owner == self)
                return FAILURE;

        /*
        Perform a test-and-set primitive in a loop.
```

```
        In this implementation, yield the processor if failure.
        Other solutions include: spin (continuously check);
                 or yield after a fixed number of checks.
        */
        while (_check_lock(mutex->lock, LOCK_FREE, LOCK_TAKEN))
                yield();

        mutex->owner = self;
        return SUCCESS;
} /* end of my_mutex_lock */

int my_mutex_unlock(my_mutex_t *mutex)
{
        /*
        Perform various checks:
          is mutex a valid pointer?
          has the mutex been initialized?
        */
        ...

        /* test that the caller owns the mutex */
        if (mutex->owner != thread_self())
                return FAILURE;

        _clear_lock(mutex->lock, LOCK_FREE);
        return SUCCESS;
} /* end of my_mutex_unlock */
```

ข้อมูลที่เกี่ยวข้อง:

การล็อกเคอร์เนลเซอร์วิส

การใช้ Multiprocessor-Safe Timer Services

bindprocessor

compare_and_swap

pthread_mutex_unlock

disable_lock

i_disable

unlock_enable

คำสั่ง bindprocessor

โปรแกรมอำนวยความสะดวกการติดตาม ProbeVue แบบไดนามิก

คุณสามารถใช้ตัวช่วย trace แบบไดนามิกสำหรับ ProbeVue สำหรับทั้งการวิเคราะห์ผลการทำงานและการดีบักปัญหา ProbeVue ใช้ภาษาโปรแกรม Vue เพื่อระบุจุด trace แบบไดนามิก และเตรียมการดำเนินการเพื่อรันที่จุด trace ที่ระบุไว้

ProbeVue สอดแทรกคุณลักษณะดังต่อไปนี้:

- ไม่ต้องมี trace hook ที่คอมไพล์ล่วงหน้า ProbeVue จะทำงานบนเคอร์เนลหรือแอ็พพลิเคชันสำหรับผู้ใช้ที่ไม่มีการแก้ไข

- trace hook ไม่ต้องคอมไพล์ล่วงหน้า แต่จะถูกคอมไพล์โดยเป็นส่วนหนึ่งของซอร์สโปรแกรม ProbeVue จะทำงานบนเคอร์เนลหรือแอ็พพลิเคชันสำหรับผู้ใช้ที่ไม่มีการแก้ไข
- trace hook จะไม่ได้รับผลกระทบ (ไม่มีอยู่) จนกว่าจะเปิดใช้งานแบบไดนามิก
- การดำเนินการ trace (ระบุด้วยโค้ดการจัดให้มีเครื่องมือ) ที่ต้องการออกคำสั่งที่ trace hook จะถูกจัดเตรียมไว้ในเวลาที่เปิดใช้งาน trace hook
- ข้อมูล trace ที่ดักจับไว้เป็นส่วนหนึ่งของการดำเนินการ trace จะพร้อมใช้งานสำหรับการดูโดยทันที และสามารถแสดงเป็นเอาต์พุตเทอร์มินัล หรือบันทึกลงในไฟล์เพื่อดูในภายหลังได้

หมายเหตุ: dbx และ Probevue ไม่สามารถดีบักโพรเซสได้พร้อมกัน ในบางครั้งการลองดีบักที่เริ่มต้นโดย ProbeVue อาจทำให้โพรเซส dbx ต้องรอให้โพรเซสอื่นเชื่อมต่อ

ภาษาและสคริปต์

ProbeVue ใช้โพรบสำหรับการกำหนดปัญหา และการวิเคราะห์ผลการทำงาน *โพรบ* คือกลไกของซอฟต์แวร์ที่อินเตอร์รัปต์การดำเนินการของระบบปกติเพื่อตรวจสอบ และขอรับข้อมูลเกี่ยวกับบริบทและสถานะของระบบปัจจุบัน

และโดยปกติแล้ว ยังอ้างเป็นการ trace การดำเนินการ trace หรือการดำเนินการโพรบยังดักจับข้อมูล โดยเขียนค่าปัจจุบันของโกลบอลและข้อมูลที่ระบุเฉพาะบริบทให้กับบัฟเฟอร์ trace ข้อมูลนี้จะถูกเรียกข้อมูล trace ระบบจะจัดเตรียมสิ่งอำนวยความสะดวกเพื่ออ่านข้อมูลของบัฟเฟอร์ และทำให้พร้อมใช้งานกับผู้ใช้ของระบบ

จุดโพรบ จะระบุจุดระหว่างกิจกรรมระบบปกติที่มีความสามารถโพรบได้ ด้วยการ trace แบบไดนามิก จุดโพรบจะไม่มีโพรบใดๆ ที่พ่วงกับจุดโพรบใดๆ นอกจากจะถูก trace *การเปิดใช้งาน* โพรบ คือการดำเนินการพ่วงต่อโพรบกับจุดโพรบและ *การปิดใช้งาน* โพรบคือการดำเนินการย้ายโพรบจากจุดโพรบ โพรบจะถูก *ทริกเกอร์* หรือ *เกิดขึ้น* เมื่อเข้าถึงจุดโพรบที่เปิดใช้งาน ในระหว่างกิจกรรมของระบบและดำเนินการกับการดำเนินการติดตาม

ProbeVue สนับสนุนหมวดหมู่ของจุดโพรบสองแบบดังนี้ :

ตำแหน่งของโพรบ
ตำแหน่งที่อยู่ในโค้ดของผู้ใช้หรือโค้ดของเคอร์เนล ซึ่งดำเนินการกับการดำเนินการติดตามบางอย่าง โพรบที่เปิดใช้งานที่ตำแหน่งโพรบ เมื่อเธรดที่รันโค้ดเข้าถึงตำแหน่งนั้น

เหตุการณ์ของโพรบ
เหตุการณ์ที่น่าสนใจซึ่งมีการดำเนินการติดตามบางเหตุการณ์ เกิดขึ้น เหตุการณ์โพรบไม่ได้แม็พกับตำแหน่งโค้ดที่ระบุเฉพาะได้โดยง่าย โพรบที่เปิดใช้งานซึ่งบ่งชี้ถึงเหตุการณ์โพรบจะถูกปรับแต่ง เมื่อเหตุการณ์นั้นเกิดขึ้น

ProbeVue ยังแยกจุดโพรบด้วยชนิด ชนิดโพรบระบุชุดของจุดโพรบที่แบ่งใช้คุณสมบัติทั่วไปบางอย่าง ตัวอย่างเช่น จุดโพรบที่ระบุ entry และทางออกของระบบที่เรียก หรือจุดโพรบที่บ่งชี้อัพเดตระบบสถิติ การแบ่งแยกโพรบด้วยชนิดที่กำหนดโครงสร้าง ให้กับชนิดที่หลากหลายของจุดโพรบ

ProbeVue ต้องการตัวจัดการโพรบ ที่เชื่อมโยงกับชนิดของโพรบแต่ละชนิด ตัวจัดการโพรบคือโค้ดของซอฟต์แวร์ที่กำหนดและจัดเตรียมชุดของจุดโพรบของชนิดโพรบที่เหมือนกัน ตัวอย่างเช่น ตัวจัดการโพรบของระบบที่เรียก

ภาษาโปรแกรมมิง **Vue**

ภาษาโปรแกรมมิง Vue จัดเตรียมข้อกำหนดคุณสมบัติการติดตามให้กับ ProbeVue สคริปต์ Vue หรือโปรแกรม Vue คือ โปรแกรมที่เขียนลงใน Vue คุณสามารถใช้สคริปต์ Vue สำหรับวัตถุประสงค์ต่อไปนี้:

- ระบุจุดโพรบที่เปิดใช้งานโพรบแบบไดนามิก
- ระบุเงื่อนไข ถ้ามี ซึ่งต้องเป็นการแก้ปัญหาสาเหตุของการโพรบที่เกิดขึ้น
- ระบุการดำเนินการที่ออกคำสั่ง ซึ่งรวมถึงข้อมูลการติดตาม ที่คุณต้องการดักจับ

สคริปต์ Vue จะบอกให้ ProbeVue ทราบถึงที่ที่ต้องติดตาม เวลาที่ต้องติดตาม และสิ่งที่ต้องติดตาม สคริปต์ Vue ควรมี ส่วนต่อท้ายไฟล์ที่เป็น ".e" เพื่อแยกสคริปต์ออกจากไฟล์ชนิดอื่น

หมายเหตุ Vue เป็นทั้งภาษาโปรแกรมมิงและภาษาสคริปต์: ซึ่งเป็นภาษาการติดตามเฉพาะงานแบบไดนามิก Vue สนับสนุนเซ็ตย่อยของ C และไวยากรณ์การสร้างสคริปต์ที่มีประโยชน์สำหรับวัตถุประสงค์ในการติดตามแบบไดนามิก

อิลิเมนต์ของ C

Vue สนับสนุนเซ็ตย่อยของ C ตารางต่อไปนี้อธิบายถึงวิธีที่ ProbeVue ให้การสนับสนุนคีย์เวิร์ดในภาษา C ที่ระบุไว้ คีย์เวิร์ดใน ภาษา C ทั้งหมดยังคงมีข้อจำกัดอยู่ใน Vue การใช้คีย์เวิร์ดเหล่านี้เป็นชื่อตัวแปร หรือสัญลักษณ์อื่นๆ ไม่ได้แฟล็กไว้ว่าเป็นข้อผิดพลาดทางไวยากรณ์ อย่างไรก็ตาม ลักษณะการทำงานของการใช้ไม่ได้ถูกนิยามไว้

หมายเหตุ: คีย์เวิร์ดในคอลัมน์ที่สองสามารถแสดงอยู่ในชนิด หรือนิยามโครงสร้างหรือการประกาศฟังก์ชัน คอมไพล์เลอร์ Vue จะละเว้นคีย์เวิร์ดเหล่านี้ อย่างไรก็ตาม คีย์เวิร์ดเหล่านี้ต้องไม่ถูกนำมาใช้ เมื่อประกาศตัวแปรสคริปต์ Vue

สนับสนุน	อนุญาตให้ใช้ในไฟล์ส่วนหัวหรือส่วนที่มีการประกาศ	ไม่สนับสนุน
char	auto	break
double	const	case
else	extern	continue
enum	register	default
float	static	do
if	typedef	for
int	volatile	goto
long		switch
return		while
short		
signed		
sizeof		
struct		
union		
unsigned		
void		

รายการต่อไปนี้อธิบายถึงชุดของคุณลักษณะในภาษา C ที่ Vue สนับสนุน:

คำสั่ง คำสั่งในภาษา C ทั้งหมดยกเว้น for loop และคำสั่ง control flow บางคำสั่ง

ตัวดำเนินการ

ตัวดำเนินการ unary, binary และ ternary ในภาษา C ยกเว้นตัวดำเนินการ ที่เป็นเครื่องหมายคอมมา ตัวดำเนินการที่นำหน้าและเชื่อมโยงจะใช้กฎของภาษา C

ชนิดข้อมูล

การนิยามชนิดตัวแปรให้ถูกต้องตามกฎ C-89 ซึ่งรวมถึงคำสั่งและคีย์เวิร์ดทั้งหมด (struct, union, enum, typedef และอื่นๆ) สำหรับการประกาศชนิดจะมีข้อจำกัด ซึ่งรวมถึงชนิดของตัวแปรและพารามิเตอร์ของเคอร์เนล หรือตัวแปรและพารามิเตอร์ของแอ็พพลิเคชัน

หมายเหตุ: Vue มีกฎของตัวเอง ที่เกี่ยวกับขอบเขตและคลาสหน่วยเก็บข้อมูล

การแปลงชนิด

การแปลงชนิดข้อมูลโดยนัยและการแปลงโดยชัดแจ้งด้วยการหลอมตัวพิมพ์

รูทีนย่อย

ไวยากรณ์สำหรับการเรียกรูทีนย่อย และการส่งพารามิเตอร์ไปยังฟังก์ชัน อย่างไรก็ตาม มีข้อจำกัดเกี่ยวกับฟังก์ชันที่สามารถเรียกใช้ได้

ชื่อตัวแปร

หลักการตั้งชื่อสำหรับตัวแปรให้ปฏิบัติตามกฎ C identifier ข้อกำหนดคุณลักษณะของตัวแปรโดยสมบูรณ์สามารถรวมโคลอนได้ ถ้าชื่อคลาสตัวแปรอยู่นำหน้าชื่อตัวแปร

ไฟล์ส่วนหัว

ไฟล์ส่วนหัวสามารถรวมไว้ เพื่อประกาศชนิดของตัวแปรโกลบอลสำหรับเคอร์เนลหรือต้นแบบของฟังก์ชันในแอ็พพลิเคชัน และเคอร์เนล ซึ่งมีข้อจำกัดบางข้อ เกี่ยวกับไฟล์ส่วนหัวที่รวมไว้

Punctuator

punctuator ในภาษา C ทั้งหมดได้รับการสนับสนุนและจะบังคับใช้โดยกฎ ดังนั้น คำสั่งจะต้องคั่นด้วยเครื่องหมายเซมิโคลอน (;) กฎเกี่ยวกับช่องว่างในภาษา C ทั้งหมดสามารถนำมาปฏิบัติได้

ตัวอักษร

การแสดงของสตริง (โดยใช้เครื่องหมายคำพูดคู่(″)) อักขระตามตัวอักษร (โดยใช้อักขระเครื่องหมายคำพูดเดี่ยว (’)) เลขจำนวนเต็มฐานแปดและ ฐานสิบหก และอักขระพิเศษ เช่น **\n** และ **\t** ลำดับ escape

ความคิดเห็น

ความคิดเห็นเกี่ยวกับลักษณะของภาษา C- และ C++- ความคิดเห็นสามารถปรากฏขึ้นได้ทั้งภายในและภายนอก clause บรรทัดที่ขึ้นต้นด้วยอักขระ # จะถูกละเว้น ให้หลีกเลี่ยงอักขระ ที่บ่งชี้ถึงบรรทัดความคิดเห็น

ข้อแตกต่างจากภาษา C

Vue มีลักษณะการทำงานที่แตกต่างกันสำหรับคุณลักษณะในภาษา C ข้อจำกัดบางข้อจะถูกกำหนดไว้ สำหรับการรักษาประสิทธิภาพในการทำงาน หรือเพื่อทำให้มั่นใจว่า สคริปต์ Vue สามารถรันได้ด้วยความปลอดภัยภายในเคอร์เนล และไม่มีผลกระทบกับกระบวนการโพรบ

คำสั่ง Loop

คำสั่ง Loop ไม่ถูกยอมรับให้ใช้ในสคริปต์ Vue นี่คือข้อควรระวัง ในการป้องกันโพรบ Vue ใดๆ จากการทำงานที่ไม่เสร็จสิ้น

คำสั่งเกี่ยวกับการไหลของเงื่อนไข

เฉพาะคำสั่ง control flow "if-else" เท่านั้นที่สามารถนำมาใช้ได้ในสคริปต์ Vue การไหลของโลจิกส่วนใหญ่สามารถทำผ่านการใช้คำสั่ง "if" ได้อย่างถูกต้อง เพรดิเคตเป็นวิธีที่ได้ประสิทธิภาพในการทำงานในการทำโลจิกแบบมีเงื่อนไขในระดับสูง สำหรับสคริปต์ Vue

คำสั่ง Return

คำสั่ง return ไม่สามารถใช้งานได้โดย Vue เพื่อส่งสัญญาณว่า การประมวลผลของบล็อกการดำเนินงานจะสิ้นสุดลงในทันที อย่างไรก็ตาม คำสั่ง return ไม่ได้ใช้นิพจน์ใดๆ ใน Vue เนื่องจากบล็อกการดำเนินงานของ Vue ไม่มีค่าสำหรับส่งคืน

รูทีนย่อย

สคริปต์ Vue ไม่มีสิทธิในการเข้าถึงฟังก์ชันที่จัดเตรียมไว้โดยระบบ AIX หรือไลบรารีผู้ใช้ทั่วไป ไม่มีส่วนสนับสนุนสำหรับการสร้างไฟล์เก็บถาวร (ไลบรารีของฟังก์ชัน) หรือฟังก์ชันผู้ใช้ ที่สามารถเรียกทำงานได้ภายในโพรบ แต่ไลบรารีภายในแบบพิเศษจะพร้อมให้คุณใช้งานแทน ซึ่งจะจัดเตรียมชุดของฟังก์ชันที่มีประโยชน์สำหรับโปรแกรมติดตามแบบไดนามิก สำหรับโปรแกรมติดตามแบบไดนามิก

เลขทศนิยม

ตัวแปรเลขทศนิยมไม่สามารถยอมรับได้โดย clause ใดๆ ที่เชื่อมโยงกับจุดโพรบของเคอร์เนล คุณสามารถใช้ตัวแปรเลขทศนิยมในคำสั่งกำหนดค่าแบบง่ายๆ และเป็นพารามิเตอร์ในฟังก์ชัน Vue ที่พิมพ์ข้อมูล ภาษาที่ Vue สนับสนุนสำหรับตัวแปรเลขทศนิยม จะถูกจำกัดเพื่อดักจับ

การเปลี่ยนแปลงตัวแปร

ตัวแปรภายนอกไม่สามารถยอมรับโดยด้านซ้ายของคำสั่งกำหนดค่า นั่นคือ ตัวแปรภายนอกไม่สามารถแก้ไขได้ในสคริปต์ Vue

ไฟล์ส่วนหัว

Vue ไม่ได้สนับสนุนการรวมกันของไฟล์ส่วนหัวในตัวสคริปต์ Vue เอง แต่ชื่อของไฟล์ส่วนหัวที่ต้องการรวม ต้องถูกส่งผ่านไปยังอาร์กิวเมนต์บรรทัดรับคำสั่งในคำสั่ง **probevue** ตัวดำเนินการหรือคำสั่งของตัวประมวลผลก่อนในภาษา C จะละเว้นไฟล์ส่วนหัว ซึ่งอาจเป็นเพราะ ลักษณะการทำงานที่ไม่คาดคิด หากต้องการหลีกเลี่ยงสาเหตุนี้ ให้เขียนโค้ดสำหรับไฟล์ส่วนหัวด้วยตนเอง หรือรันตัวประมวลผลก่อนในภาษา C ที่อยู่บนชุดของไฟล์ส่วนหัวที่เกี่ยวข้องโดยตรง และสร้างไฟล์ส่วนหัวหลังการประมวลผลเพื่อรวมเข้าด้วยกัน คุณสามารถสอดแทรกฟังก์ชันต้นแบบและโครงสร้างหรือนิยามแบบ union ในตัวสคริปต์ Vue ถ้าฟังก์ชันและโครงสร้างถูกแทรกอยู่ในตอนต้น ก่อนที่จะเริ่มต้น clauses โพรบใดๆ

ตัวประมวลผลก่อนในภาษา C

ตัวดำเนินการของตัวประมวลผลก่อนในภาษา C นิยามแมโคร บรรทัดหรือคำสั่ง pragma และชื่อแมโครที่นิยามไว้ล่วงหน้าจะถูกละเว้น

การดำเนินการกับตัวชี้

Vue ไม่ยอมรับตัวชี้ไปยังตัวแปรสคริปต์ ตัวอย่างเช่น แอดเดรสของตัวแปรสคริปต์ไม่สามารถนำมาใช้ได้ อย่างไรก็ตาม แอดเดรสของตัวแปรเคอร์เนลสามารถนำมาใช้ได้ และกำหนดให้กับตัวแปร Vue pointer และการดำเนินการกับตัวชี้ที่สนับสนุนการใช้ตัวแปร pointer

เบ็ดเตล็ด

- Trigraphs ไม่สามารถยอมรับได้
- ตัวดำเนินการที่เป็นเครื่องหมายจุลภาคไม่สามารถยอมรับได้
- Declaration statements ไม่สามารถมีการกำหนดค่าเริ่มต้นใดๆ ได้

คำสั่ง probevue

คำสั่ง **probevue** จะเริ่มต้นเซสชันการ trace แบบไดนามิก หรือเซสชัน ProbeVue คำสั่ง **probevue** จะใช้สคริปต์ Vue เป็นการอ่านอินพุตจากไฟล์ หรือจากบรรทัดรับคำสั่ง และเรียกใช้งานเซสชัน ProbeVue ข้อมูล trace ใดๆ ที่ถูกดักจับโดยเซสชัน ProbeVue สามารถพิมพ์ไปยังเทอร์มินัล หรือบันทึกเป็นไฟล์ที่ระบุสำหรับผู้ใช้ต่ออ็อพชันที่ส่งผ่านในบรรทัดรับคำสั่ง

เซสชัน ProbeVue จะยังคงแอ็คทีฟจนกว่าคุณจะพิมพ์ Ctrl-C บนเทอร์มินัล หรือออกคำสั่งดำเนินการออกจากสคริปต์ Vue

การเรียกใช้คำสั่ง **probevue** แต่ละครั้ง จะเรียกใช้เซสชันการ trace แบบไดนามิกที่แบ่งแยกกัน เซสชันการ trace จำนวนมากสามารถแอ็คทีฟได้ในหนึ่งครั้ง แต่เซสชันแต่ละตัวจะแสดงเฉพาะข้อมูล trace ที่ดักจักอยู่ในเซสชันนั้น เซสชันแบบพร้อมเพียงกันจะไม่ถูกรับรู้สำหรับเซสชันอื่นๆ

การรันคำสั่ง **probevue** เป็นการดำเนินการที่มีอภิสิทธิ์ และผู้ใช้ที่ไม่ใช่ผู้ใช้ root ต้องมี privilege ในการเริ่มต้นเซสชันการ trace แบบไดนามิก

Vueสคริปต์

สคริปต์ Vue ประกอบด้วยส่วนที่มีการประกาศอื่นๆ ตามด้วยหนึ่งใน clause ของคุณ ถ้าคุณระบุ clause ลงในสคริปต์ Vue ตามลำดับใดๆ ต่อไปนี้คือโครงสร้างทั่วไปของสคริปต์ Vue :

การประกาศ	ตัวอย่าง

optional BEGIN clause	`@@BEGIN`
	`{`
	`  <statement 1> ;`
	`  ...`
	`  <statement n> ;`
	
	`}`

โพรบ clause ที่มากกว่าหนึ่งตัว	`<probe point tuple>,....,<probe point tuple>`
	`when (<predicate>)  <---- optional predicate`
	`{`
	`  <statement 1> ;`
	`  ...`
	`  <statement n> ;`
	`  <----- action block`
	
	`}`

การประกาศ	ตัวอย่าง

END clause แบบเผื่อเลือก	@@END
	{
	    <statement 1> ;
	    ...
	    <statement n> ;
	
	}

คุณสามารถใช้ส่วนที่มีการประกาศที่เป็นตัวเลือกเพื่อประกาศชนิดตัวแปร และตัวแปรโกลบอล ซึ่งต้องมีคำสั่งที่สามารถเรียกทำงานได้

clause ของสคริปต์ Vue แต่ละส่วนประกอบด้วย อิลิเมนต์สามอย่างต่อไปนี้:

ข้อกำหนดคุณสมบัติของจุดโพรบ
ข้อกำหนดคุณสมบัติของจุดโพรบจะระบุชุดของจุดโพรบ ที่ต้องการเปิดใช้งานแบบไดนามิก ซึ่งประกอบด้วยจุดโพรบ tuples ตั้งแต่หนึ่งจุดขึ้นไป ความยาวทั้งหมดของสตริงนี้ ถูกจำกัดอยู่ที่ 1023 ตัวอักษร

บล็อกการดำเนินการ
บล็อกการดำเนินการระบุลำดับของการดำเนินการโพรบ ที่ต้องการดำเนินการเมื่อใช้โพรบ

เพรดิเคตที่เป็นตัวเลือก
เพรดิเคต ถ้ามีอยู่ จะระบุเงื่อนไขที่ถูกตรวจสอบ ณ เวลาที่โพรบถูกทริกเกอร์ เพรดิเคตต้องถูกประเมินผลเป็น TRUE สำหรับการดำเนินการโพรบของ clause ที่ต้องการออกคำสั่ง

สคริปต์ Vue สามารถ มีความคิดเห็นได้ คุณสามารถวางความคิดเห็นได้ทุกที่ในสคริปต์ Vue: ที่ด้านบนสุดของสคริปต์ Vue ระหว่าง Vue clause สอง clause หรือภายใน Vue clause

ส่วนที่มีการประกาศ

ส่วนของการประกาศซึ่งเป็นค่าเผื่อเลือกมีอิลิเมนต์ใดๆ ต่อไปนี้:

- ชนิด โครงสร้าง union และนิยาม enum
- การประกาศฟังก์ชันต้นแบบสำหรับฟังก์ชันที่โพรบแล้ว
- การประกาศสำหรับตัวแปรโกลบอลและเคอร์เนลที่เข้าถึงได้ในสคริปต์

ไม่ต้องมีคำสั่งที่สามารถเรียกทำงานได้ ซึ่งรวมถึง การกำหนดค่าเริ่มต้นลักษณะของตัวแปร

การประกาศและนิยามทั้งหมดใน Vue จะตามด้วยไวยากรณ์ของข้อกำหนดคุณสมบัติภาษาแบบ C-89 สำหรับภาษา C แม้ว่าจะมีข้อจำกัดที่ระบุเฉพาะ Vue นอกจากนี้ Vue ยังสนับสนุนชนิดข้อมูลเพียงเล็กน้อย และสนับสนุนคีย์เวิร์ดเพื่อบ่งชี้คลาสของตัวแปร Vue

หมายเหตุ: Vue สนับสนุนการประกาศและการกำหนดนิยามตัวแปรโกลบอลเหล่านั้น หรือภายใน Vue clause ใดๆ การประกาศหรือการกำหนดนิยามใดๆ ที่คุณรวมไว้ในการประกาศ สามารถเพิ่มลงใน Vue clause ได้ อย่างไรก็ตาม คอมไพลเลอร์ Vue ต้องการประกาศหรือกำหนดนิยามสำหรับตัวแปรหรือฟังก์ชันที่นำหน้าคำสั่ง ซึ่งมีการพึ่งพาอยู่ ตัวอย่างเช่น คุณไม่สามารถอ้างถึงตัวแปรโกลบอลในหนึ่ง Vue clause และการประกาศใน Vue clause อื่นที่ตามมา แม้ว่าคุณสามารถประกาศ

ตัวแปรโกลบอลใน Vue clause ก่อนหน้านี้ และอ้างถึง Vue clause ในภายหลัง อย่างไรก็ตาม สิ่งนี้สามารถทำให้สคริปต์เข้าใจและดีบักได้ยาก วัตถุประสงค์ของส่วนของการประกาศเผื่อเลือกนี้ จะจัดเตรียมตำแหน่งที่มีความสะดวกเพื่อสอดแทรกการประกาศแบบโกลบอลของคุณ การประกาศฟังก์ชัน และอื่นๆ ที่จุดเริ่มต้นของสคริปต์ Vue เพื่อทำให้สามารถอ่านได้มากขึ้น

ข้อกำหนดคุณสมบัติของจุดโพรบ

ข้อกำหนดคุณสมบัติของจุดโพรบประกอบด้วยจุดโพรบ tuples ตั้งแต่หนึ่งจุดขึ้นไป แต่ละจุดโพรบ tuple จะระบุตำแหน่งโค้ดที่มีการประมวลผล หรือเหตุการณ์ที่เกิดขึ้นต้องทริกเกอร์การดำเนินการโพรบ จุดโพรบจำนวนมาก สามารถเชื่อมโยงกับชุดของการดำเนินการโพรบและเพรดิเคตเดียวกันได้ ถ้ามี โดยจัดเตรียมรายการของโพรบ tuple ที่คั่นด้วยเครื่องหมายจุลภาค ที่ด้านบนสุดของ Vue clause

ต่อไปนี้เป็นโพรบบางชนิดที่สนับสนุน:

- โพรบรายการฟังก์ชันของผู้ใช้ (หรือโพรบ **uft**)
- โพรบระบบเรียกรายการหรือโพรบทางออก (หรือโพรบ **syscall**)
- โพรบที่เกิดขึ้นในช่วงเวลาที่ระบุ (หรือ **ช่วงเวลา** โพรบ)

สำหรับรายการที่สมบูรณ์ของชนิดโพรบที่ได้รับการสนับสนุน โปรดอ้างถึงส่วนของโปรแกรม จัดการโพรบ

จุดโพรบ tuple คือรายการของฟิลด์ที่เรียงลำดับแล้วซึ่งคั่นโดยเครื่องหมายคอมมา ที่ระบุจุดโพรบไว้โดยเฉพาะ ซึ่งมีรูปแบบทั่วไปดังต่อไปนี้ แม้ว่าฟิลด์ตำแหน่งจะถูกแสดงขึ้น หากจุดโพรบคือตำแหน่งโพรบเท่านั้น

```
@@ <probetype>:
<one or more probetype-specific fields separated by colons>:<location>
```

ตัวจัดการโพรบจะกำหนดค่าที่สามารถยอมรับได้สำหรับฟิลด์ที่ระบุเฉพาะสำหรับชนิดโพรบ ในจุดโพรบ tuple และความยาวของโพรบ tuple อย่างไรก็ตาม กฎโดยทั่วไปต่อไปนี้จะปฏิบัติตามตัวจัดการโพรบทั้งหมด เมื่อกำหนดจุดโพรบ tuples:

- จุดโพรบ tuple แต่ละจุดจะมี tuple อย่างน้อย 3 tuple นั่นคือ ต่ำสุด 3 ฟิลด์
- ฟิลด์แรกจะระบุชนิดโพรบเสมอ ดังเช่นตัวจัดการโพรบ
- สำหรับตัวจัดการโพรบที่สนับสนุนการ trace ที่ระบุเฉพาะกระบวนการ ฟิลด์ที่สองต้องเป็น ID ของกระบวนการ
- สำหรับตัวจัดการโพรบที่สนับสนุนรายการฟังก์ชันหรือโพรบออก ฟิลด์ตำแหน่ง (ฟิลด์สุดท้าย) ต้องใช้คีย์เวิร์ด **entry** หรือ **exit**
- ฟิลด์ที่แยกด้วยเครื่องหมายโคลอน (:)
- เครื่องหมายดอกจันหรือสัญลักษณ์ "*" สำหรับฟิลด์ในจุดโพรบ tuple ระบุว่า ตรวกับค่าที่เป็นไปได้สำหรับฟิลด์นั้น ตัวอย่างเช่น ตัวจัดการโพรบ **syscall** อนุญาตสำหรับการเรียกระบบของกระบวนการที่ระบุเฉพาะ หรือสำหรับกระบวนการทั้งหมดที่ต้องการโพรบ ในกรณีแรก ฟิลด์ที่สองต้องเป็น ID กระบวนการ ที่ต้องการโพรบ ใน กรณีหลัง เมื่อกระบวนการทั้งหหมดถูกโพรบแล้ว ฟิลด์ที่สอง ต้องเป็นสัญลักษณ์ "*" การใช้สัญลักษณ์เครื่องหมายดอกจันฟิลด์สำหรับฟิลด์ที่สองเป็นการอนุญาตสำหรับโพรบขนาดเล็กในอนาคต ขณะที่รักษาความเข้ากันได้แบบไบนารีด้วยสคริปต์ที่มีอยู่ ตัวอย่างเช่น ตัวจัดการโพรบ **uft** ต้องการฟิลด์ที่สามที่เป็นเครื่องหมายดอกจัน ในอนาคต ยังสามารถสนับสนุนชื่อโมดูลสำหรับฟิลด์ที่สามเพื่อจำกัดโพรบ เพื่อให้ทำหน้าที่ตามที่ได้กำหนดไว้ในโมดูลนั้น
- ความยาวสูงสดของข้อมูลจำเพาะเกี่ยวกับโพรบคือ 1023 ตัวอักษร

ตัวอย่างเช่น:

@@uft:34568:*:foo:entry

โพรบที่รายการในฟังก์ชันเรียกว่า **foo** ในกระบวนการที่มี ID = 34568 เครื่องหมายดอกจันในฟิลด์ที่สามจะบ่งชี้ว่าฟังก์ชัน **foo** จะถูกโพรบ ถ้ามีอยู่ในโมดูลของกระบวนการใดๆ

@@syscall:*:read:exit

โพรบที่ออกของการเรียกของระบบ **read** เครื่องหมายดอกจันบ่งชี้ว่าการเรียกระบบ **read** สำหรับกระบวนการทั้งหมดจะถูกโพรบ

@@interval:*:clock:500

โพรบที่เรียกใช้ทุกๆ 500 มิลลิวินาที (เวลานาฬิการติดผนัง) เครื่องหมายดอกจัน คือ placeholder สำหรับการสนับสนุนจุดโพรบในอนาคต

ID กระบวนการสำหรับกระบวนการที่ไม่รู้จักในเวลาที่เขียนสคริปต์ Vue Vue จัดเตรียมตัวอย่างเมธอดเพื่อหลีกเลี่ยงการฮาร์ดโค้ด ID กระบวนการ ในฟิลด์ที่สองของข้อกำหนดคุณสมบัติโพรบ หรือที่ใดๆ ในสคริปต์ Vue (ตัวอย่างเช่น ในส่วนของเพรดิเคต)

สคริปต์ Vue เดี่ยวสามารถมีจุดโพรบจากหลายกระบวนการ ในพื้นที่ของผู้ใช้และในเคอร์เนล เอาต์พุตการ trace ใดๆ ที่สร้างขึ้นจะถูกแสดงตามลำดับเวลา

นอกจากจุดโพรบปกติที่ได้กำหนดโดยตัวจัดการโพรบแล้ว Vue สนับสนุนจุดโพรบพิเศษสองจุด แต่ละสคริปต์ Vue สามารถมีจุดโพรบ @@BEGIN เพื่อบ่งชี้การดำเนินการใดๆ ที่ต้องการออกคำสั่งก่อนที่จะเปิดใช้งานโพรบใดๆ และจุดโพรบ @@END ที่บ่งชี้การดำเนินการใดๆ ที่ต้องการออกคำสั่ง หลังจากยกเลิกการ trace แล้ว

บล็อกการดำเนินการ

บล็อกการดำเนินการระบุการดำเนินการ trace ที่ต้องดำเนินการ เมื่อจุดโพรบที่เชื่อมโยงถูกทริกเกอร์ การดำเนินการที่สนับสนุนไม่ได้จำกัดการดักจับ และการจัดรูปแบบข้อมูล trace แต่เต็มไปด้วยกำลังของภาษา Vue ที่สามารถใช้ได้

บล็อกการดำเนินการใน Vue จะคล้ายคลึงกับโพรซีเดอร์ในภาษาเชิงโพรซีเดอร์ ซึ่งประกอบด้วยลำดับของคำสั่ง ที่ออกไว้ตามลำดับ การไหลของการประมวลผลจะเป็นไปตามลำดับ ข้อยกเว้นเดียวคือการเรียกทำงานแบบมีเงื่อนไข สามารถใช้คำสั่ง "if-else" และการควบคุมสามารถถูกส่งคืนจาก บล็อกการดำเนินการโดยใช้คำสั้ง "return" Vue ยังสนับสนุนฟังก์ชัน **exit** ที่ยกเลิกสคริปต์ทั้งหมด และจุดสิ้นสุดของเซสชันการ trace ไม่มี การสร้าวสำหรับการวนลูปใน Vue และคำสั่งภาษา C ดังนั้นจึงไม่สนับสนุน "for", "do", "goto" และอื่นๆ

ไม่เหมือนกับโพรซีเดอร์ในภาษาเชิงโพรซีเดอร์ บล็อกการดำเนินการใน Vue ไม่มีเอาต์พุตหรือค่าสำหรับส่งคืน และยังไม่มีการสืบทอดที่สนับสนุน สำหรับชุดของพารามิเตอร์อินพุต หรืออีกนัยหนึ่ง ข้อมูลบริบทที่จุดที่โพรบใช้เป็นทางเข้าสามารถเข้าถึงได้ภายใน บล็อกการดำเนินการ ตัวอย่างเช่น พารามิเตอร์ที่ส่งผ่านไปยังฟังก์ชัน สามารถอ้างอิงได้ภายในบล็อกการดำเนินการของ Vue clause หากจุดโพรบอยู่ที่ entry point ของฟังก์ชัน

เพรดิเคต

คุณต้องใช้เพรดิเคตเมื่อการประมวลผล clause ที่จุดโพรบ ต้องถูกดำเนินการตามเงื่อนไข ส่วนของเพรดิเคตจะถูกระบุไว้โดยการมีอยู่ของคีย์เวิร์ด **when** ในทันที หลังส่วนของข้อกำหนดคุณสมบัติโพรบ ด้วยตัวของเพรดิเคตเองประกอบด้วยนิพจน์แบบมีเงื่อนไข C-style ซึ่งครอบด้วยเครื่องหมายวงเล็บ

เพรดิเคตมีรูปแบบต่อไปนี้:

```
   when ( <condition> )
```

ตัวอย่างเช่น:

```
when ( __pid == 1678 )
```

Vue script example

สคริปต์ต่อไปนี้คือตัวอย่างของสคริปต์ Vue:

```
/* Global variables are auto-initialized to zero */ [1]

 int count; /[2]
 /*
  * File: count.e
  *
  * Count number of times the read or write system call is entered
  * by process with Id 400
  */

 @@BEGIN
 {
  printf("Start probing\n");
 }

 @@syscall:*:read:entry, @@syscall:*:write:entry [3]
  when (__pid == 400)[4]

 {[5]

  count++;
  /* Print a message for every 20 system calls */
  if (count % 20 == 0)
   printf("Total read/writes so far: %d\n", count);
  /* Exit when we exceed 100 system calls */
  if (count > 100)
   exit();
 } [6]

 /* print some statistics at exit */
 @@END
 {
  printf("Terminating probe after %d system calls.\n", count);
 }
```

ตัวยกต่อไปนี้ที่ใช้ในตัวอย่างข้างต้นระบุอิลิเมนต์ที่แตกต่างกันกับสคริปต์ Vue :

1. ความคิดเห็น
2. (เผื่อเลือก) ส่วนที่มีการประกาศ
3. ข้อกำหนดคุณสมบัติโพรบ
4. (เผื่อเลือก) เพรดิเคต
5. จุดเริ่มต้นของบล็อกการดำเนินการ
6. จุดสิ้นสุดของบล็อกการดำเนินการ

คุณสามารถเริ่มต้นสคริปต์ง่ายๆ นี้ได้โดยออกคำสั่งต่อไปนี้ หมายเหตุ ตัวอย่างนี้จะแสดงเอาต์พุตตัวอย่างบางตัว

```
# probevue count.e
 Total read/writes so far: 20
 Total read/writes so far: 40
 Total read/writes so far: 60
 ...
 ...
```

การรันคำสั่ง **probevue** ต้องการ privilege หากต้องการออกคำสั่งข้างต้นให้เป็นผลสำเร็จ คุณต้องล็อกอินในฐานะเป็น superuser หรือได้รับสิทธิ privilege ในการเรียกระบบโพรบ ที่สร้างโดยกระบวนการใดๆ ในระบบ

อิลิเมนต์ของภาษา

ส่วนนี้นิยามถึงไวยากรณ์พื้นฐานของภาษา Vue และอิลิเมนต์ของภาษาทั่วไปสำหรับสคริปต์ Vue ต่างๆ

หัวข้อนี้มีหัวข้อย่อยต่อไปนี้:

- ตัวแปร
- คลาสตัวแปร
- ตัวแปรคลาสแบบอัตโนมัติ
- ตัวแปรคลาส เธรดโลคัล
- ตัวแปรคลาสโกลบอล
- ตัวแปรเคอร์เนลโกลบอลคลาส
- ตัวแปรคลาส entry
- ตัวแปรคลาส exit
- ตัวแปรคลาสในตัว
- การกำหนดค่าและชนิด
- ตัวแปรภายนอก
- ตัวแปรสคริปต์
- การกำหนดชนิดโดยนัยสำหรับชนิดกลุ่ม
- การกำหนดชนิดโดยในสำหรับชนิดสตริง
- การกำหนดชนิดโดยนัยสำหรับชนิดลิสต์
- ประโยชน์ของตัวแปรเคอร์เนล
- ชนิดตัวแปรที่มีขนาดคงที่
- ชนิดตัวแปรที่มีขนาดที่ผันแปร
- ชนิดข้อมูลใน Vue
- ชนิดข้อมูลที่มาจากภาษา C
- ชนิดข้อมูล Range และ bucket
- ชนิดการติดตามสแต็ก
- การพิมพ์ตัวแปรชนิดการติดตามสแต็ก
- ชนิดสตริง

- ชนิดลิสต์
- ชนิดของอาร์เรย์ที่เชื่อมโยง
- ชนิดข้อมูลการประทับเวลา
- ชนิดข้อมูลพาธไฟล์
- ชนิดข้อมูล MAC address
- ชนิดข้อมูล IP แอดเดรส
- ชนิดข้อมูล Net info
- ฟังก์ชัน Vue
- ฟังก์ชันคุณลักษณะอื่นๆ
- เพรดิเคต
- ค่าคงที่สัญลักษณ์
- อิลิเมนต์ shell ที่สนับสนุน
- ตัวช่วยดักจับการติดตาม
- การ trace ชั่วคราว

ตัวแปร

ภาษา Vue สนับสนุนชนิดข้อมูลสำหรับภาษา C แบบดั้งเดิมเป็นส่วนใหญ่ กล่าวคือ ภาษาเหล่านั้นสามารถจดจำได้โดยข้อกำหนดคุณลักษณะแบบ C-89 นอกจากนี้ Vue ยังสอดแทรกส่วนขยายบางอย่างเพื่อทำให้ trace โปรแกรมแบบไดนามิกที่เขียนแบบง่ายๆ ได้อย่างมีประสิทธิภาพ

Vue สนับสนุนตัวแปรที่มีกฎสำหรับขอบเขตที่แตกต่างกันสามข้อ:
- ตัวแปรที่เป็นตัวแปรบนโลคัลซึ่งดำเนินการเพียงหนึ่งบล็อกเท่านั้น
- ตัวแปรที่มีขอบเขตแบบโกลบอล
- ตัวแปรที่มีขอบเขตแบบเธรดโลคัล

นอกจากนี้ Vue ยังสามารถเข้าถึงตัวแปรด้วยขอบเขตภายนอก เช่นเดียวกับตัวแปรแบบโกลบอลที่อยู่ในเคอร์เนล หรือข้อมูลผู้ใช้ที่อยู่ในแอ็พพลิเคชัน ที่กำลังถูกโพรบ

โดยทั่วไปแล้ว ตัวแปรจำเป็นต้องประกาศไว้ก่อนที่จะใช้ในสคริปต์เป็นครั้งแรก แม้ว่า Vue ยังคงสนับสนุนรูปแบบที่มีข้อจำกัดสำหรับการจดจำชนิดทางอ้อม คำสั่งในการประกาศตัวแปรที่อยู่ภายในบล็อกการดำเนินการ ต้องปรากฏอยู่ก่อนคำสั่งที่สามารถเรียกทำงานใดๆ ซึ่งไม่สามารถอยู่ภายในบล็อกที่ซ้อนกันซึ่งดูคล้ายกับอยู่ภายคำสั่ง if ในบางกรณี คุณสามารถประกาศตัวแปรภายนอกบล็อกการดำเนินการใดๆ ได้ แต่ในกรณีนี้ การประกาศทั้งหมดต้องปรากฏอยู่ก่อนบล็อกการดำเนินการแรก

คลาสตัวแปร

Vue สนับสนุนคลาสต่างๆ ของตัวแปรที่มีการเปลี่ยนกฎระเบียบในขอบเขตที่คลาสเหล่านั้นได้กำหนดค่าเริ่มต้นไว้ ไม่ว่าคลาสเหล่านั้นสามารถอัพเดตได้หรือไม่ก็ตาม ซึ่งรวมถึงวิธีการพิจารณาชนิดด้วยเช่นกัน เนื่องจากในภาษา C declaration statement ใดๆ สำหรับตัวแปรต้องอยู่นำหน้าการใช้ส่วนแรก ในสคริปต์

Vue จัดเตรียมตัว qualifier ชนิดพิเศษที่ถูกเพิ่มลงใน declaration statement เพื่อบ่งชี้คลาสของตัวแปรที่ต้องการประกาศไว้ ตัวอย่างเช่น คีย์เวิร์ด **__global** เป็น qualifier คลาสที่คุณสามารถ รวมในคำสั่งการประกาศเพื่อระบุว่าตัวแปร ที่ต้องการประกาศมีคลาส "global"

ในตัวอย่างต่อไปนี้ ทั้ง **foo** และ **bar** จะถูกประกาศเป็นตัวแปรของคลาสโกลบอล:

```
__global int foo, bar;
```

Vue ยังสนับสนุนการจดจำชนิดของตัวแปรที่แน่นอน ซึ่งอ้างอิงตามการใช้ครั้งแรกในสคริปต์ ในกรณีนี้จะไม่มี declaration statement แต่คลาสของตัวแปรยังคงสามารถจัดการได้โดยพ่วงต่อ qualifier คลาสกับตัวแปรที่ตามด้วยการอ้างอิงข้อความแรก ในสคริปต์:

```
global:count = 5; /* First reference to variable count in the script */
```

จากตัวอย่างก่อนหน้านี้ คีย์เวิร์ด **global:** คือ qualifier ที่ระบุตัวแปร **count** ให้เป็นตัวแปรคลาสโกลบอล ตัวแปรนี้จะยังคงถูกกำหนดชนิด int เนื่องจากการอ้างอิงอันดับแรกคือ การกำหนดนิพจน์ที่มีเลขจำนวนเต็มที่เป็นค่าคงที่อยู่ทางด้านขวา

หมายเหตุ: คุณจำเป็นต้องใช้คีย์เวิร์ด **__global** ขณะที่ระบุ qualifier คลาสด้วย declaration statement แต่คีย์เวิร์ด **global:** เมื่อนิยามไว้ที่การใช้ตัวแปรอันดับแรกในสคริปต์ กฎไวยากรณ์จะคล้ายคลึงกับ qualifier คลาสอื่นที่สนับสนุนโดย Vue

ตัวแปรคลาสแบบอัตโนมัติ

ตัวแปรอัตโนมัติคือ ตัวแปรที่ระบุเฉพาะ clause และเป็นตัวแปรที่คล้ายคลึงกับตัวแปรแบบอัตโนมัติ หรือตัวแปรสแต็กใน C ซึ่งมีขอบเขตอยู่ภายในบล็อกการดำเนินการในส่วนของ clause เท่านั้นโดยที่ตัวแปรนั้นจะถูกนิยามหรือถูกใช้ และถูกสร้างขึ้นใหม่สำหรับการเรียกใช้บล็อกการดำเนินการในแต่ละครั้ง ตัวแปรแบบอัตโนมัติ ยังไม่ได้นิยามไว้ที่จุดเริ่มต้นของบล็อกการดำเนินการ และต้องถูกกำหนดค่าเริ่มต้นผ่านข้อความสั่งกำหนดค่า ก่อนที่คุณสามารถใช้ตัวแปรเหล่านั้นได้ในนิพจน์ หรือในคำสั่งที่สามารถเรียกใช้งานอื่นใดได้

ตัวแปรแบบอัตโนมัติจะถูกระบุโดยใช้คำนำหน้า **auto:** ตัวอย่างเช่น `auto:lticks` บ่งชี้ถึงตัวแปรแบบอัตโนมัติ คุณยังสามารถประกาศตัวแปรแบบอัตโนมัติโดยใช้ **__auto** declaration ซึ่งคำนำหน้า **auto:** สามารถละเว้นได้

คุณไม่สามารถใช้ตัวแปรคลาสแบบอัตโนมัติในส่วนเพรดิเคตของ Vue clause ได้

สคริปต์ต่อไปนี้คือตัวอย่างของ **__auto** declaration statement:

```
__auto int i;      /* Explicit declaration */
auto:j = 0;        /* Implicit declaration */
```

ตัวแปรคลาส เธรดโลคัล

ตัวแปรเธรดโลคัลจะถูกแสดงเป็นตัวอย่างต่อเธรดที่ trace แล้ว แล้วในครั้งแรกที่ออกคำสั่งให้บล็อกการดำเนินการกำหนดค่าให้กับตัวแปร หากสร้างขึ้นแล้ว ตัวแปรเธรดโลคัลจะมีอยู่ตราบเท่าที่สคริปต์ Vue ยังคงแอ็คทีฟ และเธรดที่ trace แล้วจะไม่มีอยู่อีกต่อไป ค่าของตัวแปรเธรดโลคัลคือ ค่าที่ระบุเฉพาะสำหรับเธรด และยังคงไว้ในระหว่างการประมวลผล clause ของโปรแกรมเดียวกัน หรืออีกนัยหนึ่ง ตัวแปรของคลาสสามารถมองเห็นได้ทุกที่ภายในสคริปต์ Vue อย่างไรก็ตาม เธรดแต่ละตัวที่ออกคำสั่งสคริปต์ Vue จะขอรับสำเนาของตัวแปรเหล่านี้ที่ตนเองเป็นเจ้าของ และตัวแปรในแต่ละสำเนาจะสามารถเข้าถึงได้และแก้ไขได้ทุกที่ภายในสคริปต์ ด้วยเธรดที่ถูกแสดงเป็นตัวอย่างเท่านั้น

ตัวแปรเธรดโลคัลยังถูกแบ่งแยกได้โดยใช้คำนำหน้า **thread:** ตัวอย่างเช่น `thread:count` บ่งชี้ถึงตัวแปรเธรดโลคัล คุณยังสามารถประกาศตัวแปรเธรดโลคัลได้โดยใช้ **__thread** declaration ซึ่งคำนำหน้า **thread:** สามารถละเว้นได้ด้วยข้อยกเว้นหนึ่งข้อต่อไปนี้

คุณสามารถใช้ตัวแปรเธรดโลคัลในส่วนเพรดิเคตของ Vue ก่อนที่จะถูกแสดงเป็นตัวอย่าง เพรดิเคต ที่มีตัวแปรเธรดโลคัลที่ไม่ได้ถูกแสดงเป็นตัวอย่างจะถูกประเมินผลให้มีค่า FALSE เสมอ เมื่อใช้ในส่วนเพรดิเคต คำนำหน้า **thread:** ต้องถูกสอดแทรกไว้เสมอเพื่อระบุให้เป็นตัวแปรเธรดโลคัล

สคริปต์ต่อไปนี้คือตัวอย่างของ **__thread** declaration statement:

```
__thread int i;      /* Explicit declaration */
thread:j = 0;        /* Implicit declaration */
```

หมายเหตุ: แม้ว่าคุณจะสามารถประกาศเธรดโลคัลภายในโพรบ **@@BEGIN** และ **@@END** แล้ว การอ้างอิงอื่นใดถึงเธรดโลคัลเหล่านั้นในโพรบพิเศษเหล่านี้ สามารถสร้างลักษณะการทำงานที่ไม่ได้นิยามไว้ declaration statement ไม่ได้เป็นสาเหตุทำให้ตัวแปรเธรดโลคัลถูกแสดงเป็นตัวอย่าง

ตัวแปรคลาสโกลบอล

ตัวแปรของคลาสโกลบอลมีขอบเขตแบบโกลบอล และสามารถมองเห็นได้ทุกที่ภายในสคริปต์ Vue คุณสามารถใช้ตัวแปรโกลบอลใน clause ของสคริปต์ Vue ตั้งแต่หนึ่ง clause ได้ ซึ่งยังสามารถประกาศได้ที่จุดเริ่มต้นของข้อความก่อน clause ตัวแรก ตัวแปรโกลบอลจะถูกกำหนดค่าเริ่มต้นให้เป็นศูนย์หรือ NULL ได้ตามความเหมาะสม

ตัวแปรทั้งหมดในสคริปต์ Vue คือคลาสโกลบอลที่กำหนดไว้ตามค่าดีฟอลต์ ยกเว้นว่า ตัวระบุคลาสที่ไม่ใช่โกลบอลจะถูกนำหน้าการประกาศ ซึ่งคุณยังสามารถประกาศตัวแปรโกลบอลได้โดยใช้ตัวระบุคลาส **__global** ขณะที่กำลังประกาศตัวแปร ตัวแปรลิสต์จะถูกสร้างเป็นตัวแปรของคลาสโกลบอลเสมอ ตามนิยาม

การอ่านและอัพเดตตัวแปรโกลบอลไม่ได้ถูก serialized ยกเว้นแต่ตัวแปรนั้นมีชนิดเป็นลิสต์ ซึ่งไม่มีการรับประกันใดๆ เกี่ยวกับชนิดข้อมูลเมื่อใช้โพรบพร้อมกัน ตัวแปรโกลบอลที่ไม่ได้มีชนิดเป็นลิสต์ จะมีประโยชน์สำหรับการเก็บรวบรวมการทำโปรไฟล์และการทำสถิติอื่นๆ

คุณสามารถใช้ตัวแปรโกลบอลในส่วนเพรดิเคตของ Vue clause ได้

สคริปต์ต่อไปนี้คือตัวอย่างสำหรับการกำหนดค่าเริ่มต้น เริ่มต้น และการใช้ตัวแปรโกลบอล:

```
int wcount;   /* Global variable declared before first clause */

@@BEGIN
{
 int f_count;  /* Global variable declared inside @@BEGIN */
 __global int z_count; /* Global variable declared with __global prefix */

 f_count = 12;
}

@@syscall:*:read:entry
 when (z_count == 0)
{
 int m_count;  /* Global variable declared inside a probe */
 m_count += f_count; /* f_count already declared in earlier probe */
```

```
 printf("m_count = %d\n", m_count);
 if (wcount == 1)
  exit();
}


@@syscall:*:write:entry
{
 m_count++;  /* m_count already declared in earlier probe */

}

@@syscall:*:write:exit
{
 wcount = 1;  /* w_count declared globally */
}
```

ตัวแปรเคอร์เนลโกลบอลคลาส

ใน ProbeVue ผู้ใช้ที่มีอภิสิทธิ์สามารถเข้าถึงตัวแปรโกลบอลเคอร์เนล ภายในบล็อกการดำเนินการของ Vue clause สำหรับจุดโพรบในพื้นที่สำหรับผู้ใช้ เช่น จุดโพรบ uft ก่อนการใช้ หรืออ้างถึงตัวแปรเคอร์เนลในสคริปต์ Vue คุณต้องประกาศตัวแปรเคอร์เนลโดยใช้ **__kernel** declaration statement เฉพาะตัวแปรที่เอ็กซ์พอร์ตโดยเคอร์เนลเท่านั้น นั่นคือ ตัวแปรที่แสดงอยู่ในรายการเอ็กซ์พอร์ตของ **/unix** ที่สามารถเข้าถึงได้

คุณสามารถเข้าถึงตัวแปรเคอร์เนลชนิดที่ใช้ร่วมกัน และตัวแปรเคอร์เนลที่เป็นโครงสร้างหรือ unions และตัวชี้ได้ นอกจากนี้ คุณยังสามารถอ้างถึงชื่อสมาชิกของโครงสร้างเคอร์เนลและ unions ในสคริปต์ Vue ได้เช่นกัน เคอร์เนลอาร์เรย์ยังสามารถเข้าถึงได้ แต่ไม่สนับสนุนการคัดลอกข้อมูลอักขระเคอร์เนลลงในสตริง ProbeVue

เข้าถึงตัวแปรเคอร์เนลที่ตรึงไว้ ถ้าเพจมีตัวแปรเคอร์เนลที่ไม่ได้อยู่ในหน่วยความจำ (ลบออกไปแล้ว) ProbeVue จะส่งคืนค่าศูนย์สำหรับตัวแปรนั้น

สำหรับตัวอย่างของวิธีที่ตัวแปรเคอร์เนลสามารถประกาศและใช้ในสคริปต์ Vue

ตัวแปรเคอร์เนลไม่สามารถปรากฏในส่วนของเพรดิเคตของ clause ตัวแปรเคอร์เนลถูกใช้เป็นตัวแปรแบบอ่านอย่างเดียวในสคริปต์ Vue ความพยายามใดๆ ที่เขียนไปยังตัวแปรเคอร์เนลเป็นสาเหตุทำให้เกิดข้อผิดพลาดทางไวยากรณ์ หรือล้มเหลวในภายหลังด้วยสคริปต์ข้อความที่ยกเลิก

ตัวแปรคลาส entry

clauses ที่เชื่อมโยงกับจุดโพรบที่อยู่ในตำแหน่ง entry ซึ่งชี้การเรียกระบบ หรือฟังก์ชันผู้ใช้สามารถเข้าถึงอาร์กิวเมนต์ ที่ส่งผ่านไปยังการเรียกของระบบหรือฟังก์ชันที่โพรบ

โพรบที่จุดตำแหน่ง entry จะได้รับการสนับสนุนโดยการเรียกระบบ และตัวจัดการโพรบการติดตามฟังก์ชันของผู้ใช้ ตัวอย่าง เช่น การเรียกระบบการอ่านจะใช้อาร์กิวเมนต์สามตัว: file descriptor ID ตัวชี้ไปยังบัฟเฟอร์ผู้ใช้ และค่าสำหรับจำนวนไบต์ของข้อมูลที่ต้องการ อ่าน ค่าของอาร์กิวเมนต์สามตัวนี้สามารถเข้าถึงได้ ถ้าข้อกำหนดคุณสมบัติโพรบคือ **@@syscall:*:read:entry** ซึ่งระบุโพรบที่ entry point ของการเรียกระบบการอ่าน

พารามิเตอร์ในฟังก์ชันจะอ้างถึงการใช้ชื่อตัวแปรคลาส entry ในตัวพิเศษ **__arg1**, **__arg2**, **__arg3**, ... จนถึงจำนวนของอาร์กิวเมนต์ที่ส่งผ่านไปยังฟังก์ชัน ตัวอย่างเช่น ใน clause ที่เชื่อมโยงกับ entry point ของการเรียกระบบการอ่าน **__arg1** อ้างถึงค่าของพารามิเตอร์ file descriptor id **__arg2** อ้างถึงค่าของพารามิเตอร์ตัวชี้บัฟเฟอร์ และ **__arg3** คือขนาดของข้อมูลที่ต้องการอ่าน

หมายเหตุ: เมื่อระบุจุดโพรบ tuples หนึ่งตัวหรือมากกว่า ดังนั้นจะไม่อนุญาตให้ใช้ตัวแปร __arg <x> ใน Action Block และจะทำให้เกิดข้อผิดพลาดดังแสดงในตัวอย่าง ด้านล่าง

```
@@syscall:*:read:entry, @@syscall:*:write:entry
{
        char *argument;
        argument=__arg2;  -> Not Allowed.
}
```

Probevue จะจบการทำงานพร้อมข้อผิดพลาดต่อไปนี้: *arg builtin cannot be used No defined function*

การใช้ตัวแปลคลาส entry ใน Vue จะถูกต้อง หากการประกาศของฟังก์ชันในลักษณะของภาษา C ที่จะโพรบ โดยเฉพาะอย่างยิ่งชนิดข้อมูลของพารามิเตอร์ที่ต้องการส่งไปยังฟังก์ชัน จะถูกจัดเตรียมไว้ในสคริปต์ Vue ซึ่งต้องปรากฏขึ้นเป็นข้อความ ก่อน Vue clause แรกที่อ้างอิงถึง entry clause ให้วางการประกาศเป็นข้อความก่อน Vue clause ใดๆ ที่ด้านบนของสคริปต์ Vue

สคริปต์ต่อไปนี้คือตัวอย่างการใช้ตัวแปรคลาส entry:

```
int read(int fd, char *buf, unsigned long size);

@@syscall:*:read:entry
{
 printf("Number of bytes to read = %d\n", __arg3);
}
```

หมายเหตุ: ในตัวอย่างก่อนหน้านี้ นิยามของฟังก์ชันการเรียกระบบการอ่าน ที่ระบุในสคริปต์ไม่ตรงกับที่กำหนดไว้ในไฟล์ **/usr/include/unistd.h** แต่ระบบทำงานได้เช่นกัน

ข้อกำหนดที่สองคือ ข้อกำหนดคุณสมบัติโพรบที่เชื่อมโยงกับ clause ที่ระบุจุดโพรบเฉพาะ ตัวแปรคลาส entry ไม่สามารถใช้ได้ใน Vue clause ที่มีจุดโพรบจำนวนมาก ที่ระบุในข้อกำหนดคุณสมบัติโพรบโดยไม่คำนึงถึงฟังก์ชันที่โพรบ ซึ่งเป็นฟังก์ชันเดียวกันหรือคล้ายกับฟังก์ชันต้นฉบับ สคริปต์ต่อไปนี้ คือสคริปต์ที่ผิดกฎเกณฑ์ และจะเป็นสาเหตุที่ทำให้คอมไพลเลอร์ ProbeVue เกิดความล้มเหลวด้วยข้อผิดพลาดทางไวยากรณ์ เนื่องจากข้อกำหนดคุณสมบัติโพรบ ประกอบด้วยจุดโพรบสองจุด:

```
int read(int fd, char *buf, unsigned long size);
int write(int fd, char *buf, unsigned long size);

@@syscall:*:read:entry, @@syscall:*:write:entry
{
 /* Cannot use __arg3 in here, as this clause has multiple probe
  * points associated with it. This script will fail with a
  * syntax error in the compilation phase of the probevue command.
  */
 printf("Number of bytes to read/write = %d\n", __arg3);
}
```

สคริปต์ที่แก้ไขแล้วต่อไปนี้จะสามารถทำงานได้:

```
int read(int fd, char *buf, unsigned long size);
int write(int fd, char *buf, unsigned long size);

@@syscall:*:read:entry
{
 printf("Number of bytes to read = %d\n", __arg3);
}
@@syscall:*:write:entry
{
 printf("Number of bytes to write = %d\n", __arg3);
}
```

ตัวแปรคลาส exit

คลาสที่เชื่อมโยงกับจุดโพรบที่อยู่ที่จุดตำแหน่งทางออกของการเรียกระบบ หรือฟังก์ชันของผู้ใช้สามารถเข้าถึงค่าส่งคืนของการเรียกของระบบ หรือฟังก์ชันของผู้ใช้

ซึ่งมีเพียงตัวแปรคลาส exit เพียงตัวเดียวเท่านั้นที่ถูกนิยามโดยภาษา Vue นี่ค่าค่าส่งคืนจากฟังก์ชันหรือการเรียกของระบบที่สามารถเข้าถึงได้โดยใช้ชื่อตัวแปรในตัวพิเศษ **__rv**

โพรบที่จุดตำแหน่งทางออกจะได้รับการสนับสนุนโดยตัวจัดการโพรบของ การเรียกของระบบ ตัวอย่างเช่น การเรียกของระบบการอ่านส่งคืนจำนวนไบต์ที่อ่านได้จริง หรือโค้ดสำหรับส่งคืนข้อผิดพลาด –1 ค่าที่ส่งคืนนี้ สามารถเข้าถึงได้ที่จุดโพรบ **@@syscall:*:read:exit** ซึ่งระบุจุดทางออกทั้งหมดจากการเรียกของระบบการอ่าน

เช่นเดียวกับตัวแปรคลาส entry การใช้ตัวแปรคลาส exit ใน Vue จะถูกต้องหากข้อกำหนดคุณสมบัติที่เชื่อมโยงกับ clause ที่ระบุจุดโพรบเฉพาะ ดังนั้น **__rv** จึงไม่สามารถใช้ได้ใน Vue ที่มีจุดโพรบจำนวนมาก ที่ระบุในข้อกำหนดคุณสมบัติโพรบ นอกจากนี้ การประกาศของฟังก์ชันในลักษณะของภาษา C ที่จะโพรบ โดยเฉพาะอย่างยิ่งชนิดข้อมูลของค่าส่งคืน ต้องถูกจัดเตรียมไว้ในสคริปต์ Vue ในความเป็นจริงแล้ว ค่านี้เป็นข้อผิดพลาดที่ระบุการประกาศฟังก์ชันโดยไม่ได้เตรียมชนิดของการส่งคืน

คุณสามารถใช้ตัวแปรคลาส exit ในส่วนของเพรดิเคตของ clause ได้

สคริปต์ตัวอย่างต่อไปนี้คือสคริปต์ที่ผิดกฎเกณฑ์ และจะเป็นสาเหตุทำให้คอมไพลเลอร์ ProbeVue เกิดความล้มเหลวด้วยข้อผิดพลาดทางไวยากรณ์ เนื่องจากชนิดที่ส่งคืนของฟังก์ชัน **read** ไม่ได้ระบุไว้:

```
/* Bad example.  */

 int read(int fd, char *buf, unsigned long size);

 @@syscall:*:read:exit
  when (__rv > 0)
 {
  /* Entered on read success: return value = # of bytes read */
  printf("Number of bytes read = %d\n", __rv);
 }
```

สคริปต์ที่แก้ไขแล้วต่อไปนี้จะสามารถทำงานได้:

```
/* Good example.  */

 int read(int fd, char *buf, unsigned long size);
```

```
@@syscall:*:read:exit
 when (__rv > 0)
{
 /* Entered on read success: return value = # of bytes read */
 printf("Number of bytes read = %d\n", __rv);
}
```

ตัวแปรคลาสในตัว

นอกจากตัวแปรคลาสในตัวแบบพิเศษแล้ว **__arg1** จนถึง **__arg32** และ **__rv**, Vue ยังกำหนดชุดของตัวแปรในตัวที่มีวัตถุประสงค์ทั่วไป ตัวแปร built-in ที่มีวัตถุประสงค์ทั่วไปเหล่านี้ อธิบายรายละเอียดเพิ่มเติมไว้ในส่วนนี้ และตัวแปร built-in เฉพาะ probe manager บางตัวอธิบายไว้ในส่วนของ probe manager ดังกล่าว ตัวแปรคลาสบิวด์อินคือฟังก์ชัน แต่ไม่ได้ใช้เป็นตัวแปรโดย ProbeVue ดังนั้น คุณจึงสามารถใช้ตัวแปรบิวด์อินในส่วนของเพรดิเคตของ Vue clause

ตัวแปรบิวด์อินต่อไปนี้ได้รับการสนับสนุนใน Vue:

`__tid`
: ID เธรดของเธรดที่ติดตามแล้ว

`__pid`
: ID การประมวลผลของเธรดที่ติดตามแล้ว

`__ppid`
: ID การประมวลผลหลักของเธรดที่ติดตามแล้ว

`__pgid`
: ID กลุ่มการประมวลผลของเธรดที่ติดตารมแล้ว

`__pname`
: ชื่อการประมวลผลของเธรดที่ติดตามแล้ว

`__uid, __euid`
: ID ผู้ใช้จริงและ ID ผู้ใช้ที่มีผลบังคับใช้ของเธรดที่ติดตามแล้ว

`__trcid`
: ID การประมวลผลของการประมวลผลที่กำลังติดตาม (นั่นคือ คำสั่ง **probevue**)

`__errno`
: ค่า errno ปัจจุบันสำหรับเธรดที่ติดตามแล้ว

`__kernelmode`
: โหมดที่สามารถเรียกทำงานได้ในปัจจุบัน: 1 (ในโหมดเคอร์เนล) หรือ 0 (ในโหมดผู้ใช้)

`__r3, ..., __r10`
: ค่ารีจิสเตอร์สำหรับวัตถุประสงค์ทั่วไป (สำหรับพารามิเตอร์ฟังก์ชันหรือค่าส่งคืน)

`__curthread`
: เธรดปัจจุบัน

`__curproc`
: กระบวนการปัจจุบัน

__ublock
พื้นที่ผู้ใช้ของกระบวนการปัจจุบัน

__mst
ตัวแปรแบบ Built-in เพื่อเข้าถึงเนื้อหาการลงทะเบียนฮาร์ดแวร์ของพื้นที่ machine state save (MST) ของเธรดปัจจุบัน'

__stat
ตัวแปร Built-in เพื่อจัดเตรียมการเข้าถึงสถิติระบบสำหรับคอมโพเนนต์เคอร์เนล AIX® ต่างๆ

ตัวอย่างสคริปต์ต่อไปนี้คือตัวอย่างการใช้ตัวแปรแบบในตัว:

```
@@syscall:*:read:entry
{
 printf("Thread ID:%d, Process ID:%d, Parent Process ID:%d\n",
   __tid, __pid, __ppid);
 printf("Process Group ID: %d\n", __pgid);
 printf("Process name = %s\n", __pname);

 printf("Real UID=%d, Effective UID=%d\n", __uid, __euid);S

 printf("probevue command process ID = %d\n", __trcid);

 printf("Errno = %d\n", __errno);
 printf("Mode = %s\n", __kernelmode == 1 ? "kernel" : "user");

 printf("Current values of GPRs: r3=0x%016llx, r4=0x%016llx, r5=0x%016llx\n",
   __r3, __r4, __r5);
 printf("                        r6=0x%016llx, r7=0x%016llx, r8=0x%016llx\n",
   __r6, __r7, __r8);
 printf("                        r9=0x%016llx, r10=0x%016llx\n",
   __r9, __r10);
}
```

ตัวแปร __curthread แบบ built-in

__curthread เป็นการใช้ built-in พิเศษที่ผู้ใช้สามารถเข้าถึงข้อมูลที่สัมพันธ์กับเธรดบางอย่างสำหรับเธรดปัจจุบัน ข้อมูลสามารถเข้าถึงได้โดยใช้ตัวดำเนินการ -> บน **__curthread** แบบ built-in อย่างไรก็ตาม ในตัวนี้ไม่สามารถใช้ในโพรบ **systrace**, **BEGIN** และ **END** นอกจากนี้ยังสามารถใช้ในโพรบ interval เฉพาะถ้ามีการระบุ **PID** โดยทั่วไป built-in นี้จะมีการทำงานที่เหมือนกับ **getthrds/getthrds64** จะจะจำกัดเฉพาะเธรดปัจจุบันเท่านั้น ข้อมูลที่สามารถเข้าถึงคือ

tid
ID ของเธรด

pid
ID โปรเซส

นโยบาย
นโยบายการกำหนดเวลา

pri
ลำดับความสำคัญ

`cpusage`
การใช้งาน CPU

`cpuid`
ตัวประมวลผลที่เธรดปัจจุบันเชื่อมอยู่

`sigmask`
สัญญาณที่บล็อกบนเธรด

`lockcount`
จำนวนของเคอร์เนลล็อกที่ใช้โดยเธรด

`ptid`
ตัวระบุ pthread ของเธรดนี้ (0 หากเป็นเธรดเคอร์เนล 1 หากเป็นแอ็พพลิเคชัน เธรดเดี่ยว)

`homecpu`
โฮม CPU ของเธรด

`homesrad`
โฮม `srad` ของเธรด

ตัวอย่างการใช้งาน

```
Tid ของเธรดปัจจุบันสามารถเข้าถึงได้โดยใช้  __curthread->tid
```

ตัวแปร __curproc แบบ built-in

__curproc เป็นการใช้ built-in พิเศษที่ผู้ใช้สามารถเข้าถึงข้อมูลที่สัมพันธ์กับเธรดบางอย่างสำหรับเธรดปัจจุบัน ข้อมูลสามารถเข้าถึงได้โดยใช้ตัวดำเนินการ **->**บน **__curproc** แบบ built-in อย่างไรก็ตาม ในตัวนี้ไม่สามารถใช้ในโพรบ **systrace**, **BEGIN** และ **END** นอกจากนี้ยังสามารถใช้ในโพรบ interval เฉพาะถ้ามีการระบุ **PID** โดยทั่วไป built-in นี้จะมีการทำงานที่เหมือนกับ **getproc** จะจะจำกัดเฉพาะเธรดปัจจุบันเท่านั้น ข้อมูลที่สามารถเข้าถึงคือ

`pid`
Process ID.

`ppid`
ID ของกระบวนการพาเรนต์

`pgid`
ID กลุ่มกระบวนการ

`uid`
ID ผู้ใช้จริง

`suid`
ID ผู้ใช้ที่ถูกบันทึก

`pri`
ลำดับความสำคัญ

`nice`
ค่า Nice

`cpu`
การใช้งานตัวประมวลผล

`adspace`
พื้นที่แอดเดรสกระบวนการ

`majflt`
I/O Page Fault

`minflt`
Non I/O Page Fault

ขนาด
ขนาดของอิมเมจในเพจ

`sigpend`
สัญญาณที่ค้างอยู่บนกระบวนการ

`sigignore`
สัญญาณที่กระบวนการข้าม

`sigcatch`
สัญญารที่ถูกจับได้โดยกระบวนการ

`forktime`
เวลาการสร้างของกระบวนการ

`threadcount`
จำนวนของเธรดในกระบวนการ

`cwd`
ไดเร็กทอรีการทำงานปัจจุบัน หากไม่มีบริบท page ที่ว่าง หรือ ขนาดสแต็กการคำนวณต่อ CPU น้อยกว่า 96 KB หรือในโพรบที่ไม่อนุญาตให้ใช้ page fault (เช่น โพรบ interval) ดังนั้น built-in นี้จะคืนค่าสตริง null

ตัวอย่างการใช้งาน

```
id กระบวนการพาเรนต์สามารถเข้าถึงได้โดยใช้  __curproc->ppid
```

ตัวแปร __ublock แบบ built-in

__ublock เป็นการใช้ built-in พิเศษที่ผู้ใช้สามารถเข้าถึงข้อมูลที่สัมพันธ์กับเธรดบางอย่างสำหรับเธรดปัจจุบัน อย่างไรก็ตาม ในตัวนี้ไม่สามารถใช้ในโพรบ **systrace, BEGIN** และ **END** นอกจากนี้ยังสามารถใช้ในโพรบ interval เฉพาะถ้ามีการระบุ **PID** ข้อมูลสามารถเข้าถึงได้โดยใช้ตัวดำเนินการ -> บน **__ublock** แบบ built-in ข้อมูลที่สามารถเข้าถึงคือ

ข้อความ
เริ่มต้นของขอความ

`tsize`
ขนาดของข้อความ (ไบต์)

ข้อมูล
เริ่มต้นของข้อมูล

`sdata`
ขนาดของข้อมูลปัจจุบัน (ไบต์)

`mdata`
ขนาดของข้อมูลสูงสุด (ไบต์)

`stack`
เริ่มต้นของสแต็ก

`stkmax`
สแต็กสูงสุด (ไบต์)

`euid`
ID ผู้ใช้แบบ Effective

`uid`
ID ผู้ใช้แบบ Real

`egid`
id กลุ่มแบบ Effective

`gid`
id กลุ่มแบบ Real

`utime`
เวลาการใช้งานรีซอร์สผู้ใช้กระบวนการหน่วยเป็นวินาที

`stime`
เวลาการใช้งานรีซอร์สระบบกระบวนการหน่วยเป็นวินาที

`maxfd`
ค่า Max fd ในผู้ใช้

`is64u`
ตั้งค่าเป็น 1 หากอยู่ในคอนเท็กซ์ของกระบวนการแบบ 64 บิต

ตัวอย่างการใช้งาน

จุดเริ่มต้นของข้อมูลสำหรับกระบวนการปัจจุบันสามารถเข้าถึงได้โดยใช้ `__ublock->text`

ตัวแปร __mst แบบ built-in

__mst เป็นตัวแปรแบบ built-in พิเศษที่คุณสามารถเข้าถึงเนื้อหาการลงทะเบียนฮาร์ดแวร์ของเธรดปัจจุบัน built-in นี้ไม่สามารถใช้ได้ในโพรบ **systrace, BEGIN** และ **END** นอกจากนี้ตัวแปรแบบ built-in นี้สามารถใช้ในโพรบ interval เฉพาะถ้ามีการระบุ **PID** ข้อมูลสามารถเข้าถึงได้โดยใช้ตัวดำเนินการ -> บน **__ublock** แบบ built-in การลงทะเบียนที่สามารถเข้าถึงมีดังต่อไปนี้:

`r1-r10`
การลงทะเบียบทั่วไป r1 ถึง r10

`r14-r31`
การลงทะเบียบทั่วไป r14 ถึง r31

`iar`
การลงทะเบียนแอดเดรสคำสั่ง

`lr` การลงทะเบียนลิงก์

`isisr`
ตั้งค่าหากอยู่ในโหมดอินเตอร์รัปต์หรือคอนเท็กซ์ข้อยกเว้น

ตัวอย่างการใช้งาน

เมื่อต้องการเข้าถึงค่า **lr** ในโพรบ ให้ใช้คำสั่งต่อไปนี้:

```
__mst->lr
```

__stat built-in variable

ตัวแปรที่มีในตัวนี้จัดเตรียมการเข้าถึงสถิติระบบ สำหรับคอมโพเนนต์เคอร์เนล AIX ต่างๆ โดยใช้สคริปต์ Vue สถิติระบบถูกจัดเตรียมไว้เป็นตัวนับที่ทำงานอยู่ที่สามารถ เข้าถึงได้จากจุดโพรบของสคริปต์ ProbeVue ใดๆ จุดโพรบใหม่ไม่ถูกเพิ่มเพื่อสนับสนุนสถิติระบบ เมื่อต้องการเข้าถึงสถิติระบบ คุณต้องมีสิทธิ์การติดตามเคอร์เนล `aix.ras.probevue.trace`

สถิติระบบมีประโยชน์สำหรับเหตุผลต่อไปนี้:

- สถิติสามารถเข้าถึงได้โดยไม่ต้องเปิดใช้งานการติดตามระบบ หรือ การติดตามคอมโพเนนต์
- ฟิลด์ที่จำเป็นเท่านั้นที่จะถูกแสดง ซึ่งไม่สามารถทำได้โดยใช้คำสั่งสถิติปัจจุบัน คุณสามารถหลีกเลี่ยงการคัดลอกข้อมูลจำนวนมาก โดยการเข้าถึงฟิลด์โดยตรงจากโครงสร้างของเคอร์เนล
- ตอนนี้ สถิติระบบพร้อมใช้งานในสคริปต์ Vue เพื่อให้การดำเนินการตรรกะท่างคณิตศาสตร์ทำได้ง่ายขึ้น เช่น คุณสามารถใช้ ProbeVue เพื่อเพิ่มจำนวนของการดำเนินการระหว่างดิสก์สองลูก

Vue ตัวแปร **__stat** ที่มีในตัว อนุญาตให้สามารถดึงสถิติทั้งที่ระดับโกลบอลและระดับ คอมโพเนนต์แต่ละระดับ ข้อมูลถูกจัดเตรียมเมื่อตัวนับรันอยู่และสคริปต์ Vue ต้องการเข้าถึง ตัวนับเป็นระยะๆ คุณสามารถบันทึกค่าที่คุณต้องการ และคำนวณค่าเดลต้าเพื่อให้ได้ผลลัพธ์ที่ต้องการ คุณสามารถเข้าถึง สถิติระบบเนื่องจากเหตุผลต่อไปนี้:

- เมื่อต้องการเขียนเครื่องมือสถิติอย่างง่ายๆ โดยใช้สคริปต์ Vue โดยไม่ต้องเรียกใช้ C/C++ API เพื่อพิมพ์ค่าเดลต้าสำหรับตัวนับทุกวินาที หรือตามช่วงเวลาที่ผู้ใช้ร้องขอ
  - เมื่อต้องการมอนิเตอร์ค่าตัวนับ หากค่าเดลต้าเกินขีดจำกัด เมื่อค่าเกินขีดจำกัด สคริปต์จะบันทึกข้อความ
  - เมื่อต้องการใช้สคริปต์ Vue เพื่อบันทึกข้อความเมื่อค่าจริงของตัวนับ ซึ่งไม่ใช่ค่าเดลต้า เกินค่าขีดจำกัด (เช่น เวลาให้บริการของดิสก์)

มีโหมดที่แตกต่างกันที่จัดเตรียมโดย ตัวแปร **__stat** ที่มีในตัวเพื่อเข้าถึงสถิติจากแหล่งที่มา แหล่งที่มาของสถิติระบบได้มาจากคอมโพเนนต์ระบบ ที่แตกต่างกัน โหมดการเข้าถึงสถิติมีดังต่อไปนี้

การเข้าถึงโหมดซิงโครนัส

ProbeVue จัดเตรียม การเข้าถึงสถิติระบบโดยตรงเมื่อสคริปต์ Vue รัน การเข้าถึงโดยตรง ชนิดนี้เป็นการเข้าถึงข้อมูลสด โดยดีฟอลต์ ProbeVue จะเลือกการเข้าถึงข้อมูลสด หากสามารถทำได้ สถิติทั้งหมดอาจไม่สามารถเข้าถึงได้ ในโหมดนี้ เนื่องจากคอมโพเนนต์ไม่ได้จัดเตรียมการเข้าถึงโดยตรงหรือ การเข้าถึงโดยตรงไม่สามารถทำได้ในเธรดปัจจุบัน

โหมดอะซิงโครนัส หรือโหมดแคช

ข้อมูลถูกรวบรวมจากแหล่งที่มาเป็นระยะๆ และแคชไว้ใน ProbeVue สคริปต์ Vue ใช้แคช เพื่อเข้าถึงข้อมูล ช่วงเวลารีเฟรชแคชถูกปรับที่เซสชันเดียว หรือที่ระดับเซสชันทั้งหมด คอมโพเนนต์ต้นทางทั้งหมดมีวิธี สำหรับการเข้าถึงสถิติในโหมดอะซิงโครนัส ในกรณีดังกล่าว คุณสามารถ จัดเตรียมการเข้าถึงสถิติทั้งหมดในโหมดแคชโดยใช้ **fetch_stats_async_only** ที่ปรับได้โหมดแคชใช้เมื่อคุณไม่สามารถเข้าถึงข้อมูลสดได้ หรือคุณต้องการมอนิเตอร์ระบบที่ไม่ต้องการข้อมูลสด

ตัวอย่าง

1. ตัวอย่างต่อไปนี้จะพิมพ์จำนวนของการถ่ายโอน I/O สำหรับดิสก์ที่ชื่อ hdisk9 ทุก วินาที:

```
@@interval:*:clock:1000
{
    printf("Number of transfers = %lld\n", __stat.io.disk.hdisk9->transfers);
}
```

2. ตัวอย่างต่อไปนี้แสดงหน่วยของเวลาเซอร์วิส เป็นไมโครวินาที:

```
@@syscall:*:read:exit
{
    rdservtime = __stat.io.disk.hdisk10->rd_service_time;
    printf("rdservtime=%lld microseconds\n", rdservtime);
}
```

__stat built-in syntax

ไวยากรณ์ทั่วไปสำหรับการใช้นิพจน์ที่มีในตัว **__stat** เป็นดังต่อไปนี้: `__stat.<level1_keyword>[.<level2_keyword>.....][.<inst1_keyword>....]->fieldname`

ระดับและอินสแตนซ์ที่กำหนดไว้ล่วงหน้าสำหรับ Storage IO Statistics แสดงไว้ใน ตารางต่อไปนี้:

ตารางที่ 1. ระดับและอินสแตนซ์ที่กำหนดไว้ล่วงหน้า

Built-in	ระดับที่ 0	ระดับ 1	อินสแตนซ์ 0	อินสแตนซ์ 1	ชื่อฟิลด์
__stat	io	ดิสก์	hdisk[0...n]		• โปรดดูที่ ตารางที่ 2 ในหน้า 243
			hdisk[0...n]	path[0...n]	• โปรดดูที่ ตารางที่ 3 ในหน้า 244
__stat	io	adapter	vscsi[0....n]		• โปรดดูที่ ตารางที่ 4 ในหน้า 244 • โปรดดูที่ ตารางที่ 5 ในหน้า 245
__stat	io	adapter	fcs[0...n]		• โปรดดูที่ ตารางที่ 6 ในหน้า 245

SCSI Disk I/O statistics

ตารางต่อไปนี้แสดงชื่อฟิลด์ที่สนับสนุนสำหรับสถิติ I/O ของดิสก์ Small Computer System Interface (SCSI) ฟิลด์เหล่านี้เป็นฟิลด์เฉพาะอินสแตนซ์ดิสก์ ซึ่งสามารถเข้าถึง เป็น `__stat.io.disk.<hdisk0...n>->fieldname` สถิติต่อไปนี้สามารถเข้าถึงได้ในโหมดซิงโครนัส และอะซิงโครนัส

ตารางที่ 2. สถิติ I/O ของดิสก์ SCSI

ชื่อฟิลด์สคริปต์ Vue	ชนิดข้อมูล	คำอธิบาย
ชื่อ	String[32]	ชื่อดิสก์
block_size	unsigned long long	ขนาดบล็อกของดิสก์หน่วยเป็นไบต์
transfers	unsigned long long	จำนวนของการถ่ายโอนไปยัง หรือ จากดิสก์
rd_block_count	unsigned long long	จำนวนของการอ่านบล็อกของดิสก์
rd_service_time	unsigned long long	ชื่อเซอร์วิสการอ่านทั้งหมด หรือเวลาเซอร์วิสที่ได้รับหน่วยเป็นไมโครวินาที
rd_min_service_time	unsigned long long	เวลาเซอร์วิสการอ่านที่น้อยที่สุดหน่วยเป็นไมโครวินาที
rd_max_service_time	unsigned long long	เวลาเซอร์วิสการอ่านสูงสุดหน่วยเป็นไมโครวินาที
rd_timeouts	unsigned long long	จำนวนของการอ่านที่หมดเวลา
rd_failures	unsigned long long	จำนวนของการอ่านที่ล้มเหลว
wr_block_count	unsigned long long	จำนวนของบล็อกที่ถูกเขียน
wr_service_time	unsigned long long	เวลาเซอร์วิสการเขียนทั้งหมดหน่วยเป็นไมโครวินาที
wr_min_service_time	unsigned long long	เวลาเซอร์วิสการเขียนที่น้อยที่สุดหน่วยเป็นไมโครวินาที
wr_max_service_time	unsigned long long	เวลาเซอร์วิสการเขียนสูงสุดหน่วยเป็นไมโครวินาที
wr_timeouts	unsigned long long	จำนวนของการเขียนที่หมดเวลา
wr_failures	unsigned long long	จำนวนของการเขียนที่ล้มเหลว
wait_queue_depth	unsigned long long	ความลึกของคิวการรอของไดรเวอร์
accum_wait_queue_time	unsigned long long	เวลาคิวการรอสะสมหน่วยเป็นไมโครวินาที
min_wait_queue_time	unsigned long long	เวลาคิวการรอที่น้อยที่สุดหน่วยเป็นไมโครวินาที
max_wait_queue_time	unsigned long long	เวลาคิวการรอสูงสุดหน่วยเป็นไมโครวินาที
num_queue_full	unsigned long long	จำนวนของการนับคิว in-flight แบบ full

สถิติ I/O พาธของดิสก์ SCSI

ตารางต่อไปนี้แสดงชื่อฟิลด์ที่สนับสนุนสำหรับสถิติ I/O พาธของดิสก์ Small Computer System Interface (SCSI) ฟิลด์เหล่านี้เป็นฟิลด์เฉพาะสำหรับดิสก์และ อินสแตนซ์พาธ ซึ่งสามารถเข้าถึงเป็น `__stat.io.disk.<hdisk0...n>.path[0...n]->fieldname` สถิติต่อไปนี้สามารถเข้าถึงได้ในโหมดซิงโครนัส และอะซิงโครนัส

หมายเหตุ: สถิติ I/O พาธของดิสก์ SCSI สนับสนุนเฉพาะ IBM® multi-path driver (MPIO)

ตารางที่ 3. สถิติ I/O พาธของดิสก์ SCSI

ชื่อฟิลด์สคริปต์ Vue	ชนิดข้อมูล	คำอธิบาย
ชื่อ	String[32]	ชื่อดิสก์
block_size	unsigned long long	ขนาดบล็อกของดิสก์หน่วยเป็นไบต์
transfers	unsigned long long	จำนวนของการถ่ายโอนไปยัง หรือ จากดิสก์
rd_block_count	unsigned long long	จำนวนของการอ่านบล็อกของดิสก์
wr_block_count	unsigned long long	จำนวนของบล็อกที่ถูกเขียน

สถิติ I/O ไคลเอ็นต์ vSCSI

ตารางต่อไปนี้แสดงชื่อฟิลด์ที่สนับสนุนสำหรับสถิติ I/O ไคลเอ็นต์ SCSI เสมือน (vSCSI) ฟิลด์เหล่านี้เป็นฟิลด์เฉพาะสำหรับอินสแตนซ์ไคลเอ็นต์ vSCSI ซึ่งสามารถ เข้าถึงเป็น `__stat.io.adapter.<vscsi0...n>->fieldname` สถิติต่อไปนี้สามารถเข้าถึงได้ในโหมดซิงโครนัส และอะซิงโครนัส

ตารางที่ 4. สถิติ I/O ไคลเอ็นต์ vSCSI

ชื่อฟิลด์สคริปต์ Vue	ชนิดข้อมูล	คำอธิบาย
ชื่อ	String[32]	ชื่ออุปกรณ์
transfers	unsigned long long	จำนวนของการถ่ายโอนไปยัง หรือ จากอุปกรณ์
rd_block_count	unsigned long long	จำนวนของการอ่านบล็อกข
rd_service_time	unsigned long long	จำนวนของการอ่านหรือได้รับเวลาเซอร์วิสทั้งหมดหน่วยเป็นไมโครวินาที
rd_min_service_time	unsigned long long	เวลาเซอร์วิสการอ่านที่น้อยที่สุดหน่วยเป็นไมโครวินาที
rd_max_service_time	unsigned long long	เวลาเซอร์วิสการเขียนสูงสุดหน่วยเป็นไมโครวินาที
wr_block_count	unsigned long long	จำนวนของบล็อกที่ถูกเขียน
wr_service_time	unsigned long long	เวลาเซอร์วิสการเขียนทั้งหมดหน่วยเป็นไมโครวินาที
wr_min_service_time	unsigned long long	เวลาเซอร์วิสการเขียนที่น้อยที่สุดหน่วยเป็นไมโครวินาที
wr_max_service_time	unsigned long long	เวลาเซอร์วิสการเขียนสูงสุดหน่วยเป็นไมโครวินาที
wait_queue_depth	unsigned long long	ความลึกของคิวการรอสำหรับไดรเวอร์
accum_wait_queue_time	unsigned long long	เวลาคิวการรอสะสมหน่วยเป็นไมโครวินาที
min_wait_queue_time	unsigned long long	เวลาคิวการรอที่น้อยที่สุดหน่วยเป็นไมโครวินาที
max_wait_queue_time	unsigned long long	เวลาคิวการรอสูงสุดหน่วยเป็นไมโครวินาที
num_queue_full	unsigned long long	จำนวนของการนับคิว in flight แบบ full

สถิติไดรเวอร์ไคลเอ็นต์ vSCSI

ตารางต่อไปนี้แสดงชื่อฟิลด์ที่สนับสนุนสำหรับสถิติไดรเวอร์ไคลเอ็นต์ SCSI เสมือน (vSCSI) ฟิลด์เหล่านี้เป็นฟิลด์เฉพาะสำหรับอินสแตนซ์ไคลเอ็นต์ vSCSI ซึ่งสามารถเข้าถึง เป็น `_stat.io.adapter.<vscsi[0...n]>->fieldname` สถิติต่อไปนี้สามารถ เข้าถึงได้ในโหมดซิงโครนัส และอะซิงโครนัส

ตารางที่ 5. สถิติไดรเวอร์ไคลเอ็นต์ vSCSI

ชื่อฟิลด์สคริปต์ Vue	ชนิดข้อมูล	คำอธิบาย
no_dma_failures	unsigned char	จำนวนครั้งที่ระบบล้มเหลวในการส่งคำสั่ง I/O เนื่องจาก Direct Memory Space (DMA) ไม่เพียงพอ ตัวอย่างเช่น DMA_NORES
no_cmd_elem_failures	unsigned char	จำนวนครั้งที่ระบบล้มเหลวในการส่งคำสั่ง I/O เนื่องจาก ไม่มีอิลิเมนต์คำสั่งว่างที่มีไคลเอ็นต์ไดรเวอร์
num_ping_timeouts	unsigned char	จำนวนครั้งที่คำร้องขอ ping ของไคลเอ็นต์ไดรเวอร์ไปยัง Virtual I/O Server (VIOS) ที่แม็พอยู่ล้มเหลว
num_bad_mad	unsigned char	จำนวนครั้งที่ระบบล้มเหลวในการประมวลผลดาตาแกรมการจัดการเนื่องจาก อะแด็ปเตอร์ไม่อยู่ในสถานะที่แอ็คทีฟ
num_hcall_drops	unsigned char	จำนวนครั้งที่ระบบล้มเหลวในการส่งคำสั่งไปยังโฮสต์ CRQ (บน VIOS) เนื่องจากคิว command-response (CRQ) เต็ม ตัวอย่างเช่น พารามิเตอร์ `H_SEND_CRQ()` ล้มเหลวกับพารามิเตอร์ `H_DROPPED`

สถิติไดรเวอร์ไฟเบอร์แชนแนล

ตารางต่อไปนี้แสดงชื่อฟิลด์ที่สนับสนุนสำหรับสถิติไดรเวอร์ ไฟเบอร์แชนแนล ฟิลด์เหล่านี้เป็นฟิลด์เฉพาะสำหรับอินสแตนซ์อุปกรณ์ไฟเบอร์แชนแนล ไวยากรณ์ของ คำสั่ง Vue คือ `__stat.io.adapter.fcs[0...n]->fieldname` สถิติต่อไปนี้สามารถเข้าถึงในโหมดอะซิงโครนัสเท่านั้น

ตารางที่ 6. สถิติไดรเวอร์ไฟเบอร์แชนแนล

ชื่อฟิลด์สคริปต์ Vue	ชนิดข้อมูล	คำอธิบาย
secs_since_last_reset	unsigned long long	เวลาเป็นวินาทีตั้งแต่การรีเซ็ตครั้งล่าสุด
tx_frames	unsigned long long	จำนวนของเฟรมที่ส่ง
tx_words	unsigned long long	kbyte ของไฟเบอร์แชนแนลที่ส่ง
rx_frames	unsigned long long	จำนวนของเฟรมที่ได้รับ
rx_words	unsigned long long	kbyte ของไฟเบอร์แชนแนลที่ได้รับ
lip_count	unsigned long long	จำนวนของเหตุการณ์ Loop Initiation Primitive (LIP) บน Fibre Channel Arbitrated Loop (FC-AL)
nos_count	unsigned long long	จำนวนของเหตุการณ์ Operating System (NOS)

ตารางที่ 6. สถิติไดรเวอร์ไฟเบอร์แชนแนล (ต่อ)

ชื่อฟิลด์สคริปต์ Vue	ชนิดข้อมูล	คำอธิบาย
error_frames	unsigned long long	จำนวนของเฟรมที่ได้รับที่มีข้อผิดพลาด cyclic redundancy check (CRC) หรือ ทิ้งเฟรม แต่ละอะแด็ปเตอร์มีฟิลด์ที่กำหนดฟิลด์นี้แตกต่างกัน
lost_frames	unsigned long long	จำนวนของเฟรมที่หายไป
link_fail_count	unsigned long long	จำนวนของลิงก์ที่ล้มเหลว
sync_loss_count	unsigned long long	จำนวนของการซิงค์ที่หายไป
sig_loss_count	unsigned long long	จำนวนของสัญญาณที่หายไป
prim_seq_proto_errcount	unsigned long long	จำนวนของข้อผิดพลาด primitive sequence
inval_words_received	unsigned long long	จำนวนของคำการส่งที่ไม่ถูกต้องที่ได้รับ
inval_crc_count	unsigned long long	จำนวนของข้อผิดพลาด CRC ในเฟรมที่ได้รับ
num_interrupts	unsigned integer	จำนวนทั้งหมดของอินเตอร์รัปต์
num_spurious_interrupts	unsigned integer	จำนวนทั้งหมดของอินเตอร์รัปต์ปลอม
elastic_buf_overrun_errcount	unsigned integer	จำนวนครั้งที่อินเตอร์เฟสลิงก์เกิน elastic buffer overrun
in_reqs	unsigned long long	คำร้องขออินพุต
out_reqs	unsigned long long	คำร้องขอเอาต์พุต
ctrl_reqs	unsigned long long	คำร้องขอการควบคุม
in_bytes	unsigned long long	ไบต์อินพุต
out_bytes	unsigned long long	ไบต์เอาต์พุต
no_dma_resource_count	unsigned long long	จำนวนของ DMA ที่ล้มเหลวเนื่องจากรีซอร์ส DMA ไม่พร้อมใช้งาน
no_adap_elems_count	unsigned long long	จำนวนของความล้มเหลวในการจัดสรรอิลิเมนต์คำสั่งอะแด็ปเตอร์ เนื่องจากไม่มีอิลิเมนต์คำสั่งเหลือ
no_cmd_resource_count	unsigned long long	จำนวนของความล้มเหลวในการจัดสรรคำสั่งเนื่องจากไม่มีรีซอร์สคำสั่ง
adap_num_active_cmds	unsigned integer	จำนวนของคำสั่งที่แอ็คทีฟในไดรเวอร์อะแด็ปเตอร์
adap_active_high_wmark	unsigned integer	High water mark ของคำร้องขอที่แอ็คทีฟในไดรเวอร์อะแด็ปเตอร์
adap_num_pending_cmds	unsigned integer	จำนวนของคำสั่งที่ค้างอยู่ในไดรเวอร์อะแด็ปเตอร์
adap_pending_high_wmark	unsigned integer	High water mark ของคำร้องขอที่ค้างอยู่ในไดรเวอร์อะแด็ปเตอร์
adap_heldoff_num_cmds	unsigned integer	จำนวนของคำสั่งในคิว held off ของไดรเวอร์อะแด็ปเตอร์

ตารางที่ 6. สถิติไดรเวอร์ไฟเบอร์แชนแนล (ต่อ)

ชื่อฟิลด์สคริปต์ Vue	ชนิดข้อมูล	คำอธิบาย
adap_heldoff_high_wmark	unsigned integer	High water mark ของจำนวนของคำสั่งในคิว held off ของไดรเวอร์อะแด็ปเตอร์
proto_num_active_cmds	unsigned integer	จำนวนของคำสั่งที่แอ็คทีฟในไดรเวอร์ SCSI-FC
proto_active_high_wmark	unsigned integer	High water mark ของคำร้องขอที่แอ็คทีฟในไดรเวอร์ SCSI-FC
proto_num_pending_cmds	unsigned integer	จำนวนของคำสั่งที่ค้างอยู่ในไดรเวอร์ SCSI-FC
proto_pending_high_wmark	unsigned integer	High water mark ของคำร้องขอที่ค้างอยู่ในไดรเวอร์ SCSI-FC

ระดับและอินสแตนซ์ที่กำหนดไว้ล่วงหน้าสำหรับ Network Statistics แสดงไว้ใน ตารางต่อไปนี้:

ตารางที่ 7. ระดับและอินสแตนซ์ที่กำหนดไว้ล่วงหน้า

Built-in	ระดับที่ 0	ระดับ 1	ระดับ 2	อินสแตนซ์ 0	ชื่อฟิลด์
__stat	net	adapter		ent[0...n]	• โปรดดูที่ ตารางที่ 8 ในหน้า 248
__stat	net	interface		en[0....n]	• โปรดดูที่ ตารางที่ 9 ในหน้า 251
__stat	net	protocal	ip		• โปรดดูที่ ตารางที่ 10 ในหน้า 254
__stat	net	protocal	ipv6		• โปรดดูที่ ตารางที่ 11 ในหน้า 256
__stat	net	protocal	tcp		• โปรดดูที่ ตารางที่ 12 ในหน้า 258
__stat	net	protocal	udp		• โปรดดูที่ ตารางที่ 13 ในหน้า 262
__stat	net	protocal	icmp		• โปรดดูที่ ตารางที่ 14 ในหน้า 262
__stat	net	protocal	icmpv6		• โปรดดูที่ ตารางที่ 15 ในหน้า 263
__stat	net	protocal	igmp		• โปรดดูที่ ตารางที่ 16 ในหน้า 265
__stat	net	protocal	arp		• โปรดดูที่ ตารางที่ 17 ในหน้า 266

สถิติไดรเวอร์อุปกรณ์เครือข่าย

ตารางต่อไปนี้แสดงชื่อฟิลด์ที่สนับสนุนสำหรับสถิติ ไดรเวอร์อุปกรณ์เครือข่าย ฟิลด์เหล่านี้เป็นฟิลด์เฉพาะสำหรับอินสแตนซ์อุปกรณ์เครือข่าย สถิติไดรเวอร์อุปกรณ์เครือข่าย สามารถเข้าถึงเป็น "__stat.net.adapter.<ent0...n>->fieldname"

ตารางที่ 8. สถิติไดรเวอร์อุปกรณ์เครือข่าย

ชื่อฟิลด์สคริปต์ Vue	ชนิดข้อมูล	คำอธิบาย	ชนิดการเข้าถึงฟิลด์ (อะซิงโครนัส หรือทั้งคู่)
แฟล็ก	unsigned int	ค่าแฟล็กของอะแด็ปเตอร์ ฟิลด์นี้สามารถมีค่าต่อไปนี้: • NDD_UP • NDD_BROADCAST • NDD_DEBUG • NDD_RUNNING • NDD_SIMPLEX • NDD_DEAD • NDD_LIMBO • NDD_PROMISC • NDD_ALTADDRS • NDD_MULTICAST • NDD_DETACHED • NDD_64BIT • NDD_HIGHFUNC_QOS • NDD_MEDFUNC_QOS • NDD_MINFUNC_QOS • NDD_QOS • NDD_CHECKSUM_OFFLOAD • NDD_PSEG • NDD_ETHERCHANNEL • NDD_VLAN • NDD_SPECFLAGS ค่าเหล่านี้พร้อมใช้งานเป็นค่าคงที่สัญลักษณ์	both
max_mtu	unsigned int	หน่วยการส่งผ่านสูงสุด	both
min_mtu	unsigned int	หน่วยการส่งข้อมูลที่น้อยที่สุด	both
ชนิด	unsigned int	ชนิดอินเตอร์เฟส ฟิลด์นี้สามารถมีค่าต่อไปนี้: • NDD_ETHER • NDD_ISO88023 • NDD_ISO88024 • NDD_ISO88025 • NDD_ISO88026 ค่าเหล่านี้พร้อมใช้งานเป็นค่าคงที่สัญลักษณ์ **หมายเหตุ:** ไม่ใช่ค่าชนิดอินเตอร์เฟสที่เป็นไปได้ทั้งหมดที่ถูกกำหนดไว้และดังนั้นจึงอาจมีอ็อพชันอื่นอยู่ในค่า	both
physaddr	mac_addr_t	ฟิสิคัล หรือ MAC แอดเดรส	both

ตารางที่ 8. สถิติไดรเวอร์อุปกรณ์เครือข่าย (ต่อ)

ชื่อฟิลด์สคริปต์ Vue	ชนิดข้อมูล	คำอธิบาย	ชนิดการเข้าถึงฟิลด์ (อะซิงโครนัส หรือทั้งคู่)
adapter_type	unsigned int	ส่วนขยายของฟิลด์แฟล็ก ฟิลด์นี้สามารถมีค่าต่อไปนี้: • NDD_2_SEA • NDD_2_VIOENT • NDD_2_VASI • NDD_2_HEA • NDD_2_IPV6_LSO • NDD_2_IPV6_CSO • NDD_2_IPV6_PARTIAL_CSO • NDD_2_IPV4_PARTIAL_CSO • NDD_2_LARGE_RECEIVE • NDD_2_ARPINPUT • NDD_2_ECHAN_ELEM • NDD_2_SEA_ELEM, • NDD_2_ROCE • NDD_2_VIRTUAL_PORT • NDD_2_PHYS_LINK_UP • NDD_2_VNIC ค่าเหล่านี้พร้อมใช้งานเป็นค่าคงที่สัญลักษณ์ **หมายเหตุ:** ไม่ใช่ส่วนขยายของค่าฟิลด์แฟล็กที่เป็นไปได้ทั้งหมดที่ถูกกำหนดไว้ และดังนั้นจึงอาจมีอ็อพชันอื่นอยู่ในค่า	both
vlan_id	unsigned int	ตัวบ่งชี้ Virtual LAN (VLAN) (บิต 0 – 11 ใช้สำหรับ VLAN ID)	both
vlan_pri	unsigned int	ลำดับความสำคัญ VLAN (บิต 13 – 15 ใช้สำหรับลำดับความสำคัญของ VLAN)	both
alias (นามแฝง)	String[16]	ชื่อของนามแฝงอะแด็ปเตอร์เครือข่าย	แบบอะซิงโครนัส
nobufs	unsigned long long	จำนวนครั้งที่บัฟเฟอร์เครือข่าย (MBUFs) ไม่พร้อมใช้งานสำหรับไดรเวอร์อุปกรณ์	แบบอะซิงโครนัส
tx_packets	unsigned long long	จำนวนของแพ็กเก็ตที่ส่งสำเร็จโดยอุปกรณ์เครือข่าย	แบบอะซิงโครนัส
tx_bytes	unsigned long long	จำนวนของไบต์ที่ส่งสำเร็จโดยอุปกรณ์เครือข่าย	แบบอะซิงโครนัส
tx_interrupts	unsigned long long	จำนวนของอินเตอร์รัปต์การส่งที่ได้รับโดยไดร์เวอร์ จากอะแด็ปเตอร์	แบบอะซิงโครนัส
tx_errors	unsigned long long	จำนวนของข้อผิดพลาดการส่งบนอุปกรณ์เครือข่าย นี่เป็นตัวนับสำหรับ การส่งข้อมูลที่ไม่สำเร็จเนื่องจากข้อผิดพลาดฮาร์ดแวร์หรือเครือข่าย	แบบอะซิงโครนัส
tx_packets_dropped	unsigned long long	จำนวนครั้งที่แพ็กเก็ตถูกดร็อปขณะที่ส่งข้อมูล จำนวนของแพ็กเก็ต ที่ได้รับโดยไดรเวอร์อุปกรณ์สำหรับการสงข้อมูลที่ไม่ได้กำหนดให้กับ อุปกรณ์	แบบอะซิงโครนัส
tx_queue_overflow	unsigned long long	จำนวนของแพ็กเก็ตขาออกที่โอเวอร์โฟลว์คิวการส่ง ซอฟต์แวร์	แบบอะซิงโครนัส
tx_queue_size	unsigned long long	จำนวนสูงสุดของแพ็กเก็ตขาออกที่เข้าคิวกับคิวการส่ง ซอฟต์แวร์	แบบอะซิงโครนัส

ตารางที่ 8. สถิติไดรเวอร์อุปกรณ์เครือข่าย (ต่อ)

ชื่อฟิลด์สคริปต์ Vue	ชนิดข้อมูล	คำอธิบาย	ชนิดการเข้าถึงฟิลด์ (อะซิงโครนัส หรือทั้งคู่)
tx_queue_len	unsigned long long	จำนวนของแพ็กเก็ตขาออกที่ค้างอยู่บนคิวการส่งซอฟต์แวร์และฮาร์ดแวร์ ปัจจุบัน	แบบอะซิงโครนัส
tx_broadcast_packets	unsigned long long	จำนวนของแพ็กเก็ตบรอดคาสต์ที่ส่ง	แบบอะซิงโครนัส
tx_multicast_packets	unsigned long long	จำนวนของแพ็กเก็ตมัลติคาสต์ที่ส่ง	แบบอะซิงโครนัส
tx_carrier_sense	unsigned long long	จำนวนของการส่งข้อมูลที่ไม่สำเร็จเนื่องจากข้อผิดพลาดไม่พบ carrier	แบบอะซิงโครนัส
tx_DMA_underrun	unsigned long long	จำนวนของการส่งข้อมูลที่ไม่สำเร็จเนื่องจากข้อผิดพลาด direct memory access (DMA) underrun	แบบอะซิงโครนัส
tx_lost_CTS_errors	unsigned long long	จำนวนของการส่งข้อมูลที่ไม่สำเร็จเนื่องจากข้อผิดพลาดสัญญาณ Clear-to-Send หายไป	แบบอะซิงโครนัส
tx_timeout_errors	unsigned long long	จำนวนของการส่งข้อมูลที่ไม่สำเร็จเนื่องจากข้อผิดพลาดการหมดเวลาที่รายงานโดย อะแด็ปเตอร์เครือข่าย	แบบอะซิงโครนัส
tx_max_collision_errors	unsigned long long	จำนวนของการส่งข้อมูลที่ไม่สำเร็จเนื่องจากแพ็กเก็ตที่ส่งชนกันจำนวนมาก ในกรณีนี้ จำนวนของแพ็กเก็ตที่ชนกันที่พบเกินจำนวน การลองใหม่บนอะแด็ปเตอร์เครือข่าย	แบบอะซิงโครนัส
tx_late_collision_errors	unsigned long long	จำนวนของการส่งข้อมูลที่ไม่สำเร็จเนื่องจากข้อผิดพลาดการชนกัน ล่าช้า	แบบอะซิงโครนัส
tx_deferred	unsigned long long	จำนวนของแพ็กเก็ตที่เลื่อนออกไปสำหรับการส่งข้อมูล	แบบอะซิงโครนัส
tx_hw_q_len	unsigned long long	จำนวนของแพ็กเก็ตขาออกที่มีอยู่บนคิวการส่งฮาร์ดแวร์	แบบอะซิงโครนัส
tx_sw_q_len	unsigned long long	จำนวนของแพ็กเก็ตขาออกที่มีอยู่บนคิว การส่งของซอฟต์แวร์	แบบอะซิงโครนัส
tx_single_collision_count	unsigned long long	จำนวนของข้อผิดพลาดการชนกันเดียวที่การส่งข้อมูล	แบบอะซิงโครนัส
tx_multiple_collision_count	unsigned long long	จำนวนข้อผิดพลาดการชนกันจำนวนมากที่การส่งข้อมูล	แบบอะซิงโครนัส
sqe_test	unsigned long long	จำนวนของการทดสอบ Signal Quality Error (SQE) ที่ดำเนินการสำเร็จ ระหว่างการส่งข้อมูล	แบบอะซิงโครนัส
ucast_pkts_reqs	unsigned long long	จำนวนของแพ็กเก็ตยูนิคาสต์ขาออกที่ร้องขอโดย อุปกรณ์เครือข่าย	แบบอะซิงโครนัส
mcast_pkts_reqs	unsigned long long	จำนวนของแพ็กเก็ตมัลติคาสต์ขาออกที่ร้องขอโดย อุปกรณ์เครือข่าย	แบบอะซิงโครนัส
bcast_pkts_reqs	unsigned long long	จำนวนของแพ็กเก็ตบรอดคาสต์ขาออกที่ร้องขอโดย อุปกรณ์เครือข่าย	แบบอะซิงโครนัส
rx_packets	unsigned long long	จำนวนของแพ็กเก็ตที่ได้รับสำเร็จโดยอุปกรณ์เครือข่าย	แบบอะซิงโครนัส
rx_bytes	unsigned long long	จำนวนของไบต์ที่ได้รับสำเร็จโดยอุปกรณ์เครือข่าย	แบบอะซิงโครนัส
rx_interrupts	unsigned long long	จำนวนของอินเตอร์รัปต์การรับที่ได้รับโดยไดรเวอร์ จากอะแด็ปเตอร์	แบบอะซิงโครนัส
rx_errors	unsigned long long	จำนวนของข้อผิดพลาดอินพุตที่พบบนอุปกรณ์นี้ นี่เป็นตัวนับสำหรับ การรับที่ไม่สำเร็จเนื่องจากข้อผิดพลาดฮาร์ดแวร์หรือเครือข่าย	แบบอะซิงโครนัส
rx_packets_dropped	unsigned long long	จำนวนครั้งที่แพ็กเก็ตถูกดร็อปขณะที่รับข้อมูล จำนวน ของแพ็กเก็ตที่ได้รับโดยไดรเวอร์อุปกรณ์จากอุปกรณ์นี้ซึ่งไม่ถูกกำหนดให้กับ demuxer แพ็กเก็ต	แบบอะซิงโครนัส

ตารางที่ 8. สถิติไดรเวอร์อุปกรณ์เครือข่าย (ต่อ)

ชื่อฟิลด์สคริปต์ Vue	ชนิดข้อมูล	คำอธิบาย	ชนิดการเข้าถึงฟิลด์ (อะซิงโครนัส หรือทั้งคู่)
rx_bad_packets	unsigned long long	จำนวนของแพ็กเก็ตที่เสียที่ได้รับโดยไดรเวอร์อุปกรณ์	แบบอะซิงโครนัส
rx_broadcast_packets	unsigned long long	จำนวนของแพ็กเก็ตบรอดคาสต์ที่ได้รับ	แบบอะซิงโครนัส
rx_multicast_packets	unsigned long long	จำนวนของแพ็กเก็ตมัลติคาสต์ที่ได้รับ	แบบอะซิงโครนัส
rx_noresource_errors	unsigned long long	จำนวนของแพ็กเก็ตขาเข้าที่ถูกดร็อปโดยฮาร์ดแวร์เนื่องจาก ไม่มีข้อผิดพลาดรีซอร์ส	แบบอะซิงโครนัส
rx_alignment_errors	unsigned long long	จำนวนของแพ็กเก็ตขาเข้าที่มีข้อผิดพลาดการจัดตำแหน่ง	แบบอะซิงโครนัส
rx_DMA_overrun	unsigned long long	จำนวนของแพ็กเก็ตขาเข้าที่มีข้อผิดพลาด DMA overrun	อะซิงโครนัส
rx_CRC_errors	unsigned long long	จำนวนของแพ็กเก็ตขาเข้าที่มีข้อผิดพลาด checksum	อะซิงโครนัส
rstart_cnt	unsigned long long	จำนวนครั้งที่มีการดำเนินการกู้คืนข้อผิดพลาดอะแด็ปเตอร์	อะซิงโครนัส
rx_collision_errors	unsigned long long	จำนวนของแพ็กเก็ตขาเข้าที่มีข้อผิดพลาดการชนกันระหว่างการรับ	อะซิงโครนัส
rx_packet_tooshort_errors	unsigned long long	จำนวนของแพ็กเก็ตขาเข้าที่มีข้อผิดพลาดของความยาวระบุว่าขนาดแพ็กเก็ตน้อยกว่าขนาดแพ็กเก็ตน้อยที่สุดของอีเทอร์เน็ต	อะซิงโครนัส
rx_packet_toolong_errors	unsigned long long	จำนวนของแพ็กเก็ตขาเข้าที่มีข้อผิดพลาดของความยาวระบุว่าขนาดแพ็กเก็ตใหญ่กว่าขนาดแพ็กเก็ตสูงสุดของอีเทอร์เน็ต	อะซิงโครนัส
rx_packets_discardedbyadapter	unsigned long long	จำนวนของแพ็กเก็ตขาเข้าที่ถูกดร็อปโดยอะแด็ปเตอร์เนื่องจากสาเหตุอื่น	อะซิงโครนัส
rx_start	unsigned long long	จำนวนครั้งที่ตัวรับบนอะแด็ปเตอร์เริ่มทำงาน	แบบอะซิงโครนัส

สถิติที่เกี่ยวกับอินเตอร์เฟสเครือข่าย

ตารางต่อไปนี้แสดงชื่อฟิลด์ที่สนับสนุนสำหรับสถิติที่เกี่ยวกับ อินเตอร์เฟสเครือข่าย ฟิลด์เหล่านี้เป็นฟิลด์เฉพาะสำหรับอินสแตนซ์อินเตอร์เฟสเครือข่าย สถิติที่เกี่ยวกับอินเตอร์เฟสเครือข่าย สามารถเข้าถึงได้ในโหมดซิงโครนัสและอะซิงโครนัส สถิติที่เกี่ยวกับอินเตอร์เฟสเครือข่าย สามารถเข้าถึงเป็น "`__stat.net.interface.<en0...n>->fieldname`"

ตารางที่ 9. สถิติที่เกี่ยวกับอินเตอร์เฟสเครือข่าย

ชื่อฟิลด์สคริปต์ Vue	ชนิดข้อมูล	คำอธิบาย
mtu	unsigned long long	หน่วยการส่งผ่านสูงสุด ขนาดสูงสุดของแพ็กเก็ตหน่วยเป็นไบต์ ที่ถูกส่ง โดยใช้อินเตอร์เฟส

ตารางที่ 9. สถิติที่เกี่ยวกับอินเตอร์เฟสเครือข่าย (ต่อ)

ชื่อฟิลด์สคริปต์ Vue	ชนิดข้อมูล	คำอธิบาย
แฟล็ก	unsigned long long	แฟล็กอินเตอร์เฟส แฟล็กนี้สามารถมีค่าต่อไปนี้: • IFF_UP • IFF_BROADCAST • IFF_DEBUG • IFF_LOOPBACK • IFF_POINTOPOINT • IFF_VIPA • IFF_NOTRAILERS • IFF_RUNNING • IFF_PROMISC • IFF_NOARP ค่าเหล่านี้พร้อมใช้งานเป็นค่าคงที่สัญลักษณ์ **หมายเหตุ:** ไม่ใช่ค่าแฟล็กอินเตอร์เฟสที่เป็นไปได้ทั้งหมดที่ถูกกำหนดไว้และดังนั้นจึงอาจมีอ็อพชันอื่นอยู่ในค่า
ชนิด	unsigned int	ชนิดอินเตอร์เฟส ฟิลด์นี้สามารถมีค่าต่อไปนี้: • IFT_ETHER • IFT_IB • IFT_LOOP • IFT_FDDI • IFT_ISO88023 • IFT_ATM • IFT_OTHER • IFT_TUNNEL ค่าเหล่านี้พร้อมใช้งานเป็นค่าคงที่สัญลักษณ์ **หมายเหตุ:** ไม่ใช่ค่าแฟล็กอินเตอร์เฟสที่เป็นไปได้ทั้งหมดที่ถูกกำหนดไว้และดังนั้นจึงอาจมีอ็อพชันอื่นอยู่ในค่า
ipackets	unsigned long long	จำนวนของแพ็กเก็ตที่ได้รับบนอินเตอร์เฟสเครือข่ายนี้
ibytes	unsigned long long	จำนวนของไบต์ที่ได้รับบนอินเตอร์เฟสเครือข่ายนี้
ierrors	unsigned long long	จำนวนของข้อผิดพลาดอินพุต ตัวอย่างเช่น แพ็กเก็ตที่รูปแบบไม่ถูกต้อง ข้อผิดพลาด checksum, หรือพื้นที่ว่างบัฟเฟอร์ในไดรเวอร์อุปกรณ์ไม่เพียงพอ
opackets	unsigned long long	จำนวนของแพ็กเก็ตที่ส่งบนอินเตอร์เฟสเครือข่ายนี้
obytes	unsigned long long	จำนวนของไบต์ที่ส่งบนอินเตอร์เฟสเครือข่ายนี้
oerrors	unsigned long long	จำนวนของข้อผิดพลาดเอาต์พุต ตัวอย่างเช่น ความบกพร่องในการเชื่อมต่อโลคัล โฮสต์หรือการโอเวอร์รันเอาต์พุตคิวของอะแด็ปเตอร์
collisions	unsigned long long	จำนวนของการชนกันของแพ็กเก็ตที่พบบนอินเตอร์เฟส carrier sense multiple access (CSMA)
if_arpdrops	unsigned long long	ถูกดร็อปเนื่องจากการไม่มีตอบสนอง Address Resolution Protocol (ARP)

ตารางที่ 9. สถิติที่เกี่ยวกับอินเตอร์เฟสเครือข่าย (ต่อ)

ชื่อฟิลด์สคริปต์ Vue	ชนิดข้อมูล	คำอธิบาย
if_iqdrops	unsigned long long	จำนวนครั้งที่แพ็กเก็ตถูกดร็อปในขณะรับข้อมูลบน อินเตอร์เฟสเครือข่ายนี้
index	unsigned int	หมายเลขดัชนีอินเตอร์เฟส
tx_mcasts	unsigned long long	จำนวนของแพ็กเก็ตมัลติคาสต์ที่ส่งบนอินเตอร์เฟสเครือข่ายนี้
rx_mcasts	unsigned long long	จำนวนของแพ็กเก็ตมัลติคาสต์ที่ได้รับบนอินเตอร์เฟสเครือข่ายนี้
no_proto	unsigned long long	โปรโตคอลที่ไม่สนับสนุน
bitrate	unsigned int	อัตราบิตเป็นอัตราที่ข้อมูลถูกส่งบนสาย
dev_num	unsigned long long	หมายเลขของอุปกรณ์
อ็อพชัน	unsigned int	ฟิลด์อ็อพชัน ฟิลด์นี้สามารถมีค่าต่อไปนี้: • IFO_FLUSH • IFO_HIGHFUNC_QOS • IFO_MEDFUNC_QOS • IFO_MINFUNC_QOS • IFO_QOS • IFO_THREAD • IFO_LARGESEND • IFO_PKTCHAIN • IFO_AACCT • IFO_MONITOR • IFO_VIRTUAL_ETHERNET • IFO_CSO_IPV6 • IFO_LSO_IPV6 • IFO_PARTIAL_CSO_IPV6 • IFO_PARTIAL_CSO_IPV4 • IFO_RNIC • IFO_FIRSTALIAS • IFO_PSEUDO_CLUSTER ค่าเหล่านี้พร้อมใช้งานเป็นค่าคงที่สัญลักษณ์ **หมายเหตุ:** ไม่ใช่ค่าอ็อพชันอินเตอร์เฟสที่เป็นไปได้ทั้งหมดที่ถูกกำหนดไว้และดังนั้นจึงอาจมีอ็อพชันอื่นอยู่ในค่า

สถิติที่เกี่ยวกับโปรโตคอลเครือข่าย

ตารางต่อไปนี้แสดงชื่อฟิลด์ที่สนับสนุนสำหรับสถิติที่เกี่ยวกับ โปรโตคอลเครือข่าย ฟิลด์เหล่านี้เป็นฟิลด์เฉพาะสำหรับโปรโตคอลเครือข่ายที่ระบุ ซึ่งสามารถเข้าถึงได้เป็น "__stat.net.protocol.<protocol name>->fieldname" ตัวอย่างเช่น สถิติเฉพาะโปรโตคอล IPv4 สามารถเข้าถึงได้เป็น "__stat.net.protocol.ip->fieldname"สถิติเหล่านี้ สามารถเข้าถึงได้ในโหมดซิงโครนัสและอะซิงโครนัส

สถิติที่เกี่ยวกับโปรโตคอล IPv4 สามารถเข้าถึงเป็น "__stat.net.protocol.ip->fieldname"

ตารางที่ 10. สถิติที่เกี่ยวกับโปรโตคอลเครือข่าย (IPv4)

ชื่อฟิลด์สคริปต์ Vue	ชนิดข้อมูล	คำอธิบาย
ipackets	unsigned long long	จำนวนทั้งหมดของแพ็กเก็ต IP ที่ได้รับ
rx_bytes	unsigned long long	จำนวนทั้งหมดของไบต์ที่ได้รับใน IP datagram
tx_bytes	unsigned long long	จำนวนทั้งหมดของไบต์ข้อมูล IP ที่ส่ง
bad_cksum	unsigned long long	จำนวนของแพ็กเก็ต IP ที่มี checksum ส่วนหัวไม่ถูกต้อง
shorts_pkts	unsigned long long	บัฟเฟอร์ที่เก็บแพ็กเก็ต IP มีจำนวนไบต์น้อยกว่าไบต์ที่แสดงในฟิลด์ความยาว ของส่วนหัว IP (ความยาวทั้งหมดรวมความยาวของส่วนหัว IP + ข้อมูล)
small_pkts	unsigned long long	บัฟเฟอร์ที่เก็บแพ็กเก็ต IP มีจำนวนไบต์น้อยกว่าไบต์ที่แสดงในฟิลด์ความยาวฟิลด์ IPv4
bad_hdr_len	unsigned long long	จำนวนของแพ็กเก็ต IP ที่มีส่วนหัว IP ไม่ถูกต้อง ฟิลด์ความยาวส่วนหัวแพ็กเก็ต IP มีความยาวไม่ถูกต้อง (ความยาวส่วนหัว IP มีขนาดน้อยกว่าขนาดแพ็กเก็ต IP ที่น้อยที่สุด)
bad_data_len	unsigned long long	จำนวนของแพ็กเก็ต IP ที่มีความยาวไม่ถูกต้อง ฟิลด์ความยาวทั้งหมดของแพ็กเก็ต IP มีจำนวนไบต์น้อยกว่าความยาวส่วนหัว IP (ความยาวทั้งหมดรวมความยาวส่วนหัว IP + ข้อมูล) หรือ ขนาดข้อมูล IP มีขนาดใหญ่กว่า ขนาดแพ็กเก็ตสูงสุดที่สนับสนุน (IP_MAXPACKET)
bad_opts	unsigned long long	จำนวนของแพ็กเก็ต IP ที่มีอ็อพชันไม่ถูกต้อง
bad_vers	unsigned long long	จำนวนของแพ็กเก็ต IP ที่มีหมายเลขเวอร์ชันไม่ถูกต้อง
rx_frags	unsigned long long	จำนวนของแฟรกเมนต์ IP ที่ได้รับ
frag_drops	unsigned long long	จำนวนของแฟรกเมนต์ IP ที่ถูกดร็อป (ซ้ำหรือพื้นที่ไม่เพียงพอ)
frag_timeout	unsigned long long	จำนวนของแฟรกเมนต์ IP ที่ถูกดร็อปหลังจากหมดเวลา
reassembled	unsigned long long	จำนวนทั้งหมดของแพ็กเก็ต IP ที่ประกอบใหม่สำเร็จ
เดินหน้า	unsigned long long	จำนวนของแพ็กเก็ต IP ที่ถูกส่งต่อ
no_proto	unsigned long long	จำนวนของแพ็กเก็ตโปรโตคอลที่ไม่รู้จักหรือไม่สนับสนุน
cant_fwd	unsigned long long	จำนวนของแพ็กเก็ตที่ไม่สามารถส่งต่อได้ แพ็กเก็ตที่ได้รับจากปลายทาง ที่ไม่สามารถเข้าถึงได้
tx_redirect	unsigned long long	จำนวนของการเปลี่ยนทิศทางการส่งข้อมูล
tx_drops	unsigned long long	จำนวนของแพ็กเก็ตเอาต์พุตที่ถูกดร็อปเนื่องจากบัฟเฟอร์เครือข่ายไม่พร้อมใช้งาน (MBUF)
no_route	unsigned long long	จำนวนของแพ็กเก็ตเอาต์พุตที่ถูกทิ้งเนื่องจากไม่มีเส้นทาง
tx_frags	unsigned long long	จำนวนของแฟรกเมนต์เอาต์พุตที่สร้างขึ้น
cant_frag	unsigned long long	จำนวนของดาต้าแกรมที่ไม่สามารถแบ่งแฟรกเมนต์ แฟล็ก **don't fragment** ถูกตั้งค่า
ถูกแฟรกเมนต์	unsigned long long	จำนวนของดาต้าแกรมเอาต์พุตที่แบ่งแฟรกเมนต์สำเร็จ

ตารางที่ 10. สถิติที่เกี่ยวกับโปรโตคอลเครือข่าย (IPv4) (ต่อ)

ชื่อฟิลด์สคริปต์ Vue	ชนิดข้อมูล	คำอธิบาย
threads_pkts	unsigned long long	จำนวนของแพ็กเก็ต IP ที่ประมวลผลโดยเธรดเคอร์เนล (dog)
thread_drops	unsigned long long	จำนวนของแพ็กเก็ต IP ที่ถูกดร็อปโดยเธรดเคอร์เนลเนื่องจากไม่มีแพ็กเก็ตที่สามารถ เข้าคิว
iqueueoverflow	unsigned long long	จำนวนของแพ็กเก็ต IP ที่ถูกดร็อปเนื่องจาก socket receive buffer เต็ม
pmtu_disc	unsigned long long	จำนวนของรอบการค้นหา Maximum Transmission Unit (MTU) ของพาธที่สำเร็จ
pmtu_redisc	unsigned long long	จำนวนของการพยายามดำเนินการรอบการค้นหา MTU ของพาธอีกครั้ง
pmtu_guesses	unsigned long long	จำนวนของการเดาหรือการประมาณการค้นหา MTU ของพาธเนื่องจากไม่มีการตอบสนอง
pmtu_timeouts	unsigned long long	จำนวนของการตอบสนองการค้นหา MTU พาธที่หมดเวลา
pmtu_decs	unsigned long long	จำนวนของการลดการค้นหา MTU พาธที่พบ
tx_pmtu_pkts	unsigned long long	จำนวนของแพ็กเก็ตการค้นหา MTU พาธที่ส่ง
pmtu_nomem	unsigned long long	จำนวนของการจัดสรรหน่วยความจำการค้นหา MTU พาธที่ล้มเหลว
tx_dgd_pkts	unsigned long long	จำนวนของแพ็กเก็ต dead gateway detection ที่ส่ง
dgd_nomem	unsigned long long	จำนวนของแพ็กเก็ต dead gateway detection (DGD) ที่ไม่ถูกส่งเนื่องจากการจัดสรร ล้มเหลว
dgd_nogw	unsigned long long	จำนวนของเกตเวย์ dead gateway detection (DGD) ที่ไม่ถูกเพิ่มเนื่องจากการจัดสรร ล้มเหลว
bad_src	unsigned long long	จำนวนของแพ็กเก็ตที่มีแอดเดรสต้นทางไม่ถูกต้อง
delivered	unsigned long long	จำนวนของแพ็กเก็ต IP ที่ใช้
tx_local	unsigned long long	จำนวนทั้งหมดของแพ็กเก็ต IP ที่สร้างขึ้น
tx_raw	unsigned long long	จำนวนทั้งหมดของแพ็กเก็ต RAW IP ที่สร้างขึ้น
hdr_errs	unsigned long long	จำนวนของข้อผิดพลาดส่วนหัว
addr_errs	unsigned long long	จำนวนของดาต้าแกรมที่มีข้อผิดพลาด IP แอดเดรส
rx_discards	unsigned long long	จำนวนของอินพุตดาต้าแกรมที่ถูกทิ้ง
mcast_addr_errs	unsigned long long	จำนวนของแพ็กเก็ต IP มัลติคาสต์ที่ถูกดร็อปเนื่องจากไม่มีผู้รับ
rx_mcast_bytes	unsigned long long	จำนวนของไบต์ IP มัลติคาสต์ที่ได้รับ
tx_mcast_bytes	unsigned long long	จำนวนของไบต์ IP มัลติคาสต์ที่ส่ง
rx_mcast_pkts	unsigned long long	จำนวนของดาต้าแกรม IP มัลติคาสต์ที่ได้รับ
tx_mcast_pkts	unsigned long long	จำนวนของดาต้าแกรม IP มัลติคาสต์ที่ส่ง
rx_bcast_pkts	unsigned long long	จำนวนของดาต้าแกรม IP บรอดคาสต์ที่ได้รับ
tx_bcast_pkts	unsigned long long	จำนวนของดาต้าแกรม IP บรอดคาสต์ที่ส่ง

ตารางที่ 10. สถิติที่เกี่ยวกับโปรโตคอลเครือข่าย (IPv4) (ต่อ)

ชื่อฟิลด์สคริปต์ Vue	ชนิดข้อมูล	คำอธิบาย
tx_mls_drops	unsigned long long	จำนวนของแพ็กเก็ต IP ขาออกที่ถูกดร็อปเนื่องจากตัวกรอง Multi Level Security (MLS)
rx_mls_drops	unsigned long long	จำนวนของแพ็กเก็ต IP ขาเข้าที่ถูกดร็อปเนื่องจากตัวกรอง MLS

สถิติที่เกี่ยวกับโปรโตคอล Internet Protocol เวอร์ชัน 6 (IPv6) สามารถเข้าถึงเป็น "`__stat.net.protocol.ipv6->fieldname`"

ตารางที่ 11. สถิติที่เกี่ยวกับโปรโตคอลเครือข่าย (IPv6)

ชื่อฟิลด์สคริปต์ Vue	ชนิดข้อมูล	คำอธิบาย
ipackets	unsigned long long	จำนวนทั้งหมดของแพ็กเก็ต IPv6 ที่ได้รับ
rx_bytes	unsigned long long	จำนวนทั้งหมดของไบต์ที่ได้รับในดาต้าแกรม IPv6
tx_bytes	unsigned long long	จำนวนทั้งหมดของไบต์ข้อมูล IPv6 ที่ส่ง
raw_cksum	unsigned long long	จำนวนของแพ็กเก็ต IPv6 ที่ไม่ถูกจัดส่งเนื่องจาก raw IPv6 checksum ไม่ถูกต้อง
shorts_pkts	unsigned long long	MBUF มีพื้นที่ไม่เพียงพอเพื่อเก็บแพ็กเก็ต IPv6 (ส่วนหัว IPv6 + ข้อมูล)
small_pkts	unsigned long long	MBUF มีพื้นที่ไม่เพียงพอเพื่อเก็บส่วนหัว IPv6
rx_nomem	unsigned long long	จำนวนครั้งที่บัฟเฟอร์เครือข่าย (MBUFs) ไม่พร้อมใช้งานสำหรับ แพ็กเก็ตอินพุต
tx_nomen	unsigned long long	จำนวนครั้งที่บัฟเฟอร์เครือข่าย (MBUFs) ไม่พร้อมใช้งานสำหรับ แพ็กเก็ตเอาต์พุต
no_proto	unsigned long long	จำนวนของแพ็กเก็ตโปรโตคอลที่ไม่รู้จักหรือไม่สนับสนุน
bad_vers	unsigned long long	จำนวนของแพ็กเก็ต IPv6 ที่มีหมายเลขเวอร์ชันไม่ถูกต้อง
rx_frags	unsigned long long	จำนวนของแฟรกเมนต์ IPv6 ที่ได้รับ
frag_drops	unsigned long long	จำนวนของแฟรกเมนต์ IPv6 ที่ถูกดร็อป (ซ้ำหรือพื้นที่ไม่เพียงพอ)
frag_timeout	unsigned long long	จำนวนของแฟรกเมนต์ IPv6 ที่ถูกดร็อปหลังจากหมดเวลา
ถูกแฟรกเมนต์	unsigned long long	จำนวนของดาต้าแกรมเอาต์พุตที่แบ่งแฟรกเมนต์สำเร็จ
tx_frags	unsigned long long	จำนวนของแฟรกเมนต์เอาต์พุตที่สร้างขึ้น
reassembled	unsigned long long	จำนวนทั้งหมดของแพ็กเก็ต IPv6 ที่ประกอบใหม่สำเร็จ

ตารางที่ 11. สถิติที่เกี่ยวกับโปรโตคอลเครือข่าย (IPv6) (ต่อ)

ชื่อฟิลด์สคริปต์ Vue	ชนิดข้อมูล	คำอธิบาย
cant_frag	unsigned long long	จำนวนของดาต้าแกรมที่ไม่สามารถแบ่งแฟรกเมนต์ แฟล็ก **don't fragment** ถูกตั้งค่า
เดินหน้า	unsigned long long	จำนวนของแพ็กเก็ต IPv6 ที่ถูกส่งต่อ
cant_fwd	unsigned long long	จำนวนของแพ็กเก็ตที่ไม่สามารถส่งต่อได้ แพ็กเก็ตที่ได้รับจากปลายทาง ที่ไม่สามารถเข้าถึงได้
bad_src	unsigned long long	จำนวนของแพ็กเก็ตที่มีแอดเดรสต้นทางไม่ถูกต้อง
tx_drops	unsigned long long	จำนวนของแพ็กเก็ตเอาต์พุตที่ถูกดร็อปเนื่องจากบัฟเฟอร์เครือข่ายไม่พร้อมใช้งาน (MBUF)
no_route	unsigned long long	จำนวนของแพ็กเก็ตเอาต์พุตที่ถูกทิ้งเนื่องจากไม่มีเส้นทาง
delivered	unsigned long long	จำนวนของแพ็กเก็ต IPv6 ที่ใช้
tx_local	unsigned long long	จำนวนทั้งหมดของแพ็กเก็ต IPv6 ที่สร้างขึ้น
iqueueoverflow	unsigned long long	จำนวนของแพ็กเก็ต IPv6 ที่ถูกดร็อปเนื่องจาก socket receive buffer เต็ม
big_pkts	unsigned long long	จำนวนของแพ็กเก็ต IPv6 ที่ไม่ถูกส่งต่อเนื่องจากขนาดของแพ็กเก็ตใหญ่กว่า MTU
tx_raw	unsigned long long	จำนวนทั้งหมดของแพ็กเก็ต raw IPv6 ที่สร้างขึ้น
hdr_errs	unsigned long long	จำนวนของข้อผิดพลาดส่วนหัว
addr_errs	unsigned long long	จำนวนของดาต้าแกรมที่มีข้อผิดพลาด IPv6 แอดเดรส
rx_discards	unsigned long long	จำนวนของอินพุตดาต้าแกรมที่ถูกทิ้ง
rx_mcast_bytes	unsigned long long	จำนวนของไบต์ IPv6 มัลติคาสต์ที่ได้รับ
tx_mcast_bytes	unsigned long long	จำนวนของไบต์ IPv6 มัลติคาสต์ที่ส่ง
rx_mcast_pkts	unsigned long long	จำนวนของดาต้าแกรม IPv6 มัลติคาสต์ที่ได้รับ
tx_mcast_pkts	unsigned long long	จำนวนของไบต์ IPv6 มัลติคาสต์ที่ส่ง
rx_bcast_pkts	unsigned long long	จำนวนของดาต้าแกรม IPv6 บรอดคาสต์ที่ได้รับ
tx_bcast_pkts	unsigned long long	จำนวนของดาต้าแกรม IPv6 มัลติคาสต์ที่ส่ง
tx_mls_drops	unsigned long long	จำนวนของแพ็กเก็ต IPv6 ขาออกที่ถูกดร็อปเนื่องจากตัวกรอง MLS
rx_mls_drops	unsigned long long	จำนวนของแพ็กเก็ต IPv6 ขาเข้าที่ถูกดร็อปเนื่องจากตัวกรอง MLS

สถิติเกี่ยวกับ Transmission Control Protocol (TCP) สามารถเข้าถึงเป็น "`__stat.net.protocol.tcp->fieldname`"

ตารางที่ 12. สถิติที่เกี่ยวกับโปรโตคอลเครือข่าย (TCP)

ชื่อฟิลด์สคริปต์ Vue	ชนิดข้อมูล	คำอธิบาย
tx_total	unsigned long long	จำนวนทั้งหมดของแพ็กเก็ต TCP ที่ส่ง จำนวนนี้รวมถึง **ข้อมูล** และแพ็กเก็ต **ack**
rx_total	unsigned long long	จำนวนทั้งหมดของแพ็กเก็ต TCP ที่ได้รับ จำนวนนี้รวมถึง **ข้อมูล** และแพ็กเก็ต **ack**
opackets	unsigned long long	จำนวนของแพ็กเก็ตข้อมูล TCP ที่ส่ง
ipackets	unsigned long long	จำนวนของแพ็กเก็ตข้อมูล TCP ที่ได้รับ
tx_bytes	unsigned long long	จำนวนของไบต์ข้อมูล TCP ที่ส่ง
rx_bytes	unsigned long long	จำนวนของไบต์ข้อมูล TCP ที่ได้รับตามลำดับ
rexmit_pkts	unsigned long long	จำนวนของแพ็กเก็ตข้อมูล TCP ที่ส่งใหม่
rexmit_bytes	unsigned long long	จำนวนของไบต์ข้อมูล TCP ที่ส่งใหม่
tx_ack_pkts	unsigned long long	จำนวนของแพ็กเก็ต TCP ACK only ที่ส่ง
rx_ack_pkts	unsigned long long	จำนวนของแพ็กเก็ต TCP ACK only ที่ได้รับ
rx_ack_bytes	unsigned long long	จำนวนของไบต์ TCP ACK ที่ได้รับ
rx_dup_pkts	unsigned long long	จำนวนของแพ็กเก็ต duplicate-only TCP ที่ได้รับ
rx_dup_bytes	unsigned long long	จำนวนของไบต์ duplicate-only TCP ที่ได้รับ
rx_part_dup_pkts	unsigned long long	จำนวนของแพ็กเก็ตที่มีข้อมูลบางอย่างซ้ำกัน (แพ็กเก็ต part-duplicate) ที่ได้รับ
rx_part_dup_bytes	unsigned long long	จำนวนของไบต์ที่ซ้ำกันที่ได้รับจากแพ็กเก็ต part-duplicate
rx_dup_ack_pkts	unsigned long long	จำนวนของแพ็กเก็ต TCP duplicate ACK ที่ได้รับ
tx_win_probe	unsigned long long	จำนวนของแพ็กเก็ต TCP window probe ที่ส่ง
rx_win_probe	unsigned long long	จำนวนของแพ็กเก็ต TCP window probe ที่ได้รับ
tx_win_update	unsigned long long	จำนวนของแพ็กเก็ต TCP window update ที่ส่ง
rx_win_update	unsigned long long	จำนวนของแพ็กเก็ต TCP window update ที่ได้รับ
tx_delay_ack_pkts	unsigned long long	จำนวนของแพ็กเก็ต TCP delayed ACK ที่ส่ง
tx_urg_pkts	unsigned long long	จำนวนของแพ็กเก็ต URG only ที่ส่ง
tx_ctrl_pkts	unsigned long long	จำนวนของแพ็กเก็ตควบคุม (SYN|FIN|RST) ที่ส่ง
tx_large_send_pkts	unsigned long long	จำนวนของแพ็กเก็ต large send ที่ส่ง
tx_large_send_bytes	unsigned long long	จำนวนของไบต์ที่ส่งโดยใช้อ็อพชัน large send offload
tx_large_send_max	unsigned long long	จำนวนของไบต์ที่สามารถส่งโดยใช้อ็อพชัน large send offload
rx_ack_unsent_data	unsigned long long	จำนวนของ ACK ที่ได้รับสำหรับข้อมูลที่ไม่ถูกส่ง

ตารางที่ 12. สถิติที่เกี่ยวกับโปรโตคอลเครือข่าย (TCP) (ต่อ)

ชื่อฟิลด์สคริปต์ Vue	ชนิดข้อมูล	คำอธิบาย
rx_out_order_pkts	unsigned long long	จำนวนของแพ็กเก็ต out-of-order ที่ได้รับ
rx_out_order_bytes	unsigned long long	จำนวนของไบต์ out-of-order ที่ได้รับ
rx_after_close_pkts	unsigned long long	จำนวนของแพ็กเก็ตที่ได้รับหลังจากการเชื่อมปิดแล้ว
fast_lo_conns	unsigned long long	จำนวนของการเชื่อมต่อ fastpath loopback
tx_fast_lo_pkts	unsigned long long	จำนวนของแพ็กเก็ตที่ส่งผ่านการเชื่อมต่อ fast path loopback
rx_fast_lo_pkts	unsigned long long	จำนวนของแพ็กเก็ตที่ได้รับผ่านการเชื่อมต่อ fast path loopback
tx_fast_lo_bytes	unsigned long long	จำนวนของไบต์ที่ส่งผ่านการเชื่อมต่อ path loopback
rx_fast_lo_bytes	unsigned long long	จำนวนของไบต์ที่ได้รับผ่านการเชื่อมต่อ fast path loopback
rx_bad_hw_cksum	unsigned long long	จำนวนของแพ็กเก็ตที่ได้รับโดยมี hardware checksum ที่ไม่ถูกต้อง
rx_bad_cksum	unsigned long long	จำนวนของแพ็กเก็ตที่ถูกทิ้งเนื่องจาก hardware checksum ไม่ถูกต้อง
rx_bad_off	unsigned long long	จำนวนของแพ็กเก็ตที่ถูกทิ้งเนื่องจากฟิลด์ error offset ไม่ถูกต้อง
rx_short_pkts	unsigned long long	จำนวนของแพ็กเก็ตที่ถูกทิ้งเนื่องจากแพ็กเก็ตสั้น ขนาดแพ็กเก็ตมีขนาดสั้นกว่า ขนาดแพ็กเก็ต TCP ที่น้อยที่สุด
rx_queue_ovflow	unsigned long long	จำนวนของแพ็กเก็ตที่ถูกทิ้งเนื่องจากคิวตัวรับฟังเต็ม
rx_after_win_pkts	unsigned long long	จำนวนของแพ็กเก็ตที่ได้รับโดยมีข้อมูลที่เกินขนาดหน้าต่างของผู้รับ
rx_after_win_bytes	unsigned long long	จำนวนของไบต์ที่ได้รับโดยมีข้อมูลที่เกินขนาดหน้าต่างของผู้รับ
initiated	unsigned long long	จำนวนของคำร้องขอสำหรับการเชื่อมต่อ TCP
ถูกยอมรับ	unsigned long long	จำนวนของการเชื่อมต่อ TCP ที่ยอมรับ
established	unsigned long long	จำนวนของการเชื่อมต่อ TCP ที่สร้างขึ้น
ปิด	unsigned long long	จำนวนของการเชื่อมต่อ TCP ที่ปิดรวมถึงการเชื่อมต่อถูกดร็อป
dropped	unsigned long long	จำนวนของการเชื่อมต่อ TCP ที่ถูกดร็อป
ecn_conns	unsigned long long	จำนวนของการเชื่อมต่อที่มีความสามารถ Explicit Congestion Notification (ECN)
ecn_congestion	unsigned long long	จำนวนครั้งที่ตอบสนองต่อ ECN

ตารางที่ 12. สถิติที่เกี่ยวกับโปรโตคอลเครือข่าย (TCP) (ต่อ)

ชื่อฟิลด์สคริปต์ Vue	ชนิดข้อมูล	คำอธิบาย
conn_drops	unsigned long long	จำนวนของการเชื่อมต่อ embryonic ที่ถูกดร็อป
segs_timed	unsigned long long	จำนวนครั้งที่เซ็กเมนต์พยายามอัพเดต round trip time (RTT)
rtt_updated	unsigned long long	จำนวนครั้งที่เซ็กเมนต์อัพเดต RTT
ecnce	unsigned long long	จำนวนของเซ็กเมนต์ที่มีการตั้งค่าบิต congestion experienced(CE)
ecnwr	unsigned long long	จำนวนของเซ็กเมนต์ที่มีการตั้งค่าบิต congestion window reduced (CWR)
pmtu_resends	unsigned long long	จำนวนของการส่งใหม่เนื่องจาก path MTU discovery
pmtu_halts	unsigned long long	จำนวนของการยกเลิก path MTU discovery เนื่องจากการส่งใหม่
rexmt_timeout	unsigned long long	จำนวนของการหมดเวลาการส่งใหม่
timeout_drops	unsigned long long	จำนวนของการเชื่อมต่อที่ดร็อปเนื่องจากการหมดเวลาการส่งใหม่
fast_rxmt	unsigned long long	จำนวนของ fast retransmit
new_reno_rxmt	unsigned long long	จำนวนของ NewReno fast retransmit
false_fast_rxmt	unsigned long long	จำนวนครั้งของ false fast retransmits ที่หลีกเลี่ยง
persist_timeouts	unsigned long long	จำนวนของ persist timeout
persist_drops	unsigned long long	จำนวนของการเชื่อมต่อที่ดร็อปเนื่องจาก persist timeout
keep_alive_timeout	unsigned long long	จำนวนของการหมดเวลา keep alive
keep_alive_probe	unsigned long long	จำนวนของ keep alive probes ที่ส่ง
keep_alive_drops	unsigned long long	จำนวนของการเชื่อมต่อที่ดร็อปโดย keep alive
delay_ack_syn	unsigned long long	จำนวนของ delayed ACKs for SYN
delay_ack_fin	unsigned long long	จำนวนของ delayed ACKs for FIN
sack_blocks_upd	unsigned long long	จำนวนครั้งที่อาร์เรย์บล็อก Selective Acknowledgments (SACK) ถูกขยาย
sack_holes_upd	unsigned long long	จำนวนครั้งที่อาร์เรย์ SACK holes ถูกขยาย
tx_drops	unsigned long long	จำนวนของแพ็กเก็ตที่ถูกดร็อปเนื่องจากการจัดสรรหน่วยความจำล้มเหลว
time_wait_reuse	unsigned long long	จำนวนครั้งที่การเชื่อมต่อที่มีอยู่ในสถานะ TIME_WAIT ถูกนำมาใช้ใหม่สำหรับการเชื่อมต่อขาออกใหม่
send_and_disc	unsigned long long	จำนวนของการส่งและการตัดการเชื่อมต่อ

ตารางที่ 12. สถิติที่เกี่ยวกับโปรโตคอลเครือข่าย (TCP) (ต่อ)

ชื่อฟิลด์สคริปต์ Vue	ชนิดข้อมูล	คำอธิบาย
spliced_conns	unsigned long long	จำนวนของการเชื่อมต่อ TCP spliced
splice_closed	unsigned long long	จำนวนของการเชื่อมต่อ TCP spliced ที่ปิด
splice_resets	unsigned long long	จำนวนของการเชื่อมต่อ TCP spliced ที่รีเซ็ต
splice_timeouts	unsigned long long	จำนวนของการเชื่อมต่อ TCP spliced ที่หมดเวลา
splice_persist_drops	unsigned long long	จำนวนของการหมดเวลาที่คงอยู่ของการเชื่อมต่อ TCP spliced
splice_keep_drops	unsigned long long	จำนวนของการหมดเวลา keep live ของการเชื่อมต่อ TCP spliced
bad_ack_conn_drops	unsigned long long	จำนวนของการเชื่อมต่อที่ดร็อปเนื่องจาก ACK ไม่ถูกต้อง
dup_syn_conn_drops	unsigned long long	จำนวนของการเชื่อมต่อที่ดร็อปเนื่องจากแพ็กเก็ต SYN ซ้ำ
auto_cksum_offload	unsigned long long	จำนวนของการเชื่อมต่อที่ checksum offload ถูกปิดใช้งานแบบไดนามิก
bad_syn	unsigned long long	จำนวนของแพ็กเก็ตที่ไม่ถูกต้องที่ทิ้งโดยตัวรับฟัง
limit_transmit	unsigned long long	จำนวนครั้งที่ fast retransmit ได้รับความช่วยเหลือโดยอัลกอริทึม **limited transmit**
pred_acks	unsigned long long	จำนวนครั้งที่ส่วนหัวแพ็กเก็ต ACK มีการคาดการณ์อย่างถูกต้อง
pred_dat	unsigned long long	จำนวนครั้งที่ส่วนหัวแพ็กเก็ตข้อมูลมีการคาดการณ์อย่างถูกต้อง
paws_drops	unsigned long long	จำนวนของเซ็กเมนต์ทีดร็อปเนื่องจาก PAWS
persist_drops	unsigned long long	จำนวนของการเชื่อมต่อที่ดร็อปในสถานะ persist
fake_syn_drops	unsigned long long	จำนวนของเซ็กเมนต์ fake SYN ที่ถูกดร็อป
fake_rst_drops	unsigned long long	จำนวนของเซ็กเมนต์ fake RST ที่ถูกดร็อป
data_inject_drops	unsigned long long	จำนวนของเซ็กเมนต์ data injection ที่ถูกดร็อป
tr_max_conn_drops	unsigned long long	การเชื่อมต่อสูงสุดที่ถูกดร็อปสำหรับกฎทราฟฟิก TCP
tr_nomem_drops	unsigned long long	จำนวนของการเชื่อมต่อที่ถูกดร็อปสำหรับกฎทราฟฟิกเนื่องจากไม่มีหน่วยความจำ
tr_max_per_host	unsigned long long	การเชื่อมต่อสูงสุดต่อโฮสต์ที่ถูกดร็อปสำหรับกฎทราฟฟิก

สถิติที่เกี่ยวกับ User Datagram Protocol (UDP) สามารถเข้าถึงเป็น `"__stat.net.protocol.udp->fieldname"`

ตารางที่ 13. สถิติที่เกี่ยวกับโปรโตคอลเครือข่าย (UDP)

ชื่อฟิลด์สคริปต์ Vue	ชนิดข้อมูล	คำอธิบาย
opackets	unsigned long long	จำนวนทั้งหมดของ UDP ดาต้าแกรมที่ส่ง
ipackets	unsigned long long	จำนวนทั้งหมดของ UDP ดาต้าแกรมที่ได้รับ
hdr_drops	unsigned long long	จำนวนของแพ็กเก็ตที่มีขนาดเล็กกว่าขนาดของส่วนหัว ส่วนหัว IP และ UDP ไม่พอดีกับ memory buffer (MBUF) เดียว
bad_cksum	unsigned long long	จำนวนของแพ็กเก็ต UDP ที่ได้รับที่มี checksum ไม่ถูกต้อง
bad_len	unsigned long long	จำนวนของแพ็กเก็ตที่มีความยาวที่ไม่ถูกต้องที่ได้รับ ความยาวของ UDP ที่ระบุในแพ็กเก็ตมีขนาดใหญ่กว่าขนาดแพ็กเก็ตทั้งหมดในส่วนหัว IP หรือเล็กกว่าขนาดของส่วนหัว UDP
no_socket	unsigned long long	จำนวนของแพ็กเก็ตที่ถูกดร็อปเนื่องจากซ็อกเก็ตบนพอร์ต
sock_buf_overflow	unsigned long long	จำนวนครั้งที่ซ็อกเก็ตบัฟเฟอร์โอเวอร์โฟลว์
dgm_no_socket	unsigned long long	จำนวนของบรอดคาสต์ หรือมัลติคาสต์ดาต้าแกรมที่ถูกดร็อปเนื่องจากไม่มีซ็อกเก็ต
pcb_cache_miss	unsigned long long	จำนวนครั้งที่แพ็กเก็ตอินพุตไม่มีแคช PCB

สถิติที่เกี่ยวกับ Internet Control Message Protocol (ICMP) สามารถเข้าถึงเป็น "`__stat.net.protocol.icmp->fieldname`"

ตารางที่ 14. สถิติที่เกี่ยวกับโปรโตคอลเครือข่าย (ICMP)

ชื่อฟิลด์สคริปต์ Vue	ชนิดข้อมูล	คำอธิบาย
sent	unsigned long long	จำนวนทั้งหมดของแพ็กเก็ต ICMP ที่ส่ง
received	unsigned long long	จำนวนทั้งหมดของแพ็กเก็ต ICMP ที่ได้รับ
ข้อผิดพลาด	unsigned long long	จำนวนของข้อผิดพลาด ICMP
bad_cksum	unsigned long long	จำนวนของข้อความ ICMP ที่ได้รับที่มี checksum ไม่ถูกต้อง
bad_len	unsigned long long	จำนวนของข้อความ ICMP ที่ได้รับที่มีความยาวไม่ถูกต้อง
bad_code	unsigned long long	จำนวนของข้อความ ICMP ที่ได้รับที่มีฟิลด์โค้ดไม่ถูกต้อง ข้อความเหล่านี้มี **icmp_code** ที่ไม่อยู่ในช่วง
old_msg	unsigned long long	จำนวนของข้อผิดพลาดที่ไม่ถูกสร้างขึ้นเนื่องจากโปรโตคอลของแพ็กเก็ตเก่า คือ ICMP
old_short_msg	unsigned long long	จำนวนของข้อผิดพลาดที่ไม่ถูกสร้างขึ้นเนื่องจากแพ็กเก็ต IP เก่าสั้นเกินไป

ตารางที่ 14. สถิติที่เกี่ยวกับโปรโตคอลเครือข่าย (ICMP) (ต่อ)

ชื่อฟิลด์สคริปต์ Vue	ชนิดข้อมูล	คำอธิบาย
short_msg	unsigned long long	ขนาดของข้อความ ICMP มีขนาดน้อยกว่าขนาดที่น้อยที่สุดของข้อความ ICMP (ขนาดแพ็กเก็ต <ICMP_MINLEN)
reflect	unsigned long long	จำนวนของการตอบสนองข้อความ ICMP ที่สร้างขึ้น

สถิติที่เกี่ยวกับโปรโตคอล ICMPV6 สามารถเข้าถึงเป็น "__stat.net.protocol.icmpv6->fieldname"

ตารางที่ 15. สถิติที่เกี่ยวกับโปรโตคอลเครือข่าย (ICMPV6)

ชื่อฟิลด์สคริปต์ Vue	ชนิดข้อมูล	คำอธิบาย
tx_echo_reply	unsigned long long	จำนวนทั้งหมดของการตอบกลับ ICMPv6 echo ที่ส่ง
rx_echo_reply	`unsigned long long	จำนวนทั้งหมดของการตอบกลับ ICMPv6 echo ที่ได้รับ
ข้อผิดพลาด	unsigned long long	จำนวนของข้อผิดพลาด ICMPv6
rx_bad_cksum	unsigned long long	จำนวนของข้อความ ICMPv6 ที่ได้รับที่มีข้อผิดพลาด checksum ไม่ถูกต้อง
rx_bad_len	unsigned long long	จำนวนของข้อความ ICMPv6 ที่ได้รับที่มีความยาวไม่ถูกต้อง
bad_code	unsigned long long	จำนวนของข้อความ ICMPv6 ที่มีฟิลด์โค้ดไม่ถูกต้อง ข้อความเหล่านี้มี **icmp6_code** ที่ไม่อยู่ในช่วง
old_msg	unsigned long long	จำนวนของข้อผิดพลาดที่ไม่ถูกสร้างขึ้นเนื่องจากโปรโตคอลของแพ็กเก็ตเก่า คือ ICMPv6
short_msg	unsigned long long	ขนาดของข้อความ ICMPv6 มีขนาดน้อยกว่าขนาดที่น้อยที่สุดของข้อความ ICMPv6 (ขนาดแพ็กเก็ต <ICMP6_MINLEN)
reflect	unsigned long long	จำนวนของการตอบสนองข้อความ ICMPv6 ที่สร้างขึ้น
err_rate_limit	unsigned long long	จำนวนของข้อผิดพลาด ICMPv6 ที่เกินกว่าขีดจำกัดของอัตราข้อผิดพลาด
tx_unreach	unsigned long long	จำนวนของข้อความที่ไม่สามารถเข้าถึงได้ที่ส่ง
rx_unreach	unsigned long long	จำนวนของข้อความที่ไม่สามารถเข้าถึงได้ที่ได้รับ
tx_big_pkt	unsigned long long	จำนวนครั้งที่แพ็กเก็ต ICMPv6 ขนาดใหญ่ถูกส่ง
rx_big_pkt	unsigned long long	จำนวนครั้งที่ได้รับแพ็กเก็ต ICMPv6 ขนาดใหญ่
tx_timxceed	unsigned long long	จำนวนครั้งที่เกินเวลาที่ส่งข้อความ ICMPv6
rx_timxceed	unsigned long long	จำนวนครั้งที่เกินเวลาเวลาที่รับข้อความ ICMPv6
tx_param_prob	unsigned long long	จำนวนครั้งที่ส่งข้อความ ICMPv6 ที่มีปัญหาพารามิเตอร์

ตารางที่ 15. สถิติที่เกี่ยวกับโปรโตคอลเครือข่าย (ICMPV6) (ต่อ)

ชื่อฟิลด์สคริปต์ Vue	ชนิดข้อมูล	คำอธิบาย
rx_param_prob	unsigned long long	จำนวนครั้งที่รับข้อความ ICMPv6 ที่มีปัญหาพารามิเตอร์
tx_echo_req	unsigned long long	จำนวนครั้งที่ส่งข้อความคำร้องขอ echo
rx_echo_req	unsigned long long	จำนวนครั้งที่ได้รับข้อความคำร้องขอ echo
tx_mld_qry	unsigned long long	จำนวนครั้งที่ส่งคำร้องขอเดียวรีกลุ่ม
rx_mld_qry	unsigned long long	จำนวนครั้งที่ได้รับคำร้องขอเดียวรีกลุ่ม
tx_mld_report	unsigned long long	จำนวนครั้งที่ส่งรายงานกลุ่ม
rx_mld_report	unsigned long long	จำนวนครั้งที่ได้รับรายงานกลุ่ม
rx_bad_mld_qry	unsigned long long	จำนวนครั้งที่ได้รับเดียวรีกลุ่มที่ไม่ถูกต้อง
rx_bad_mld_report	unsigned long long	จำนวนครั้งที่ได้รับรายงานกลุ่มที่ไม่ถูกต้อง
rx_our_mld_report	unsigned long long	จำนวนครั้งที่ได้รับรายงานกลุ่มของเรา
tx_mld_term	unsigned long long	จำนวนครั้งที่มีการส่งการยกเลิกกลุ่ม
rx_mld_term	unsigned long long	จำนวนครั้งที่ได้รับการยกเลิกกลุ่ม
rx_bad_mld_term	unsigned long long	จำนวนครั้งที่ได้รับการยกเลิกกลุ่มที่ไม่ถูกต้อง
tx_redirect	unsigned long long	จำนวนครั้งที่ส่งการเปลี่ยนทิศทาง
rx_redirect	unsigned long long	จำนวนครั้งที่ได้รับการเปลี่ยนทิศทาง
rx_bad_redirect	unsigned long long	จำนวนครั้งที่ได้รับการเปลี่ยนทิศทางที่ไม่ถูกต้อง
tx_router_sol	unsigned long long	จำนวนครั้งที่ส่ง router solicitations
rx_router_sol	unsigned long long	จำนวนครั้งที่ได้รับ router solicitations
rx_bad_router_sol	unsigned long long	จำนวนครั้งที่ได้รับ router solicitation ที่ไม่ถูกต้อง
tx_router_adv	unsigned long long	จำนวนครั้งที่ส่ง router advertisements
rx_router_adv	unsigned long long	จำนวนครั้งที่ได้รับ router advertisements
rx_bad_router_adv	unsigned long long	จำนวนครั้งที่ได้รับ router solicitation ที่ไม่ถูกต้อง
tx_nd_sol	unsigned long long	จำนวนครั้งที่ส่ง neighbor solicitations
rx_nd_sol	unsigned long long	จำนวนครั้งที่ได้รับ neighbor solicitations
rx_bad_nd_sol	unsigned long long	จำนวนครั้งที่ได้รับ neighbor solicitation ที่ไม่ถูกต้อง
tx_nd_adv	unsigned long long	จำนวนครั้งที่ส่ง neighbor advertisements
rx_nd_adv	unsigned long long	จำนวนครั้งที่ได้รับ neighbor advertisements
rx_bad_nd_adv	unsigned long long	จำนวนครั้งที่ได้รับ neighbor advertisement ที่ไม่ถูกต้อง
tx_router_renum	unsigned long long	จำนวนครั้งที่ส่ง router re-numberings

ตารางที่ 15. สถิติที่เกี่ยวกับโปรโตคอลเครือข่าย (ICMPV6) (ต่อ)

ชื่อฟิลด์สคริปต์ Vue	ชนิดข้อมูล	คำอธิบาย
rx_router_renum	unsigned long long	จำนวนครั้งที่ได้รับ router re-numberings
tx_haad_req	unsigned long long	จำนวนครั้งที่ส่งคำร้องขอ home agent address discovery (HAAD)
rx_haad_req	unsigned long long	จำนวนครั้งที่ได้รับคำร้องขอ HAAD
rx_bad_haad_req	unsigned long long	จำนวนครั้งที่ได้รับคำร้องขอ HAAD ที่ไม่ถูกต้อง
tx_haad_reply	unsigned long long	จำนวนครั้งที่ส่งการตอบกลับ HAAD
rx_haad_reply	unsigned long long	จำนวนครั้งที่ได้รับการตอบกลับ HAAD
rx_bad_haad_reply	unsigned long long	จำนวนครั้งที่ได้รับการตอบกลับ HAAD ที่ไม่ถูกต้อง
tx_prefix_sol	unsigned long long	จำนวนครั้งที่ส่ง prefix solicitations
rx_prefix_sol	unsigned long long	จำนวนครั้งที่ได้รับ prefix solicitations
rx_bad_prefix_sol	unsigned long long	จำนวนครั้งที่ได้รับ prefix solicitation ที่ไม่ถูกต้อง
tx_prefix_adv	unsigned long long	จำนวนครั้งที่ส่ง prefix advertisements
rx_prefix_adv	unsigned long long	จำนวนครั้งที่ได้รับ prefix advertisements
rx_bad_prefix_adv	unsigned long long	จำนวนครั้งที่ได้รับ prefix solicitation ที่ไม่ถูกต้อง
no_mobility	unsigned long long	จำนวนของ mobility calls เมื่อไม่เริ่มทำงาน
ndp_q_drops	unsigned long long	จำนวนของ held packet ที่ถูกดร็อปขณะรอเพื่อให้ ndp เสร็จสมบูรณ์

สถิติเกี่ยวกับ Internet Group Management Protocol (IGMP) สามารถเข้าถึงเป็น "`__stat.net.protocol.igmp->fieldname`"

ตารางที่ 16. สถิติที่เกี่ยวกับโปรโตคอลเครือข่าย (IGMP)

ชื่อฟิลด์สคริปต์ Vue	ชนิดข้อมูล	คำอธิบาย
rx_total	unsigned int	จำนวนทั้งหมดของแพ็กเก็ต IGMP ที่ได้รับ
rx_queries	unsigned int	จำนวนของเคียวรี IGMP membership ที่ได้รับ
tx_reports	unsigned int	จำนวนของรายงาน IGMP membership ที่ส่ง
rx_reports	unsigned int	จำนวนของรายงาน IGMP membership ที่ได้รับ
rx_our_reports	unsigned int	จำนวนของรายงาน IGMP membership ที่ได้รับสำหรับกลุ่มของเรา
rx_bad_cksum	unsigned int	จำนวนที่ได้รับข้อความ IGMP ที่มีข้อผิดพลาด checksum ไม่ถูกต้อง
rx_short_msg	unsigned int	จำนวนที่ได้รับข้อความ IGMP ที่มีไบต์จำนวนน้อยที่มีขนาดเล็กกว่าขนาดที่เล็กที่สุด ของข้อความ IGMP

ตารางที่ 16. สถิติที่เกี่ยวกับโปรโตคอลเครือข่าย (IGMP) (ต่อ)

ชื่อฟิลด์สคริปต์ Vue	ชนิดข้อมูล	คำอธิบาย
rx_bad_queries	unsigned int	จำนวนของเคียวรี IGMP membership ที่ได้รับที่มีฟิลด์ไม่ถูกต้อง
rx_bad_reports	unsigned int	จำนวนของรายงาน IGMP membership ที่ได้รับที่มีฟิลด์ไม่ถูกต้อง

สถิติที่เกี่ยวกับ Address Resolution Protocol (ARP) สามารถเข้าถึงเป็น "`__stat.net.protocol.arp->fieldname`"

ตารางที่ 17. สถิติที่เกี่ยวกับโปรโตคอลเครือข่าย (ARP)

ชื่อฟิลด์สคริปต์ Vue	ชนิดข้อมูล	คำอธิบาย
purged	unsigned int	จำนวนของแพ็กเก็ต ARP ที่ถูกล้างออก เมื่อไม่มีที่ว่างในบักเก็ต ให้ลบรายการ ARP ที่เก่าที่สุด ออกจากบักเก็ต
sent	unsigned int	จำนวนทั้งหมดของแพ็กเก็ต ARP ที่ส่ง

ProbeVue เป็นคำสั่งที่รันนานและสคริปต์ the Vue เป็นสคริปต์ที่รันหลายครั้ง สคริปต์ Vue อาจอ้างถึงไปยังอุปกรณ์หรือรีซอร์สเพื่อพิมพ์สถิติ และรีซอร์สและอุปกรณ์เหล่านั้นจะไม่พร้อมใช้งานชั่วคราว หรือ ไม่พร้อมใช้งานถาวร ดังนั้น คำสั่งสถิติระบบ ProbeVue จะมีลักษณะการทำงานต่อไปนี้สำหรับเงื่อนไขที่ต่างกัน:

ตารางที่ 18. เงื่อนไขสถิติระบบ ProbeVue และ ลักษณะการทำงาน

เลขลำดับ	เงื่อนไข	Behavior
1	ไม่มีรีซอร์สที่ระบุในคำสั่ง	ระหว่างการตรวจสอบสคริปต์ Vue หาก ProbeVue ไม่พบรีซอร์สที่ระบุในสคริปต์ Vue การคอมไพล์จะล้มเหลว
2	อุปกรณ์หรือรีซอร์สไม่เปิด/แอ็คทีฟ	• ProbeVue จะพิมพ์ศูนย์ทั้งหมดสำหรับสถิติ • อุปกรณ์อาจถูกเปิดเพื่อรันฟังก์ชัน **ioctl()** และจากนั้นถูกปิด
3	อุปกรณ์หรือรีซอร์สถูกย้ายจากสถานะที่แอ็คทีฟเป็นสถานะที่ไม่แอ็คทีฟ	ProbeVue จะพิมพ์ศูนย์ หรือ ข้อมูลค้าง
4	อุปกรณ์ถูกปิดขณะรัน	ผลลัพธ์เดี่ยวดับกับเมื่อรีซอร์สถูกย้ายไปเป็นสถานะที่ไม่แอ็คทีฟ
5	API การดึงข้อมูลแบบอะซิงโครนัสใช้เวลามากกว่าช่วงเวลาการดึงข้อมูลที่ตั้งค่าโดย ผู้ใช้	ProbeVue จะไม่สามารถรับประกัน ช่วงเวลาการดึงข้อมูลได้ หากการดึงข้อมูลใช้เวลามากกว่าช่วงเวลา
6	การดำเนินการ LPAR แบบไดนามิกขณะรันไทม์และรีซอร์สที่ระบุโดยสคริปต์ Vue ถูกลบออก	ProbeVue จะพิมพ์ศูนย์ทั้งหมด หรือ ข้อมูลค้าง
7	การเรียกใช้การดึงสถิติอุปกรณ์ถูกบล็อกเนื่องจากหมดเวลาใน ไดรเวอร์อุปกรณ์	ProbeVue จะไม่สามารถรับประกัน ช่วงเวลาการดึงข้อมูลและผู้ใช้อาจเห็นข้อมูลค้าง

การกำหนดค่าและชนิด

การจัดประเภทในส่วนก่อนหน้านี้คือหนึ่งวิธี ที่ใช้ดูตัวแปรในสคริปต์ Vue คลาสตัวแปร สามารถตรวจสอบได้จากเปอร์สเปคทีฟต่างๆ วิธีการรับค่าต่างๆ ภายใต้มุมมอง คุณสามารถแบ่งตัวแปร ออกเป็นสองหมวดหมู่:

ตัวแปรภายนอก

ตัวแปรคลาสเคอร์เนล ตัวแปรคลาส entry และ exit และตัวแปรในตัว คือตัวแปรภายนอกทั้งหมด ซึ่งจะมีอยู่เป็นอิสระของกรอบงาน ProbeVue และได้รับค่าภายนอกบริบทของสคริปต์ Vue ใดๆ ProbeVue อนุญาตให้ค่าปัจจุบันของตัวแปรภายนอกถูกทำให้พร้อมใช้งานภายในสคริปต์ Vue ตัวแปรเหล่านี้จะเป็นแบบอ่านอย่างเดียว ภายในบริบทของสคริปต์ Vue คำสั่งโปรแกรมใดๆ ที่พยายามแก้ไขค่าของตัวแปรภายนอก จะถูกแฟล็กโดยคอมไพลเลอร์ว่าเป็นคำสั่งที่ผิดกฎเกณฑ์

แม้ว่าตัวแปรภายนอกจะมีชนิดที่ได้กำหนดไว้ก่อน ProbeVue ต้องการการประกาศตัวแปรภายนอกให้ชัดเจน ยกเว้นตัวแปรในตัว ในสคริปต์ Vue ที่เข้าถึง ตารางต่อไปนี้อธิบายถึง วิธีการกำหนดชนิดของตัวแปรภายนอก:

Variable	ชนิก
เคอร์เนลโกลบอลคลาส	จากคำสั่งการประกาศ **__kernel** ของตัวแปรเคอร์เนล
คลาส entry	จากการประกาศฟังก์ชันต้นแบบ ในสคริปต์ Vue ต้องระบุชนิดข้อมูลของอาร์กิวเมนต์แต่ละตัว ที่จะใช้ในสคริปต์ Vue
ค่าส่งคืนจากฟังก์ชันเคอร์เนล	จากการประกาศฟังก์ชันต้นแบบ ในสคริปต์ Vue ต้องจัดเตรียมชนิดของค่าส่งคืน
ในตัว	ตัวแปรที่พึ่งพา ตัวแปรเคอร์เนล ชนิดที่นิยามไว้และเทียบเท่ากับชนิด ProbeVue มีดังต่อไปนี้:

Built-in	**ชนิดที่นิยามไว้**	**ProbeVue type**
__tid	tid_t	long long
__pid	pid_t	long long
__ppid	pid_t	long long
__pgid	pid_t	long long
__pname	char [32]	String [32]
__uid	uid_t	unsigned int
__euid	uid_t	unsigned int
__trcid	pid_t	long long
__errno	int	int
__kernelmode	int	int
__r3..__r10	32 บิตสำหรับการประมวลผลแบบ 32 บิต 64 บิตสำหรับการประมวลผลแบบ 64 บิต	unsigned long
__curthread	N/A	สมาชิกทั้งหมดคือ long long
__curproc	N/A	สมาชิกทั้งหมดยกเว้นสำหรับ **cwd** เป็น long long สมาชิก **cwd** เป็นสตริงชนิด
__ublock	N/A	สมาชิกทั้งหมดคือ long long

หมายเหตุ: ขนาดสูงสุดของข้อมูลที่ส่งคืน สามารถเล็กกว่าขนาดของชนิด ตัวอย่างเช่น ID การประมวลผลใน AIX จะเหมาะกับเลขจำนวนเต็ม 32 บิต ขณะที่ชนิดข้อมูล pid_t คือเลขจำนวนเต็ม 64 บิตสำหรับการประมวลผลแบบ 64 บิต และเคอร์เนล

ตัวแปรสคริปต์

ตัวแปรสคริปต์คือตัวแปรแบบอัตโนมัติ ตัวแปรเธรดแบบโลคัล หรือตัวแปรคลาสโกลบอล อย่างใดอย่างหนึ่ง ตัวแปรสคริปต์จะมีอยู่ภายในบริบทของสคริปต์ Vue และค่าจะถูกกำหนดจากสคริปต์ ยิ่งไปกว่านั้น ตัวแปรเหล่านี้ยังสามารถเข้าถึงหรือแก้ไขได้ภายในสคริปต์ ที่กำหนดตัวแปรเหล่านั้นไว้

โดยทั่วไป คุณต้องประกาศชนิดข้อมูลของตัวแปรสคริปต์ ผ่าน declaration statement อย่างไรก็ตาม คอมไพลเลอร์สามารถกำหนด ชนิดข้อมูลของตัวแปรโปรแกรมในกรณีที่มีข้อจำกัดบางอย่าง หากการอ้างอิงถึงตัวแปรคือการดำเนินการกำหนดค่าด้วยตัวแปรทางด้านซ้ายและขวา ของตัวดำเนินการ

การกำหนดชนิดโดยนัยสำหรับชนิดกลุ่ม

หากต้องการกำหนดชนิดกลุ่ม ด้านขวามือของการกำหนด ต้องเป็นหนึ่งในสถานการณ์ต่อไปนี้:

- ตัวเลขคงที่
- ตัวแปรอื่นๆ ของชนิดกลุ่มที่ประกอบด้วยตัวแปรในตัว การกำหนดจากตัวแปรที่มีชนิดที่ไม่รู้จักคือข้อผิดพลาด
- ฟังก์ชัน Vue ที่ส่งคืนชนิดกลุ่ม เช่น ฟังก์ชัน **diff_time**
- การคัดเลือกนิพจน์ทางด้านขวาให้มีชนิดกลุ่ม แม้ว่าการดำเนินการนี้จะแสดงคำเตือนในบางกรณี
- นิพจน์ที่เกี่ยวข้องกับสถานการณ์ก่อนหน้าใดๆ

ตัวแปรที่ใช้ชนิดที่เพิ่มเติมจากค่า ที่อ้างอิงตามนิพจน์ทางด้านขวา นอกจากนี้ คลาสของตัวแปร สามารถกำหนดให้กับตัวแปรได้โดยใส่คำนำหน้าในตัวแปร สคริปต์ตัวอย่างต่อไปนี้ จะสาธิตตัวอย่างบางส่วน:

```
/*
 * File: implicit2.e
 * Usage: Demonstrates implicit assignment for integer types
 */

int read(int fd, char *p, long size);

@@BEGIN
{
 count = 404;   /* count: int of global class */
 zcount = 2 * (count - 4); /* zcount: int of global class */
 llcount = 33459182089021LL;   /* lcount: long long of global class */
 lxcount = 0xF00000000245B20LL; /* xcount: long long of global class */

}

@@syscall:$1:read:entry
{
 __auto probev_timestamp_t ts1, ts2;
 int gsize;
 ts1 = timestamp();
 auto:dcount = llcount - lxcount;  /* dcount: long long of auto class */

 auto:mypid = __pid; /* mypid:  pid_t (64-bit integer) of automatic class */
```

```
 fd = __arg1;  /* fd: int of global class */

 /* The following cast will likely cause a compiler warning
  * but can be ignored here
  */
 global:bufaddr = (long)__arg2; /* bufaddr: long of global class */

 gsize = __arg3;
 thread:size =  gsize + 400; /* size: int of thread-local class */

 printf("count = %d, zcount = %lld\n", count, zcount);
 printf("llcount = %lld, lxcount = 0x%016llx, diff = %lld\n",
   llcount, lxcount, dcount);
 printf("mypid = %ld, fd = %d, size = %d\n", mypid, fd, size);
 printf("bufaddr = 0x%08x\n", bufaddr);
 ts2 = timestamp();

 auto:diff = diff_time(ts1, ts2, MICROSECONDS); /* diff: int of automatic class */

 printf("Time to execute = %d microseconds\n", diff);

 exit();
}
```

หมายเหตุ: การมีอยู่ในสคริปต์ที่นำหน้าของตำแหน่ง shell เช่นเดียวกับพารามิเตอร์ นั่นคือ สัญลักษณ์ **$1** ในข้อกำหนดคุณสมบัติโพรบ **@@syscall:$1:read:entry** ตัวจัดการโพรบ **syscall** อนุญาตให้ ID การประมวลผล ID สำหรับฟิลด์ที่สองเพื่อบ่งชี้ว่า จุดโพรบของการเรียกของระบบต้องเปิดใช้งาน สำหรับการประมวลผลโดยเฉพาะเท่านั้น นอกเหนือจากฮาร์ดโค้ด ID การประมวลผลที่ระบุแล้ว ฟิลด์ที่สองได้ถูกตั้งค่าพารามิเตอร์เชิงตำแหน่งของ shell ในสคริปต์นี้ เพื่ออนุญาตให้ ID การประมวลผลที่เกิดขึ้นจริงส่งผ่าน เป็นอาร์กิวเมนต์ในเวลาที่ออกสคริปต์ คำสั่ง **probevue** จะแทนที่พารามิเตอร์เชิงตำแหน่ง shell ใดๆ ในสคริปต์ด้วยอาร์กิวเมนต์ตามลำดับ ที่ส่งผ่านบนบรรทัดรับคำสั่ง

สมมุติว่าคุณกำลังโพรบการประมวลผลที่มี ID การประมวลผล ID 250000 สคริปต์ต่อไปนี้จะแสดงตัวอย่างการรันสคริปต์ **implicit2.e**

```
# probevue implicit2.e 250000
WRN-100: Line:29 Column:26 Incompatible cast
count = 404, zcount = 800
llcount = 33459182089021, lxcount = 0x0f00000000245b20, diff = -1080830451389212643
mypid = 250000, fd = 10, size = 4496
bufaddr = 0x20033c00
Time to execute = 11 microseconds
```

ในตัวอย่างก่อนหน้านี้ สัญลักษณ์ **$1** ในสคริปต์จะแทนที่ด้วย "250000" โดยอัตโนมัติ ดังนั้น การจำกัดจุดโพรบ entry ของการเรียกของระบบจะชี้ไปยังการประมวลผลด้วย ID การประมวลผลที่เท่ากับ 250000

การกำหนดชนิดโดยในสำหรับชนิดสตริง

หากต้องการกำหนดชนิดสตริง ด้านขวามือของการกำหนด ต้องเป็นหนึ่งในสถานการณ์ต่อไปนี้:

- สตริงตามตัวอักษรที่เรียงลำดับอักขระภายในเครื่องหมายอัญประกาศ คู่
- ตัวแปรอื่นของชนิดสตริงซึ่งประกอบด้วยตัวแปรบิวด์อิน

- ฟังก์ชัน Vue ที่ส่งคืนสตริง เช่น ฟังก์ชัน **et_userstring**
- นิพจน์ที่เกี่ยวข้องกับสถานการณ์ข้างต้นใดๆ

ตัวอย่างต่อไปนี้จะสาธิตการกำหนดชนิดของสตริงโดยนัย:

```
/*
 * File: implicit3.e
 * Usage: Demonstrates implicit assignment for string types
 */

int write(int fd, char *p, long size);

@@BEGIN
{
 s1 = "Write system call:\n";
}

@@syscall:$1:write:entry
{
 String s2[40];

 wbuf = get_userstring(__arg2, __arg3);

 s2 = s1;

 zbuf = s2;

 pstring = zbuf + wbuf;

 printf("%s\n", pstring);
}

@@syscall:$1:write:exit
{
 ename = __pname;
 printf("Exec name = %s\n", ename);
 exit();
}
```

ID การประมวลผลต้องถูกส่งเป็นอาร์กิวเมนต์ไปยังสคริปต์ เมื่อออกคำสั่งเพื่อแทนที่ตัวแปรพารามิเตอร์เชิงตำแหน่งของ **$1** shell

การกำหนดชนิดโดยนัยสำหรับชนิดลิสต์

หากต้องการกำหนดชนิดลิสต์ ที่ด้านขวาของการกำหนดต้องเป็น ฟังก์ชัน **list( )** ฟังก์ชัน **list( )** สามารถเรียกใช้จาก clause ใดๆ ก็ได้

ประโยชน์ของตัวแปรเคอร์เนล

ตารางต่อไปนี้แสดงตัวอย่างเล็กน้อยของประโยชน์ของตัวแปรเคอร์เนล ที่สามารถเข้าถึงได้จากภายในสคริปต์ Vue โปรดระมัดระวัง ขณะใช้ตัวแปรในสคริปต์ Vue เนื่องจากชื่อของตัวแปรเหล่านั้น หรือความหมายของตัวแปรเหล่านั้นสามารถเปลี่ยนแปลงได้ระหว่างรีลีสที่ต่างกันของ AIX ตัวแปรเคอร์เนลเหล่านี้ทั้งหมดจะถูกตรึงไว้ในหน่วยความจำ และเอ็กซ์พอร์ต

จากเคอร์เนล

ตัวแปรเคอร์เนล	คำอธิบาย	ไฟล์ส่วนหัวที่เชื่อมโยง
struct **system_configuration** **_system_configuration**	โครงสร้างคอนฟิกูเรชันระบบ	sys/systemcfg.h
struct var **v**	พารามิเตอร์เคอร์เนลพื้นฐานที่สามารถปรับแต่งได้ (และอื่นๆ)	sys/var.h
struct timestruc_t **tod**	หน่วยความจำ เวลาที่แม็พของนาฬิการายวัน วินาที และนาโนวินาที ตั้งแต่ช่วงเวลาในอดีตที่สำคัญ	sys/time.h
cpu_t **high_cpuid**	CPU ID แบบโลจิคัลที่สูงที่สุดที่เคยออนไลน์	sys/encap.h
struct vminfo **vmminfo**	โครงสร้างข้อมูลที่มีข้อมูลที่แสดงด้วยคำสั่ง **vmstat**	sys/vminfo.h
time_t **lbolt**	จำนวนของการทำเครื่องหมายตั้งแต่บูตครั้งสุดท้าย	sys/time.h
char **spurr_version**	ระบุว่า ระบบปัจจุบันสนับสนุน SPURR register 0=No SPURR, 1=CPUs have SPURR หรือไม่	sys/sysinfo.h
struct **utsname utsname**	โครงสร้างชื่อระบบที่สอดแทรกชื่อระบบปฏิบัติการ ชื่อโหนด ระดับของรีลีส และอื่นๆ	sys/utsname.h

แบบจำลองข้อมูลสำหรับการประมวลผลแบบ 32 บิต และ 64 บิต

AIX สนับสนุนสภาวะแวดล้อมการพัฒนาสองแบบ: สภาวะแวดล้อมการพัฒนาแบบ 32 บิต และแบบ 64 บิต ดังนั้น คอมไพล์เลอร์บน AIX จะนำเสนอแบบจำลองการโปรแกรมมิงสองแบบดังนี้:

ILP32 สภาวะแวดล้อมการโปรแกรม ILP32 ตัวย่อสำหรับเลขจำนวนเต็ม แบบยาว และตัวชี้ 32 นั่นคือ 32 บิต ใน AIX แบบจำลองข้อมูล ILP32 จะจัดเตรียมพื้นที่แอดเดรสขนาด 32 บิตด้วยหน่วยความจำที่จำกัดที่ 4 GB

LP64 สภาวะแวดล้อมการโปรแกรม LP64 เลขจำนวนเต็มแบบยาว และตัวชี้ 64 นั่นคือ 64บิต บน AIX ด้วยข้อยกเว้นของขนาดของชนิดข้อมูลและการจัดตำแหน่ง LP64 สนับสนุนคุณลักษณะโปรแกรมมิงเช่นเดียวกับแบบจำลอง ILP32 และจะเข้ากันได้กับรุ่นที่ผ่านมาที่ใช้ชนิดข้อมูล int

โปรแกรม AIX สามารถคอมไพล์เพื่อรันเป็นโปรแกรมแบบ 32 บิตหรือ 64 บิตก็ได้ สคริปต์ Vue ตัวเดียวกันสามารถออกคำสั่งสำหรับการประมวลผลที่รันบนโหมด 32 บิต หรือ 64 บิต สำหรับข้อกำหนดคุณสมบัติของข้อมูล ตัวแปรภายนอกชนิดในแบบยาวจะเข้าถึงได้ในสคริปต์ Vue ต้องถูกใช้เป็น 4 ไบต์แบบยาว เมื่อโพรบ (หรือติดตามแล้ว) การประมวลผลคือการประมวลผลแบบ 32 บิต ตัวแปรเดียวกัน ต้องใช้ด้วยขนาด 8 ไบต์แบบยาว เมื่อการประมวลผลที่โพรบแล้วอยู่ในการประมวลผลแบบ 64 บิต โครงร่างและขนาดของโครงสร้างหรือ union ต้องมีสมาชิกที่มีตัวชี้ หรือตัวแปรแบบยาว ซึ่งจะขึ้นอยู่กับมุมมองจากเปอร์สเปคทีฟของ การประมวลผลแบบ 32 บิตหรือการประมวลผลแบบ 64 บิต หากต้องการหลีกเลี่ยงความสับสน Vue ได้จัดเตรียมกฎของความหมาย สำหรับการจัดการแบบจำลองข้อมูลต่างๆ ด้วยวิธีแบบโลจิคัลและสอดคล้องกับ คลาสของตัวแปร

ชนิดตัวแปรที่มีขนาดคงที่

ชนิดตัวแปรต่อไปนี้มีขนาดคงที่ ซึ่งจะมีขนาดเดียวกันในโหมด 32 บิต หรือ 64 บิต โดยไม่คำนึงถึงคลาสของตัวแปร ที่ประกาศไว้

ชนิด	ขนาด
long long	8
int	4
short	2
char	1

ชนิดตัวแปรที่มีขนาดที่ผันแปร

ชนิดตัวแปรต่อไปนี้มีทั้งในโหมด 32 บิต และ 64 บิต:

ชนิด	ขนาดแบบ 32 บิต	ขนาดแบบ 64 บิต
long	4	8
ชนิดของตัวชี้	4	8

ในตารางก่อนหน้านี้ ชนิดของตัวชี้จะอ้างถึงชนิดที่เหมือนกับ **char** *, **int** *, **struct foo** *, **unsigned long** * และอื่นๆ

กฎความหมายต่อไปนี้จะใช้กับตัวแปรที่ถูกนิยาม ด้วยชนิดที่นำหน้าใดๆ นั่นคือ สำหรับ ″longs″ และ ″pointers″ กฎจะใช้ตัวแปรที่เป็นสมาชิกของโครงสร้าง หรือ union หรือประกาศตัวแปรเป็นตัวแปรแต่ละตัว:

คลาสแบบอัตโนมัติ
: โหมดของตัวแปรที่จะขึ้นอยู่กับโหมดของการประมวลผลโพรบ (32 หรือ 64)

คลาสของเธรดแบบโลคัล
: โหมดของตัวแปรที่จะขึ้นอยู่กับโหมดของการประมวลผลโพรบ (32 หรือ 64)

คลาสโกลบอล
: ตัวแปรที่ใช้เป็นโหมด 64 บิต โดยไม่คำนึงถึงโหมดของการประมวลผลที่โพรบแล้ว ซึ่งอนุญาตให้ตัวแปรถูกใช้อย่างปลอดภัยโดยการประมวลแบบ 32 บิตและ 64 บิตโดยเสียข้อมูลใดๆ

เคอร์เนลโกลบอลคลาส
: ตัวแปรเคอร์เนลที่เป็นเลขจำนวนเต็มแบบยาวหรือตัวชี้จะอยู่ในโหมด 64 บิตเนื่องจากมีการสนับสนุนเคอร์เนลสำหรับ AIX 6.1 และใกล้กับเคอร์เนล 64 บิต

คลาส entry
: ถ้าตัวเลขแบบยาวหรือชนิดตัวชี้ถูกกำหนดในฟังก์ชันต้นแบบสำหรับพารามิเตอร์ใดๆ ในฟังก์ชัน โหมดของตัวแปรคลาส entry ที่สอดคล้องกัน (**__arg1** จนถึง **__arg32**) จะขึ้นอยู่กับโหมดของการประมวลผลโพรบ (32 หรือ 64)

คลาส exit
: ถ้าตัวเลขแบบยาวหรือชนิดตัวชี้ถูกกำหนดไว้ในฟังก์ชันต้นแบบ สำหรับชนิดของค่าส่งคืนของฟังก์ชัน โหมดของตัวแปรคลาส exit (**__rv**) จะขึ้นอยู่กับโหมดของการประมวลผล ที่โพรบแล้ว (32 หรือ 64)

คลาสในตัว
: ตัวแปรเหล่านี้จะมีชนิดที่มีขนาดไม่ผันแปรด้วยข้อยกเว้นของ **__r3** จนถึง **__r10** แบบในตัวซึ่งถูกกำหนดให้มีชนิด unsigned long และ 32 บิตแบบยาว สำหรับการประมวลผลแบบ 32 บิต และ 64 บิตสำหรับการประมวลผลแบบ 64 บิต

โพรบ **@@BEGIN** และ **@@END** จะออกคำสั่งในโหมด 64 บิตเสมอ

ชนิดข้อมูลใน Vue

ภาษา Vue จะยอมรับชนิดข้อมูลพิเศษที่เพิ่มเติมจากชนิดข้อมูล C-89 แบบเก่า

ชนิดข้อมูลที่มาจากภาษา C

ภาษา Vue จะสนับสนุนชนิดข้อมูลส่วนใหญ่ที่กำหนดไว้ในข้อกำหนดคุณสมบัติ C-89 ซึ่งประกอบด้วยเวอร์ชันที่มีเครื่องหมายและไม่มีเครื่องหมาย ของชนิดข้อมูลกลุ่ม: **char, short, int, long** และ **long long** ″plain″ **char** จะใช้เป็นแบบไม่มีเครื่องหมาย ขณะที่ชนิดกลุ่มอื่นๆ ถ้าไม่ผ่านการรับรอง จะถูกใช้เป็นแบบมีเครื่องหมาย เครื่องนี้จะจับคู่การนำไปปฏิบัติของ C บน PowerPC ภาษา Vue ยังสนับสนุนชนิดเลขทศนิยม: **float** และ **double** นอกจากชนิดพื้นฐานของภาษา C แล้ว Vue ยังสนับสนุนชนิดที่ได้รับมา เช่น อาร์เรย์ โครงสร้าง และชนิดของตัวชี้ ชนิดการเลียนแบบและชนิดที่ไม่สมบูรณ์ บางชนิด เช่น **void**

ชนิดเหล่านี้ทั้งหมดมีไวยากรณ์และความหมายเหมือนกับใน Vue ซึ่งเทียบเท่ากับชนิดในภาษา C ที่มีข้อยกเว้นดังนี้:

ชนิดเลขทศนิยม
: คุณสามารถใช้ชนิดเลขทศนิยามได้ในการกำหนดนิพจน์ง่ายๆ และเป็นอาร์กิวเมนต์สำหรับฟังก์ชัน Vue เช่น **printf** เท่านั้น โดยเฉพาะอย่างยิ่ง คุณไม่สามารถใช้ตัวแปรเลขทศนิยม เป็นตัวถูกดำเนินการของ unary หรือตัวดำเนินการแบบไบนารี ที่ไม่ใช่ตัวดำเนินการกำหนดค่า

ชนิดของตัวชี้
: คุณสามารถใช้ตัวชี้เพื่อยกเลิกการอ้างถึงเคอร์เนลหรือข้อมูลแอ็พพลิเคชัน อย่างไรก็ตาม คุณไม่สามารถประกาศตัวชี้เป็นตัวแปรสคริปต์ Vue หรือใช้แอดเดรสได้

อาร์เรย์อักขระ
: คุณไม่สามารถใช้อักขระอาร์เรย์เป็นสตริงในภาษา C แต่ต้องใช้ชนิดข้อมูลสตริง

ชนิดที่ไม่สมบูรณ์
: คุณไม่สามารถใช้ชนิดอาร์เรย์ที่มีขนาดที่ไม่รู้จัก

ชนิดฟิลด์บิต
: คอมไพลเลอร์ Vue จะละเว้นการประกาศฟิลด์บิต และโครงร่างของโครงสร้างหรือชนิด union ที่มีสมาชิก ที่เป็นฟิลด์บิต ซึ่งถูกยกเลิกการนิยาม

โมเดลข้อมูล ILP32 และ LP64
: โดยทั่วไป โปรแกรมภาษา C สามารถคอมไพล์ในโหมด 32 บิต โดยที่ปฏิบัติตามแบบจำลอง ILP32 หรือในโหมด 64 บิต โดยที่ปฏิบัติตามแบบจำลอง LP64 เนื่องจาก Vue clause ที่เหมือนกันสามารถออกคำสั่งได้ทั้งการประมวลผลแบบ 32 บิตและ 64 บิต Vue จะสนับสนุนทั้งสองแบบในเวลาเดียวกัน

ชนิดข้อมูล Range และ bucket

ชนิดข้อมูล range ใน Vue ถูกออกแบบมาเพื่อจัดการ การแจกแจงจุดข้อมูลสำหรับช่วงที่กำหนดแน่นอน ในแต่ละช่วง ของตัวแปรถูกกำหนดด้วยชนิดข้อมูล range ที่แสดงจำนวน นับอิลิเมนต์ภายในค่าช่วงที่สอดคล้องกัน ชนิด ข้อมูล range จะสนับสนุนชนิด Integral และ String ของช่วง การแจกแจงของค่าช่วงสำหรับช่วง integral สามารถเป็นการแจกแจง ของเลขยกกำลังของสอง หรือการแจกแจงเชิงเส้น ตัวอย่างของการแจกแจง เชิงเส้นและแบบยกกำลังของสองของค่าช่วงดังนี้:

การกระจายแบบเชิงเส้น:

ช่วง	Count
0 – 5	2
5 – 10	4
10 – 15	1
อื่นๆ	20

การกระจาย **Power® 2:**

ช่วง	Count
1 – 2	2
2 – 4	9
4 – 8	2005
8 – 16	4
16 – 32	1999
32 – 64	7
อื่นๆ	5

การแจกแจงก่อนหน้านี้ระบุจำนวนนับ ของอิลิเมนต์ที่มากกว่าหรือเท่ากับขอบล่าง ของช่วง และน้อยกว่าขอบบนของค่าช่วง ตัวอย่าง เช่น ในการแจกแจงของยกกำลังสอง จำนวนข้อมูลเริ่มต้น จาก 4 และน้อยกว่า 8 คือ 2005 จำนวนของค่าที่ไม่อยู่ ภายใต้ช่วงที่กำหนดจะถูกแสดงในช่วง อื่นๆ

ตัวอย่างของช่วงของสตริง

ตัวอย่าง:

ช่วง	Count
Read, write, open	87
Close, foo1	3
foo2	1
อื่นๆ	51

ในตัวอย่างก่อนหน้านี้ การแจกแจงระบุ จำนวนครั้งที่สตริงหนึ่งๆ เกิดขึ้นภายในค่าช่วง ในตัวอย่างนี้ read, write และ open ถูกเรียกใช้ 87 ครั้ง

การประกาศ และการกำหนดค่าเริ่มต้น ของชนิดข้อมูล range:

ชนิดข้อมูล range สามารถประกาศโดยใช้คีย์เวิร์ด `range_t` ตัวอย่างเช่น การประกาศต่อไปนี้ในสคริปต์ Vue จะกำหนด ตัวแปรชนิดข้อมูล range สองตัวแปร:

```
range_t T1, T2;  // T1 และ T2 คือตัวแปรชนิดข้อมูล Range
```

โดยรูทีน `set_range` และ `add_range` ถูกใช้เตรียมข้อมูลเบื้องต้นช่วง integral และสตริงสำหรับตัวแปรชนิดข้อมูล range เฉพาะใดๆ

การเตรียมข้อมูลเบื้องต้นชนิดข้อมูล integral range: รูทีน `set_range` จะถูกใช้เตรียมข้อมูลเบื้องต้นช่วง integral ไวยากรณ์ของ `set_range` จาก จากการแจกแจงเชิงเส้นและยกกำลังของสองของค่าช่วง ไวยากรณ์ ของรูทีน `set_range` สำหรับการแจกแจงเชิงเส้น เป็นดังนี้:

```
void set_range(range_t range_data, LINEAR, int min, int max, int step);
```

ตัวอย่าง:

```
set_range(T1, LINEAR, 0, 100, 10);
```

ในตัวอย่างก่อนหน้านี้ รูทีน set_range เตรียมข้อมูลเบื้องต้นข้อมูลช่วง T1 ข้อมูลช่วง T1 มีการแจกแจงเชิงเส้น ของค่า ขอบล่างของ T1 คือ 0 และขอบบนคือ 100 ขนาดของแต่ละช่วงคือ 10 การแจกแจงสำหรับตัวอย่างก่อนหน้านี้ จะมีคล้ายต่อไปนี้:

ช่วง	Count
0 - 10	...
10 - 20	...
20 - 30	...
...	...
...	...
90 - 100	...

ไวยากรณ์สำหรับการกำหนดค่าเริ่มต้นของการแจกแจง ยกกำลังของ 2 เป็นดังนี้:

```
set_range(range_t range_data, POWER, 2);
```

ตัวอย่าง:

```
set_range(T2, POWER, 2);
```

ในตัวอย่างนี้ รูทีนเตรียมข้อมูลเบื้องต้นชนิดข้อมูล range ของ T2 เป็นชนิดช่วงการแจกแจงของยกกำลังของ 2

การเตรียมข้อมูลเบื้องต้นชนิดข้อมูล range สตริง: รูทีน add_range เตรียมข้อมูลเบื้องต้นชนิดข้อมูลช่วง range สตริง

ไวยากรณ์:

```
void add_range(range_t range_data , String S1, String S2, ..., String Sn);
```

ตัวอย่าง:

```
add_range(T1, "read", "write", "open");
```

รูทีนนี้เพิ่มสตริง read, write และ open ใน สล็อตเดียวของข้อมูล range_t T1 อีก **add_range** บน ข้อมูล range_t T1 เดียวกันเพิ่มสตริงใน สล็อตถัดไป

```
add_range(T1, "close", "func1", "func2");
```

รูทีนนี้เพิ่มสตริง close, func1 และ func2 ไปยังข้อมูล range_t T1 ในสล็อตถัดไป

หมายเหตุ: ชนิดข้อมูล range **range_t** เป็นชนิดข้อมูล พิเศษสำหรับ Vue ซึ่งสามารถใช้เพื่อเก็บเป็นค่าภายในอาร์เรย์ ที่สัมพันธ์กันเท่านั้น สำหรับการดำเนินการอื่นใด (เช่น ทางคณิตศาสตร์ โลจิคัล, bitwise และเชิงสัมพันธ์) ในชนิดข้อมูล **range_t** การดำเนินการจะล้มเหลว และเกิดข้อผิดพลาด

หมายเหตุ: ข้อมูลนี้ อธิบายการใช้งานและรูทีนการกำหนดค่าเริ่มต้นต่างๆ ของชนิดข้อมูล **range_t**

1. การประกาศของชนิดข้อมูล **range_t** สามารถทำใน คำสั่ง @@BEGIN เท่านั้น
2. การกำหนดค่าเริ่มต้นของรูทีน **set_range** สามารถใช้ภายใน คำสั่ง @@BEGIN เท่านั้น
3. ชนิดข้อมูลช่วงที่ค่าของช่วงเป็น integral สามารถกำหนดค่าเริ่มต้น ได้ครั้งเดียว ตัวแปรเดียวกันไม่สามารถกำหนดค่าเริ่มต้นได้สองครั้ง

   ตัวอย่าง:

```
set_range(T1, LINEAR, 0, 50, 5);        // Valid syntax
    set_range(T1, LINERA, 10, 100, 10); // Error, cannot initialize an already
                                        // initialized T1.
    set_range(T1, POWER, 2);            // Error, T1 has already initialized.
    add_range(T1, "read", "write");     // Error, T1 has already initialized.
```

4. พารามิเตอร์ของ min, max และ step เป็น ค่าคงที่สำคัญสำหรับรูทีน **set_range**

การเก็บและการพิมพ์ชนิดมูล range:

ชนิดข้อมูล range สามารถเก็บในอาร์เรย์ เชิงสัมพันธ์กับค่าโดยใช้รูทีน qrange รูทีน qrange ค้นหาหมายเลขสล็อตที่มีความถี่และจำนวนที่ต้องการเพิ่ม

ตัวอย่าง:

สำหรับตัวอย่างนี้ T1 คือชนิดข้อมูล **range_t** ที่มีค่าช่วงเป็นชนิด integral

```
qrange(aso["read"], T1, time_spent);
```

ในตัวอย่างนี้ รูทีน qrange ค้นหาหมายเลขสล็อตซึ่ง *time_spent* ล้มเหลวและ จำนวนสำหรับหมายเลขสล็อตนั้นที่ถูกเพิ่มสำหรับอาร์เรย์ aso ที่สัมพันธ์ สอดคล้องกับคีย์ที่อ่าน

ในตัวอย่างต่อไปนี้ T2 คือชนิดข้อมูล **range_t** และค่าช่วยที่มีชนิดสตริง

```
qrange(aso["function usage"], T2,get_function());
```

ในตัวอย่างนี้ รูทีน qrange ค้นหาหมายเลขสล็อตซึ่งฟังก์ชันถูกส่งเป็นอาร์กิวเมนต์ที่สาม ล้มเหลว และเพิ่มจำนวนสำหรับสล็อตนั้นสำหรับอาร์เรย์ aso ที่ สัมพันธ์กันที่สอดคล้องกับคีย์ **การใช้ฟังก์ชัน**

หมายเหตุ:

1. สำหรับ ASO มีตัวแปรชนิด **range_t** หนึ่งตัวเท่านั้นที่สามารถเก็บ เป็นค่า การใช้ **qrange** สำหรับชนิดต่างกันของชนิดตัวแปร **range_t** สำหรับ ASO เดียวกันจะล้มเหลว

   ตัวอย่าง:

   ```
   qrange(aso["read"], T1,time_spent);  // Correct syntax.
   qrange(aso["read"], T2,time_spent);  // Error. Two different range_t types
                                        // cannot be used for the same ASO.
   ```

   ฟังก์ชัน quantize และ lquantize ของอาร์เรย์ที่สัมพันธ์กันที่มีชนิดค่าเป็น **range_t** แสดง quantization เสมือนของความถี่และจำนวนนับของช่วง
2. ขณะพิมพ์ช่วงสตริงสูงสุด 40 อักขระ (รวม เครื่องหมายจุลภาค) สามารถพิมพ์ได้สำหรับสล็อตเฉพาะ ถ้าสตริงในสล็อตมีมากกว่า 40 ตัวอักษร ช่วงของสตริง จะถูกตัดปลาย และพิมพ์ด้วยอักขระ 3 ตัวสุดท้ายในรูปของ จุด(...)

ตัวอย่างของชนิดข้อมูล range และรูทีน qrange:

```
@@BEGIN
{
  __thread start ;
  range_t T1;
  set_range(T1, LINEAR, 0, 150, 10) ;
}
@@syscall :$__CPID :read :entry
```

```
{
  thread :tracing = 1 ;
  start = timestamp() ;
}
@@syscall :$__CPID :read :exit
         when(thread :tracing == 1)
{
  __auto long time_spent;
  currtime = timestamp() ;
  time_spent = diff_time(start, currtime, MICROSECONDS);
  qrange(aso[“read”], T1, time_spent);
}
@@END
{
  print(aso);
  quantize(aso);
}
```

ต้องการเอาต์พุตสำหรับตัวอย่างนี้:

คีย์ **ค่า**

```
Read            Range                   count
                0-11                    4
                10-20                    6
                60-70                    7
                Others                  32
```

คีย์ **ค่า**

```
Read            Range                   count
                0-10                    4       ===
                10-20                    6       ====
                60-70                    7       =====
                Others                  32      ================
```

ชนิดการติดตามสแต็ก

ตัวแปรของ **stktrace_t** ถูกใช้เพื่อเก็บค่าที่ส่งคืนจากฟังก์ชัน ProbeVue **get_stktrace** ซึ่งจะคืนค่าการติดตามสแต็กปัจจุบัน การติดตามสแต็กที่ส่งคืนเป็นการติดตามสแต็กของเธรดปัจจุบัน ตัวแปรนี้ยังสามารถถูกเก็บในอาร์เรย์ที่เชื่อมโยงทั้งที่เป็นคีย์และเป็นค่า ชนิด **stktrace_t** เป็นชนิดของมูลแบบ abstract และตัวแปรนี้ไม่สามารถใช้โดยตรงกับตัวดำเนินการ uninary หรือ binary ภาษา C มาตรฐาน ตัวแปรนี้เป็นอาร์เรย์แบบ longs ที่ไม่มีเครื่องหมาย

Vue สนับสนุนลักษณะและการดำเนินการต่อไปนี้สำหรับตัวแปรชนิดการติดตามสแต็ก:

การประกาศของตัวแปรชนิดการติดตามสแต็ก

ตัวแปรสามารถประกาศให้เป็นชนิดการติดตามสแต็กโดยการประกาศดังในสคริปต์ต่อไปนี้:

```
      stktrace_t  st;              // st is a stktrace_t variable.
      st = get_stktrace(5);         //  Get the stack trace up to five levels.
      a_st[0] = get_stktrace(-1);  // Get the stack trace up to the extent possible and
                                  // store in the associative array a_st as value.
```

ไม่สนับสนุน qualifiers signed, unsigned, register, static, auto, thread, kernel, และ const สำหรับตัวแปรชนิด **stktrace_t**

การดำเนินการการกำหนดค่า

การกำหนดค่าตัวดำเนินการ **(=)** อนุญาตให้ใช้ตัวแปรชนิด **stktrace_t** ที่ต้องกำหนดค่าให้กับ ตัวแปรชนิด **stktrace_t** ค่าเดิมในตัวแปร **stktrace_t** เป้าหมายจะถูกทำลาย ไม่อนุญาตให้ใช้ชนิด casting จากหรือกับชนิดตัวแปร **stktrace_t** ในตัวอย่างต่อไปนี้ เนื้อหาของการติดตามสแต็ก *t1* ถูกกำหนดให้กับ *t2*

```
stktrace_t     t1, t2;          // Declares two stack trace variables.
t1 = get_stktrace();            // Get the current stack trace in t1.
t2 = t1 ;                       // Get the content of t1 into t2.
```

การดำเนินการเปรียบเทียบ

อนุญาตให้ใช้เฉพาะตัวดำเนินการเท่ากับ **(==)** และไม่เท่ากับ **(!=)** สำหรับตัวแปร **stktrace_t** ผลลัพธ์ของตัวดำเนินการเหล่านี้จะเป็น True(1) หรือ False(0)ตามรายการทั้งหมดของตัวแปร **stktrace_t** ไม่อนุญาตให้เปรียบเทียบรายการแต่ละรายการของตัวแปร **stktrace_t** ไม่อนุญาตให้ใช้โอเปอเรเตอร์การเปรียบเทียบอื่น (>=, >, < หรือ =<) สำหรับตัวแปรชนิด **stktrace_t**

```
if( t1 == t2)                  // comparing two stktrace_t type variables.
        printf("Entries are similar");
  else
        printf("Entries are not similar");
```

การพิมพ์ตัวแปรชนิดการติดตามสแต็ก

คุณสามารถพิมพ์ตัวแปร **stktrace_t** ด้วยตัวระบุรูปแบบ **%t** ในฟังก์ชัน **printf** ของ Vue เอาต์พุตจะเป็นการติดตามสแต็ก สัญลักษณ์ของเธรดที่ถูกบันทึกในตัวแปร สัญลักษณ์ที่มีแอดเดรส (สัญลักษณ์บวกกับแอดเดรส) จะถูกพิมพ์เฉพาะเมื่อเธรดที่สอดคล้องกับตัวแปร **stktrace_t** อยู่ในสถานะที่กำลังทำงาน ไม่เช่นนั้นจะพิมพ์เฉพาะเฉพาะการติดตามสแต็กที่เป็นแอดเดรสสำหรับตัวแปร

ตัวแปรชนิด **stktrace_t** ที่เก็บอยู่ในอาร์เรย์ที่เชื่อมโยงทั้งที่เป็นคีย์หรือค่าสามารถพิมพ์ได้ด้วยฟังก์ชัน print สำหรับอาร์เรย์ที่เชื่อมโยง แอดเดรสพร้อมกับสัญลักษณ์ (ชื่อสัญลักษณ์ + ออฟเซ็ต) จะถูกพิมพ์ถ้าเธรดที่สอดคล้องกับชนิด **stktrace_t** ที่ถูกเก็บในอาร์เรย์ที่เชื่อมโยงกำลังรันอยู่ ไม่เช่นนั้นจะพิมพ์เฉพาะแอดเดรส

```
stktrace_t t1;
t1 = get_stktrace (5);
printf ("%t", t1);       // Displays the stack trace stored in variable t1.
a[__tid] = t1;           // Store t1 as value in an associative array a.
 print(a) ;              // Print associative array a, whose value
              // type is stktrace_t variable.
```

ข้อจำกัดสำหรับตัวแปรชนิดการติดตามสแต็ก

- ไม่สามารถประกาศอาร์เรย์ของตัวแปร **stktrace_t**
- ไม่สามารถใช้ตัวแปร **stktrace_t** เป็นสมาชิกของ struct หรือ union
- ไม่อนุญาตให้เข้าถึงรายการใดๆ ของการติดตามสแต็ก
- ไม่สนับสนุนการดำเนินการ (การกำหนดค่า การเปรียบเทียบ และการพิมพ์) ของตัวแปรชนิด **stktrace_t** ในโพรบ systrace

ชนิดข้อมูลพิเศษ

นอกจากชนิดข้อมูล C-89 แบบดั้งเดิมแล้ว ภาษา Vue ยังยอมรับข้อมูลชนิดพิเศษเจ็ดชนิด

ชนิดสตริง

ชนิดข้อมูลสตริงจะแสดงถึงสตริงตามตัวอักษร ไม่เหมือนกับ C สตริงคือชนิดพื้นฐานใน Vue การมีชนิดสตริง จะหลีกเลี่ยงความสับสนบางอย่างใน C ซึ่งไม่สนับสนุนชนิดสตริง แต่อนุญาตให้สตริงแสดงได้โดยตัวชี้ชนิด **char** type และอักขระอาร์เรย์

คุณสามารถประกาศตัวแปรสตริงได้โดยใช้ declaration statement ตัวแปรสตริงที่ประกาศไว้อย่างชัดเจน ต้องระบุความยาวสตริงสูงสุด (คล้ายกับการประกาศอักขระอาร์เรย์ใน C) ไม่เหมือน C สตริงใน Vue จะไม่ถูกยกเลิกโดยอักขระ null และคุณไม่จำเป็นต้องจองพื้นที่สำหรับตัวแปรไว้

```
String s[40];  /* Defines a string 's' of length 40 */
s = "probevue";
```

นอกจากนี้ สตริงตามตัวอักษรใดๆ ที่เขียนในลักษณะของ C ด้วยการครอบด้วยเครื่องหมายอัญประกาศคู่ จะถูกกำหนดให้เป็นชนิดข้อมูลสตริง Vue จะแปลงตัวแปรภายนอกที่ถูกประกาศเป็นชนิดข้อมูลอักขระในลักษณะของ C (**char** * หรือ **char**[]) ให้เป็นชนิดข้อมูลสตริงตามความต้องการ

คุณสามารถใช้ตัวดำเนินการต่อไปนี้สำหรับชนิดข้อมูลสตริง:

- ตัวดำเนินการต่อข้อมูล: "+"
- ตัวดำเนินการกำหนดค่า: "="
- โอเปอเรเตอร์ที่เกี่ยวข้องสำหรับการเปรียบเทียบสตริง: "==", "!=", ">", ">=", "<" และ "<="

คุณสามารถตั้งค่าตัวแปรสตริงให้กับสตริงที่ว่างโดยกำหนด "" ตามตัวอย่างต่อไปนี้:

```
s = "";  /* Sets s to an empty string */
```

ไม่เหมือนกับภาษา C คู่ของสตริงที่เชื่อมต่อเข้าด้วยกันตามตัวอักษร จะไม่เชื่อมต่อโดยอัตโนมัติ ตัวดำเนินการต่อข้อมูล (+) ต้องถูกใช้อย่างชัดเจนในตัวอย่างต่อไปนี้:

```
String s[12];

 // s = "abc" "def";
   /* ERROR: Commented out as this will result in a syntax error */
 s = "abc" + "def"; /* Correct way to concatenate strings */
```

Vue จะสนับสนุนฟังก์ชันต่างๆ ที่ยอมรับชนิดข้อมูลสตริง เป็นพารามิเตอร์ที่มีชนิดข้อมูลสตริง

ชนิดลิสต์

ตัวแปรของชนิดลิสจะเก็บรวบรวมชุดของค่าชนิดกลุ่ม ชนิดลิสต์คือ abstract data type และคุณไม่สามารถใช้ตัวแปรลิสต์ได้โดยตรงด้วยตัวดำเนินการ unary หรือตัวดำเนินการแบบไบนารีในมาตรฐาน C คุณสามารถใช้ตัวดำเนินการต่อไปนี้ สำหรับชนิดลิสต์:

- ฟังก์ชัน constructor, **list( )** เพื่อสร้าง ตัวแปร list ใหม่หากไม่ได้กำหนดไว้ก่อน ถ้าตัวแปรถูกกำหนดไว้แล้ว – ตัวแปรก็ควรถูกเคลียร์ค่า
- ฟังก์ชันการต่อข้อมูล **append** เพื่อเพิ่มรายการให้กับลิสต์หรือเชื่อมสองลิสต์เข้าด้วยกัน
- ตัวดำเนินการ "=" ที่อนุญาตให้ลิสต์ถูกกำหนดให้กับลิสต์อื่น

- ชุดของฟังก์ชันการรวมที่ทำงานบนตัวแปรลิส และส่งคืนค่าสเกลาร์ (เลขจำนวนเต็ม) เช่น **sum, avg, min, max** และอื่นๆ

แม้ว่า คุณจะสามารถใช้ตัวแปรลิสต์เพื่อเก็บรวบรวมค่ากลุ่มใดๆ ค่าจะบันทึกเป็นเลขจำนวนเต็มที่มีเครื่องหมายแบบ 64 บิต

ฟังก์ชัน **list( )** จะส่งคืนลิสต์ว่างใหม่ ที่สามารถกำหนดให้กับตัวแปรของชนิดลิสต์ได้ ซึ่งจะสร้างตัวแปรใหม่ ของชนิดลิสต์ ถ้าตัวแปรลิสต์ทางฝั่งซ้ายของตัวดำเนินการกำหนดค่า ยังไม่ถูกกำหนดค่าในลิสต์ก่อนหน้านั้น ซึ่งอาจถูกกำหนดค่าเป็นตัวแปรลิสต์ก่อนหน้า ในกรณีที่ ค่าใดๆ ที่รวบรวมไว้ในลิสต์เป้าหมายถูกละทิ้ง นอกจากนี้ ตัวแปรสามารถประกาศให้เป็นชนิดลิสต์ได้ โดยประกาศไว้ในสคริปต์ Vue ดังต่อไปนี้:

```
__list l_opens;
```

ผลของการกระทำนี้เหมือนฟังก์ชัน **list( )** ถูกเพิกถอนในโพรบ @@BEGIN และค่าส่งคืนถูกกำหนด ให้กับตัวแปรลิสต์นี้

ตัวอย่างต่อไปนี้จะสร้างตัวแปรลิสต์ใหม่ที่เรียกว่า**l_opens**:

```
l_opens = list();
```

ฟังก์ชัน **list** สามารถเรียกใช้งานจาก clause ใดๆ ได้ ถ้าคุณระบุชื่อลิสต์ที่มีอยู่ขณะเรียกใช้งานฟังก์ชัน **list** ลิสต์ที่มีอยู่นั้นจะถูกลบทิ้ง

คุณสามารถใช้ฟังก์ชัน **append( )** เพื่อเพิ่มค่า ในตัวแปรรายการ แต่ละครั้งที่เรียกฟังก์ชัน **append** ให้เพิ่มค่าใหม่ลงในชุดของค่าที่บันทึกไว้แล้วในตัวแปรลิสต์ ตัวอย่างต่อไปนี้จะแสดงขนาดของตัวแปรลิสต์ที่โตขึ้น พร้อมกับการเรียกฟังก์ชัน **append**:

```
append(l_opens, n_opens1); /* l_opens = {n_opens1} */
append(l_opens, n_opens2); /* l_opens = {n_opens1, n_opens2} */
append(l_opens, n_opens3); /* l_opens = {n_opens1, n_opens2, n_opens3} */
append(l_opens, n_opens4); /* l_opens = {n_opens1, n_opens2, n_opens3, n_opens4} */
```

พารามิเตอร์ที่สองไปยังฟังก์ชัน **append( )** ยังสามารถเป็นตัวแปรชนิดรายการซึ่งจะผนวกค่าทั้งหมด ท้ายรายการปลายทางที่ระบุโดยพารามิเตอร์แรก ดังนั้นฟังก์ชัน append สามารถใช้ในการเชื่อมต่อ ลิสต์สองชุดเข้าด้วยกัน

ในตัวอย่างต่อไปนี้ เนื้อหาของลิสต์ b จะถูกเพิ่มให้กับ a:

```
a=list()
b=list()
append(a,b)
```

หมายเหตุ: ค่าที่เพิ่มให้กับลิสต์ต้องเป็นพารามิเตอร์ของชนิดลิสต์หรือกลุ่ม และอาจมีข้อผิดพลาดได้หากตัวแปรใดๆ **n_opens1 –n_opens4** ไม่มีชนิดกลุ่ม ชนิดใดๆ ที่เล็กกว่า **long long** (เช่น short หรือ int) อาจถูกพัฒนาเป็นชนิด **long long**

คุณยังสามารถใช้ append เพื่อเชื่อมลิสต์สองตัวเข้าด้วยกัน อาร์กิวเมนต์แรกคือ ลิสต์เป้าหมาย และอาร์กิวเมนต์ตัวที่สองคือ ลิสต์ของซอร์สลิสต์ ในตัวอย่างต่อไปนี้ เนื้อหาของลิสต์ b จะถูกเพิ่มให้กับ a:

```
a=list()
b=list()
append(a,b)
```

ฟังก์ชัน **append( )** ไม่มีค่าส่งคืน

ลิสต์สามารถกำหนดให้กับลิสต์อื่นได้โดยใช้ตัวกำหนดค่า ค่าเดิมในลิสต์เป้าหมายจะถูกลบทิ้ง ในตัวอย่างต่อไปนี้ เนื้อหาของลิสต์ **l_opens2** จะหายไป (รายการจะถูกลบทิ้ง) และเนื้อหาของลิสต์ **l_opens** จะถูกคัดลอกไปยังลิสต์ **l_opens2**

```
l_opens2 = list();
append(l_opens2, n_opens5);

l_opens2 = l_opens;
  /* l_opens and l_opens2 => {n_opens1, n_opens2, n_opens3, n_opens4}  */
```

ฟังก์ชันการรวมสามารถใช้กับตัวแปรลิสต์ตามที่แสดงอยู่ใต้วอย่างต่อไปนี้ :

```
/* below we assume n_opens1=4, n_opens2=6, n_opens3=2 and n_opens4 = 4
 * at the time they were added to the l_opens list variable
 */
x = avg(l_opens); /* this will set x to 4 */
y = min(l_opens); /* this will set y to 2 */
z = sum(l_opens); /* this will set z to 16 */
a = count(l_opens) /* this will set a to 4 */
```

ตัวแปรลิสจะมีประโยชน์เมื่อค่าที่รวมกันที่ถูกต้องต้องการเก็บเร็กคอร์ดไว้ ตัวแปรลิสต์จะถูกอัพเดตโดยอัตโนมัติ ดังนั้น ใช้ลิสต์เมื่อจำเป็น เนื่องจากลิสต์มีประสิทธิภาพน้อยกว่าตัวแปรปกติอื่นๆ

ชนิดของอาร์เรย์ที่เชื่อมโยง

อาร์เรย์แบบเชื่อมโยงเป็นการแม็พหรือตารางการค้นดูที่ประกอบด้วยชุดของคีย์และค่าที่เกี่ยวข้องของคีย์ มีการแม็พแบบหนึ่งต่อหนึ่งระหว่างชุดของคีย์และค่า อาร์เรย์ที่สัมพันธ์กันจะได้รับการสนับสนุน โดย Perl, ksh93 และภาษาอื่นๆ อีกหลายภาษา

ใน Vue แต่ละการแม็พมาจากหนึ่งหรือหลายคีย์ (มิติ) กับค่าเดียว คีย์ Associative Array สามารถมีชนิดต่อไปนี้

- integral
- floating point (จุดทศนิยม)
- สตริง
- การประทับเวลา
- stacktrace
- path
- MAC แอดเดรส
- IP แอดเดรส

ค่า Associative Array สามารถมีชนิดต่อไปนี้

- integral
- floating point (จุดทศนิยม)
- สตริง
- การประทับเวลา
- stacktrace
- path
- MAC แอดเดรส
- IP แอดเดรส
- list

- ช่วง

Associative arrays เป็นชนิดข้อมูล abstract ใน Vue แอ็คชันต่อไปนี้สามารถดำเนินการบน ชนิดข้อมูล associative array

- การเชื่อมโยงคีย์กับค่า:

  หากยังไม่มีอินสแตนซ์ของคีย์ แอ็คชันนี้จะเพิ่มคีย์หรือชุดของคีย์ พร้อมกับค่าที่เชื่อมโยงเข้ากับ associative array ไม่เช่นนั้น แอ็คชันนี้จะแทนที่ ค่าเก่าที่เชื่อมโยงด้วยค่าใหม่ คีย์ที่ไม่เชื่อมโยงจะมีค่าดีฟอลต์เป็น 0 สำหรับชนิด `numerical` ค่าสตริงว่างสำหรับชนิด `string` หรือค่า `NULL` สำหรับคีย์ชนิดอื่น

  ตัวอย่างต่อไปนี้อธิบายการเชื่อมโยงคีย์กับ ค่า

```
/* single key dimension */
count["ksh"] = 1;
/* multiple key dimensions */
var[0][“a”][2.5] = 1;
var[1][“a”][3.5] = 2;
```

  การใช้ตัวแปร associative array เป็นครั้งแรกจะตั้งค่าชนิดของคีย์ ขนาดมิติของคีย์ และชนิดของค่า ซึ่งยังคงต้องเหมือนกันกับตำแหน่งอื่นทั้งหมดในสคริปต์ Vue

  สำหรับคีย์ใน ASO คุณสามารถเชื่อมโยงค่า LIST โดยดำเนินการแอ็คชัน ที่แตกต่างกันต่อไปนี้:

  1. โดยการกำนดค่าตัวแปร LIST:

```
assoc_array["ksh"]=ll  /* copies ll list into associative array */
assoc_array["ksh"]=assoc_array["abc"]; /* copies a list in ASO to another list in ASO.
                                          ต่อไปนี้เป็นชนิดค่าของ assoc_array ใน LIST */
```

  2. โดยการกำหนดค่ารายการว่างที่ส่งคืนโดยฟังก์ชัน **list( )** constructor:

```
assoc_array["ksh"]=list(); /* assigns an empty list */
```

  3. โดยการผนวกรายการหรือค่า integral ต่อท้าย

```
append(assoc_array["ksh"], 5); /* integral value 5 is appended to the list in ASO */
append(assoc_array["ksh"], ll); /* appends the contents of LIST variable ll to the list in ASO*/
append(assoc_array["ksh"], assoc_array["abc"]); /* appends the contents of list in ASO to another list in ASO */
```

- การยกเลิกการเชื่อมโยงคีย์หรือชุดของคีย์และการลบค่าที่เชื่อมโยง: ฟังก์ชัน **delete( )** ใช้เพื่อลบคีย์และค่าที่เชื่อมโยงจาก associative array คีย์ที่ยกเลิกการเชื่อมโยงถูกกำหนดค่าเป็น 0 หรือสตริงว่าง

  ตัวอย่างต่อไปนี้อธิบายวิธีใช้ ฟังก์ชันลบสำหรับการยกเลิกการเชื่อมโยง คีย์

```
delete(count, "ksh");
delete(var, 0, “a”, 2.5);
```

  อาร์กิวเมนต์แรกเป็นชื่อของตัวแปร associative array ชื่อตัวแปรอาร์เรย์ ต้องตามด้วยคีย์ที่คั่นด้วยเครื่องหมายคอมม่า จำนวน N โดยที่เป็นขนาดมิติของคีย์ หากคุณต้องการลบเฉพาะ ค่าที่เชื่อมโยงบนพื้นฐานของมิติของคีย์ที่ไม่ใช่ N คุณสามารถระบุ ANY สำหรับมิติของคีย์นั้น เช่น เมื่อต้องการลบอิลิเมนต์ทั้งหมดที่มีสตริง “a” เป็นมิติที่สอง ให้ป้อน คำสั่งต่อไปนี้

```
delete(var, ANY, “a”, ANY);
```

| คีย์ทั้งหมดในฟังก์ชัน delete() สามารถระบุเป็น ANY ในกรณีนี้ อิลิเมนต์ทั้งหมดของ associative จะถูกลบออก ฟังก์ชันนี้
| จะคืนค่า 0 หากพบอิลิเมนต์ที่ตรงกัน และจะถูกลบออก ไม่เช่นนั้น ฟังก์ชัน delete() จะคืนค่าที่เป็น 1

- | การค้นหาค่าสำหรับชุดของคีย์: การดำเนินการนี้จะค้นหาค่าที่ เชื่อมโยงกับคีย์เดียวหรือหลายคีย์

```
total = count["ksh"] + count["csh"];
prod = var[0][“a”][2.5] * var[1][“a”][3.5];
```

ค่า LIST สำหรับคีย์สามารถเรียกค้นโดยทำดัชนี อาร์เรย์ที่สัมพันธ์กันกับคีย์ ฟังก์ชัน LIST ทั้งหมด sum(), min(), max(), count() และ avg() สามารถใช้บน List ในอาร์เรย์ Associative คุณยังสามารถกำหนดรายการในอาร์เรย์ที่สัมพันธ์กันกับตัวแปร LIST

ตัวอย่าง:

```
/* copies associative array list into list variable "ll" */
ll=assoc_array["ksh"];
/* prints the sum of all elements of list in associative array indexed with ksh" */
printf("sum of assoc_array %d\n",sum(assoc_array["ksh"]) );
/* prints the minimum value */
printf("min of assoc_array %d\n",min(assoc_array["ksh"]) );
/* prints the maximum value */
printf("max of assoc_array %d\n",max(assoc_array["ksh"]) );
/* prints the number of values in list */
printf("count of assoc_array %d\n",count(assoc_array["ksh"]) );
/* prints average value of the list */
printf("avg of assoc_array %d\n",avg(assoc_array["ksh"]) );
```

- | การตรวจสอบว่าคีย์หรือชุดของคีย์มีอยู่หรือไม่: ฟังก์ชัน exists() จะตรวจสอบว่า associative array มีอิลิเมนต์ใดๆ ที่สอด
| คล้องกับคีย์ที่กำหนดหรือไม่ ฟังก์ชัน exists() จะคืนค่าที่เป็น 1 หากพบอิลิเมนต์ และฟังก์ชัน จะคืนค่า 0 หากไม่พบอิลิ
| เมนต์

| บล็อกของโค้ดต่อไปนี้จะตรวจสอบว่าคีย์หรือชุดของคีย์ มีอยู่หรือไม่

```
| if (exists(count, "ksh"))
|   printf("Number of ksh calls = %d\n", count["ksh"]);
| if (exists(var, 0, “a”, 2.5))
|   printf(“Found value = %d\n”, var[0][“a”][2.5]);
```

| หาก คุณระบุคีย์เวิร์ด ANY สำหรับมิติของคีย์เฉพาะเจาะจง มิติจะกลายเป็น ไม่มีนัยสำคัญสำหรับการดำเนินการค้นหา คีย์
| ทั้งหมดในฟังก์ชัน exists() สามารถระบุเป็น ANY ในกรณีนี้ ฟังก์ชัน exists() จะตรวจสอบว่า Associative Array มีอิลิ
| เมนต์ใดๆ หรือไม่

```
| my_key = “a”;
| if (exists(var, ANY, my_key, ANY))
|   printf(“Found element with second key as %s \n”, my_key);
```

- | การดำเนินการเพิ่มค่าและลดค่า: การดำเนินการนี้สามารถใช้ เพื่อเพิ่มหรือลดค่าอาร์เรย์ที่สัมพันธ์กัน เมื่อต้องการใช้การ
| ดำเนินการนี้ คุณต้องระบุเลขจำนวนเต็มเป็น ชนิดของค่าสำหรับคีย์ ตัวอย่างต่อไปนี้แสดงการใช้การดำเนินการ increment
| และ decrement :

1. printf(“Incremented value = %d\n”, ++count[“ksh”]);
2. printf(“Incremented value = %d\n”, count[“ksh”]++);
3. printf(“Decremented value = %d\n”, --count[“ksh”]);
4. printf (“Decremented value = %d\n”, count[“ksh”]--);

ในตัวอย่างที่ 1 ค่าที่สอดคล้องกับคีย์ **ksh** จะถูกเพิ่ม และ ค่าที่เพิ่มจะถูกพิมพ์

ในตัวอย่างที่ 2 ค่าที่สอดคล้องกับ **ksh** จะถูกพิมพ์ก่อนและจากนั้น ค่าจะเพิ่มขึ้น การดำเนินการลดค่า ทำงานในแบบเดียวกัน อย่างไรก็ตาม การดำเนินการเพิ่มหรือลดค่า สามารถดำเนินการบนอาร์เรย์ที่สัมพันธ์กันที่มีชนิดค่าเป็นเลขจำนวนเต็มเท่านั้น การดำเนินการเพิ่มหรือลดค่ายังสามารถใช้เป็นตัวรวมค่า โดยชนิดค่าของอาร์เรย์ที่สัมพันธ์กันโดยดีฟอลต์ถูกตั้งค่าเป็น เลขจำนวนเต็ม ตัวอย่างเช่น เมื่อพบข้อความสั่ง a[100]++ เป็น ครั้งแรก อาร์เรย์ที่สัมพันธ์กัน a ถูกสร้างโดยมีชนิดคีย์ เลขจำนวนเต็ม และชนิดค่าเลขจำนวนเต็ม ค่าถูกที่เก็บสำหรับคีย์ 100 คือ 1 อย่างไรก็ตามสำหรับ a[100]-- นั้น -1 จะถูกเก็บเป็นค่าสำหรับคีย์ 100 เมื่อพบการดำเนินการเพิ่มค่าหรือลดค่าต่อมา สำหรับอาร์เรย์ที่มีความสัมพันธ์เดียวกัน การดำเนินการเพิ่มและลดค่า a จะถูกดำเนินการกับค่าสำหรับคีย์ที่ระบุ

ลักษณะการทำงาน increment และ decrement จะเหมือนกันสำหรับ associative arrays ที่มีมากกว่าหนึ่ง มิติคีย์:

```
++var[0]["a"][2.5];
var[0]["a"][2.5]++;
--var[1]["a"][3.5];
var[1]["a"][3.5]--;
```

- การพิมพ์เนื้อหาของ associative array: การดำเนินการนี้จะพิมพ์คีย์และ ค่าที่เชื่อมโยงของอิลิเมนต์สำหรับ associative array คุณสามารถระบุอ็อพชันการพิมพ์ต่อไปนี้:

อ็อพชันการพิมพ์อาร์เรย์ที่เชื่อมโยง	คำอธิบาย	ค่าที่เป็นไปได้	ค่าดีฟอลต์
num-of-entries	ระบุเพื่อพิมพ์ตัวเลขแรกของคู่ คีย์-ค่า	*n*>=0. (ถ้า 0 รายการทั้งหมด ถูกแสดง)	0
sort-type	ระบุลำดับการเรียง	SORT_TYPE_ASCEND, SORT_TYPE_DESCEND	SORT_TYPE_ASCEND
sort-by	ระบุว่าเรียงลำดับตามคีย์ หรือค่า	SORT_BY_KEY, SORT_BY_VALUE	SORT_BY_KEY
list-value	ระบุแอ็ตทริบิวต์ LIST ใดที่จะเรียงลำดับ หรือ quantize เมื่อค่าของอาร์เรย์ที่สัมพันธ์กันเป็นชนิดรายการ	USE_LIST_SUM, USE_LIST_MIN, USE_LIST_MAX, USE_LIST_COUNT, USE_LIST_AVG	USE_LIST_AVG
sort-key-index	ระบุดัชนีคีย์ (มิติ) โดยอิงตามเอาต์พุตที่จะ ถูกเรียงลำดับ	-1 หรือ k โดยที่ 0 <= k < number_of_keys	0

เมื่อแฟล็ก sort-by คือ SORT_BY_KEY, SORT_BY_VALUE และ คู่ของคีย์ (กำหนดโดย sort-key-index) และค่าจะเป็นชนิดที่ไม่สามารถเรียงลำดับได้ ดังนั้น อ็อพชัน *num-of-entries* และอ็อพชันการพิมพ์อื่นจะถูกนำมาใช้กับการพิมพ์แต่ละ คู่ของคีย์และค่า หากใช้ได้ ตัวอย่างเช่น ถ้าการเรียงลำดับเป็นไปตามชนิดช่วง อ็อพชัน *num-of-entries* และอ็อพชันการพิมพ์อื่นๆ จะถูกสงวนสำหรับสล็อตของแต่ละช่วง

อ็อพชันการพิมพ์ของอาร์เรย์ที่สัมพันธ์กันดีฟอลต์ สามารถถูกเปลี่ยนโดยฟังก์ชัน **set_aso_print_options()** ใน BEGIN probe

ตัวอย่าง:

```
set_aso_print_options (10, SORT_TYPE_DESCEND|SORT_BY_VALUE);
```

ดังแสดงในตัวอย่าง สามารถระบุหลาย แฟล็กโดยการแทรก สัญลักษณ์แถวแนวดิ่ง ระหว่าง แฟล็ก

หมายเหตุ: อ็อพชัน sort-key-index ไม่สามารถตั้งค่าโดยฟังก์ชัน set_aso_print_options() เนื่องจากไม่สามารถทำให้เป็นแบบทั่วไปสำหรับ associative arrays ของขนาดมิติคีย์ ที่แตกต่างกัน

ฟังก์ชัน print() จะพิมพ์คีย์และค่าที่เชื่อมโยง สำหรับอิลิเมนต์ทั้งหมด หรือชุดย่อยของอิลิเมนต์ของ associative array โดยใช้อ็อพชันการพิมพ์ ดีฟอลต์ เมื่อต้องการแทนที่อ็อพชันการพิมพ์ดีฟอลต์ คุณต้องใช้อาร์กิวเมนต์เพิ่มเติมในฟังก์ชัน print() สำหรับข้อมูลเพิ่มเติมเกี่ยวกับฟังก์ชัน print() โปรดดูที่หัวข้อ ฟังก์ชัน Vue

ฟังก์ชัน **print( )** พิมพ์ คู่คีย์และค่าของอาร์เรย์ที่สัมพันธ์กันโดยใช้อ็อพชัน การพิมพ์ดีฟอลต์ ถ้าคุณต้องการดูเนื้อหาอาร์เรย์ที่สัมพันธ์กัน ในรูปแบบต่างกัน จะมีอ็อพชัน *num-of-entries* และแฟล็กอ็อพชันการพิมพ์เป็นพารามิเตอร์เพิ่มเติมของฟังก์ชัน **print( )**

ตัวอย่าง:

```
/* uses default print options to display the contents of associative array 'count' */
print(count);
/* prints the first 10 entries of sorted associative array 'count'.
Default sort-by and sort-type options are used */
print(count, 10);
/* sorts the associative array 'count' in descending order of values and
displays the first 10 entries of 'count' */
print(count, 10, SORT_BY_VALUE|SORT_TYPE_DESCEND);

/* print elements which have first key as 0 */
print(var[0][ANY][ANY]);
```

- รูทีน clear() ใช้เพื่อล้างคีย์และค่าที่เชื่อมโยงสำหรับ อิลิเมนต์ของ associative array รูทีน clear() ยังใช้เพื่อรีเซ็ต ค่าของคีย์ associated array โดยไม่ล้างคีย์ออก หากรูทีนย่อย clear() ล้างหนึ่งหรือหลายอิลิเมนต์ออกสำเร็จ ค่า 0 จะถูกส่งคืน และรูทีนย่อยจะคืนค่า 1 เมื่อไม่มีอิลิเมนต์ที่ถูกล้างออก

  ```
  clear(count);              // count is an associative array.
  ```

  รูทีนก่อนหน้านี้ที่มีเพียงอาร์กิวเมนต์เดียวของชนิดของอาร์เรย์แบบเชื่อมโยงจะล้างคู่ของคีย์ทั้งหมดที่อยู่ในอาร์เรย์แบบเชื่อมโยง **count** หลังจากการดำเนินการล้างก่อนหน้านี้ อาร์เรย์แบบเชื่อมโยง **count** จะว่างเปล่า

  ```
  clear(count, RESET_VALUE);       // count is an associative array.
  clear(var);          //  var is an associative array with three key dimensions
  ```

  รูทีน clear ก่อนหน้านี้จะรีเซ็ตค่าของคู่ของคีย์ในอาร์เรย์แบบเชื่อมโยงโดยไม่ล้างคีย์ ค่าดีฟอลต์ต่อไปนี้จะถูกรีเซ็ตขึ้นอยู่กับชนิดของค่าของอาร์เรย์แบบเชื่อมโยง:

  เมื่อต้องการล้างอิลิเมนต์ที่มีค่าเฉพาะ คุณต้องระบุคีย์ในอาร์กิวเมนต์แรก นอกจากนี้ เมื่อต้องการข้ามมิติคีย์เฉพาะเจาะจงใดๆ (ค่าทั้งหมดของมิติคีย์เฉพาะเจาะจงดังกล่าวตรงกัน) คุณสามารถระบุ ANY หากมีการระบุคีย์ มิติคีย์ทั้งหมดของ associative array ต้องระบุเป็นค่าจริงของชนิดของคีย์ที่ตรงกัน หรือ ANY

  ```
  clear(var[ANY]["a"][ANY]);  // clear all elements with second key as "a"
  ```

  คุณสามารถระบุพารามิเตอร์ที่สองในรูทีน clear() เป็น RESET_VALUE หากคุณระบุ RESET_VALUE คีย์ของ associative array จะถูกเก็บรักษาไว้ และจะรีเซ็ต เฉพาะค่า

  ```
  clear(count, RESET_VALUE);
  clear(var[0][ANY][ANY], RESET_VALUE);
  ```

  RESET_VALUE ขึ้นอยู่กับชนิดของค่า ตารางต่อไปนี้แสดงชนิดข้อมูลและค่าดีฟอลต์ที่ชนิดข้อมูลจะถูกรีเซ็ต:

ชนิด	ค่าดีฟอลต์
ชนิด Integral (int, long, short, long long)	0
LIST	ว่าง
Foat และ Double	0.0000000
สตริง	ว่าง
stktrace_t	ว่าง

ชนิด	ค่าดีฟอลต์
probev_timestamp_t	0
path_t	ว่าง
mac_addr_t	0
ip_addr_t	0

- การดำเนินการ Quantizeจะพิมพ์คีย์และค่าของ associative array ที่กำหนด ในรูปแบบกราฟิกโดยอิงตามสเกลแบบลิเนียร์ของค่า

  ```
  quantize(count);
  ```

  count คือ อาร์เรย์ที่สัมพันธ์ และพิมพ์เนื้อหาต่อไปนี้:

  ```
  คีย์                 ค่า
  1                      1                ========
  2                      2                =========
  3                      3                ==========
  4                      4                ===========
  5                      5                ============
  6                      6                =============
  ```

คล้ายกับฟังก์ชัน **print( )** คุณสามารถระบุฟังก์ชัน **quantize( )** ที่พิมพ์อ็อพชันเพื่อแทนที่อ็อพชันการพิมพ์ดีฟอลต์

ตัวอย่าง:

```
/* sorts the associative array 'count' in descending order of values and displays
the first 10 entries of 'count' in graphical format*/
quantize(count, 10, _BY_VALUE|SORT_TYPE_DESCEND);
```

สำหรับ associative arrays ที่มีคีย์แบบหลายมิติ คีย์สามารถระบุใน อิลิเมนต์แรกเพื่อจำกัด อิลิเมนต์เฉพาะ:

```
quantize(var[0][ANY][ANY]);  //quantize elements with first key as 0
```

- Lquantize บน associative array: การดำเนินการนี้จะพิมพ์คีย์และค่าของ associative array ที่กำหนดในรูปแบบกราฟิกโดยอิงตามสเกลแบบล็อกาลิทึมของ ค่า

  ```
  lquantize (count);
  ```

  โดยที่ count เป็นอาร์เรย์แบบเชื่อมโยงจะพิมพ์เนื้อหาต่อไปนี้:

  ```
  คีย์                 ค่า
  500                    500              ====
  1000                   1000             ====
  2000                   2000             =====
  4000                   4000             =====
  8000                   8000             ======
  16000                 16000             ======
  32000                 32000             =======
  64000                 64000             =======
  ```

คล้ายกับฟังก์ชัน **print( )** คุณสามารถระบุด้วย ฟังก์ชัน **lquantize( )** ที่พิมพ์อ็อพชันเพื่อแทนที่ อ็อพชันการพิมพ์ดีฟอลต์

ตัวอย่าง:

```
/* sorts the associative array 'count' in descending order of values, and displays
the first 10 entries of 'count' in graphical
format based on the logarithmic value*/
lquantize(count, 10, _BY_VALUE|SORT_TYPE_DESCEND);
```

สำหรับ associative arrays **ที่มีคีย์แบบหลายมิติ คีย์สามารถระบุใน อิลิเมนต์แรกเพื่อจำกัด อิลิเมนต์เฉพาะ:**

```
lquantize(var[0][ANY][ANY]);  //lquantize elements with first key as 0
```

| ตัวอย่างต่อไปนี้แสดงวิธีใช้ associative array:

ตัวอย่าง:

```
# Trace all the alloc- related calls and store the entry
# Time in 'entry_time' associative array
#
@@uft:$__CPID:*:"/alloc/":entry
{
        entry_time[get_function()]=timestamp();
}
#
# At exit, first check if entry for this function was traced
# If so, delete the entry time from 'entry_time' associative array
# To ensure that next time no action is taken on exit if entry was not traced.

@@uft:$__CPID:*:"/alloc/":exit
{
   func =get_function();
   if(exists(entry_time, func) )
   {
             append(time_taken[func],
                   diff_time(timestamp(),entry_time[func],MICROSECONDS));
             delete(entry_time, func);
   }
}
#
# Print the list attributes sum, min, max, count, and avg time taken in every
# Alloc function.
#
@@syscall:$__CPID:exit:entry
{
  print(time_taken);
  exit();
}
```

หมายเหตุ: วิธีนี้คุณไม่ต้องกำหนดตัวแปรรายการ หลายตัวโดยชัดแจ้ง และยังคงได้รับฟังก์ชันของรายการ อย่างครบถ้วน ด้วยความช่วยเหลือของอาร์เรย์ที่สัมพันธ์กัน

ชนิดข้อมูลการประทับเวลา

ตัวแปรชนิด **probev_timestamp_t** จะเก็บค่าจากฟังก์ชัน **timestamp** ProbeVue ซึ่งส่งคืนการประทับเวลาในรูปแบบ AIX ภายใน ตัวแปรนี้สามารถส่งผ่านในรูปของพารามิเตอร์ได้ในภายหลังไปยัง ฟังก์ชัน **diff_time** ซึ่งจะส่งคืนค่าความต่างระหว่างการประทับเวลาสองครั้ง ชนิดข้อมูลนี้ ยังถูกเก็บไว้ในอาร์เรย์ที่เชื่อมโยงในรูปของคีย์หรือค่า

แม้ว่าคอมไพลเลอร์ ProbeVue จะไม่พิมพ์การตรวจสอบเมื่อคุณใช้ชนิดข้อมูล **long** แทนชนิดข้อมูล **probev_timestamp_t** สำหรับเก็บเวลาประทับ ถ้าเป็นไปได้ให้หลีกเลี่ยงการใช้นี้

การดำเนินการต่อไปนี้จะสามารถยอมรับได้สำหรับตัวแปรของชนิด **probev_timestamp_t** :

- สามารถกำหนดค่าเริ่มต้นให้เป็นศูนย์

  หมายเหตุ: ตัวแปรการประทับเวลา ของคลาสโกลบอลหรือคลาสเธรดแบบโลคัลจะถูกกำหนดค่าเริ่มต้นให้เป็นศูนย์ สำหรับการเริ่มต้นเซสชัน ProbeVue
- สามารถเปรียบเทียบกับค่าศูนย์ ค่าการประทับเวลาที่ส่งคืนโดยฟังก์ชัน **timestamp** จะมีค่ามากกว่าศูนย์
- สามารถเปรียเทียบกับตัวแปรการประทับเวลาอื่นๆ ได้ การประทับเวลาภายหลัง ที่ถูกรับประกันว่ามีขนาดใหญ่กว่าการประทับเวลาในตอนต้น
- สามารถส่งผ่านเป็นพารามิเตอร์ไปยังฟังก์ชัน **diff_time**
- สามารถพิมพ์โดยใช้ฟังก์ชัน **printf** หรือ **trace**

ชนิดข้อมูลพาธไฟล์

ตัวแปรชนิด `path_t` สามารถใช้เพื่อเก็บค่า `__file->path` (อ้างถึง **__file** ที่บิวด์ในโปรแกรมจัดการโพรบ I/O) หรือ `function fd_path()` เฉพาะตัวแปรโลคัลหรือโกลบอลชนิด `path_t` ได้รับการสนับสนุนเท่านั้น ตัวแปรชนิดนี้ยังสามารถเป็นคีย์หรือค่าในอาร์เรย์ที่เชื่อมโยง

การประกาศตัวแปรพาธไฟล์

```
path_t pth; // global variable of type path_t
auto:pth2 = fd_path(fd); // store in a local path_t variable
my_aso[__file->fname] = __file->path; // store in associative array
```

signed, unsigned, register, static, thread และ kernel qualifier ไม่ได้รับการสนับสนุนสำหรับตัวแปรชนิด `path_t`

การดำเนินการการกำหนดค่า

ตัวดำเนินการกำหนดค่า (=) อนุญาตให้ใช้ตัวแปร `path_t` หนึ่งตัว ที่ต้องถูกกำหนดให้กับตัวแปร `path_t` ค่าเดิมของตัวแปรจะถูกเขียนทับ

ในตัวอย่างต่อไปนี้ หลังจากการกำหนดค่าแล้ว p1 และ p2 จะอ้างถึงพาธไฟล์ เดียวกัน:

```
path_t p1, p2;
p1 = fd_path(fd); // fd is a valid file descriptor value
p2 = p1;
```

การดำเนินการเปรียบเทียบ

เฉพาะตัวดำเนินการเท่ากับ (==) และไม่เท่ากับ (!=) เท่านั้นที่อนุญาตให้ใช้สำหรับตัวแปร `path_t` ผลลัพธ์ของตัวดำเนินการเท่ากับเป็น true (1) หากทั้งสอง มีค่าพาธไฟล์สัมบูรณ์ที่เหมือนกัน หรือเป็น false (0) หากเป็นกรณีอื่น

ตัวดำเนินการไม่เท่ากับเป็นส่วนเสริมของลักษณะการทำงานนั้น ไม่มีตัวดำเนินการ เปรียบเทียบ (>=, >, < หรือ =<) ที่ได้รับอนุญาตให้ใช้สำหรับตัวแปรชนิด `path_t`

การพิมพ์ตัวแปรชนิดพาธไฟล์

ตัวแปรชนิด path_t สามารถพิมพ์ด้วยตัวระบุรูปแบบ "%p" ในฟังก์ชัน printf()

การพิมพ์ไฟล์นี้เกี่ยวข้องกับการดำเนินการค้นหาไฟล์ที่ใช้เวลาในระบบไฟล์ ที่สอดคล้องกัน ดังนั้น จึงต้องใช้ในสคริปต์ Vue

หมายเหตุ: สำหรับไฟล์ชั่วคราว ซึ่งอาจไม่มีอยู่เมื่อข้อความ printf() สามารถ ขอรับ printed; null string จะถูกพิมพ์เป็นพาธไฟล์

อาร์เรย์ที่สัมพันธ์กันที่มีตัวแปรชนิด path_t เป็นคีย์หรือค่า (หรือทั้งสองอย่าง) สามารถพิมพ์ได้โดยใช้ฟังก์ชัน print()

```
printf("file path=[%p]\n", __file->path);
my_aso[0] = fd_path(fd); // fd is valid file descriptor value
print(my_aso);
```

ข้อจำกัดเกี่ยวกับตัวแปรชนิดพาธไฟล์

- อาร์เรย์ของตัวแปร path_t ไม่สามารถประกาศได้
- ตัวแปร path_t ไม่สามารถใช้เป็นสมาชิกของ struct หรือ union
- ตัวชี้ไปยังตัวแปร path_t ไม่อนุญาตให้ใช้
- การพิมพ์ตัวแปร path_t ไปเป็นชนิดอื่นหรือพิมพ์ชนิดอื่นไปเป็น path_t ไม่ได้รับการอนุมัติ
- ไม่มีตัวดำเนินการทางคณิตศาสตร์ (+, -, *, /, ++, -- และอื่นๆ) ที่สามารถใช้กับตัวแปรชนิด path_t

ชนิดข้อมูล MAC address

ตัวแปรชนิด mac_addr_t ถูกใช้เพื่อเก็บค่า MAC address (อ้างถึงบิวด์อิน __etherhdr และ __arphdr ในส่วนโปรแกรมจัดการโพรบ เครือข่ายสำหรับการใช้ตัวแปรชนิด mac_addr_t)

ชนิดข้อมูลแบบย่อนี้ไม่สามารถใช้ได้โดยตรงกับตัวดำเนินการที่ไม่ใช่อาร์เรย์ C หรือตัวดำเนินการไลบรารี เฉพาะตัวแปรโลคัลหรือโกลบอลของชนิด mac_addr_t เท่านั้นที่ได้รับการสนับสนุน

ตัวแปรของชนิดนี้ยังสามารถเก็บในอาร์เรย์ที่เชื่อมโยง เป็นคีย์หรือค่า

Vue สนับสนุนคุณสมบัติและการดำเนินการต่อไปนี้สำหรับตัวแปรชนิด MAC address :

การประกาศตัวแปร MAC แอดเดรส

```
mac_addr_t m1;          // global variable of type
__auto mac_addr_t m2;                          // auto variable of type
m2 = __etherhdr->src_addr;                     // store source MAC address in a local variable
mac_aso["src_mac_addr"] = __etherhdr->src_addr ;  // store in an associative array.
```

signed, unsigned, register, static, thread และ kernel qualifier ไม่ได้รับการสนับสนุนสำหรับตัวแปรชนิด mac_addr_t

การดำเนินการการกำหนดค่า

ตัวดำเนินการกำหนดค่า (=) ไม่ได้รับอนุญาตให้ใช้ตัวแปรชนิด mac_addr_t ที่ต้องกำหนดค่าเป็นตัวแปร ชนิด mac_addr_t อื่นๆ ค่าเดิมของตัวแปรจะถูกเขียนทับ ไม่อนุญาตให้พิมพ์จากหรือเป็นตัวแปรชนิด mac_addr_t

ในตัวอย่างต่อไปนี้ เนื้อหาของ mac_addr_t m1 ถูกกำหนดให้กับ m2

```
mac_addr_t m1, m2;              // Declares two MAC address variables.
m1 = __etherhdr->src_addr;      // Get the source MAC address of the packet in m1.
m2 = m1 ;                       // Get the content of m1 into m2.
```

การดำเนินการเปรียบเทียบ

เฉพาะตัวดำเนินการเท่ากับ (==) และไม่เท่ากับ (!=) ได้รับอนุญาตให้ใช้สำหรับตัวแปร mac_addr_t ผลลัพธ์ของตัวดำเนินการเท่ากับเป็น true (1) หากทั้งสองมีค่า MAC address ที่เหมือนกันหรือเป็น false (0) ในกรณีอื่น

ตัวดำเนินการไม่เท่ากับเป็นส่วนเสริมของลักษณะการทำงานนั้น ไม่มีตัวดำเนินการเปรียบเทียบ (>=, >, <, or =<) ที่ได้รับอนุญาตให้ใช้สำหรับตัวแปรชนิด mac_addr_t

```
if ( m1 == m2) // comparing two mac_addr_t type variables.
printf("Mac addresses are equal"); else printf("Mac addresses are not equal");
```

การพิมพ์ตัวแปรชนิด MAC แอดเดรส

ตัวแปรชนิด mac_addr_t สามารถพิมพ์ด้วยตัวระบุรูปแบบ "%M" ในฟังก์ชัน printf() ของ Vue อาร์เรย์ที่สัมพันธ์กันที่มีตัวแปรชนิด mac_addr_t เป็นคีย์หรือค่า (หรือทั้งสองอย่าง) สามารถพิมพ์ได้โดยใช้ฟังก์ชัน print()

```
printf(" Source MAC address=[%M]\n", __etherhdr->src_addr);
mac_aso["src_mac_address"] = __etherhdr->src_addr ; // Store source MAC address as value in an associative array mac_aso.
print(mac_aso);
```

ข้อจำกัดสำหรับตัวแปรชนิด MAC แอดเดรส

- อาร์เรย์ของตัวแปร mac_addr_t ไม่สามารถประกาศได้
- ไม่สามารถใช้ตัวแปร mac_addr_t เป็นสมาชิกของ struct หรือ union
- ตัวชี้ไปยังตัวแปร mac_addr_t ไม่อนุญาตให้ใช้
- การพิมพ์ตัวแปร mac_addr_t ไปเป็นชนิดอื่นหรือพิมพ์ชนิดอื่นไปเป็นชนิด mac_addr_t ไม่อนุญาตให้ใช้
- ไม่มีตัวดำเนินการทางคณิตศาสตร์ (+, -, *, /, ++, -- etc) ที่สามารถใช้กับตัวแปรชนิด mac_addr_t

ชนิดข้อมูล IP แอดเดรส

ตัวแปรชนิด ip_addr_t ถูกใช้เพื่อเก็บค่า IP แอดเดรส (อ้างถึงบิวด์อิน __ip4hdr, __ip6hdr และ __proto_info ในโปรแกรมจัดการโพรบเครือข่ายสำหรับการใช้ตัวแปรชนิด ip_addr_t)

สิ่งนี้เป็นชนิดข้อมูลย่อและไม่สามารถใช้ได้โดยตรงกับตัวดำเนินการที่ไม่ใช่อาร์เรย์ C หรือ ตัวดำเนินการไบนารี เฉพาะตัวแปรโลคัลหรือโกลบอลชนิด ip_addr_t ได้รับการสนับสนุนเท่านั้น ตัวแปรของชนิดนี้ยังสามารถเก็บในอาร์เรย์ที่เชื่อมโยงเป็นคีย์หรือค่า

Vue สนับสนุนคุณสมบัติและการดำเนินการต่อไปนี้สำหรับ ตัวแปรชนิด IP แอดเดรส:

การประกาศตัวแปร IP แอดเดรส

```
ip_addr_t i1;                     // global variable of type ip_addr_t
__auto ip_addr_t i2;                        // auto variable of type
ip_addr_t i2 = __ip4hdr->src_addr;          // store source IP address in a local ip_addr_t variable.
ip_aso["src_ip_addr"] = __ip4hdr->src_addr; // store in an associative array.
```

qualifiers signed, unsigned, register, static, thread และ kernel ไม่ได้รับการสนับสนุนสำหรับตัวแปรชนิด ip_addr_t

การดำเนินการการกำหนด

ตัวดำเนินการการกำหนด (=) อนุญาตให้กำหนดตัวแปรชนิด ip_addr_t ให้กับตัวแปนชนิด ip_addr_t อื่นและยังอนุญาต ให้กำหนดค่าคงที่ IP แอดเดรสหรือชื่อโฮสต์ให้กับตัวแปรชนิด ip_addr_t ค่าเดิมของตัวแปร จะถูกเขียนทับ ไม่อนุญาตให้ใช้ชนิด casting จากหรือไปยังชนิดตัวแปร ip_addr_t

ในตัวอย่างต่อไปนี้ เนื้อหาของ ip_addr_t i1 ถูกกำหนดให้กับ i2®

```
ip_addr_t i1, i2; // Declares two IP address variables.
ip_addr_t i3, i4, i5;               // Declares three IP address variables.
i1 = __ip4hdr->src_addr;   // Get the source IP address of the packet in i1.
i2 = i1 ;                  // Get the content of i1 into i2.
i3 = "10.10.10.1";                  // Assign the constant IPv4 address to i3 variable.
i4 = "fe80::2c0c:33ff:fe48:f903";   // Assign the Ipv6 address to i4 variable.
i5 = "example.com";                 // Assign the hostname to i5 variable.
                                   // Get the content of i1 into i2.
```

การดำเนินการเปรียบเทียบ

เฉพาะตัวดำเนินการเท่ากับ (==) และไม่เท่ากับ (! =) อนุญาตให้ใช้สำหรับตัวแปรชนิด ip_addr_t อนุญาตให้ใช้การเปรียบเทียบ ระหว่างสองตัวแปรชนิด ip_addr_t และที่มีสตริงค่าคงที่ (IP แอดเดรสหรือชื่อโฮสต์ระบุในเครื่องหมายคำพูดคู่ "192.168.1.1" หรือ "example.com")

ผลลัพธ์ของตัวดำเนินการเท่ากับ คือ True (1) หากทั้งสองมี ชนิด IP แอดเดรส (IPV4 หรือ IPV6) และค่าเดียวกัน หรือ false (0) ในกรณีอื่น ตัวดำเนินการไม่เท่ากับเป็นส่วนเสริมของ สิ่งนั้น ไม่มีตัวดำเนินการเปรียบเทียบอื่น (>=, >, < หรือ =<) ที่ได้รับอนุญาตให้ใช้สำหรับตัวแปรชนิด ip_addr_t

```
if( i1 == i2) // comparing two ip_addr_t type variables.
                          //IP address string
printf("IP addresses are equal");
else printf("IP addresses are not equal");
หรือ
if( i1 == "192.168.1.1") // comparing ip_addr_t type variable and constant string.
 printf("IP addresses are equal");
else printf("IP addresses are not equal");
หรือ
if (i1 = "example.com")          // comparing ip_addr_t type variable and constant
                                 //IP address string
printf("IP addresses are equal");
else printf("IP addresses are not equal");
```

การพิมพ์ตัวแปรชนิด IP แอดเดรส

ตัวแปรชนิด ip_addr_t สามารถพิมพ์ได้ด้วยตัวระบุรูปแบบ "%I" เพื่อพิมพ์ IP แอดเดรส ในรูปของทศนิยมแบบมีจุดหรือรูปแบบเลขฐานหก และตัวระบุรูปแบบ "%H" เพื่อพิมพ์ชื่อโอสต์ในฟังก์ชัน printf() ของ Vue การพิมพ์ชื่อโฮสต์นี้เกี่ยวข้องกับเวลาที่ใช้ในการดำเนินการคนหา dns ดังนั้น จึงควรใช้ในสคริปต์ VUE

หมายเหตุ: เมื่อผู้ใช้ใช้ตัวระบุรูปแบบ "%H" เพื่อพิมพ์ชื่อโฮสต์สำหรับ IP แอดเดรสที่อาจไม่มีอยู่ใน dns สำหรับ IP แอดเดรสเหล่านั้น จะพิมพ์ IP แอดเดรสในรูปแบจุดทศนิยม/เลขฐานสิบหก แทนชื่อโฮสต์

อาร์เรย์ที่สัมพันธ์กันที่มีตัวแปรชนิด ip_addr_t เป็นคีย์หรือค่า (หรือทั้งสองอย่าง) สามารถพิมพ์ ได้โดยใช้ฟังก์ชัน print()

```
printf(" Source IP address=[%I]\n", __ip4hdr->src_addr);
ip_aso["src_ip_address"] = __ip4hdr->src_addr ; // Store source IP address as value in an associative array
print(ip_aso);
```

ข้อจำกัดสำหรับตัวแปรชนิด IP แอดเดรส

- อาร์เรย์ของตัวแปร ip_addr_t ไม่สามารถประกาศได้
- ตัวชี้ไปยังตัวแปร ip_addr_t ไม่อนุญาตให้ใช้
- การพิมพ์ตัวแปร ip_addr_t ไปยังชนิดอื่นใดหรือพิมพ์ชนิดอื่นใดไปเป็น ip_addr_t ไม่ได้รับอนุญาต
- ไม่มีตัวดำเนินการทางคณิตศาสตร์ (+, -, *, /, ++, -- etc) ที่สามารถใช้ได้ด้วยตัวแปรชนิด ip_addr_t

net_info_t data type

ตัวแปร net_info_t เป็นตัวแปรโครงสร้าง หรือคอมโพสิตที่ใช้เพื่อเก็บข้อมูล tuple เครือข่าย 4 ตัว (โลคัลหรือรีโมต IP แอดเดรส และหมายเลขพอร์ต) จาก ตัวให้คำอธิบายซ็อกเก็ตเฉพาะผ่านฟังก์ชัน sockfd_netinfo Vue สมาชิกของ โครงสร้างนี้ถูกเข้าถึงเหมือนกับโครงสร้างที่นิยามโดยผู้ใช้ในสคริปต์ Vue ชนิด net_info_t เป็นชนิดข้อมูลย่อและตัวแปรนี้ไม่สามารถใช้ได้โดยตรง ด้วยตัวดำเนินการที่ไม่ใช่อาร์เรย์ C แบบมาตรฐานหรือตัวดำเนินการไบนารี ตัวแปรนี้เป็นโครงสร้างที่มีข้อมูล tuple ตัวที่สี่ อิลิเมนต์ตัวแปรนี้สามารถเข้าถึงโดยใช้ตัวดำเนินการ "." เช่น อิลิเมนต์โครงสร้าง C

อิลิเมนต์ของชนิดตัวแปร net_info_t เป็นดังนี้:

```
net_info_t
{
    int local_port;
    int remote_port;
    ip_addr_t local_addr;
    ip_addr_t remote_addr; };
```

Vue สนับสนุนคุณสมบัติและการดำเนินการสำหรับตัวแปรชนิด net_info_t:

การประกาศตัวแปรชนิด net_info_t

```
net_info_t n1,n2 // n1 is variable of type net_info_t
sockfd_netinfo(fd, n1);
// fd is socket descriptor and n1 contains network
// four tuple information from sockfd_netinfo Vue function.
n2.local_addr = __ip4hdr->src_addr; n2.remote_addr = __ip4hdr->dst_addr; n1.local_port = __tcphdr->src_port;
n1.remote_port = __tcphdr->dst_port;
```

signed signed, unsigned, register, static, thread, local, global และ kernel qualifiers ไม่ได้รับการสนับสนุน สำหรับตัวแปรชนิด net_info_t

ข้อจำกัดสำหรับตัวแปรชนิด net_info_t

- ไม่สนับสนุนโครงสร้างและตัวแปรสมาชิกร่วม
- ไม่สามารถประกาศตัวชี้ไปยังตัวแปร net_info_t
- ตัวแปรนี้ไม่ได้รับการสนับสนุนในอาร์เรย์ที่สัมพันธ์กัน
- อาร์เรย์ของตัวแปร net_info_t ไม่สามารถประกาศได้
- การพิมพ์ตัวแปร net_info_t ไปเป็นชนิดอื่นหรือพิมพ์ชนิดอื่น ไปเป็นชนิด net_info_t ไม่ได้รับการอนุมัติ
- ตัวดำเนินการทางคณิตศาสตร์ (+, -, *, /, ++, -- และอื่นๆ) ไม่สามารถใช้ได้กับตัวแปรชนิด net_info_t

ฟังก์ชัน Vue

ไม่เหมือนกับโปรแกรมที่เขียนโดย C หรือ FORTRAN หรือภาษาท้องถิ่น สคริปต์ที่ถูกเขียนใน Vue ไม่ได้เข้าถึงรูทีนย่อย ที่จัดเตรียมไว้โดยไลบรารีระบบ AIX หรือไลบรารีผู้ใช้ใดๆ อย่างไรก็ตาม Vue สนับสนุนไลบรารีของฟังก์ชันภายในเป็นพิเศษ ซึ่งมีประโยชน์สำหรับโปรแกรมการติดตามแบบไดนามิก

ฟังก์ชันที่ระบุเฉพาะสำหรับการติดตาม

get_function
ส่งคืนชื่อของฟังก์ชันที่ครอบคลุมด้วยโพรบปัจจุบัน เมื่อฟังก์ชัน **get_function** ถูกเรียกใช้จากประโยค interval, systrace, BEGIN และ END ฟังก์ชันจะคืนค่าสตริงว่างๆ

การประทับเวลา
ส่งคืนการประทับเวลาปัจจุบัน

diff_time
ค้นหาความแตกต่างระหว่างการประทับเวลาสองครั้งในหน่วยไมโครวินาที หรือมิลลิวินาที

ฟังก์ชันการดักจับการติดตาม

printf จัดรูปแบบและพิมพ์ค่าของตัวแปรและนิพจน์

การติดตาม
พิมพ์ข้อมูลที่ไม่มีการจัดรูปแบบ

stktrace
จัดรูปแบบและพิมพ์การติดตามสแต็ก

ฟังก์ชันลิสต์

list แสดงตัวอย่างตัวแปรลิสต์

append ผนวกรายการใหม่ลงในลิสต์

sum, max, min, avg, count
ฟังก์ชันการรวมที่สามารถใช้กับตัวแปรลิสต์ได้

ฟังก์ชันไลบรารี C

atoi, strstr
ฟังก์ชันสตริงมาตรฐาน

ฟังก์ชันที่สนับสนุนการทดสอบการติดตาม

start_tentative, end_tentative
ตัวบ่งชี้สำหรับการเริ่มต้นและสิ้นสุดการทดสอบการติดตาม

commit_tentative, discard_tentative
Commits หรือละทิ้งข้อมูลการทดสอบการติดตาม

ฟังก์ชันอื่นๆ

exit ยกเลิกสคริปต์ Vue

get_userstring
อ่านสตริง (หรือข้อมูล) จากหน่วยความจำผู้ใช้

ptree พิมพ์แผนผังกระบวนการของกระบวนการที่ถูกโพรบ

คุณสามารถใช้ฟังก์ชัน Vue เฉพาะกับตัวแปรชนิดสตริง และไม่ใช่ตัวแปรชนิดตัวชี้ ฟังก์ชันสตริงมาตรฐานเช่น like **strcpy**, **strcat** และอื่นๆ ไม่ใช่ฟังก์ชันที่จำเป็นใน Vue เนื่องจากมีการสนับสนุนผ่าน ไวยากรณ์ภาษาด้วยตัวของมันเอง

คอมไพลเลอร์ ProbeVue จะตรวจสอบชนิดข้อมูลของพารามิเตอร์ ที่ส่งผ่านไปยังฟังก์ชัน Vue

สำหรับฟังก์ชัน **printf** การตรวจสอบความถูกต้องจะถูกกระทำเพื่อตรวจสอบว่ามีการระบุอาร์กิวเมนต์ในฟังก์ชัน **printf** สำหรับตัวระบุรูปแบบแต่ละตัวที่ระบุในสตริงรูปแบบ จำนวนทั้งหมดของตัวระบุรูปแบบและจำนวนทั้งหมดของอาร์กิวเมนต์ที่ถูกผ่านไปยังฟังก์ชัน **printf** ควรเท่ากัน นอกจากนั้น การตรวจสอบความถูกต้องยังถูกทำเพื่อเปรียบเทียบว่า ชนิดของอาร์กิวเมนต์ที่ถูกผ่านเข้ากันได้กับชนิดที่แท้จริงที่ระบุเป็นตัวระบุรูปแบบในสตริงรูปแบบ ถ้าการตรวจสอบเหล่านี้ล้มเหลว Probevue จะแสดงข้อความแสดงข้อผิดพลาด

ตัวอย่างเช่น,

```
printf("hello world %s, %d\n", str);
```

จะแสดงข้อความแสดงข้อผิดพลาดจากคอมไพลเลอร์ว่าไม่มีอาร์กิวเมนต์ที่ผ่านสำหรับ **%d** เช่นเดียวกัน

```
Printf("The total count of elements is %d\n", str);
```

ยังข้อความแสดงข้อผิดพลาดเป็น รูปแบบที่ระบุเป็น **%d** ที่อาร์กิวเมนต์ถูกผ่าน ตัวแปร **str** เป็นสตริง

ฟังก์ชันคุณลักษณะอื่น

อย่างไรก็ตาม เมื่อระบุเป็น

```
printf ("The total count of elements is %lld\n", i);
```

โดยที่ *i* เป็นตัวแปรชนิด int จะไม่แสดงข้อความแสดงข้อผิดพลาดเนื่องจากตัวแปร *i* เป็นชนิดที่เข้ากันได้สำหรับตัวระบุที่ร้องขอ ดังนั้น จะไม่มีการตรวจสอบที่แท้จริง อย่างไรก็ตาม การตรวจสอบชนิดที่เข้ากันได้ คือ

คุณไม่สามารถวางฟังก์ชันในส่วนของเพรดิเคตของ Vue clause ได้

เพรดิเคต

คุณไม่สามารถใช้เพรดิเคตได้ เมื่อประมวลผล clause ที่จุดโพรบ ซึ่งต้องดำเนินการตามเงื่อนไข ส่วนของเพรดิเคตจะถูกระบุไว้โดยการมีอยู่ของคีย์เวิร์ด **when** ในทันที หลังส่วนของข้อกำหนดคุณสมบัติโพรบ ด้วยตัวของเพรดิเคตเองประกอบด้วยนิพจน์แบบมีเงื่อนไข C-style ซึ่งครอบด้วยเครื่องหมายวงเล็บ

มีข้อจำกัดบางข้อเกี่ยวกับนิพจน์ภายในส่วนของเพรดิเคต :

- ตัวแปรคลาสเคอร์เนลไม่อนุญาตในเพรดิเคต
- ตัวแปรคลาสแบบอัตโนมัติไม่อนุญาตให้ใช้ในเพรดิเคต
- ตัวแปรชนิดเลขทศนิยมไม่อนุญาตให้ใช้ในเพรดิเคต
- ฟังก์ชัน Vue ไม่อนุญาตให้ใช้ในเพรดิเคต
- เอฟเฟ็กต์ไม่อนุญาตให้ใช้ในเพรดิตเคต และตัวดำเนินการกำหนดค่า = และการลอกเลียน เช่น +=, |= และอื่นๆ ไม่อนุญาตให้ใช้
- พารามิเตอร์อันดับที่เก้าและสูงกว่าจะส่งผ่านไปยังฟังก์ชัน (ตัวแปรรายการคลาส entry **__arg9**, **__arg10**, และอื่นๆ) ไม่อนุญาตให้ใช้ในเพรดิเคต

การประมวลผลตามเงื่อนไขของการดำเนินการที่ระบุภายใน clause อาจเกิดขึ้นได้โดยใช้คำสั่ง **if ... else** ซึ่งทำงานเหมือนกับคำสั่ง ในภาษา C อย่างไรก็ตาม ถ้า clause ทั้งหมดถูกออกคำสั่งตามเงื่อนไข ก็จะมีความต้องการใช้เพรดิเคตแทน เนื่องจาก ProbeVue ถูกออกแบบมาเพื่อออปติไมซ์การประมวลผลของเพรดิเคต

หมายเหตุ: เมื่อจุดโพรบสามารถเรียกใช้งานสำหรับหนึ่งการประมวลผล การใช้ตัวแปรเธรดแบบโลคัลภายในเพรดิเคตจะเป็นวิธีที่ดีที่สุด เพื่อลดผลการทำงานทั้งหมดที่กระทบต่อโพรบที่เปิดใช้งาน การวางเงื่อนไขจะตรวจสอบเพรดิเคตภายในที่ต้องการใช้คำสั่ง **if**

สคริปต์ต่อไปนี้ใช้ตัวแปรเธรดแบบโลคัลภายในเพรดิเคต เพื่อตรวจพบเมื่อสตริงอักขระ ถูกเขียนลงในไฟล์ที่ระบุเฉพาะ และยังแสดงตัวอย่างการใช้คำสั่ง **if** ภายในบล็อกการดำเนินการของ clause ด้วยเพรดิเคต ทั้งชื่อไฟล์และสตริงอักขระจะถูกส่งผ่านเป็นพารามิเตอร์ ไปยังสคริปต์ที่ใช้พารามิเตอร์ shell

```
/*
 * Filename : chkfilewrite.e
 *
 * Capture when someone writes a particular word to a specific file
 * takes 2 arguments: filename and word
 *
 * assumes file name is < 128
 *
 * Usage: probevue chkfilewrite.e \"<filename>\" \"<string>\"
 *
 * The backslashes above are necessary to prevent shell
 * from stripping the double quotation mark.
 */

int open(char *fname, int m, int p);
int write(int fd, char *s, int size);

@@syscall:*:open:entry
{
 __auto String fname[128];

 fname = get_userstring(__arg1, -1);

 if (fname == $1)
  thread:opening = 1;
}
```

```
@@syscall:*:open:exit
 when (thread:opening == 1)
{
 thread:fd = __rv;
 thread:opening = 0;

}

@@syscall:*:write:entry
 when (thread:fd == __arg1)
{
 __auto String buf[128];

 if (__arg3 < 128)
  buf = get_userstring(__arg2, __arg3);
 else
  buf = get_userstring(__arg2, 128);

 if (strstr(buf, $2)) {
  printf("%d wrote word to file.\n", __pid);
  exit();
 }
}
```

หากต้องการรันโปรแกรมนี้เพื่อตรวจสอบผู้ที่เขียนสตริง ″Error″ ลงในไฟล์ **foo.log** คุณสามารถออกคำสั่งต่อไปนี้ได้ :

```
probevue chkfilewrite.e \"foo.log\" \"Error\"
```

หมายเหตุ: คุณสามารถพัฒนาสคริปต์ก่อนหน้านี้ได้โดยเพิ่มโพรบ **close** เพื่อตรวจพบเมื่อไฟล์ถูกปิด เพื่อป้องกันสคริปต์จากการจัดคำหลังจากไฟล์ต้นฉบับถูกปิด และมีไฟล์ใหม่เปิดอยู่ และมีหมายเลข file descriptor ที่เหมือนกันซึ่งนำกลับมาใช้ใหม่

ค่าคงที่สัญลักษณ์

Vue สนับสนุนค่าคงที่สัญลักษณ์ที่กำหนดไว้ก่อน ซึ่งถูกใช้ในขณะโปรแกรมมิง AIX ค่าคงที่เหล่านี้จะใช้เป็นคีย์เวิร์ดใน Vue ในระหว่างการคอมไพล์ ค่าคงที่จะถูกแทนที่โดย นิยามในไฟล์ส่วนหัวของระบบ ค่าคงที่สัญลักษณ์ที่ระบุเฉพาะโปรแกรมจัดการโพรบถูกกล่าวถึงใน ส่วนต่างๆ ตามลำดับ ต่อไปนี้คือค่าคงที่สัญลักษณ์ทั่วไป

AF_INET
: ซึ่งระบุตระกูลของแอดเดรสชนิด IPv4 ซึ่งทำให้แน่ใจว่า ข้อมูลจะเป็นชนิด IPV4

AF_INET6
: ซึ่งระบุตระกูลของแอดเดรสชนิด IPv6 ซึ่งทำให้แน่ใจว่า ข้อมูลจะเป็นชนิด IPV6

NULL หากต้องการตั้งค่าชนิดตัวชี้ให้กับ NULL หรือค่าศูนย์ คุณสามารถใช้ NULL เพื่อตั้งค่าตัวแปร String ให้กับสตริงว่าง

หมายเลขข้อผิดพลาดหรือชื่อ ″errno″
: ต่อไปนี้คือชื่อข้อผิดพลาดมาตรฐาน เช่น **EPERM**, **EAGAIN**, **ESRCH**, **ENOENT**, และอื่นๆ ซึ่งระบุโดย POSIX และ ANSI มาตรฐาน และกำหนดไว้ในส่วนหัวของไฟล์ **/usr/include/sys/errno.h**

: สคริปต์ต่อไปนี้ จะติดตามเมื่อการเรียกของระบบ **ที่โยง** ไว้ ล้มเหลวด้วย errno ที่ตั้งค่าเป็น **EADDRINUSE** (แอดเดรสที่ใช้แล้ว)

```
/*
 * File: bind.e
 */

/*
 * Okay to use void for parameters since we are not planning to
 * access them in this script.
 */
int bind(void);

@@syscall:*:bind:exit
 when (__rv == -1)
{
 /*
  * The following check could also be moved to the predicate,
  * although it may not buy a lot because we are already in an
  * error path that should be executed only rarely
  */
 if (__errno == EADDRINUSE)
 /* This check could also be moved to the predicate */
  printf("%d failed with EADDRINUSE for bind() call.\n", __pid);
}
```

ชื่อสัญลักษณ์

ต่อไปนี้คือชื่อสัญลักษณ์มาตรฐาน เช่น **SIGSEGV, SIGHUP, SIGILL, SIGABRT,** และอื่นๆ ซึ่งระบุโดย ANSI มาตรฐานและกำหนดในไฟล์ส่วนหัว **/usr/include/sys/signal.h**

สคริปต์ต่อไปนี้ แสดงวิธีการดีบัก "who" หยุดทำงานการประมวลผลโดยส่งสัญญาณที่ระบุเฉพาะ

```
/*
 * File: signal.e
 *
 * Who sent SIGKILL to my process ?
 */

/* Process IDs are < 2^32, so using an 'int' here instead of pid_t is
 * good enough
 */
int kill(int pid, int signo);

@@syscall:*:kill:entry
 when (__arg1 == $1 && __arg2 == SIGKILL)
{
 /* Trace sender of SIGKILL */
 printf("Stack trace of %s: (PID = %d)\n", __pname, __pid);
 stktrace(PRINT_SYMBOLS|GET_USER_TRACE, -1);
 exit();
}
```

FUNCTION_ENTRY

ระบุว่า จุดโพรบคือฟังก์ชัน entry point หรือไม่ใช้กับฟังก์ชัน **get_location_point**

FUNCTION_EXIT

ระบุว่า จุดโพรบคือฟังก์ชัน exit point หรือไม่ใช้กับฟังก์ชัน **get_location_point**

ไฟล์ส่วนหัว

คุณสามารถสอดแทรกไฟล์ส่วนหัวในสคริปต์ Vue ผ่านอ็อพชัน **-I** ของคำสั่ง **probevue** ไฟล์ส่วนหัวต้องถูกเขียนในไวยากรณ์ของภาษา C ซึ่ง สามารถสอดแทรกชนิด โครงสร้างและนิยาม union ตามข้อกำหนดคุณสมบัติ C-89 อย่างไรก็ตาม ต้องไม่มีคำสั่งที่สามารถเรียกทำงานได้ที่เป็นภาษา C ตัวประมวลผล C ตัวดำเนินการ และคำสั่ง ถ้ามีอยู่ จะถูกละเว้น แต่ตัวดำเนินการ เช่น **#ifdef** และ **#if** สามารถเป็นสาเหตุทำให้ลักษณะการทำงานเป็นแบบไม่กำหนดไว้ ดังนั้น ไฟล์ส่วนหัว AIX มาตรฐานจากไดเร็กทอรี **/usr/include** ไม่สามารถสอดแทรกได้โดยตรง คุณต้องรันตัวประมวลผลก่อนในภาษา C สำหรับชุดของไฟล์ส่วนหัวที่เกี่ยวข้องและสร้างไฟล์ส่วนหัวหลังการประมวลผล ซึ่งสามารถส่งผ่านไปยังคำสั่ง **probevue** อ็อพชันอื่นๆคือ การเขียนโค้ดด้วยมือลงในไฟล์ส่วนหัว เพื่อสอดแทรกชนิดของนิยามและการประกาศฟังก์ชันที่คุณใช้ในสคริปต์ Vue ของคุณ

ตัวอย่างต่อไปนี้จะแสดงไฟล์ส่วนหัวที่เขียนโค้ดด้วยมือด้วยนิยาม typedef ที่ถูกใช้ในสคริปต์ Vue :

```
/* My header file : myheader.i */

 typedef int myid_t;
```

The following Vue script uses this **typedef** definition:

```
/* Program name: myscript.e */
 @@BEGIN
 {
  myid_t id;
  id = 0;
 }
```

คุณสามารถรันคำสั่ง **probevue** :

```
probevue -I myheader.i myscript.e
```

คุณสามารถระบุไฟล์ส่วนหัวบนบรรทัดรับคำสั่ง โดยแยกไฟล์ส่วนหัวด้วยเครื่องหมายจุลภาค (ไม่มีช่องว่างระหว่าง เครื่องหมายคอมมา และชื่อไฟล์) หรือโดยการระบุไฟล์หนึ่งไฟล์และแยกด้วยแฟล็ก **-I** ตัวอย่างสองตัวอย่างต่อไปนี้เหมือนกัน :

```
probevue -I myheader.i,myheader2.i myscript.e
probevue -I myheader.i -I myheader2.i myscript.e
```

สามารถรวมไฟล์ส่วนหัว C++ สำหรับคำนิยาม struct/class และอนุญาตให้สคริปต์ **probevue** เพื่อเข้าถึงฟิลด์ข้อมูล struct/class ผ่านตัวชี้ ไฟล์ส่วนหัว C++ ทั้หมดสามารถแสดงรายการได้โดยใช้ **#include** directives ระหว่าง **##C++** และ **##Vue** directive ในสคริปต์ ProbeVue สำหรับการใช้อ็อพชันนี้ ต้องติดตั้ง IBM คอมไพลเลอร์ C++ บนระบบ อ็อพชันอื่นเพื่อรวมไฟล์ส่วนหัว C++ คือการใช้ preprocess ไฟล์ส่วนหัว C++ แรกที่มีอ็อพชัน **-P** ของ **probevue** และจากนั้นรวมไฟล์ที่ถูก preprocessed กับอ็อพชัน **-I** ของ **probevue** โดยใช้อ็อพชัน **-P probevue** จะสร้าง out file ที่มีชื่อเดียวกับไฟล์ส่วนหัว C++ อินพุตที่มีส่วนต่อท้าย **.Vue**

ข้อดีของการใช้อ็อพชัน **-I** สำหรับ preprocessed ไฟล์ส่วนหัว C++ คือ ไม่จำเป็นต้องติดตั้งคอมไพลเลอร์ IBM C++ บนระบบ

คุณสามารถใช้คำสั่งต่อไปนี้เพื่อ preprocess ไฟล์ส่วนหัว C++

```
probevue -P myheader.h
```

หมายเหตุ: เมื่อต้องการรันคำสั่งข้างต้น คอมไพลเลอร์ของ IBM C++ เป็นสิ่งที่จำเป็นต้องมีก่อน

คำสั่งด้านบนจะสร้างไฟล์ ที่ชื่อ **myheader.Vue** จากนั้น สามารถจัดส่งไฟล์นี้ไปยัง ระบบอื่นและสามารถใช้เพื่อโพรบแอ็พพลิเคชัน C++ โดยการรวม อ็อพชัน **–I** ของ **probevue** ขณะใช้ไฟล์ส่วนหัว preprocessed C++ ที่มีให้ สภาวะแวดล้อมระบบควรเหมือนกับระบบที่ใช้สำหรับการสร้างไฟล์ส่วนหัว preprocessed C++ และระบบที่ใช้เพื่อรวมไฟล์ส่วนหัว preprocessed ที่มีอ็อพชัน **–I** ของ **probevue** สำหรับการโพรบแอ็พพลิเคชัน C++

ไฟล์ส่วนหัว C++ ที่ถูกใช้สำหรับการคอมไพล์ล่วงหน้ากับอ็อพชัน **–P** หรือถูกรวมระหว่าง **##C++** และ **##Vue** ควรมีส่วนขยาย **.h** สำหรับการรวบไฟล์ส่วนหัว Input/Output C++ มาตรฐาน สำหรับการรวมส่วนหัว IOstream ให้ใช้ **#include<iostream.h>** แทน **#include<iostream>**

คุณสามารถรันคำสั่งต่อไปนี้สำหรับไฟล์ C++ ที่สามารถเรียกทำงานได้ที่ชื่อ **cpp_executable** และสคริปต์ที่ชื่อ **myscript.e** เพื่อโพรบแอ็พพลิเคชัน C++

```
probevue -I myheader.Vue -X cpp_executable myscript.e
```

หมายเหตุ: เมื่อต้องการรันคำสั่งข้างต้น คอมไพลเลอร์ของ IBM C++ ไม่ใช่สิ่งที่จำเป็นต้องมีก่อน

อิลิเมนต์ shell ที่สนับสนุน

ไวยากรณ์ภาษา Vue ประกอบด้วยส่วนสนับสนุนสำหรับตัวแปร shell ที่ระบุโดยคำนำหน้า **$** เช่น ตัวแปร shell ที่เอ็กซ์พอร์ตและพารามิเตอร์เชิงตำแหน่ง (อาร์กิวเมนต์ในสคริปต์)

ตัวแปร Vue shell สามารถปรากฏได้ทุกที่ในสคริปต์ Vue ซึ่งสามารถเป็นส่วนหนึ่งของข้อกำหนดคุณสมบัติโพรบ ที่ถูกใช้ในเพรดิเคตหรือภายในคำสั่งในบล็อกการดำเนินการ อย่างไรก็ตาม ไม่เหมือนกับสคริปต์ shell shelll ไม่สามารถขยายได้ ถ้าใช้ภายในสตริงเครื่องหมายอัญประกาศคู่

อาร์กิวเมนต์ที่ส่งผ่านจากบรรทัดรับคำสั่งไปยังสคริปต์ จะถูกอ้างอิงภายในสคริปต์เป็น **$1**, **$2**, **$3** และอื่นๆ โปรดพิจารณาสคริปต์ Vue ต่อไปนี้:

```
/* Program name: myscript.e */
 @@syscall:*:read:entry
  when (__pid == $1)

 {
  int count;
  count++;
 }
```

ในตัวอย่างต่อไปนี้ ID การประมวลผลของการประมวลผลที่รันโปรแกรม **myprog** จะแทนที่ **$1** ในสคริปต์ก่อนหน้า ซึ่งยอมรับว่า โปรแกรม **prgrep** shell ที่พิมพ์ ID การประมวลผลจะกำหนดชื่อของการประมวลผล ที่ถูกใช้เพื่อเรียกใช้งานสคริปต์ Vue

```
probevue myscript.e `prgrep myprog`
```

ตัวแปรสภาวะแวดล้อมที่เอ็กซ์พอร์ตจาก shell ยังสามารถอ้างอิงได้ในสคริปต์ ที่ใช้ตัวดำเนินการ **$** โปรดพิจารณาสคริปต์ Vue ต่อไปนี้:

```
/* Program name: myscript2.e */
 @@syscall:*:read:entry
  when (__pid == $PID)

 {
  int count;
```

```
 count++;
}

/* program to be traced has a function called 'foo' */

@@uft:$PID:*:foo:entry
{
 printf("Read system call was invoked %d times\n", count);
}
```

ในตัวอย่างต่อไปนี้ 3243 จะแทนที่ $PID ในสคริปต์ก่อนหน้า:

```
PID=3423 probevue myscript2.e
```

If an environment variable needs to be recognized as a string inside the ProbeVue script, the value of the environment variable must include the enclosing double quotation mark that identify it as a string. ตัวอย่างเช่น สคริปต์ต่อไปนี้จะดักจับเอาต์พุตเมื่อไฟล์ที่ระบุ ถูกเปิดในระบบ:

```
/* Program name: stringshell.e */
 int open(char *path, int oflag);
 @@syscall:*:open:entry
 {
  String s[40];
  s = get_userstring(__arg1, -1);
  if (s == $FILE_NAME) {
   printf("pid %d (uid %d) opened %s\n",__pid,__uid, s);
   exit();
  }
 }
```

The script expects that $FILE_NAME is the name of an exported shell environment variable, which includes the double quotation mark in its value. สคริปต์ต่อไปนี้คือตัวอย่าง:

```
export FILE_NAME=\"/etc/passwd\"
probevue stringshell.e
```

If the value of an existing environment variable that does not have double quotation mark is required in a script, then a new environment variable will need to be constructed using double quotation mark around the existing environment variable. สคริปต์ต่อไปนี้คือตัวอย่าง:

```
export FILE_NAME=\"$HOME\"
probevue stringshell.e
```

Vue สนับสนุนตัวแปรสภาวะแวดล้อมที่มีประโยชน์ เมื่อการประมวลผลที่ต้องการโพรบเริ่มต้นโดยคำสั่ง **probevue** โดยใช้แฟล็ก **-X** ตัวแปรสภาวะแวดล้อม **$__CPID** บ่งชี้ว่า ID การประมวลผลของ child process จะถูกสร้างโดยคำสั่ง **probevue** และตัวแปรสภาวะแวดล้อม **$__CTID** จะบ่งชี้โดย ID เธรด แฟล็ก **-X** จะมีประโยชน์ต่อโพรบในการประมวลผลระยะสั้น โดยเฉพาะอย่างยิ่งสำหรับ วัตถุประสงค์ในการดีบัก

สคริปต์ Vue สามารถเรียกใช้งานได้โดยตรง (เช่นเดียวกับสคริปต์ shell) โดยตั้งค่าบรรทัดแรกให้กับสคริปต์ดังต่อไปนี้:

```
#!/usr/bin/probevue
```

คำสั่ง **probevue** ยังสามารถอ่านสคริปต์ Vue จากอินพุตมาตรฐานได้เช่นเดียวกับที่ shell ทำ ซึ่งสามารถบรรลุเป้าหมายได้โดยยกเลิกชื่อไฟล์สคริปต์จากบรรทัดรับคำสั่ง ซึ่งจะมีประโยชน์ในการทดสอบสคริปต์สั้นๆ

Vue ไม่สนับสนุนพารามิเตอร์ shell แบบพิเศษ เช่น **$$** และ **$@** ซึ่งถูกสร้างขึ้น ภายใน shell

ตัวช่วยดักจับการติดตาม

ProbeVue สนับสนุนความเข้าใจตัวช่วยดักจับการติดตาม การดำเนินการดักจับการติดตามพื้นฐานจะจัดเตรียมไว้ผ่านฟังก์ชัน **printf** ที่สามารถเรียกใช้งานจากโพรบใดๆ ที่เป็นส่วนหนึ่งของบล็อกการดำเนินการ Vue ในเวอร์ชันของฟังก์ชัน **printf** จะประกอบด้วย กำลังของเวอร์ชันไลบรารี C เป็นส่วนใหญ่ ฟังก์ชันการดักจับการติดตามสำรอง คือฟังก์ชัน **trace** ฟังก์ชัน **trace** จะยอมรับตัวแปรเดี่ยวเป็นพารามิเตอร์ และทำสำเนาค่าในรูปแบบของเลขฐานสิบหก ที่สามารถพิมพ์ได้ให้กับบัฟเฟอร์การติดตาม ฟังก์ชันนี้จะมีประโยชน์โดยเฉพาะ สำหรับการดัมพ์เนื้อหาของสตริงและโครงสร้าง ฟังก์ชัน **stktrace** ในฟังก์ชันการดักจับการติดตาม ซึ่งดักจับการติดตามสแต็กของเธรดที่ถูกติดตามที่จุดโพรบปัจจุบัน

นอกจากค่าจองตัวแปรสคริปต์ภายในแล้ว ตัวแปรภายนอก เช่น ตัวแปรโกลบอล ข้อความที่ระบุเฉพาะกับบริบท เช่น พารามิเตอร์ในฟังก์ชันที่ต้องการโพรบ ค่าส่งคืนจากฟังก์ชัน และอื่นๆ สามารถดักจับและแสดงผลผ่านการดักจับการติดตามเหล่านี้

ตัวรายงานการติดตามจะแสดงข้อมูลการติดตามเสมอในเวลาที่เกิดขึ้น และข้อมูลที่ดักจับจาก CPU ที่ต่างกันจะถูกเรียงลำดับภายใน ก่อนที่จะเอาต์พุต

การ trace ชั่วคราว

การ trace ชั่วคราวคือตัวช่วยที่อยู่ใน ProbeVue เพื่อใช้ดักจับข้อมูล data ตามเงื่อนไขที่กำหนด ฟังก์ชัน **start_tentative** จะใช้ภายในบล็อกการดำเนินการเพื่อบ่งชี้จุดเริ่มต้นของการดักจับ ข้อมูลการติดตามที่เป็นการทดสอบ ฟังก์ชัน **end_tentative** จะแสดงจุดสิ้นสุดของการดักจับการติดตามภายในบล็อกการดำเนินการ ฟังก์ชัน **end_tentative** สามารถยกเลิกได้ ซึ่งในกรณีที่จุดสิ้นสุดของบล็อกการดำเนินการจะแสดง จุดสิ้นสุดของการทดสอบการติดตามที่เริ่มต้นในบล็อกนั้น การติดตามที่ดักจับใดขณะที่ทดสอบการติดตามถูกเปิดใช้งานจะถูกทำเครื่องหมายเป็น ข้อมูลการทดสอบการติดตาม ในเวลาสั้นๆ จุดเริ่มต้นและจุดสิ้นสุดฟังก์ชันการทดสอบการติดตาม จะสร้างส่วนของการทดสอบการติดตามภายบล็อกการดำเนินการที่อยู่ภายใน ข้อมูลการติดตามทั้งหมดที่ดักจับซึ่งจะถูกนำมาพิจารณาเป็นการทดสอบ

ข้อมูลการติดตามที่ดักจับไม่เคยพร้อมใช้งานกับผู้ใช้การติดตาม แต่จะพักอยู่ภายในบัฟเฟอร์จนกว่าจะถูก commit หรือยกเลิก อย่างใดอย่างหนึ่ง ฟังก์ชัน **commit_tentative** จะ commit ข้อมูลการทดสอบการติดตามที่เก็บรวบรวมทั้งหมด และทำให้พร้อมใช้งานกับผู้ใช้การติดตาม ฟังก์ชัน **discard_tentative** จะยกเลิกข้อมูลการติดตามที่เก็บรวบรวมไว้ทั้งหมด และล้างข้อมูลพื้นที่ในบัฟเฟอร์การติดตาม

ไม่มีการทดสอบ การติดตามปกติของข้อมูลที่อนุญาติให้ทำพร้อมกัน ด้วยการทดสอบการติดตาม ข้อมูลการติดตามทั้งหมดจะพร้อมใช้งานกับการติดตาม ในลำดับของการประทับเวลา ดังนั้นข้อมูลการทดสอบการติดตาม ที่ไม่ commit หรือยกเลิกสามารถป้องกันการติดตามข้อมูลปกติจาก การป้องกันการใช้การติดตาม ในกรณีร้ายสุด การดำเนินการสามารถหยุดการดักจับข้อมูลการติดตาม เนื่องจากบัฟเฟอร์การติดตาม จะกรอกข้อมูลด้วยข้อมูลการทดสอบการติดตามที่ไม่ได้ commit หรือละเว้น

ฟังก์ชันการทดสอบการติดตามจะใช้สตริงเพื่อระบุ ID ภายใต้ข้อมูลการทดสอบการติดตามที่ดักจับ ซึ่งอนุญาตให้ควบคุม เมื่อ commit หรือยกเลิกการดักจับข้อมูลการทดสอบ การติดตาม คุณสามารถครอบคุลมการดำเนินการดักจับการติดตามได้ใน

บล็อก ตั้งแต่หนึ่งบล็อกขึ้นไป ภายในการเริ่มต้นและสิ้นสุดฟังก์ชันการทดสอบการติดตาม ที่ส่งผ่านไปยัง ID การทดสอบการติดตาม ซึ่งอนุญาตให้ข้อมูลการทดสอบการติดตามจากตำแหน่งต่างๆ commit หรือยกเลิกได้ในบล็อกเดี่ยว Vue ต้องการสตริงตามตัวอักษรที่ต้องใช้เป็น ID การทดสอบการติดตาม

การทดสอบการติดตามอนุญาตให้กรองข้อมูล และลดจำนวนที่เกิดขึ้นจริงของข้อมูลที่แสดงให้คุณ และที่คุณต้องการวิเคราะห์ ซึ่งจะมีเอฟเฟ็กต์ของการป้องกันปัญหาบัฟเฟอร์โอเวอร์โฟลว์ ถ้าคุณสามารถมั่นใจได้ว่า ข้อมูลที่เก็บรวบรวมจะถูกละทิ้งหรือ commit เร็วขึ้น

สคริปต์ต่อไปนี้คือตัวอย่างของการใช้ฟังก์ชัน trace ชั่วคราวเพื่อดักจับข้อมูล trace เท่านั้น หากจำเป็น:

```
/*
 * File: tentative.e
 *
 * Print details when write system call takes longer than a
 * specified number of microseconds
 *
 * Usage: probevue tentative.e <processID> <microseconds>
 */
int write(int fd, char *buf, int size);

@@BEGIN
{
 probev_timestamp_t ts1, ts2;
}

@@syscall:$1:write:entry
{
 __auto String buf[256];

 if (__arg3 < 256)
  buf = get_userstring(buf, __arg3);
 else
  buf = get_userstring(buf, 256);

 start_tentative("write");

 /* print out all the data associated with the write */
 stktrace(PRINT_SYMBOLS|GET_USER_TRACE, -1);

 printf("fd = %d, size = %d\n", __arg1, __arg3);

 /* Prints 256 bytes of buf, even though size may be < 256 */
 trace(buf);

 end_tentative("write");

 /* Get timestamp for when we entered write: do this at the end of
  * the probe to reduce probe effect
  */
 ts1 = timestamp();
}

/* If we started probing in the middle of write, ts1 will be zero,
 * ignore that case with a predicate
```

```
 */
@@syscall:$1:write:exit
 when (ts1 != 0)
{
 /* diff_time() may return up to a 64-bit value, but we
  * use an int here since we don't expect the difference to
  * larger than a few hundred microseconds at the most.
  */
 int micros;

 /* Get timestamp for when we exited write: do this at the beginning of
  * the probe to reduce probe effect
  */
 ts2 = timestamp();

 micros = diff_time(ts1, ts2, MICROSECONDS);

 start_tentative("write");
 printf("Return value from write = %d\n", __rv);
 end_tentative("write");

 if (micros > $2) {
  /* Can mix normal trace with tentative also  */
  printf("Time to write = %d, limit =%d micro seconds\n",
   micros, $2);
  commit_tentative("write");
  exit();
 }
 else
  discard_tentative("write");
}
```

ฟังก์ชัน Vue

ส่วนนี้อธิบายเกี่ยวกับฟังก์ชัน Vue

ภาษา Vue สนับสนุนรายชื่อของฟังก์ชันดังต่อไปนี้:

ฟังก์ชัน	คำอธิบาย
add_range	เตรียมข้อมูลเบื้องต้นชนิดข้อมูล range สตริง
ภาคผนวก	ต่อท้ายค่าในรายการ
atoi	ส่งคืนค่าจำนวนเต็มของสตริง
avg	ส่งคืนค่าเฉลี่ยขององค์ประกอบ ในรายการ
commitdiscard	Commit หรือละทิ้งข้อมูลในบัฟเฟอร์ trace ชั่วคราว
convert_ip4_addr	แปลงแอดเดรส IPv4 (ข้อมูล) เป็นรูปแบบ ProbeVue data-type **ip_addr_t**
convert_ip6_addr	แปลงแอดเดรส IPv6 (ข้อมูล) เป็นรูปแบบ ProbeVue data-type **ip_addr_t**
copy_kdata	คัดลอกข้อมูลจากหน่วยความจำเคอร์เนลไปยังตัวแปรสคริปต์ Vue
copy_userdata	คัดลอกข้อมูลจากหน่วยความจำผู้ใช้ไปยังตัวแปรสคริปต์ Vue

ฟังก์ชัน	คำอธิบาย
count	ส่งคืนการนับของจำนวนองค์ประกอบ ในรายการ
diff_time	ส่งคืนค่าความต่างระหว่างการประทับเวลาสองครั้ง
eprintf	จัดรูปแบบและพิมพ์ข้อมูลเกี่ยวกับข้อผิดพลาดมาตรฐาน
exit	ยกเลิกสคริปต์ Vue
fd_fname	รับชื่อไฟล์สำหรับ descriptor ไฟล์ที่ระบุเฉพาะ
fd_fstype	รับชนิดระบบไฟล์สำหรับ descriptor ไฟล์ที่ระบุเฉพาะ
fd_ftype	รับชนิดไฟล์สำหรับ descriptor ไฟล์ที่ระบุเฉพาะ
fd_inodeid	รับ inode ID สำหรับ descriptor ไฟล์ที่ระบุเฉพาะ
fd_mpath	รับพาธเมาท์ของระบบไฟล์สำหรับ descriptor ไฟล์ที่ระบุเฉพาะ
fd_path	รับพาธไฟล์สัมบูรณ์สำหรับ descriptor ไฟล์ที่ระบุเฉพาะ
fpath_inodeid	รับ inode ID สำหรับพาธไฟล์ที่ระบุเฉพาะ
get_function	ส่งคืนชื่อของฟังก์ชันที่โพรบแล้ว
get_kstring	คัดลอกข้อมูลจากหน่วยความจำเคอร์เนลไปยังตัวแปรสตริง
get_location_point	ส่งคืนจุดตำแหน่งโพรบปัจจุบัน
get_probe	ส่งคืนข้อกำหนดคุณลักษณะของจุดโพรบปัจจุบัน
get_stktrace	คืนค่าการติดตามรันไทม์สแต็ก
get_userstring	คัดลอกข้อมูลจากหน่วยความจำผู้ใช้
list	สร้างและส่งคืนรายการเปล่าที่สร้างใหม่
lquantize	Quantizes เชื่อมโยงค่าอาร์เรย์ และพิมพ์คู่ key-value ในรูปแบบกราฟิก
max	ส่งคืนค่าสูงสุดขององค์ประกอบ ในรายการ
min	ส่งคืนค่าต่ำสุดขององค์ประกอบทั้งหมด ในรายการ
args	พิมพ์ชื่อของฟังก์ชันที่ถูกโพรบและอาร์กิวเมนต์ของฟังก์ชัน
print	พิมพ์คู่ key-value ไรอาร์เรย์ ที่เชื่อมโยง
printf	จัดรูปแบบและทำสำเนาข้อมูลลงในบัฟเฟอร์ trace
ptree	พิมพ์แผนผังกระบวนการสำหระบกระบวนการที่ถูกโพรบ
quantize	กำหนดจำนวนค่าอาร์เรย์ที่สัมพันธ์กันแบบเชิงเส้น และพิมพ์คู่คีย์-ค่าในรูปแบบกราฟิก
qrange	ค้นหาหมายเลขสล็อตของช่วงและเพิ่ม เข้ากับอาร์เรย์ Associative
round_trip_time	รับเวลาของการเดินทางไปกลับที่ราบเรียบสำหรับการเชื่อมต่อ TCP สำหรับ descriptor ซ็อกเก็ตที่ระบุเฉพาะ
set_aso_print_options	ระบุ sort-type, sort-by และแฟล็ก ***list-value***
set_date_format	อัพเดตรูปแบบวันที่
set_range	กำหนดค่าเริ่มต้นชนิด linear และ power range

ฟังก์ชัน	คำอธิบาย
sockfd_netinfo	ขอรับข้อมูลโลคัลและรีโมตพอร์ต และ IP พอร์ต สำหรับตัวให้คำอธิบายซ็อกเก็ตเฉพาะ
startend_tentative	บ่งชี้ถึงจุดเริ่มต้นและจุดสิ้นสุดของส่วนของ trace ชั่วคราว
stktrace	สร้างและพิมพ์ trace แบบสแต็ก ณ รันไทม์
strstr	ส่งคืนสตริงภายในสตริงอื่น
sum	ส่งคืนผลรวมของส่วนประกอบทั้งหมดที่อยู่ในรายการ
timestamp	ส่งคืนการประทับเวลาปัจจุบัน
การติดตาม	คัดลอกข้อมูลดิบไปยังบัฟเฟอร์ trace ด้วยอักขระเลขฐานสิบหก

add_range

วัตถุประสงค์:

เตรียมข้อมูลเบื้องต้นชนิดข้อมูล range สตริง และเพิ่ม สตริงในสล็อต

ไวยากรณ์:

```
add_range(range_t range_data, String S1, String S2, ..., String Sn);
```

คำอธิบาย:

รูทีนนี้เตรียมข้อมูลเบื้องต้น `range_data` เป็น ชนิดข้อมูล range สตริง และยังเพิ่มสตริงทั้งหมดที่ส่งเป็นอาร์กิวเมนต์ไปยังรูทีนในหนึ่งสล็อต ถ้ารูทีนนี้ถูกเรียกใช้ครั้งแรก สำหรับชนิดข้อมูล range สตริงจะถูกเพิ่มในสล็อตแรก มิฉะนั้น สตริงจะถูกเพิ่มในสล็อตถัดไป

พารามิเตอร์:

`range_data`
ชนิดข้อมูล **range_t**

`S1, S2,...`
สตริงที่จะเพิ่มในพารามิเตอร์ **range_data**

append

วัตถุประสงค์

ต่อท้ายค่าในรายการ

ไวยากรณ์

```
void append ( List listvar, long long val );
```

คำอธิบาย

ฟังก์ชัน **append** คือรายการฟังก์ชันการที่มีอยู่ใน Vue ซึ่งจะต่อท้ายค่าที่ระบุด้วยพารามิเตอร์สำรอง ในตัวแปร list ที่ระบุด้วยพารามิเตอร์แรก การเรียกฟังก์ชัน **append** แต่ละครั้ง จะเพิ่มค่าให้กับชุดของค่าที่บันทึกไว้แล้วในตัวแปร list และขนาดของตัวแปร list จะโตขึ้นเรื่อยๆ ฟังก์ชัน append ยังอนุญาตให้ใช้รายการอื่นเป็นอาร์กิวเมนต์ ซึ่งอนุญาตให้คุณเชื่อมต่อรายการสองรายการได้

หมายเหตุ: ค่าที่เพิ่มให้กับรายการต้องมีพารามิเตอร์ที่ครบถ้วน หรือชนิดของรายการ หรือผลลัพธ์ที่จะส่งผลให้เกิดข้อผิดพลาดทางไวยากรณ์ คอมไพลเลอร์ ProbeVue จะยอมรับพารามิเตอร์ใดๆ ที่มีชนิดข้อมูลแบบเลขจำนวนเต็ม C-89 ซึ่งจะเป็นแบบลงชื่อแล้วและยังไม่ลงชื่อ ไม่มีการคำนวณที่ต้องการ

ฟังก์ชัน **append** จะไม่มีค่าสำหรับส่งคืน

สำหรับข้อมูลเกี่ยวกับชนิดข้อมูลรายการ โปรดดูที่ ชนิดข้อมูลใน Vue ส่วน list ที่มาก่อนมีสคริปต์ตัวอย่าง ที่ใช้ฟังก์ชัน **append**

พารามิเตอร์

พารามิเตอร์	คำอธิบาย
listvar	ระบุตัวแปรของชนิด รายการ
val	ระบุค่าหรือรายการที่ต้องการต่อท้าย

atoi

วัตถุประสงค์

ส่งคืนค่าจำนวนเต็มของสตริง

ไวยากรณ์

```
int atoi( String str );
```

คำอธิบาย

ฟังก์ชัน **atoi** จะส่งคืนเลขจำนวนเต็ม ที่มีค่าที่แสดงด้วยสตริงซึ่งระบุด้วยพารามิเตอร์ *str* และจะอ่านสตริงได้มากสุดอักขระตัวแรกซึ่งไม่ใช่หลักตัวเลข (0-9) และแปลงอักขระที่ตรวจแล้วลงในเลขจำนวนเต็ม ที่มีค่าเท่ากัน การนำหน้าด้วยช่องว่างจะถูกข้าม และตัวบ่งชี้สัญญาณเผื่อเลือกสามารถนำหน้าหลักได้

ฟังก์ชัน **atoi** มีประโยชน์ในการแปลงสตริงกลับไปเป็นเลขจำนวนเต็ม ขณะที่รันสคริปต์เชลล์ sprobevue ซึ่งจะถูกตัดเครื่องหมายอัญประกาศคู่รอบๆ อาร์กิวเมนต์ทั้งหมด สคริปต์ต่อไปนี้ คือตัวอย่างที่ดักจับกระบวนการแตกสาขาได้เร็วกว่าที่คาดการณ์ไว้

```
/* File: ffork.e
 *
 * Usage: sprobevue ffork.e processname delta
 *
 * Traces whenever a specified process is forking faster than
 * the "delta" value passed. Specify a process name and the time
 * in milliseconds as parameters.
 */

/* Ignore other parameters to execve */
int execve(char *path);
```

```
@@BEGIN
{
 int done;
 int pid;

 pname = $1;  /* name of process we are monitoring */

 /*
  * Since sprobevue is used, need to extract the integer value
  * from the string (double quotes around the delta).
  */
 delta = atoi($2); /* minimum delta in millisecs between forks */
 printf("pname = %s, delta = %d\n", pname, delta);
}

@@syscall:*:execve:entry
 when (done == 0)
{
 __auto String exec[128];
 __thread int myproc;

 /* Find process being 'exec'ed */
 exec = get_userstring(__arg1, 128);

 /* Got it. Set a thread-local and reset 'done' so that we
  * avoid entering this probe from now on.
  */
 if (exec == pname) {
  pid = __pid;
  myproc = 1;
  done = 1;
  printf("Process name = %s, pid = %d\n", __pname, pid);
 }
}

@@syscall:*:fork:entry
 when (thread:myproc == 1)
{
 /* old_ts is initialized to zero */
 probev_timestamp_t old_ts, new_ts;
 unsigned long long interval;

 /* Get current time stamp */
 new_ts = timestamp();

 /* Find time from last fork */
 if (old_ts != 0) {
  interval = diff_time(old_ts, new_ts, MILLISECONDS);

  /* if below the specified limit, trace that */
  if (interval < delta)
   printf("%s (%ld) forking too fast (%d milliseconds)\n",
     pname, __pid, interval);
 }
```

```
 /* Save current fork timestamp */
 old_ts = new_ts;
}

@@syscall:*:exit:entry
 when (__pid == pid)
{
 /* Catch process exit and terminate ourselves */
 printf("Process '%s' exited.\n", pname);
 exit();
}
```

พารามิเตอร์

พารามิเตอร์	คำอธิบาย
str	ระบุสตริงที่ต้องการแปลง

avg

วัตถุประสงค์

ส่งคืนค่าเฉลี่ยขององค์ประกอบในรายการ

ไวยากรณ์

```
long long avg ( List listvar );
```

คำอธิบาย

ฟังก์ชัน **avg** ส่งคืนค่าเฉลี่ยขององค์ประกอบทั้งหมดที่ได้ผนวกเข้ากับตัวแปร list ซึ่งระบุด้วยพารามิเตอร์ *listvar*

พารามิเตอร์

พารามิเตอร์	คำอธิบาย
listvar	ระบุตัวแปรของชนิด รายการ

commit_tentative, discard_tentative

วัตถุประสงค์

Commit หรือละทิ้งข้อมูลในบัฟเฟอร์ trace ชั่วคราว

ไวยากรณ์

```
void commit_tentative( String bufID );
void discard_tentative( String bufID );
```

คำอธิบาย

ฟังก์ชัน **commit_tentative** จะ commit ข้อมูล trace ที่เชื่อมโยงกับบัฟเฟอร์ trace ชั่วคราวที่ระบุโดยพารามิเตอร์ *bufID* ซึ่งจะบันทึกข้อมูล และทำให้พร้อมใช้งานกับคอนซูมเมอร์ trace

ฟังก์ชัน **discard_tentative** จะละทิ้งข้อมูลทั้งหมดในบัฟเฟอร์ trace ชั่วคราวที่บ่งชี้ด้วยพารามิเตอร์ *bufID* และจะเพิ่มพื้นที่ว่างในบัฟเฟอร์ trace ที่ใช้งานโดยข้อมูล trace ชั่วคราว

เมื่อข้อมูล trace ชั่วคราวกำลังถูกบันทึกพร้อมกับข้อมูล trace ปกติ นั่นคือ erstwhile ชั่วคราว แต่การ commit ข้อมูล trace และข้อมูล trace ปกติจะทำให้คอนซูมเมอร์ trace พร้อมใช้งานในลำดับของการประทับเวลา ดังนั้น จึงเป็นแนวคิดที่ดีในการ commit หรือละทิ้งข้อมูลชั่วคราวในตอนแรกเพื่อเพิ่มพื้นที่ว่างบัฟเฟอร์ trace

ข้อมูล trace ชั่วคราวที่ไม่ได้ถูก commit จะถูกละทิ้ง เมื่อเซสชัน ProbeVue สิ้นสุดลง

Tentative tracing จะอธิบาย การติดตามชั่วคราวในรายละเอียดเพิ่มเติมและรวมสคริปต์ Vue ตัวอย่างที่ใช้การติดตามชั่วคราว

พารามิเตอร์

พารามิเตอร์	คำอธิบาย
bufID	ระบุค่าคงที่ของสตริงที่บ่งชี้ถึง ID บัฟเฟอร์การติดตามชั่วคราว

convert_ip4_addr

วัตถุประสงค์

แปลง IPV4 แอดเดรส (ข้อมูล) เป็นรูปแบบชนิดข้อมูลแอดเดรส ProbeVue IP

ไวยากรณ์

```
ip_addr_t convert_ip4_addr (unsigned int ipv4_data);
```

คำอธิบาย

ฟังก์ชัน **convert_ip4_addr** จะแปลงแอดเดรส IPv4 ในโครงสร้าง **in_addr** ที่กำหนดไว้ในไฟล์ **/usr/include/netinet/in.h** เป็นชนิดข้อมูล ProbeVue IP address **ip_addr_t** ฟังก์ชันนี้ส่งคืนค่า **ip_addr_t** ที่ถูกแปลง

พารามิเตอร์

`ipv4_data`
ระบุข้อมูลแอดเดรส ipv4 ที่ต้องถูกแปลงเป็นรูปแบบ **ip_addr_t**

convert_ip6_addr

วัตถุประสงค์

แปลงแอดเดรส IPv6 (ข้อมูล) เป็นรูปแบบชนิดข้อมูลแอดเดรส ProbeVue IP

ไวยากรณ์

```
ip_addr_t convert_ip6_addr (int *ipv6_data);
```

คำอธิบาย

ฟังก์ชัน **convert_ip6_addr** จะแปลงแอดเดรส IPv6 ในโครงสร้าง **in6_addr** ที่กำหนดไว้ในไฟล์ **/usr/include/netinet/in.h** เป็นชนิดข้อมูล ProbeVue IP address **ip_addr_t** ฟังก์ชันนี้ส่งคืนค่า **ip_addr_t** ที่ถูกแปลง

พารามิเตอร์

ipv6_data
ระบุข้อมูลแอดเดรส ipv6 ที่ต้องถูกแปลงเป็นรูปแบบ **ip_addr_t**

สคริปต์ต่อไปนี้เป็นตัวอย่างที่พิมพ์ข้อมูลเกี่ยวกับกระบวนการที่ถูกโพรบ ที่กำลัง ส่งข้อมูล

```
/* Declare the Function prototype */
int sendto(int s,  char * uap_buf,   int len,  int flags,  char * uap_to,  int tolen);

typedef unsigned int in_addr_t;

/* Structure Declarations */

/* Declare the in_addr structure */
struct in_addr {
        in_addr_t        s_addr;
};

/* Declare the sockaddr_in structure */
struct sockaddr_in {
        unsigned char           sin_len;
        unsigned char           sin_family;
        unsigned short          sin_port;
        struct in_addr          sin_addr;
        unsigned char           sin_zero[8];
};
/* Declare the in6_addr structure */
struct in6_addr {
        union {
                int s6_addr32[4];
                unsigned short s6_addr16[8];
                unsigned char  s6_addr8[16];
        } s6_addr;
};
/* Declare the sockaddr_in6 structure */
struct sockaddr_in6 {
        unsigned char          sin6_len;
        unsigned char          sin6_family;
        unsigned short         sin6_port;
        unsigned int           sin6_flowinfo;
        struct   in6_addr      sin6_addr;
        unsigned int           sin6_scope_id;  /* set of interfaces for a scope */
};


/* Print the information about to whom it is sending data */
@@syscall:*:sendto:entry
{
```

```
        struct sockaddr_in6 in6;
        struct sockaddr_in in4;
        ip_addr_t ip;

                /* Copy the arg5 data into sockaddr_storage variable */
                /* using copy_userdata( ) Vue function */
        copy_userdata(__arg5, in4);

       /*
        * Verify  whether the destination address is IPv4 or IPv6 and based on that call the
        * corresponding IPv4 or IPV6 conversion routine.
        */
       if (in4.sin_family == AF_INET)
       {
               /* Copy the ipv4 data into sockaddr_in structure using copy_userdata routine */
                copy_userdata(__arg5, in4);

               /* Convert Ipv4 data into ip_addr_t format */
                ip = convert_ip4_addr(in4.sin_addr.s_addr);

               /* Print the destination address and hostname  using %H and %I format specifier */
               printf("It is sending the data to node %H(%I)\n",ip,ip);
        }
        else if(in4.sin_family == AF_INET6)
        {
              /* Copy the ipv6 data into sockaddr_in6 structure using copy_userdata routine */
               copy_userdata(__arg5, in6);

              /* Convert Ipv6 data into ip_addr_t format */
               ip = convert_ip6_addr(in6.sin6_addr.s6_addr.s6_addr32);

              /* Print the destination address and hostname using %H and %I format specifier */
              printf("It is sending the data to node %H(%I)\n", ip,ip);
        }
}
```

count

วัตถุประสงค์

ส่งคืนจำนวนขององค์ประกอบในรายการ

ไวยากรณ์

```
long long count ( List listvar );
```

คำอธิบาย

ฟังก์ชัน **count** ส่งคืนจำนวนขององค์ประกอบที่ได้ผนวกไว้กับตัวแปร listที่ระบุด้วยพารามิเตอร์ *listvar*

สำหรับข้อมูลเกี่ยวกับชนิดข้อมูลรายการ โปรดดูที่ ชนิดข้อมูลใน Vue ส่วน list ที่มาก่อนมีสคริปต์ชั่วคราว ที่ใช้ฟังก์ชัน **count**

พารามิเตอร์

พารามิเตอร์	คำอธิบาย
listvar	ระบุตัวแปรของชนิด รายการ

copy_kdata

วัตถุประสงค์

คัดลอกข้อมูลจากหน่วยความจำเคอร์เนลไปยังตัวแปรสคริปต์ Vue

ไวยากรณ์

```
void copy_kdata( <type> *kaddr,<type>svar);
```

คำอธิบาย

ฟังก์ชัน **copy_kdata** อ่านข้อมูลจากหน่วยความจำเคอร์เนลไปยังตัวแปรสคริปต์ Vue ตัวแปร อาจเป็นชนิด C-89 ซึ่งได้รับการสนับสนุนโดย Vue ยกเว้นสำหรับชนิดตัวชี้ ความยาวที่ถูกคัดลอกเท่ากับขนาดของตัวแปร ตัวอย่างเช่น ขนาด 4 ไบต์ถูกคัดลอกหากตัวแปรสคริปต์ Vue เป้าหมาย มีชนิดเป็น int ขนาด 8 ไบต์ถูกคัดลอกหากมีชนิดเป็น long long และขนาด 48 ไบต์ถูกคัดลอกหากชนิดเป็นอาร์เรย์ 12 integers หรือ int[12]

ข้อมูลที่อยู่ในพื้นที่เคอร์เนลต้องถูกคัดลอกก่อนที่จะสามารถใช้ใน นิพจน์หรือส่งผ่านเป็นพารามิเตอร์ไปยังฟังก์ชัน Vue

ถ้าข้อยกเว้นเกิดขึ้นขณะที่กระบวนการกำลังรันฟังก์ชันนี้อยู่ ตัวอย่างเช่น เมื่อแอดเดรสเคอร์เนลที่ไม่อยู่ในรูปแบบที่ถูกต้องถูกส่งผ่านไปยัง ฟังก์ชัน เซสชัน ProbeVue จะถูกยกเลิกพร้อมกับแสดงข้อความแสดงข้อผิดพลาด

พารามิเตอร์

kaddr ระบุแอดเดรสของข้อมูลพื้นที่เคอร์เนล

svar ระบุตัวแปรสคริปต์ที่คัดลอกเคอร์เนล ตัวแปรสคริปต์สามารถมีชนิดเป็นข้อมูลเคอร์เนล

copy_userdata

วัตถุประสงค์

คัดลอกข้อมูลจากหน่วยความจำผู้ใช้ไปยังตัวแปรสคริปต์ Vue

ไวยากรณ์

```
void copy_userdata( <type> *uaddr,<type>svar);
```

คำอธิบาย

ฟังก์ชัน **copy_userdata** อ่านข้อมูลจากหน่วยความจำผู้ใช้ไปยังตัวแปรสคริปต์ Vue ตัวแปรอาจมีชนิดเป็น C-89 ซึ่งได้รับการสนับสนุนโดย Vue ความยาว ที่ถูกคัดลอกเท่ากับขนาดของชนิดตัวแปร ตัวอย่างเช่น ขนาด 4 ไบต์ถูกคัดลอกหากตัวแปรสคริปต์ Vue เป้าหมาย มีชนิดเป็น int ขนาด 8 ไบต์ถูกคัดลอกหากมีชนิดเป็น long long และขนาด 48 ไบต์ถูกคัดลอกหากมีชนิดเป็นอาร์เรย์ของ 12 integers หรือ int[12]

ข้อมูลในพื้นที่ผู้ใช้ต้องถูกคัดลอกก่อนที่จะสามารถใช้ในนิพจน์ หรือส่งผ่านเป็นพารามิเตอร์ไปยังฟังก์ชัน Vue

ถ้าข้อยกเว้นเกิดขึ้นขณะที่กระบวนการกำลังรันฟังก์ชันนี้อยู่ ตัวอย่างเช่น เมื่อแอดเดรสผู้ใช้ที่อยู่ในรูปแบบที่ไม่ถูกต้องถูกส่งผ่านไปยังฟังก์ชัน เซสชัน ProbeVue จะถูกยกเลิกพร้อมกับแสดงข้อความแสดงข้อผิดพลาด

พารามิเตอร์

uaddr ระบุที่อยู่ของข้อมูลพื้นที่ของผู้ใช้

svar ระบุตัวแปรสคริปต์ที่คัดลอกข้อมูลผู้ใช้ ชนิดของตัวแปรสคริปต์ ต้องเป็นชนิดข้อมูลผู้ใช้

diff_time

วัตถุประสงค์

ส่งคืนค่าความต่างระหว่างการประทับเวลาสองครั้ง

ไวยากรณ์

```
unsigned long long diff_time( probev_timestamp_t ts1, probev_timestamp_t ts2, intformat );
```

คำอธิบาย

ฟังก์ชัน **diff_time** ส่งคืนค่าความต่างของเวลาระหว่างการประทับเวลาสองครั้ง ซึ่งจะบันทึกไว้โดยใช้ฟังก์ชัน **timestamp** ฟังก์ชันนี้สามารถส่งคืนค่าความต่างของเวลาในหน่วยไมโครวินาทีหรือมิลลิวินาทีตามที่ระบุด้วยพารามิเตอร์ *format*

ส่วน get_location_point และ list มีสคริปต์ตัวอย่าง ที่ใช้ฟังก์ชัน **diff_time**

พารามิเตอร์

พารามิเตอร์	คำอธิบาย
ts1	บ่งชี้การประทับเวลาก่อนหน้า
ts2	บ่งชี้การประทับเวลาในภายหลัง
format	ตั้งค่าหนึ่งในค่าต่อไปนี้:
	MILLISECONDS ส่งคืนค่าความต่างของเวลาที่ใกล้เคียงกับมิลลิวินาที
	MICROSECONDS ส่งคืนค่าความต่างของเวลาที่ใกล้เคียงกับไมโครวินาที
	คุณไม่สามารถส่งค่าตัวแปรสำหรับพารามิเตอร์นี้ได้

eprintf

วัตถุประสงค์

จัดรูปแบบและพิมพ์ข้อมูลเกี่ยวกับข้อผิดพลาดมาตรฐาน

ไวยากรณ์

```
void eprintf ( String format[ , data, ... ]);
```

คำอธิบาย

The **eprintf** function is similar to **printf** function except that the output is sent to the standard error. The **eprintf** function converts, formats, and copies the data parameter values to the trace buffer under the control of the format parameter. As indicated by the syntax, a varying list of arguments can be passed as data parameters to the **eprintf** function. Vue สนับสนุนตัวระบุการแปลงทั้งหมด ที่ได้รับการสนับสนุนโดยรูทีนย่อย **printf** ซึ่งถูกจัดเตรียมไว้โดยไลบรารี C ยกเว้นสำหรับตัวระบุการแปลง `%p`

ฟังก์ชัน **eprintf** ไม่สามารถใช้เพื่อพิมพ์ตัวแปรของชนิดรายการได้ อย่างไรก็ตาม ตัวแปรสตริงสามารถพิมพ์ได้โดยใช้ตัวระบุการแปลง *%s* ตัวแปร ชนิด `probev_timestamp_t` ถูกพิมพ์ในรูปแบบตัวเลขโดยใช้ตัวระบุ *%lld* หรือ *%16llx* ตัวแปรชนิด `probev_timestamp_t` ถูกพิมพ์ในรูปแบบวันที่โดยใช้ตัวระบุ *%A* หรือ *%W*

พารามิเตอร์

รูปแบบ

สตริงหนึ่งตัวที่มีอักขระธรรมดาซึ่งถูกคัดลอกไปยังบัฟเฟอร์การติดตามโดยตรงโดยไม่มีการเปลี่ยนแปลงใดๆ ตัวระบุการแปลงอื่นหรือเพิ่มเติมที่จัดเตรียมการบ่งชี้ถึงวิธีการจัดรูปแบบ พารามิเตอร์ข้อมูล สำหรับข้อมูลเพิ่มเติมเกี่ยวกับตัวระบุการแปลง โปรดดู รูทีนย่อย **printf** ใน *Technical Reference: Base Operating System and Extensions, Volume 1*

ข้อมูล ระบุศูนย์หรืออาร์กิวเมนต์เพิ่มเติมที่สอดคล้องกับตัวระบุการแปลงในพารามิเตอร์ format

หมายเหตุ: การติดตามเบื้องต้นไม่ได้รับอนุญาตให้ใช้กับฟังก์ชัน **eprintf**

exit

วัตถุประสงค์

ยกเลิกสคริปต์ Vue

ไวยากรณ์

```
void exit();
```

คำอธิบาย

ฟังก์ชัน **exit** จะยกเลิกสคริปต์ Vue ซึ่งปิดใช้งานโพรบทั้งหมดที่เปิดใช้งานในเซสชันการติดตามแบบไดนามิก ละทิ้งข้อมูลการติดตามชั่วคราวใดๆ ออกคำสั่งให้ดำเนินการที่บ่งชี้ในโพรบ **@@END** และระบุข้อมูลการติดตามที่ดักจับไว้ทั้งหมด ลงในคอนซูมเมอร์การติดตาม หลังจากที่คอนซูมเมอร์การติดตามพิมพ์เอาต์พุตใดๆ ที่ trace อยู่ในโพรบ **@@END** เซสชันการติดตาม จะถูกยกเลิกและกระบวนการ **probevue** จะมีอยู่

ผลกระทบแบบเดียวกันนี้สามารถทำได้โดยพิมพ์ Ctrl-C บนเทอร์มินัลที่คำสั่ง **probevue** จะถูกออกคำสั่ง ถ้าเทอร์มินัลทำงานเป็นงานส่วนหน้า หรือ คุณสามารถส่ง **SIGINT** ไปยังกระบวนการ **probevue** ได้โดยตรงโดยใช้คำสั่ง **kill** หรือการเรียกระบบ **kill**

ส่วน list มี สคริปต์ตัวอย่างที่ใช้ฟังก์ชัน **exit** ส่วน atoi มีสคริปต์ตัวอย่าง สำหรับวิธีการออกในเวลาเดียวกับกระบวนการที่คุณกำลัง ยกเลิกการโพรบ

พารามิเตอร์

ฟังก์ชัน **exit** ไม่ได้ใช้พารามิเตอร์ใดๆ ไม่เหมือนกับรูทีนย่อย **exit** ที่ถูกจัดเตรียมไว้โดยไลบรารี C

fd_fname

วัตถุประสงค์

ส่งคืนชื่อไฟล์สำหรับ descriptor ไฟล์เฉพาะ

ไวยากรณ์

```
char * fd_fname(int fd);
```

คำอธิบาย

ฟังก์ชันนี้ขอรับชื่อของไฟล์สำหรับ descriptor ไฟล์ที่ระบุเฉพาะ ซึ่งส่งคืนค่าเหมือนกับ `__file->fname` (อ้างถึง `__file` built-in ของ I/O probe manager) สำหรับ ไฟล์เดียวกัน

หมายเหตุ: ฟังก์ชันนี้ต้องการค่า `num_pagefaults` ที่ปรับได้ ของคำสั่ง **probevctrl** ให้เป็นค่าที่มากกว่า 0 หากค่าเป็น 0 (หรือไม่เพียงพอ) ดังนั้นฟังก์ชันนี้จะคืนค่าสตริง null เป็นชื่อไฟล์

พารามิเตอร์

fd ค่าตัวให้คำอธิบายไฟล์หรือซ็อกเก็ต

fd_fstype

วัตถุประสงค์

ส่งคืนชนิดระบบไฟล์สำหรับ descriptor ไฟล์เฉพาะ

ไวยากรณ์

```
int fd_fstype(int fd);
```

คำอธิบาย

ฟังก์ชันนี้ขอรับชนิดของระบบไฟล์ที่ไฟล์ของ descriptor ไฟล์ที่ระบุ เป็นเจ้าของ โดยส่งคืน ค่าเหมือนกับ `__file->fs_type` (อ้างถึง __file built-in ของ I/O probe manager)

หมายเหตุ: ฟังก์ชันนี้ต้องการให้พารามิเตอร์ที่สามารถปรับค่าได้ `num_pagefaults` ของคำสั่ง `probevctrl` มีค่ามากกว่า 0 หากมีค่า 0 (หรือไม่เพียงพอ) ฟังก์ชันนี้ จะส่งคืนสตริง -1 เป็นชนิดของระบบไฟล์

พารามิเตอร์

fd ค่า descriptor ไฟล์

fd_ftype

วัตถุประสงค์

ส่งคืนชนิดไฟล์สำหรับ descriptor ไฟล์ที่ระบุเฉพาะ

ไวยากรณ์

```
int fd_ftype(int fd);
```

คำอธิบาย

ฟังก์ชันนี้ขอรับชนิดไฟล์สำหรับ descriptor ไฟล์ที่ระบุเฉพาะ โดยส่งคืน ค่าเหมือนกับ `__file->f_type` (อ้างถึง __file built-in ของ I/O probe manager)

หมายเหตุ: ฟังก์ชันนี้ต้องการ num_pagefaults ที่สามารถปรับค่าได้ของคำสั่ง **probevctrl** ให้มีค่ามากกว่า 0 หากมีค่า 0 (หรือไม่เพียงพอ) ฟังก์ชันนี้จะส่งคืน -1 เป็นชนิดไฟล์

พารามิเตอร์

fd ค่า descriptor ไฟล์

fd_inodeid

วัตถุประสงค์

ส่งคืน inode ID สำหรับ descriptor ไฟล์ที่ระบุเฉพาะ

ไวยากรณ์

```
unsigned long long fd_inodeid(int fd);
```

คำอธิบาย

ฟังก์ชันนี้ส่งคืน inode ID สำหรับไฟล์ที่เกี่ยวข้องกับ descriptor ไฟล์ ที่ระบุเฉพาะ inode ID เป็นค่าเฉพาะ `unsigned long long` ที่ใช้ในระบบทั่วไป (ซึ่งเป็นค่า ที่แตกต่างจากหมายเลข inode ของระบบไฟล์และสามารถเปลี่ยนค่าได้หากรีบูตระบบ) ค่านี้ตรงกับค่าที่ถูกส่งคืนโดยฟังก์ชัน `fpath_inodeid()` สำหรับไฟล์เดียวกัน

หมายเหตุ: ฟังก์ชันนี้ต้องการให้ num_pagefaults ที่สามารถปรับค่าได้ของคำสั่ง **probevctrl** มีค่ามากกว่า 0 หากมีค่า 0 (หรือไม่เพียงพอ) ฟังก์ชันนี้จะส่งคืน 0 เป็น inode ID

พารามิเตอร์

fd ค่า descriptor ไฟล์

fd_mpath

วัตถุประสงค์

รับพาธเมาท์ของระบบไฟล์สำหรับ descriptor ไฟล์ที่ระบุเฉพาะ

ไวยากรณ์

```
char * fd_mpath(int fd);
```

คำอธิบาย

ฟังก์ชันนี้ขอรับพาธเมาท์ของระบบไฟล์ที่ descriptor ไฟล์ที่ระบุเฉพาะ เป็นเจ้าของ ซึ่งส่งคืน ค่าเหมือนกับ `__file->mount_path` (อ้างถึง __file built-in สำหรับ I/O probe manager) สำหรับไฟล์เดียวกัน

หมายเหตุ: ฟังก์ชันนี้ต้องการให้ num_pagefaults ที่สามารถปรับค่าได้ของคำสั่ง `probevctrl` มีค่ามากกว่า 0 หากมีค่า 0 (หรือไม่เพียงพอ) ฟังก์ชันนี้จะส่งคืนสตริง null เป็นพาธเมาท์

พารามิเตอร์

fd ค่า descriptor ไฟล์

fd_path

วัตถุประสงค์

ส่งคืนพาธสัมบูรณ์ของไฟล์สำหรับ descriptor ไฟล์ที่ระบุเฉพาะ

ไวยากรณ์

```
path_t fd_path(int fd);
```

คำอธิบาย

ฟังก์ชันนี้ส่งคืนพาธสัมบูรณ์ของ descriptor ไฟล์ที่ระบุเฉพาะ ค่าส่งคืนเป็นชนิด `path_t` ซึ่งส่งคืนค่าที่เหมือนกับ __file->path (อ้างถึง __file built-in ของ I/O probe manager) สำหรับไฟล์เดียวกัน

หมายเหตุ: ฟังก์ชันนี้ต้องการให้ num_pagefaults ที่สามารถปรับค่าได้ของคำสั่ง `probevctrl` มีค่ามากกว่า 0 หากมีค่า 0 (หรือไม่เพียงพอ) ฟังก์ชันนี้จะส่งคืนพาธ null ที่พิมพ์ด้วยฟังก์ชัน `printf("%p")` ซึ่งพิมพ์สตริง null

พารามิเตอร์

fd ค่า descriptor ไฟล์

fpath_inodeid

วัตถุประสงค์

ส่งคืน inode ID สำหรับพาธไฟล์ที่ระบุเฉพาะ

ไวยากรณ์

```
unsigned long long fpath_inodeid(String file_path);
```

คำอธิบาย

ฟังก์ชันนี้ส่งคืน inode ID สำหรับพาธไฟล์ที่ระบุเฉพาะ inode ID เป็นค่าเฉพาะ unsigned long long ที่ใช้ในระบบทั่วไป (ซึ่งแตกต่างจากหมายเลข inode ของระบบไฟล์ และสามารถเปลี่ยนค่าได้หากรีสตาร์ทระบบ) ถ้าพาธไฟล์ไม่มีอยู่ สคริปต์ Vue จะถูกปฏิเสธโดยคำสั่ง **probevue** ค่า inode ID ยังคงเหมือนกับค่าที่จัดเตรียมโดย __file->inode_id ในเหตุการณ์โพรบ vfs สำหรับไฟล์เดียวกัน (อ้างถึง __file built-in ของ I/O probe manager)

หมายเหตุ: ฟังก์ชันนี้สามารถใช้ได้ทุกที่ในสคริปต์ Vue (เมื่ออนุญาตให้ใช้ฟังก์ชัน Vue)

พารามิเตอร์

file_path

สตริงอักขระเครื่องหมายอัญประกาศคู่ที่แทนค่าไฟล์ที่มีอยู่ เช่น "/usr/lib/boot/unix_64" ไม่สามารถเป็นตัวแปรได้

get_function

วัตถุประสงค์

ส่งคืนชื่อของฟังก์ชันที่ครอบคลุมด้วยโพรบปัจจุบัน เมื่อฟังก์ชัน **get_function** ถูกเรียกใช้จากประโยค interval, systrace, BEGIN และ END ฟังก์ชันจะคืนค่าสตริงว่างๆ

ไวยากรณ์

```
String get_function ( );
```

คำอธิบาย

ฟังก์ชัน **get_function** ส่งคืนชื่อของฟังก์ชันโพรบ ซึ่งเป็นฟังก์ชันที่ล้อมรอบจุดโพรบปัจจุบัน โดยปกติแล้ว ชื่อของฟังก์ชันที่โพรธแล้ว คือจุดโพรบของฟิลด์ tuple ที่มาก่อนฟิลด์ตำแหน่ง

ส่วน get_probe ที่มาก่อน มีสคริปต์ตัวอย่างที่ใช้ฟังก์ชัน **get_function**

ฟังก์ชัน **get_function** ส่งคืนสตริงเปล่า เมื่อเรียกจากช่วงของตัวจัดการโพรบ

พารามิเตอร์

ฟังก์ชัน **get_function** ไม่ได้ใช้พารามิเตอร์ใดๆ

get_kstring

วัตถุประสงค์

Copies data from kernel memory into a String variable.

ไวยากรณ์

```
String get_kstring( char *kaddr,int len);
```

คำอธิบาย

ฟังก์ชัน **get_kstring** จะอ่านข้อมูลในหน่วยความจำเคอร์เนลไปยังตัวแปรชนิดสตริง

สตริงในพื้นที่เคอร์เนลต้องถูกคัดลอกก่อน ที่จะสามารถใช้ได้ในนิพจน์หรือส่งผ่านเป็นพารามิเตอร์ไปยังฟังก์ชัน Vue

เป้าหมายของฟังก์ชัน **get_kstring** ต้องเป็นตัวแปรชนิดสตริง ถ้าระบุค่า *–1* ไว้สำหรับพารามิเตอร์ `len` ข้อมูลจะถูกคัดลอกจากหน่วยความจำเคอร์เนล จนกระทั่งไบต์ NULL ถูกอ่าน (ไบต์ NULL ถูกอ่าน เพื่อหยุดสตริงข้อความในภาษา C) ถ้าความยาวของสตริง มากกว่าขนาดที่ประกาศไว้ของตัวแปรสตริงเป้าหมาย เฉพาะอักขระสตริงมากสุดเท่ากับขนาดของตัวแปรที่ถูกคัดลอก ไว้ อย่างไรก็ตาม สตริงในรูปแบบนี้เป็นแบบนี้ทั้งหมดจนกว่าจะอ่านไบต์ NULL ซึ่งถูกคัดลอกไปยังพื้นที่บัฟเฟอร์สตริงชั่วคราวเมื่อเริ่มต้น ผู้ใช้ฟังก์ชันต้องระมัดระวังว่า แอดเดรสเคอร์เนลชี้ไปยัง สตริงที่ยกเลิก NULL เพื่อหลีกเลี่ยงโอเวอร์โฟลว์ของพื้นที่บัฟเฟอร์สตริงชั่วคราว ซึ่งเป็นสาเหตุทำให้เซสชัน ProbeVue หยุดทำงาน

ความยาวสูงสุดของสตริงที่ต้องอ่านจาก หน่วยความจำเคอร์เนลสามารถกำหนดค่าคงที่ได้โดยระบุค่าที่ไม่ใช่ค่าติดลบ สำหรับพารามิเตอร์ `len` ในกรณีนี้ การคัดลอกดำเนินการ ได้จนกว่าจะอ่านไบต์ NULL หรือจำนวนที่ระบุไว้ของไบต์ ถูกอ่าน คุณลักษณะนี้อนุญาตให้ใช้สตริงแบบ long ในหน่วยความจำเคอร์เนล ซึ่งต้องถูกคัดลอกเพิ่มเติม เนื่องจากการคัดลอกมีข้อจำกัดโดยค่าของพารามิเตอรร์ `len` และไม่ได้เป็นสาเหตุทำให้บัฟเฟอร์สตริงหน่วยความจำภายในของ ProbeVue เกิดโอเวอร์โฟลว์

ถ้าเกิดข้อยกเว้นเมื่อฟังก์ชันนี้กำลังรัน ตัวอย่างเช่น เมื่อแอดเดรสเคอร์เนลที่อยู่ในรูปแบบที่ไม่ถูกต้องถูกส่งผ่านไปยังฟังก์ชัน เซสชัน ProbeVue จะถูกยกเลิกด้วยข้อความแสดงข้อผิดพลาด

พารามิเตอร์

addr ระบุแอดเดรสของข้อมูลพื้นที่เคอร์เนล

len ระบุจำนวนไบต์ของข้อมูลเคอร์เนลที่ถูกคัดลอก ค่า *–1* บ่งชี้ว่า ข้อมูลเคอร์เนล ต้องถูกใช้เป็นสตริง ″C″ และการคัดลอกเพื่อดำเนินการจนกว่าจะอ่านไบต์ null (อักขระ '\0') โปรดระมัดระวังเมื่อคุณระบุ -1 เป็นค่าของพารามิเตอร์ `len`

get_location_point

วัตถุประสงค์

ส่งคืนจุดตำแหน่งโพรบปัจจุบัน

ไวยากรณ์

```
int get_location_point ( );
```

คำอธิบาย

ฟังก์ชัน **get_location_point** ส่งคืนจุดของตำแหน่งโพรบปัจจุบันที่เป็นออฟเซ็ตจากฟังก์ชันการล้อมรอบ entry point โดยเฉพาะอย่างยิ่ง จะส่งคืน **FUNCTION_ENTRY** หรือค่าศูนย์หากจุดโพรบอยู่ที่ entry point ของฟังก์ชัน และส่งคืน **FUNCTION_EXIT** หรือ –1 หากจุดโพรบอยู่ที่ exit point ใด หรือส่งคืนออฟเซ็ตของแอดเดรสจริง

สคริปต์ต่อไปนี้จะแสดงตัวอย่างของการใช้ฟังก์ชัน **get_location_point** :

```
@@syscall:$1:read:entry, @@syscall:$1:read:exit
 {
  probev_timestamp_t ts1, ts2;
  int diff;
```

```
  if (get_location_point() == FUNCTION_ENTRY)
   ts1 = timestamp();
  else if (get_location_point() == FUNCTION_EXIT) {
   ts2 = timestamp();
   diff = diff_time(ts1, ts2, MICROSECONDS);
   printf("Time for read system call = %d\n", diff);
  }
 }
```

ฟังก์ชันนี้ไม่ได้รับการสนับสนุน เมื่อเรียกจากช่วงของตัวจัดการโพรบ

พารามิเตอร์

ฟังก์ชัน **get_location_point** ไม่ได้ใช้พารามิเตอร์ใดๆ

get_probe

วัตถุประสงค์

ส่งคืนข้อกำหนดคุณลักษณะของจุดโพรบปัจจุบัน

ไวยากรณ์

```
String get_probe ( );
```

คำอธิบาย

ฟังก์ชัน **get_probe** จะส่งคืนการแทนค่าภายในของข้อกำหนดคุณลักษณะจุดโพรธ เมื่อบันทึกไว้ภายในแล้ว จุดโพรบจะไม่สอดแทรกคำนำหน้า **@@** เริ่มต้น หรือ ID กระบวนการ หากมี

สคริปต์ต่อไปนี้แสดงตัวอย่างของการใช้ฟังก์ชัน **get_probe** :

```
#cat get_probe.e
 @@uft:312678:*:run_prog:entry
 {
  printf("function ′%s′ probe ′%s′\n", get_function(), get_probe());
 }

 #probevue get_probe.e
 function ′run_prog′ probe ′uft:*:*:run_prog:entry′
```

พารามิเตอร์

ฟังก์ชัน **get_probe** ไม่ได้ใช้พารามิเตอร์ใดๆ

get_stktrace

วัตถุประสงค์

คืนค่าการติดตามสแต็กปัจจุบัน

ไวยากรณ์

```
stktrace_t get_stktrace(int level);
```

คำอธิบาย

พารามิเตอร์

วัตถุประสงค์

ไวยากรณ์

คำอธิบาย

ฟังก์ชัน **get_stktrace** ส่งคืนการติดตามแบบสแต็กของเธรดปัจจุบัน การติดตามสแต็กนี้ถูกเก็บไว้ในตัวแปรชนิด **stktrace_t** หรือถูกพิมพ์ด้วยตัวระบุ **%t** ในฟังก์ชัน **printf** ของ ProbeVue built-in พารามิเตอร์ level จะระบุจำนวนของระดับจนถึงระดับที่การติดตามสแต็กจะถูกพิมพ์ ลักษณะการทำงาน ของฟังก์ชัน **get_stktrace** ที่ใช้ภายในฟังก์ชัน **printf** จะคล้ายกับฟังก์ชัน built-in **stktrace** เฉพาะความแตกต่าง ที่เป็นสัญลักษณ์ดีฟอลต์พร้อมกับแอดเดรสเท่านั้นที่ถูกพิมพ์หากเธรดอยู่ในสถานะกำลังรัน มิฉะนั้นแอดเดรสจะถูกพิมพ์ และยังพิมพ์สแต็ก CPU ทั้งหมดโดยข้ามสถานะ เครื่องทั้งหมด

สคริปต์ต่อไปนี้จะแสดงตัวอย่างของการใช้ฟังก์ชัน **get_stktrace** :

```
t1 = get_stktrace(3)              // คืนค่าการติดตามสแต็กปัจจุบัน & ที่เก็บใน stktrace_t
                                  // type variable t1.
printf("%t\n", get_stktrace(4));  // พิมพ์การติดตามสแต็กปัจจุบันจนถึงระดับที่ 4
```

พารามิเตอร์

พารามิเตอร์	คำอธิบาย
level	ระบุจำนวนของระดับจนถึงระดับที่การติดตามสแต็กจะถูกบันทึกในตัวแปรชนิด **stktrace_t** ค่า -1 บ่งชี้ว่า กลุ่มของเร็กคอร์ดที่เชื่อมโยงกันแบบสแต็กย้อนกลับ จะเดินข้ามส่วนขยายที่เป็นไปได้ ค่าดีฟอลต์ที่เป็นค่า 0 จะติดตามย้อนกลับไปจนถึงระดับที่ 2 และบันทึก 2 รายการ ค่าใดๆ ที่เป็นค่าจำนวนเต็มบวกจะบอกถึงจำนวนของระดับที่จะบันทึกในตัวแปร ค่าสูงสุดของระดับสามารถเป็น 240 **หมายเหตุ:** ถ้ารายการต่างๆ จาก mst จำนวนมากถูกพิมพ์ ขอบเขต mst จะถูกคั่นด้วยบรรทัดที่มีอักขระ '-' บรรทัดนี้ยังถูกพิจารณาให้เป็นระดับ 1 นั่นหมายความว่า รายการต่างๆ ที่พิมพ์เป็นพารามิเตอร์ระดับที่จัดเตรียมจำนวนบรรทัดตัวคั่นที่ติดค่าลบ (ยกเว้นพารามิเตอร์ที่มีค่า -1)

get_userstring

วัตถุประสงค์

คัดลอกข้อมูลจากหน่วยความจำผู้ใช้

ไวยากรณ์

```
String get_userstring( char * addr, int len );
```

คำอธิบาย

ฟังก์ชัน **get_userstring** อ่านข้อมูลในหน่วยความจำผู้ใช้ภายในตัวแปรชนิด **String**

ข้อมูลในพื้นที่ของผู้ใช้ต้องถูกคัดลอกก่อนที่คุณจะสามารถใช้ข้อมูลนั้นในนิพจน์ หรือส่งข้อมูลในฐานะเป็นพารามิเตอร์ไปยังฟังก์ชัน Vue เป้าหมายของฟังก์ชัน **get_userstring** คือตัวแปรชนิด **String** ถ้าระบุค่า -1 สำหรับพารามิเตอร์ *len* ไว้ ข้อมูลจะถูกคัดลอกจากหน่วยความจำผู้ใช้ จนกว่า NULL ไบต์จะถูกอ่าน (NULL ไบต์จะถูกใช้เพื่อยกเลิกสตริงข้อความในภาษา C) ถ้าสตริงมีความยาวมากกว่าขนาดของตัวแปร **String** เป้าหมายที่ประกาศไว้ เฉพาะอักขระสตริงใดๆ ที่มีขนาดสูงสุด เท่ากับขนาดของตัวแปรจะถูกคัดลอกไปยังเป้าหมายนั้น อย่างไรก็ตาม สตริงภายในทั้งหมดที่ยังคงอ่าน NULL ไบต์อาจต้องการคัดลอกไปยังพื้นที่บัฟเฟอร์สตริงชั่วคราวในตอนแรก ผู้ใช้ฟังก์ชันต้องระมัดระวังว่า แอดเดรสผู้ใช้ชี้ไปยังสตริง NULL ที่ยกเลิกแล้ว เพื่อหลีกเลี่ยงการโอเวอร์โฟลว์ของพื้นที่บัฟเฟอร์สตริงชั่วคราว ซึ่งอาจเป็นสาเหตุทำให้เซสชัน ProbeVue ถูกยกเลิกได้

ความยาวจริงของสตริงที่อ่านจากหน่วยความจำผู้ใช้สามารถแก้ไขได้โดยระบุค่าพารามิเตอร์ *len* ในกรณีนี้ การคัดลอกยังคงดำเนินการจนกว่าไบต์ NULL จะถูกอ่านหรือ จำนวนของไบต์ที่ระบุไว้จะถูกอ่าน คุณลักษณะนี้อนุญาตให้ข้อมูลชนิดที่ไม่ใช่สตริง ถูกคัดลอกไปยังตัวแปร **String** ซึ่งสามารถพิมพ์ได้ในภายหลัง โดยใช้ฟังก์ชันการติดตาม

หมายเหตุ: Vue ไม่ได้รักษา NULL ไบต์เป็นเทอร์มิเนเตอร์สตริง ดังนั้น สตริงจริงต้องไม่ถูกคัดลอกไว้ในกลไกนี้

ฟังก์ชันนี้จะอนุญาตให้ใช้ในโพรบของพื้นที่ของผู้ใช้เท่านั้น (เช่นเดียวกับโพรบชนิด **uft**) หรือโพรบที่จัดเตรียมโดยตัวจัดการโพรบ **syscall** ถ้าเพจความผิดพลาดเกิดขึ้นขณะคัดลอกข้อมูล การดำเนินการคัดลอกจะถูกยกเลิก และตัวแปร String จะมีข้อมูลที่คัดลอกไว้เป็นผลสำเร็จเท่านั้น ถ้าเกิด exception ขึ้นขณะออกคำสั่งฟังก์ชันนี้ ตามตัวอย่าง ขณะที่แอดเดรสของผู้ใช้ที่เสียหายถูกส่งผ่านไปยังฟังก์ชัน เซสชัน ProbeVue จะถูกยกเลิกพร้อมกับข้อความแสดงความผิดพลาด

“โปรแกรมอำนวยความสะดวกการติดตาม ProbeVue แบบไดนามิก” ในหน้า 219 มีสคริปต์ตัวอย่างที่ใช้ฟังก์ชัน **get_userstring**

หมายเหตุ: คุณสามารถแก้ไขชนิดข้อมูลของเป้าหมายของการดำเนินการคัดลอกโดยใช้การคำนวณ แม้ว่าการดำเนินการนี้จะสร้างข้อความเตือนจากคอมไพเลอร์ก็ตาม ดังนั้น คุณสามารถใช้ฟังก์ชัน **get_userstring** เพื่อคัดลอกสตริงและข้อมูลที่เตรียมไว้เป็นโครงสร้าง และรูปแบบข้อมูลอื่นๆ จากพื้นที่ของผู้ใช้ภายในพื้นที่ ProbeVue สคริปต์ต่อไปนี้ คือตัวอย่างวิธีการดำเนินการข้อมูลชนิดนี้:

```
/* File: string2int.e
 *
 * Reads an integer passed in a pointer using get_userstring()
 *
 */
int get_file_sizep(int *fd);

@@BEGIN
{

 int i;
}

@@uft:$1:*:get_file_sizep:entry
{
 i = *(int *)(get_userstring(__arg1, 4));

 printf("fd = %d\n", i);
}
```

หมายเหตุ: เป้าหมายของการดำเนินการคัดลอกต้องเป็นตัวแปร String ซึ่งมีความยาวที่ประกาศไว้เพียงพอที่ยอมรับข้อมูลที่คัดลอก หรือเซสชัน ProbeVue สามารถยกเลิกได้ **get_userstring** จะยอมรับค่าใดๆ สำหรับขนาดของข้อมูลที่ต้องการคัดลอก แต่ความยาวสูงสุดที่สามารถคัดลอกได้จะถูกจำกัดโดยข้อจำกัดด้านหน่วยความจำของเซสชัน ProbeVue

พารามิเตอร์

พารามิเตอร์	คำอธิบาย
addr	ระบุที่อยู่ของข้อมูลพื้นที่ของผู้ใช้
len	ระบุจำนวนไบต์ของข้อมูลผู้ใช้ที่ต้องการคัดลอก ค่า -1 บ่งชี้ว่า ข้อมูลผู้ใช้ที่ต้องการใช้เป็นสตริง "C" และดำเนินการคัดลอกต่อจนกว่า null ไบต์ (อักขระ '\0') จะถูกอ่าน โปรดระวัง เมื่อระบุ -1 เป็นค่าพารามิเตอร์ *len*

list

วัตถุประสงค์

สร้างและส่งคืนรายการเปล่า

ไวยากรณ์

```
List list ( );
```

คำอธิบาย

ฟังก์ชัน **list** คือฟังก์ชัน constructor สำหรับชนิดข้อมูลลิสต์ ซึ่งส่งคืนลิสต์เปล่า และประกาศเป้าหมายที่ต้องการให้เป็นชนิดข้อมูลลิสต์แบบอัตโนมัติ ไม่มีวิธีที่แน่นอน ในการประกาศตัวแปรที่ต้องการให้เป็นชนิดข้อมูลลิสต์ ตัวแปร list จะถูกสร้างเป็นตัวแปรของคลาสแบบโกลบอล

ฟังก์ชัน **list** สามารถเรียกใช้งานจาก clause ใดๆ ได้ ถ้าคุณระบุชื่อลิสต์ที่มีอยู่ขณะเรียกใช้งานฟังก์ชัน **list** ลิสต์ที่มีอยู่นั้นจะถูกลบทิ้ง

ตัวแปร list สามารถนำมาใช้เพื่อเก็บรวบรวมค่าชนิดเลขจำนวนเต็ม ค่าใดๆ ที่เก็บอยู่ในรายการจะถูกพัฒนาไปเป็นชนิดข้อมูล **long long** (หรือเลขจำนวนเต็มขนาด 64 บิต)

สคริปต์ต่อไปนี้แสดงตัวอย่างการใช้ฟังก์ชัน **list** ซึ่งยอมรับว่าโปรแกรมเชลล์ sprobevue ซึ่งครอบคลุมด้วยเครื่องหมายอัญประกาศคู่ล้อมรอบอาร์กิวเมนต์แต่ละตัว เรียกใช้งานสคริปต์ Vue

```
/* File: list.e
 *
 * Collect execution time for read system call statistics
 *
 * Usage: sprobevue list.e <-s|-d>
 *
 * Pass -s for summary and -d for detailed information
 */

int read(int fd, void *buf, int n);

@@BEGIN
{
 String s[10];
 int detail;
```

```
 times = list();  /* declare and create a list */

 /* Check for parameters */
 s = $1;
 if (s == "-d")
  detail = 1;
 else if (s == "-s")
  detail = 0;
 else {
  printf("Usage: sprobevue list.e <-s|-d>\n");
  exit();
 }
}

@@syscall:*:read:entry
{
 /*
  * Save entry time in a thread-local to ensure that
  * in the read:exit probe point we can access our thread's value for
  * entry timestamp. If we use a global, then the variable can be
  * overlaid by the next thread to enter read and this can give
  * wrong values when we try to find the difference at read:exit
  * time since we use this later value instead of the original value.
  */

 __thread probev_timestamp_t t1;
 t1 = timestamp();
}

@@syscall:*:read:exit
 when (thread:t1 != 0)
{
 __auto t2;
 __auto long difft;

 /* Get exit time */
 t2 = timestamp();
 difft = diff_time(t1, t2, MICROSECONDS);

 /* Append read time to list */
 append(times, difft);

 /* print detail data if "-d" was passed to script */
 if (detail)
  printf("%s (%ld) read time = %d micros\n", __pname, __pid, difft);
}

@@interval:*:clock:10000
{
 /* Print statistics every 10 seconds */
 printf("Read calls so far = %d, total time = %d, max time = %d, " +
        "min = %d, avg = %d\n",
   count(times),
   sum(times),
```

```
    max(times),
    min(times),
    avg(times));
}
```

พารามิเตอร์

ฟังก์ชัน list ไม่ได้ใช้พารามิเตอร์ใดๆ

lquantize

วัตถุประสงค์

พิมพ์คีย์และค่าที่เชื่อมโยงของ associative array ในรูปแบบกราฟิก โดยการกำหนดค่าในสเกลแบบ logarithmic

ไวยากรณ์:

```
void lquantize( aso-name ,int num-of-entries, int flags, sort_key_ind)
```

คำอธิบาย:

ฟังก์ชันนี้แสดงรายการของ associative array ในรูปแบบกราฟิก โดยอิงตามค่า logarithmic ของค่าของ associative array หากคุณต้องการพิมพ์เฉพาะ อิลิเมนต์ที่มีชุดของคีย์เฉพาะเจาะจง คีย์สามารถระบุพร้อมกับชื่อตัวแปรของ associative array ในอาร์กิวเมนต์แรก เมื่อต้องการจำกัดเฉพาะคีย์มิติบางอย่าง และอนุญาตใช้ใช้ค่าใดๆ สำหรับคีย์อื่นใน คุณสามารถใช้คีย์เวิร์ด ANY สำหรับตัวอย่าง โปรดดูที่ส่วนของ ฟังก์ชัน print()

พารามิเตอร์แรกเป็นพารามิเตอร์ที่จำเป็น และพารามิเตอร์อื่นทั้งหมดเป็นทางเลือก หากคุณ ไม่ได้ระบุพารามิเตอร์ที่เป็นทางเลือก ดังนั้น จะใช้อ็อพชันการพิมพ์ดีฟอลต์

พารามิเตอร์:

aso-name
ชื่อของตัวแปร associative array ที่คุณต้องการพิมพ์ คุณยังสามารถระบุคีย์สำหรับ มิติทั้งหมดในวงเล็บ คุณสามารถใช้คีย์เวิร์ด ANY เพื่อจับคู่คีย์ทั้งหมด ในมิติคีย์

num-of-entries
ระบุจำนวนรายการที่ต้องการพิมพ์ พารามิเตอร์นี้เป็นทางเลือก ระบุ 0 เพื่อแสดง รายการทั้งหมด หากไม่ได้ระบุค่า จะใช้อ็อพชันการพิมพ์ดีฟอลต์สำหรับเซสชัน ค่าลบ เทียบเท่ากับ 0

flags
ระบุแฟล็ก sort-type, sort-by และ *list-value* พารามิเตอร์นี้เป็นทางเลือก แฟล็ก sort-type, sort-by และ list-value ถูกอธิบายไว้ภายใต้ส่วน 'Associative Array Type' หากคุณระบุ 0 จะใช้อ็อพชันการพิมพ์ดีฟอลต์สำหรับเซสชัน

sort_key_ind
ดัชนีของคีย์ (มิติคีย์) ที่เอาต์พุตถูกเก็บ หากคุณระบุ -1 คีย์แรกจะถูกใช้สำหรับการเรียงลำดับ หากคีย์แรกเป็นชนิดที่ไม่สามารถเรียงลำดับได้ เอาต์พุต จะไม่ถูกเรียงลำดับ

max

วัตถุประสงค์:

ส่งคืนค่าสูงสุดของอิลิเมนต์ทั้งหมดในรายการ

ไวยากรณ์:

```
long long max ( List listvar );
```

คำอธิบาย:

ฟังก์ชัน **max** ส่งคืนค่าสูงสุดขององค์ประกอบทั้งหมดที่ได้ผนวกกับตัวแปรลิสต์ที่ระบุด้วยพารามิเตอร์ *listvar*

สำหรับข้อมูลเพิ่มเติมเกี่ยวกับชนิดข้อมูลรายการ โปรดดูที่หัวข้อ “โปรแกรมอำนวยความสะดวกการติดตาม ProbeVue แบบไดนามิก” ในหน้า 219 ส่วน *listvar* ที่มาก่อน มีสคริปต์ตัวอย่างที่ใช้ฟังก์ชัน **max**

พารามิเตอร์

พารามิเตอร์	คำอธิบาย
listvar	ระบุตัวแปรของชนิด **รายการ**

min

วัตถุประสงค์

ส่งคืนค่าต่ำสุดขององค์ประกอบทั้งหมดในรายการ

ไวยากรณ์

```
long long min ( List listvar );
```

คำอธิบาย

ฟังก์ชัน **min** ส่งคืนค่าต่ำสุดขององค์ประกอบทั้งหมดที่ได้ผนวกเข้ากับตัวแปร list ที่ระบุด้วยพารามิเตอร์ *listvar*

ส่วน *listvar* ที่มาก่อนมีสคริปต์ตัวอย่างที่ ใช้ฟังก์ชัน **min**

พารามิเตอร์

listvar: ระบุตัวแปรชนิด **list**

print_args

วัตถุประสงค์

พิมพ์ฟังก์ชันปัจจุบันและค่าอาร์กิวเมนต์ของฟังก์ชัน

ไวยากรณ์

```
void print_args();
```

คำอธิบาย

ฟังก์ชัน **print_args** จะพิมพ์ชื่อฟังก์ชันตามด้วยอาร์กิวเมนต์ของฟังก์ชันที่คั่นด้วยเครื่องหมายคอมม่าที่อยู่ในวงเล็บ ค่าอาร์กิวเมนต์จะถูกพิมพ์ตาข้อมูลชนิดของอาร์กิวเมนต์ที่มีในตาราง traceback ของฟังก์ชัน รูทีนนี้สามารถใช้ได้ในการป้อนโพรบ uft/uftxlc++ และโพรบ syscall/syscallx ซึ่งจะมีประโยชน์ในโพรบที่ตำแหน่งของโพรบถูกระบุเป็นนิพจน์ธรรมดา

พารามิเตอร์

ฟังก์ชัน **print_args** จะไม่ใช้พารามิเตอร์ใดๆ

หมายเหตุ: รูทีน **print_args** จะไม่สร้างเอาต์พุตใดๆ ถ้าตาราง traceback ของรูทีนถูกเพจออก และไม่มีบริบท page fault ที่ว่างให้ใช้งาน จำนวนของ page faults ที่สามารถจัดการได้สามารถเพิ่มได้โดยใช้คำสั่ง `probevctrl` และสามารถลองใช้สคริปต์ใหม่ได้

print

วัตถุประสงค์

พิมพ์คีย์และค่าที่เชื่อมโยงของ associative array

ไวยากรณ์

```
void print ( aso-name , int num-of-entires , int flags, int sort_key_ind );
```

คำอธิบาย

ฟังก์ชันนี้จะพิมพ์อิลิเมนต์ของ associative array ที่ระบุโดย ตัวแปร `aso-name` เมื่อต้องการพิมพ์เฉพาะอิลิเมนต์ที่มีชุดของคีย์เฉพาะเจาะจง คุณสามารถระบุคีย์ที่มีชื่อตัวแปรของ associative array ในอาร์กิวเมนต์แรก เมื่อต้องการจำกัด เฉพาะบางคีย์มิติและอนุญาตค่าใดๆ สำหรับคีย์อื่นทั้งหมด ให้ใช้คีย์เวิร์ด `ANY`

ตัวอย่าง:

```
print(aso_var[0][ANY][ANY]);    // print all elements having first key as 0 (other keys can be anything)
print(aso_var[ANY][ANY][ANY]);  // print all; equivalent to print(aso_var);
```

พารามิเตอร์แรกเป็นพารามิเตอร์ที่จำเป็น และพารามิเตอร์อื่นทั้งหมดเป็นทางเลือก หากคุณ ไม่ได้ระบุพารามิเตอร์ที่เป็นทางเลือก จะใช้อ็อพชันการพิมพ์ดีฟอลต์

หมายเหตุ: การติดตามแบบชั่วคราวไม่อนุญาตให้ใช้กับฟังก์ชันการพิมพ์

สำหรับ associative ที่มีคีย์หลายมิติ คีย์จะถูกพิมพ์เป็นรายการที่คั่นด้วย 'I' และค่าจะถูกพิมพ์บนบรรทัดเดียวกัน หากคีย์สร้างเอาต์พุตหลายบรรทัด คีย์จะถูกพิมพ์ในบรรทัดที่แยกจากกันและค่าจะถูกพิมพ์ในบรรทัดใหม่ ตัวอย่างต่อไปนี้แสดง สคริปต์ที่มี associative array ที่มีชนิดของ int 3 มิติและค่าเป็นชนิด เลขจำนวนเต็ม:

```
aso1[0]["a"][2.5] = 100;
aso1[1]["b"][3.5] = 101;
print(aso1);
The output from previous print() function follows:
```

```
| [key1              |    key2              |    key3]             |    value
| 0                  |    a                 |    2.5000000         |    100
| 1                  |    b                 |    3.5000000         |    101
```

| ตัวอย่างต่อไปนี้ใช้ associative array ที่มีคีย์สองมิติชนิด int และ stktrace_t มีค่าเป็นชนิด สตริง:

```
| aso2[0][get_stktrace(-1)] = "abc";
| print(aso2);
|
| เอาต์พุตจากฟังก์ชัน print() ด้านบนจะคล้ายดังต่อไปนี้:
| [key1              |    key2]                                  |    value
| 0
|                    |
|                         0x100001b8
|                         0x10003328
|                         0x1000166c
|                         0x10000c30
|                         .read+0x288
|                         sc_entry_etrc_point+0x4
|                         .kread+0x0
|                                                                |
|                                                                   abc
```

พารามิเตอร์

| *aso-name*
| ชื่อของตัวแปร associative array ที่คุณต้องการพิมพ์ คุณยังสามารถระบุคีย์สำหรับ มิติทั้งหมดในวงเล็บ คุณสามารถใช้คีย์
| เวิร์ด ANY เพื่อจับคู่คีย์ทั้งหมด ในมิติคีย์

| *num-of-entires*
| ระบุจำนวนรายการที่ต้องการพิมพ์ พารามิเตอร์นี้เป็นทางเลือก ระบุ 0 เพื่อแสดง รายการทั้งหมด หากไม่ได้ระบุค่า จะ
| ใช้อ็อพชันการพิมพ์ดีฟอลต์สำหรับเซสชัน ค่าลบ เทียบเท่ากับ 0

| *flags*
| ระบุแฟล็ก sort-type, sort-by และ *list-value* พารามิเตอร์นี้เป็นทางเลือก แฟล็ก sort-type, sort-by และ list-value ถูก
| อธิบายไว้ภายใต้ส่วน 'Associative Array Type' หากคุณระบุ 0 จะใช้อ็อพชันการพิมพ์ดีฟอลต์สำหรับเซสชัน

| *sort_key_ind*
| ดัชนีคีย์ (มิติคีย์) ถูกใช้สำหรับการการเรียงลำดับเอาต์พุต หากคุณระบุ -1 คีย์แรกจะถูกใช้สำหรับการเรียงลำดับ หากคีย์
| แรกเป็นชนิดที่ไม่สามารถเรียงลำดับได้ เอาต์พุต จะไม่ถูกเรียงลำดับ

printf

วัตถุประสงค์

จัดรูปแบบและทำสำเนาข้อมูลลงในบัฟเฟอร์ trace

ไวยากรณ์

```
void printf ( String format[ , data, ... ]);
```

คำอธิบาย

ฟังก์ชัน **printf** จะแปลง จัดรูปแบบ และคัดลอกค่าพารามิเตอร์ *data* ไปยังบัฟเฟอร์ trace ภายใต้การควบคุมพารามิเตอร์ format ตามที่ได้บ่งชี้ด้วยไวยากรณ์ รายการที่เปลี่ยนแปลงของอาร์กิวเมนต์สามารถส่งผ่านเป็นพารามิเตอร์ *data* ไปยัง **printf** Vue สนับสนุนตัวระบุการแปลงทั้งหมด ที่สนับสนุนด้วยรูทีนย่อย **printf** ที่จัดเตรียมไว้โดยไลบรารี C ยกเว้นตัวระบุการแปลง *%p*

นอกเหนือจากตัวระบุ `printf()` ของไลบรารี C แล้ว ภาษา Vue สนับสนุนตัวระบุสองตัวเพิ่มเติม: *%A* และ *%W*

%A – พิมพ์ `probev_timestamp_t` 't' ในรูปแบบวันที่ ดีฟอลต์ รูปแบบนี้สามารถเปลี่ยนได้โดยใช้ฟังก์ชัน `set_date_format()`

`%W` – พิมพ์ `probev_timestamp_t` 't' ในหน่วยวินาทีและมิลลิวินาที ซึ่งเกี่ยวข้องกับจุดเริ่มต้นของเซสชัน `probevue`

%p – พิมพ์สตริงที่สอดคล้องกับพาธไฟล์สัมบูรณ์ของค่า `path_t` ที่ระบุเฉพาะ

%M – พิมพ์ MAC address ของค่า mac_addr_t ที่ระบุไว้

%I – พิมพ์ IP address ในรูปแบบจุดทศนิยมสำหรับแอดเดรส ipv4 และรูปแบบจุดสำหรับเลขฐานหกสำหรับ IPV6 address ของค่า ip_addr_t ที่ระบุเฉพาะ

%H – พิมพ์ชื่อโฮสต์ในรูปแบบสตริงหรือจุดทศนิยมหรือเลขฐานหกของค่า ip_addr_t ที่ระบุเฉพาะ

หมายเหตุ: ถ้า IP address ถูกแก้ไขโดย Domain Name System (DNS) แล้ว ฟังก์ชัน `printf` จะแสดงชื่อโฮสต์ที่สอดคล้องกัน มิฉะนั้น จะแสดง IP address ในรูปแบบจุดทศนิยมหรือเลขฐานหก

ฟังก์ชัน **printf** ไม่สามารถนำมาใช้เพื่อพิมพ์ตัวแปรชนิด **list** อย่างไรก็ตาม ตัวแปรชนิด **string** สามารถพิมพ์ได้โดยใช้ตัวระบุการแปลง *%s* ตัวแปรชนิด `probev_timestamp_t` ถูกพิมพ์ในรูปแบบตัวเลข โดยใช้ตัวระบุ *%lld* หรือ *%16llx* `probev_timestamp_t` ถูกพิมพ์ในรูปแบบวันที่โดยใช้ตัวระบุ *%A* หรือ *%W*

สคริปต์ต่อไปนี้สาธิตตัวอย่างบางส่วนของการใช้ฟังก์ชัน **printf** :

```
@@BEGIN
{
 String s[128];
 int i;
 float f;
 f = 2.3;

 s = "Test: %d, float = %f\n";
 i = 44;

 printf(s, i, f);

 s = "Note:";
 printf("%s Value of i (left justified) = %-12d and right-justified = %12d\n",
    s, i, i);

 printf("With a long format string that may span multiple lines, you " +
   "can use the '+' operator to concatenate the strings " +
```

```
   "in multiple lines into a single string: 0x%08x\n", i);

 exit();
}
```

พารามิเตอร์

format
สตริงที่มีอักขระธรรมดาที่คัดลอกไปยังบัฟเฟอร์ trace โดยตรงโดยไม่ต้องเปลี่ยนแปลงใดๆ และตัวระบุการแปลงที่มากกว่าหนึ่งตัว ซึ่งจัดเตรียมการบ่งชี้ถึงวิธีการจัดรูปแบบพารามิเตอร์ *data* สำหรับข้อมูลเพิ่มเติมเกี่ยวกับตัวระบุการแปลง โปรดดูที่รูทีนย่อย AIX **printf** ใน *Technical Reference: Base Operating System and Extensions, Volume 1*

ข้อมูล
ระบุศูนย์หรืออาร์กิวเมนต์เพิ่มเติมที่สอดคล้องกับตัวระบุการแปลงในพารามิเตอร์ *format*

ptree

วัตถุประสงค์

เมื่อต้องการพิมพ์แผนผังกระบวนการสำหรับกระบวนการที่ถูกโพรบ

ไวยากรณ์

```
void ptree ( int depth );
```

คำอธิบาย

ฟังก์ชัน **ptree** จะพิมพ์แผนผังกระบวนการสำหรับฟังก์ชันที่ถูกโพรบ ฟังก์ชันนี้จะพิมพ์ทั้งลำดับชั้นซายน์และพาเรนต์ ความลึกที่ถูกผ่านเป็นพารามิเตอร์สามารถช่วยในการควบคุมความลึกของกระบวนการซายน์ที่ต้องการพิมพ์ ฟังก์ชันนี้ไม่สามารถใช้ได้ในโพรบ BEGIN หรือ END หรือ systrace นอกจากนี้ ฟังก์ชันนี้สามารถใช้ได้ในโพรบ interval เท่านั้น ถ้ามีการระบุ PID

หมายเหตุ: ฟังก์ชันนี้จะไม่ถูกเรียกทำงานทันทีในเคอร์เนล เมื่อฟังก์ชันถูกเรียกใช้จากโพรบ แต่จะถูกกำหนดเวลาให้รันภายหลังในพื้นที่ของเคอร์เนลแทน ดังนั้น ถ้าแผนผังกระบวนการมีการเปลี่ยนแปลงในระหว่างนั้น เอาต์พุตของฟังก์ชัน **ptree** อาจไม่ตรงกับโครงสร้างของแผนผังเมื่อฟังก์ชันถูกเรียกใช้จริงๆ

Sample output

ตัวอย่างเอาต์พุตของแผนผังกระบวนการเป็นดังนี้:

```
PID             CMD

1               init
                  |
                  V
 3342460        srcmstr
                  |
                  V
 3539052        inetd
                  |
                  V
 7667750        telnetd
```

```
                |
                V
6881336         ksh
                |
                V
5112038         probevue
                |
                V
7930038           tree    <=======
6553782              |\--tree
4849828              |\--tree
6422756                  |\--tree
3408074                      |\--tree
5963846                      |\--tree
7864392                    |\--tree
7799006                       |\--tree
```

พารามิเตอร์

พารามิเตอร์	คำอธิบาย
depth	ระบุความลึกสูงสุดที่ฟังก์ชัน **ptree** จะพิมพ์ข้อมูลชายน์สำหรับกระบวนการ ถ้าผ่านค่าที่เป็น –1 ฟังก์ชันจะพิมพ์ชายน์ทั้งหมดของกระบวนการ

quantize

วัตถุประสงค์:

พิมพ์คีย์และค่าที่เชื่อมโยงของ associative array ในรูปแบบกราฟิก โดยการกำหนดค่าในสเกลแบบลิเนียร์

ไวยากรณ์:

```
void quantize ( aso-name, int num-of-entries, int flags, int sort_key_ind)
```

คำอธิบาย:

ฟังก์ชันนี้แสดงรายการของอาร์เรย์ที่เชื่อมโยง ในรูปแบบกราฟิกโดยใช้ค่า linear ของ ค่าของอาร์เรย์ที่เชื่อมโยง เมื่อต้องการพิมพ์เฉพาะอิลิเมนต์ที่มีชุดของคีย์เฉพาะเจาะจง คีย์สามารถใช้กับชื่อตัวแปรของ associative array ในอาร์กิวเมนต์แรก เมื่อต้องการจำกัดเฉพาะบางคีย์มิติ และอนุญาตคีย์ทั้งหมดในมิติที่เหลือ คุณสามารถใช้คีย์เวิร์ด ANY

นอกเหนือจากพารามิเตอร์แรกแล้ว พารามิเตอร์อื่นเป็นพารามิเตอร์ที่เป็นทางเลือก หากคุณไม่ได้ระบุ พารามิเตอร์ที่เป็นทางเลือกเหล่านี้ ดังนั้นอ็อพชันการพิมพ์ดีฟอลต์จะถูกใช้

พารามิเตอร์:

aso-name

ชื่อของตัวแปร associative array ที่คุณต้องการพิมพ์ คุณยังสามารถระบุคีย์สำหรับ มิติทั้งหมดในวงเล็บ คุณสามารถใช้คีย์เวิร์ด ANY เพื่อจับคู่คีย์ทั้งหมด ในมิติคีย์

num-of-entries

ระบุจำนวนรายการที่ต้องการพิมพ์ พารามิเตอร์นี้เป็นทางเลือก ระบุ 0 เพื่อแสดง รายการทั้งหมด หากไม่ได้ระบุค่า จะใช้อ็อพชันการพิมพ์ดีฟอลต์สำหรับเซสชัน ค่าลบ เทียบเท่ากับ 0

flags
ระบุแฟล็ก sort-type, sort-by และ *list-value* พารามิเตอร์นี้เป็นทางเลือก แฟล็ก sort-type, sort-by และ list-value ถูกอธิบายไว้ภายใต้ส่วน 'Associative Array Type' หากคุณระบุ 0 จะใช้อ็อพชันการพิมพ์ดีฟอลต์สำหรับเซสชัน

sort_key_ind

ดัชนีของคีย์ (มิติคีย์) ที่เอาต์พุตถูกเก็บ หากคุณระบุ -1 คีย์แรกจะถูกใช้สำหรับการเรียงลำดับ หากคีย์แรกเป็นชนิดที่ไม่สามารถเรียงลำดับได้ เอาต์พุต จะไม่ถูกเรียงลำดับ

qrange

รูทีนนี้จะได้รับหมายเลขสล็อตสำหรับช่วงและ เพิ่มชนิดข้อมูลช่วงเป็นชนิดของค่าสำหรับอาร์เรย์ที่เชื่อมโยง

ไวยากรณ์:

```
void qrange(aso[key], range_t range_data, int value);
void qrange(aso[key], range_t range_data, String value);
```

คำอธิบาย:

รูทีน qrange สามารถค้นหา หมายเลขสล็อตสำหรับทั้งชนิด integral และชนิด String range ถ้าชนิดของช่วง เป็นชนิด Integral ดังนั้นชนิดอาร์กิวเมนต์ชนิดที่สามควรเป็นจำนวนเต็ม ไม่เช่นนั้น สำหรับชนิดข้อมูลช่วงสตริง อาร์กิวเมนต์ตัวที่สามควร ควรเป็นชนิดสตริง รูทีน qrange จะค้นหา หมายเลขสล็อตที่ตรงกับค่าที่ถูกผ่าน การนับสำหรับจำนวนสล็อต ดังกล่าวจะถูกเพิ่มขึ้นสำหรับชนิดช่วงที่เก็บเป็นค่าไว้ใน อาร์เรย์ที่เชื่อมโยง

พารามิเตอร์:

`aso[key]`
อาร์เรย์ที่เชื่อมโยงกับคีย์ที่ระบุ

`range_data`
ชนิดข้อมูล **range_t**

`value`
ค่าสามารถเป็นเลขจำนวนเต็มหรือสามารถเป็นชนิดสตริง

round_trip_time

วัตถุประสงค์

ส่งคืนเวลาไปกลับที่ราบเรียบสำหรับการเชื่อมต่อ TCP สำหรับ descriptor ซ็อกเก็ตที่ระบุเฉพาะ

ไวยากรณ์

```
int round_trip_time(int sock_fd);
```

คำอธิบาย

ฟังก์ชัน round_trip_time แสดงเวลา smoothed round-trip time (srtt) สำหรับตัวให้คำอธิบายซ็อกเก็ตเฉพาะ ซึ่งจัดเตรียมค่าแบบไปกลับที่ถูกต้องสำหรับ descriptor ซ็อกเก็ตสตรีม และส่งคืนค่า -1 เป็นค่าเวลาแบบไปกลับสำหรับ descriptor ที่ไม่ถูกต้อง หรือไม่ใช่สตรีมซ็อก ฟังก์ชันนี้พร้อมใช้งานใน uft และโปรแกรมจัดการโพรบ syscall เท่านั้น

หมายเหตุ: ฟังก์ชันนี้ต้องการให้ค่า num_pagefaults ที่ปรับได้ของคำสั่ง probevctrl มากกว่า 0 หากค่าเป็น 0 ดังนั้น ฟังก์ชันนี้จะส่งคืน -1 เป็นเวลาแบบไปกลับ

พารามิเตอร์

fd ค่าตัวให้คำอธิบายไฟล์หรือซ็อกเก็ต

set_aso_print_options

วัตถุประสงค์

เปลี่ยนอาร์เรย์การพิมพ์ดีฟอลต์สำหรับอาร์เรย์ที่เชื่อมโยง

ไวยากรณ์

```
void set_aso_print_options( int num-of-entries, int flags);
```

คำอธิบาย

ฟังก์ชัน **set_aso_print_options( )** จะเปลี่ยน อ็อพชันการพิมพ์ดีฟอลต์สำหรับดีฟอลต์ที่เชื่อมโยง อ็อพชันการพิมพ์ ที่สามารถจัดเตรียมโดยอ็อพชันและค่าเริ่มต้นจะถูกแสดงรายการ ภายใต้ส่วน 'Associative Array type' ฟังก์ชันนี้อนุญาตให้ใช้ ได้ในโพรบ BEGIN เท่านั้น

พารามิเตอร์

num-of-entries
ระบุเพื่อพิมพ์คีย์ 'n' ตัวแรกหรือคู่ของค่า ถ้าเป็นค่า 0 รายการทั้งหมด จะถูกแสดง

flags
ระบุแฟล็ก sort-type, sort-by และ list-valueพารามิเตอร์นี้เป็นทางเลือก แฟล็ก sort-type, sort-by และ list-value ถูกกล่าวถึงภายใต้ส่วนของ 'Associative Array Type'

set_range

วัตถุประสงค์:

กำหนดค่าเริ่มต้นข้อมูลชนิดช่วง linear และ power

ไวยากรณ์:

```
void set_range(range_t range_data, LINEAR, int min, int max, int step);
void set_range(range_t range_data, POWER, 2);
```

คำอธิบาย:

มีความแตกต่างกันของ *set_range* สองชนิด เมื่อต้องการกำหนดค่าเริ่มต้นเป็นช่วง linear แฟล็ก LINEAR with *min*, *max* และ *step* จะ ถูกผ่านเป็นอาร์กิวเมนต์ เพื่อเตรียมข้อมูลเบื้องต้นช่วง Power แฟล็ก POWER ด้วยค่าสอง จะถูกส่งเป็นอาร์กิวเมนต์ รูทีนนี้จะเตรียมข้อมูลเบื้องต้นชนิด ช่วงเป็น Linear หรือเป็น Power ตามค่าอาร์กิวเมนต์ที่ส่ง ข้อมูลชนิดช่วงแบบ linear จะถูกกำหนดค่าเริ่มต้นด้วยค่า *min*, *max* และ *step* ที่ถูกผ่าน ขณะที่ข้อมูลชนิดช่วงแบบ power จะถูกกำหนดค่าเริ่มต้นด้วยค่า ยกกำลังเป็น 2

พารามิเตอร์ (สำหรับชนิดช่วงแบบ Linear):

`range_data`
ชนิดข้อมูล **range_t**

`LINEAR`
แฟล็กค่าคงที่จำนวนเต็มที่ระบุว่าการกระจายของ**range_data** เป็นแบบ linear

`min`
ระบุการเชื่อมขั้นต่ำของ **range_data**

`max`
ระบุการเชื่อมขั้นสูงของ **range_data**

`step`
ระบุขนาดของช่วงของค่าที่ระบุไว้สำหรับ แถวของ **range_data** ชนิดของ min, max & step สามารถเป็น integral (int, short, long, long, long) ไม่อนุญาตให้ใช้ชนิดอื่น

พารามิเตอร์ (สำหรับชนิดช่วง Power):

`range_data`
ชนิดข้อมูล **range_t**

`POWER`
แฟล็กค่าคงที่เลขจำนวนเต็มที่บ่งชี้ว่าการกระจายค่าเป็นการกระจายแบบ POWER

ค่าคงที่ที่ระบุค่าของการยกกำลัง ปัจจุบันสนับสนุน ยกกำลังสองเท่านั้น

set_date_format

วัตถุประสงค์

อัพเดตรูปแบบวันที่ที่ใช้สำหรับการพิมพ์ชนิดข้อมูล `probev_timestamp_t`

ไวยากรณ์

```
void set_date_format(String s);
```

คำอธิบาย

อัพเดตรูปแบบวันที่

ฟังก์ชันนี้สนับสนุนตัวระบุการแปลงที่สนับสนุนโดย `strftime()` ของไลบรารี C สำหรับรูปแบบวันที่ ตัวระบุใดๆ ซึ่งไม่ได้รับการสนับสนุนโดย `strftime()` จะไม่ถูกต้องและรูปแบบดีฟอลต์จะถูกนำมาใช้

รูปแบบดีฟอลต์

MM:DD:YYYY hh:mm:ss TZ

MM เดือนของปีในรูปของเลขฐานสิบ (01 ถึง 12)

DD วันของเดือนในรูปของเลขฐานสิบ (01 ถึง 31)

YYYY ปีในรูปของเลขฐานสิบ (ตัวอย่างเช่น 1989)

hh ชั่วโมงรูปแบบ 24 ชั่วโมงในรูปของเลขฐานสิบ (00 ถึง 23)

mm นาทีของชั่วโมงในรูปแบบเลขฐานสิบ (00 ถึง 59)

ss วินาทีของนาทีในรูปแบบเลขฐานสิบ (00 ถึง 59)

TZ ชื่อเขตเวลาหากสามารถดึงข้อมูลได้ (ตัวอย่างเช่น CDT)

หมายเหตุ: ฟังก์ชัน `set_date_format()` ถูกเรียกในโพรบ `@@BEGIN` เท่านั้น สตริงค่าคงที่ต้องส่งผ่านเป็น *รูปแบบ*

พารามิเตอร์

S – สตริงที่เก็บรูปแบบวันที่

sockfd_netinfo

วัตถุประสงค์

รับข้อมูลโลคัลและรีโมตพอร์ต และ IP แอดเดรส สำหรับตัวให้คำอธิบายซ็อกเก็ตเฉพาะ

ไวยากรณ์

```
void sockfd_netinfo(int sock_fd,  net_info_t ninfo);
```

คำอธิบาย

ฟังก์ชัน `sockfd_netinfo` ขอรับข้อมูลโลคัล IP แอดเดรส, รีโมต IP แอดเดรส หมายเลขพอร์ตโลคัล และหมายเลขรีโมตพอร์ตสำหรับตัวให้คำอธิบายซ็อกเก็ตอินพุต ฟังก์ชันนี้ ขอรับหมายเลขพอร์ตแบบโลคัลที่ถูกต้องและหมายเลขพอร์ตแบบรีโมตและข้อมูล IP address สำหรับ descriptor ซ็อกเก็ตที่ถูกต้อง ซึ่งรับค่า 0 สำหรับ descriptor ที่ม่ถูกต้องหรือหาก descriptor ไม่ได้เป็นชนิดซ็อกเก็ต

หมายเหตุ: ฟังก์ชันนี้ต้องการให้ค่า num_pagefaults ที่ประบได้ของคำสั่ง `probevctrl` มากกว่า 0 และต้องการค่า 2 หรือมากกว่า ถ้ามีค่า 0 ฟังก์ชันนี้จะรับค่าที่ไม่ถูกต้อง (0) สำหรับพอร์ตโลคัลและพอร์ตแบบรีโมตและข้อมูล IP address

พารามิเตอร์

fd ค่าตัวให้คำอธิบายไฟล์หรือซ็อกเก็ต

ninfo ระบุตัวแปรสคริปต์ `net_info_t` ที่ข้อมูล 4 tuple ของเครือข่าย (โลคัลและรีโมต IP แอดเดรส และหมายเลขพอร์ต) สำหรับตัวให้คำอธิบายไฟล์เฉพาะจะถูกคัดลอก

start_tentative, end_tentative

วัตถุประสงค์

บ่งชี้ถึงจุดเริ่มต้นและจุดสิ้นสุดของส่วนของการติดตามชั่วคราว

ไวยากรณ์

```
void start_tentative( String bufID );
void end_tentative( String bufID );
```

คำอธิบาย

ฟังก์ชันเหล่านี้บ่งชี้ถึงจุดเริ่มต้นและจุดสิ้นสุดของส่วนของการติดตามชั่วคราวภายใน Vue clause ข้อมูลการติดตามถูกสร้างโดยฟังก์ชันเอาต์พุตการติดตาม ที่ล้อมรอบอยู่ภายในส่วนของการการติดตามชั่วคราวซึ่งถูกบันทึกไว้ชั่วคราวจนกว่า **commit_tentative** หรือฟังก์ชัน **discard_tentative** จะถูกเรียกเพื่อ commit หรือละทิ้งข้อมูลนี้ ฟังก์ชัน **end_tentative** คือฟังก์ชันเผื่อเลือก และหากไม่ได้ระบุไว้ จุดสิ้นสุดของ Vue clause จะนำมาใช้เพื่อบ่งชี้ถึงจุดสิ้นสุดของส่วนของการติดตาม ชั่วคราว

ข้อมูลการการติดตามชั่วคราวที่ถูกสร้างขึ้นจะถูกบ่งชี้โดยพารามิเตอร์ *bufID* ซึ่งต้องการค่าคงที่ของสตริง หรือตัวอักษร และไม่ใช่ตัวแปร ข้อมูลการติดตามชั่วคราวสามารถควบคุมได้ภายใต้ ID อื่นๆ อย่างพร้อมเพียงกันซึ่งสามารถ commit หรือละทิ้งเป็นบล็อกๆ ได้ ProbeVue สนับสนุนบัฟเฟอร์การติดตาม ชั่วคราวได้มากสุด 16 บัฟเฟอร์ในเซสชันการติดตามแบบไดนามิกเซสชันเดียวกัน ดังนั้น ID การติดตามต่างๆ มากสุด 16 บัฟเฟอร์สามารถใช้ในสคริปต์ Vue Vue clause เดี่ยวสามารถมีส่วนของการติดตามชั่วคราวที่มากกว่าหนึ่งพร้อมกับ ID ที่ต่างกัน

พารามิเตอร์

พารามิเตอร์	คำอธิบาย
bufID	ระบุค่าคงที่ของสตริงที่บ่งชี้ถึง ID บัฟเฟอร์การติดตามชั่วคราว

stktrace

วัตถุประสงค์

สร้างและพิมพ์การติดตามแบบสแต็ก ณ รันไทม์

ไวยากรณ์

```
void stktrace ( int flags, int levels );
```

คำอธิบาย

ฟังก์ชัน **stktrace** พิมพ์การติดตามแบบสแต็ก ที่จุดโพรบปัจจุบัน ตามค่าดีฟอลต์แล้วการติดตามแบบสแต็กจะถูกสร้างโดยการบีบอัดด้วยการเรียกแอดเดรสของกลุ่มของเร็กคอร์ดที่เชื่อมโยงกัน สำหรับระดับที่มากสุดสองระดับ คุณสามารถใช้พารามิเตอร์ *แฟล็ก* และ *ระดับ* เพื่อแก้ไขรูปแบบและเนื้อหาของ การติดตามแบบสแต็ก ProbeVue ไม่สามารถอ่านข้อมูลที่เพจออกได้ ดังนั้นการติดตามสำหรับสแต็กที่ถูกตัดปลาย หากพบกับข้อบกพร่องของเพจ เมื่อเข้าถึงสแต็ก

ฟังก์ชัน **stktrace** ไม่ได้ส่งคืนค่าใดๆ

พารามิเตอร์

พารามิเตอร์	คำอธิบาย
flags	ตั้งค่า 0 เพื่อระบุลักษณะการทำงานที่เป็นค่าดีฟอลต์ หรือระบุหนึ่งในแฟล็กต่อไปนี้ อย่างใดอย่างหนึ่ง: **PRINT_SYMBOLS** พิมพ์ชื่อสัญลักษณ์แทนแอดเดรส **GET_USER_TRACE** ตามค่าดีฟอลต์แล้ว trace แบบสแต็กถูกหยุดทำงานที่เขตของการเรียกระบบ ถ้าตำแหน่งโพรบอยู่ในพื้นที่เคอร์เนล แฟล็กนี้บ่งชี้เพื่อติดตาม วิธีทั้งหมดภายในพื้นที่ของผู้ใช้ยังเพิ่มจำนวนของระดับมากสุดที่ระบุด้วยพารามิเตอร์ *levels* **GET_ALL_MSTS** ตามค่าดีฟอลต์แล้ว การติดตามแบบสแต็กจะถูกรวบรวมสำหรับหนึ่งคอนเท็กซ์ (สถานะเครื่อง) โดยที่เริ่มต้นทำงาน กับโพรบ ถ้าระบุแฟล็กนี้ไว้ การติดตามแบบสแต็กจะถูกพิมพ์ไว้สำหรับคอนเท็กซ์ที่เกี่ยวข้องทั้งหมดสำหรับ CPU นั้น ถ้าคุณต้องการส่งผ่านแฟล็กจำนวนมาก แฟล็กอื่นๆ ต้องเป็น 'or' โดยใช้ตัวดำเนินการ OR นั่นคือตัวดำเนินการ 'l' คุณไม่สามารถส่งค่าตัวแปรสำหรับพารามิเตอร์นี้ได้
levels	บ่งชี้จำนวนของระดับที่มากสุดเท่ากับการติดตาม แบบสแต็กที่ต้องการพิมพ์ ค่า -1 บ่งชี้ว่า กลุ่มของเร็กคอร์ดที่เชื่อมโยงกันแบบสแต็กย้อนกลับ จะเดินข้ามส่วนขยายที่เป็นไปได้ ค่าดีฟอลต์ 0 จะติดตามกลับไป 2 ระดับ **หมายเหตุ:** ถ้ารายการต่างๆ จาก mst จำนวนมากถูกพิมพ์ ขอบเขต mst จะถูกคั่นด้วยบรรทัดที่มีอักขระ '-' บรรทัดนี้ยังถูกพิจารณาให้เป็นระดับ 1 นั่นหมายความว่า รายการต่างๆ ที่พิมพ์เป็นพารามิเตอร์ระดับที่จัดเตรียมจำนวนบรรทัดตัวคั่นที่ติดค่าลบ (ยกเว้นพารามิเตอร์ที่มีค่า -1)

strstr

วัตถุประสงค์

ส่งคืนสตริงภายในสตริงอื่น

ไวยากรณ์

```
String strstr( String s1, String s2 );
```

คำอธิบาย

ฟังก์ชัน **strstr** ค้นหาเหตุการณ์ที่เกิดขึ้นในครั้งแรกของสตริงที่ระบุด้วยพารามิเตอร์ *s2* ในสตริงที่ระบุด้วยพารามิเตอร์ *s1* และส่งคืนสตริงใหม่ที่มีอักขระในพารามิเตอร์ *s1* ที่เริ่มต้นจากตำแหน่งนี้ ทั้งพารามิเตอร์ *s1* และพารามิเตอร์ *s2* ไม่สามารถแก้ไขด้วยการดำเนินการนี้ได้ ถ้าลำดับของอักขระที่ระบุด้วยพารามิเตอร์ *s2* ไม่ได้ปรากฏขึ้นเพียงครั้งเดียวในพารามิเตอร์ *s1* ดังนั้น ฟังก์ชันนี้ส่งคืนสตริงเปล่า

หมายเหตุ: ลักษณะการทำงานของฟังก์นี้ไม่เหมือนกับรูทีนย่อย **strstr** ในไลบรารี C

พารามิเตอร์

พารามิเตอร์	คำอธิบาย
s1	ระบุสตริงภายในที่ต้องการค้นหา
s2	ระบุสตริงที่ต้องการค้นหา

sum

วัตถุประสงค์

ส่งคืนผลรวมของส่วนประกอบทั้งหมดที่อยู่ในรายการ

ไวยากรณ์

long long sum (List *listvar* **);**

คำอธิบาย

ฟังก์ชัน **sum** ส่งคืนผลรวมขององค์ประกอบทั้งหมด ซึ่งได้ผนวกเข้ากับตัวแปร list ที่ระบุด้วยพารามิเตอร์ *listvar*

พารามิเตอร์

พารามิเตอร์	คำอธิบาย
listvar	ระบุตัวแปรของชนิด **รายการ**

timestamp

วัตถุประสงค์

ส่งคืนการประทับเวลาปัจจุบัน

ไวยากรณ์

```
probev_timestamp_t timestamp( );
```

คำอธิบาย

ฟังก์ชัน **timestamp** ส่งคืนการประทับเวลาปัจจุบันใน **probev_timestamp_t** abstract data type แม้ว่าค่า abstract จะมีคุณสมบัติต่อไปนี้:

- ส่งคืนค่าที่เท่ากันหรือใกล้เคียงกันเมื่อเรียกจาก CPU อื่นๆ อย่างพร้อมเพียงกัน
- ถ้าฟังก์ชัน **timestamp** จะถูกเรียกใช้สองครั้ง และการเรียกใช้ครั้งที่สองสามารถรับประกันเหตุการณ์ที่เกิดขึ้นในภายหลัง ค่าที่ส่งคืนสำหรับการเรียกครั้งที่สองมากกว่า หรือเท่ากับค่าที่ส่งคืนโดยการเรียกในครั้งแรก (จัดเตรียมระบบได้รีบูตในระหว่างการเรียก)

ไม่มีความสัมพันธ์ระหว่างค่าที่ส่งคืนโดยฟังก์ชัน **timestamp** สำหรับระบบสองระบบที่แตกต่างกัน แม้ว่า คอมไพลเลอร์จะอนุญาตให้คุณใช้ค่าที่ส่งคืนเป็นเลขจำนวนเต็ม 64 บิต การทำเช่นนี้ยังสามารถเป็นตัวเริ่มต้นปัญหาเกี่ยวกับความเข้ากันได้

หมายเหตุ: ตัวแปรเคอร์เนล **lbolt** ซึ่งมีค่าที่ระบุจำนวนของการทำเครื่องหมายเนื่องจากการบูต หรือตัวแปรเคอร์เนล **time** ซึ่งมีค่าที่บ่งชี้ถึงจำนวนวินาทีตั้งแต่ epoch (มกราคม 1, 1970) ที่สามารถใช้แทนฟังก์ชันนี้ได้ ถ้าการประทับเวลาการแก้ปัญหาที่ต่ำกว่าจะสามารถยอมรับได้

```
typedef long long time_t;
 __kernel time_t lbolt;    /* number of ticks since last boot     */
 __kernel time_t time;     /* memory mapped time in secs since epoch  */
```

พารามิเตอร์

ฟังก์ชัน **timestamp** ไม่ได้ใช้พารามิเตอร์ใดๆ

ติดตาม

วัตถุประสงค์

คัดลอกข้อมูลดิบไปยังบัฟเฟอร์ trace ในรูปขแงเลขฐานสิบหก

ไวยากรณ์

```
void trace ( data );
```

คำอธิบาย

ฟังก์ชัน **trace** ใช้พารามิเตอร์เดี่ยว ซึ่งต้องเป็นตัวแปร ฟังก์ชัน **trace** ไม่ยอมรับนิพจน์ใดๆ

ฟังก์ชัน **trace** คัดลอกค่าของอาร์กิวเมนต์การส่งภายใน ไปยังบัฟเฟอร์ trace อาร์กิวเมนต์สามารถเป็นชนิดข้อมูลใดๆ และขนาดของข้อมูลที่คัดลอกไปยังบัฟเฟอร์ จะอ้างอิงตามขนาดดั้งเดิมของอาร์กิวเมนต์ ดังนั้น ขนาดสี่ไบต์จะถูกคัดลอกไว้สำหรับอาร์กิวเมนต์เลขจำนวนเต็ม ขนาดสี่หรือแปดไบต์สำหรับตัวชี้ (ขึ้นอยู่กับว่า การประมวลผลนั้นอยู่ในโหมด 32 บิต หรือ 64 บิต) และขนาดของโครงสร้างสำหรับอาร์กิวเมนต์ชนิด **struct** สำหรับตัวแปรชนิด **String** จำนวนไบต์ที่คัดลอกคือความยาวสตริงที่ประกาศไว้ (ไม่ใช่ความยาวที่เหมือนกับความยาวของสตริงที่อยู่ในตัวแปร) ตัวแปรชนิด **probev_timestamp_t** จะมีความยาวอย่างน้อย 8 ไบต์

ตัวรายงาน trace จะแสดงข้อมูลเลขฐานสิบหกที่เขียนด้วยฟังก์ชัน **trace** ในกลุ่มของอักขระสี่ตัว โดยไม่มีการจัดรูปแบบเพิ่มเติม

หมายเหตุ: ฟังก์ชัน **trace** ยังยอมรับตัวแปรชนิด **list** ในฐานะเป็นพารามิเตอร์ แต่เอาต์พุตไม่สามารถนำมาใช้ประโยชน์ได้ในกรณีนี้

พารามิเตอร์

พารามิเตอร์	คำอธิบาย
ข้อมูล	อาร์กิวเมนต์ data จะถูกคัดลอกไปยังบัฟเฟอร์ trace

การรัน ProbeVue

การติดตามแบบไดนามิกจะอนุญาตให้ใช้สำหรับผู้ใช้ที่มี privilege หรือ superuser เท่านั้น

การให้สิทธิและ privilege

ซึ่งไม่เหมือนกันตัวช่วยการติดตามสถิติใน AIX ซึ่งจะบังคับใช้กับการตรวจสอบ privilege ที่ถูกจำกัด และมีเหตุผลสำหรับความต้องการ ในการรันคำสั่ง **probevue** สคริปต์ Vue สามารถสร้างผลกระทบที่รุนแรงสำหรับผลการทำงานของระบบ มากกว่าตัวช่วยการติดตามสถิติ เช่น การติดตามระบบ AIX เนื่องจากจุดโพรบสำหรับการติดตามระบบได้ถูกกำหนดไว้ก่อน และถูกจำกัด ProbeVue สามารถสนับสนุนจุดโพรบได้มากกว่า และตำแหน่งของโพรบสามารถถูกกำหนดได้ทุกที่ นอกจากนี้ การดำเนินการติดตาม ProbeVue ที่จุดโพรบสามารถใช้เวลาในการออกคำสั่งนานกว่าการดำเนินการติดตามระบบที่จุดโพรบ เนื่องจากมีข้อจำกัดในการดักจับข้อมูลที่ชัดเจน

นอกจากนี้ ProbeVue อนุญาตให้คุณติดตามการประมวลผล และอ่านตัวแปรเคอร์เนลแบบโกลบอล ซึ่งทั้งสองนี้ต้องการการควบคุม เพื่อป้องกันการเปิดเผยการรักษาความปลอดภัย เซสชัน ProbeVue ยังสามารถใช้หน่วยความจำที่ตรึงไว้จำนวนมาก และการจำกัดการใช้ ProbeVue สำหรับผู้ใช้ที่มี privilege จะลดความเสี่ยงของการปฏิเสธการจู่โจมเซอร์วิส ProbeVue ยังอนุญาตให้ผู้ดูแลระบบควบคุมการใช้หน่วยความจำของเซสชัน ProbeVue ผ่านอินเตอร์เฟส SMIT

Privilege สำหรับการติดตามแบบไดนามิกจะถูกขอรับได้ต่างกัน ขึ้นอยู่กับการเปิดใช้การควบคุมสิทธิในการเข้าถึงแบบอิงบทบาท (RBAC) หรือไม่ โปรดอ้างถึง AIX man pages สำหรับข้อมูลเพิ่มเติมเกี่ยวกับการเปิดใช้งานและปิดใช้งาน RBAC

หมายเหตุ ในโหมดดั้งเดิมหรือการปิดใช้งาน RBAC– จะไม่มีสิทธิ ผู้ใช้ปกติไม่สามารถได้รับ privilege เพื่อรันคำสั่ง **probevue** เพื่อเริ่มต้นเซสชันการติดตามแบบไดนามิก หรือรันคำสั่ง **probevctrl** เพื่อดูแลจัดการ ProbeVue เฉพาะ superuser สามารถมี privilege สำหรับฟังก์ชันทั้งสองเหล่านี้ ห้ามปิดใช้งาน RBAC เมื่อใช้ ProbeVue เว้นเสียแต่คุณต้องการจำกัดตัวช่วยนี้ในผู้ใช้ root เท่านั้น

โหมด RBAC ที่เปิดใช้งาน

Privilege ในระบบ RBAC จะถูกได้รับผ่านการให้สิทธิ การให้สิทธิคือ สตริงข้อความที่เชื่อมโยงกับฟังก์ชันที่เกี่ยวข้องกับการรักษาความปลอดภัย หรือคำสั่ง การให้สิทธิจะจัดเตรียมกลไกเพื่อให้สิทธิแก่คุณ เพื่อดำเนินการกับการดำเนินการที่กำหนด privilege ไว้ เฉพาะผู้ใช้ที่มีการให้สิทธิที่เพียงพอ สามารถออกคำสั่ง **probevue** และเริ่มต้นเซสชันการติดตามแบบไดนามิก

aix.ras.probevue.trace.user.self
การให้สิทธินี้อนุญาตให้คุณติดตามแอ็พพลิเคชัน ในพื้นที่ของผู้ใช้ ID ผู้ใช้ของการประมวลผลที่ถูกติดตามต้องเท่ากับ ID ผู้ใช้จริง ID ของผู้ใช้ที่เรียกใช้งานคำสั่ง **probevue** การให้สิทธินี้อนุญาตให้คุณเปิดใช้งานจุดโพรบที่จัดเตรียมไว้โดยตัวจัดการโพรบ uft สำหรับการประมวลผลของคุณ อย่างไรก็ตาม เพื่อให้ได้ประสิทธิผล ID ผู้ใช้จริงและ ID ผู้ใช้ที่บันทึกไว้ของการประมวลผลที่ถูกติดตามต้องเท่ากัน ดังนั้น คุณไม่สามารถติดตามโปรแกรม setuid ด้วยการให้สิทธินี้

aix.ras.probevue.trace.user
การให้สิทธินี้อนุญาตให้คุณติดตามแอ็พพลิเคชันใดๆ ในพื้นที่ของผู้ใช้ซึ่งรวมถึงโปรแกรม setuid และแอ็พพลิเคชัน

ที่เริ่มต้นด้วย superuser โปรดระวัง เมื่อจัดการนอกสิทธิที่ได้รับนี้ การให้สิทธินี้ อนุญาตให้คุณออกคำสั่ง **probevue** และเปิดใช้งานจุดโพรบที่จัดเตรียมโดยตัวจัดการโพรบ uft สำหรับการประมวลผลใดๆ บนระบบ

aix.ras.probevue.trace.syscall.self

การให้สิทธินี้อนุญาตให้คุณติดตามการเรียกของระบบที่ทำโดยแอ็พพลิเคชัน ของระบบ ความมีประสิทธิภาพ ID ผู้ใช้จริงและที่บันทึกไว้ของการประมวลผลที่ทำการเรียกระบบ ต้องเป็นข้อมูลเดียวกับ ID ผู้ใช้จริงของผู้ใช้ที่เรียกใช้งานคำสั่ง **probevue** การให้สิทธินี้อนุญาตให้คุณเปิดใช้งานจุดโพรบที่จัดเตรียมโดยตัวจัดการโพรบ syscall สำหรับการประมวลผลของคุณ ฟิลด์ที่สองของข้อกำหนดคุณสมบัติโพรบ ต้องบ่งชี้ถึง ID การประมวลผลสำหรับการประมวลผลที่เริ่มต้นขึ้น โดยคุณ

aix.ras.probevue.trace.syscall

การให้สิทธินี้อนุญาตให้คุณติดตามการเรียกของระบบที่ทำโดยแอ็พพลิเคชันใดๆ บนระบบ ซึ่งประกอบด้วยโปรแกรม setuid และแอ็พพลิเคชันที่เริ่มต้นด้วย superuser โปรดระวัง เมื่อจัดการนอกสิทธิที่ได้รับนี้ การให้สิทธินี้อนุญาตให้คุณออกคำสั่ง **probevue** และเปิดใช้งานจุดโพรบที่จัดเตรียมโดยตัวจัดการโพรบ syscall สำหรับการประมวลผลใดๆ ฟิลด์ที่สองของข้อกำหนดคุณสมบัติ สามารถตั้งค่าเป็น ID การประมวลผลเพื่อโพรบการประมวลผลที่ระบุเฉพาะ หรือเป็น * เพื่อโพรบการประมวลผลทั้งหมด

aix.ras.probevue.trace

การให้สิทธินี้อนุญาตให้คุณติดตามระบบทั้งหมด และสอดแทรกการให้สิทธิที่กำหนดในส่วนที่นำหน้า คุณยังสามารถเข้าถึง และอ่านตัวแปรเคอร์เนลได้ ขณะที่รันคำสั่ง **probevue** พร้อมกับติดตามเหตุการณ์สำหรับการติดตามระบบ โดยใช้ตัวจัดการโพรบ systrace โปรดระวัง เมื่อจัดการนอกสิทธิที่ได้รับนี้

aix.ras.probevue.manage

การให้สิทธินี้อนุญาตให้คุณดูแล ProbeVue ซึ่งประกอบด้วยการเปลี่ยนแปลงค่าของพารามิเตอร์ ProbeVue ที่แตกต่างกัน การเริ่มต้นหรือหยุดทำงาน ProbeVue และการดูรายละเอียดของเซสชันการติดตามแบบไดนามิก ของผู้ใช้ทั้งหมด เมื่อรันคำสั่ง **probevctrl** หากไม่มีการให้สิทธินี้ คุณสามารถใช้คำสั่ง **probevctrl** เพื่อดูข้อมูลเซสชันสำหรับเซสชันการติดตามแบบไดนามิกที่เริ่มต้นโดยคุณ หรือดูค่าปัจจุบันสำหรับพารามิเตอร์ ProbeVue

aix.ras.probevue.rase

การให้สิทธิ์นี้อนญาตให้คุณเข้าถึงชุดของฟังก์ชัน ″RAS events″ Vue ที่มีสิทธิ์ระดับสูง ซึ่งสามารถสร้างเร็กคอร์ดระบบและเร็กคอร์ดการติดตาม LMT สร้างดัมพ์แบบ live และแม้แต่ทำให้ระบบหยุดทำงาน privilege นี้ต้องถูกควบคุมด้วยความระมัดระวัง

aix.ras.probevue

การให้สิทธินี้จะให้ privilege การติดตามแบบไดนามิก และเทียบเท่ากับการรวมกันของการให้สิทธิก่อนหน้านี้

superuser (หรือ root) มีสิทธิเหล่านี้ที่กำหนดให้ตาม ค่าดีฟอลต์ ผู้ใช้อื่นอาจต้องการมีสิทธิที่กำหนดให้กับ ผู้ใช้เหล่านั้นด้วย การสร้างบทบาทเป็นอันดับแรกพร้อมกับชุดของการให้สิทธิ และกำหนดบทบาทให้กับผู้ใช้ ผู้ใช้จะยังไม่ต้องการสับเปลี่ยนบทบาทไปเป็นบทบาทที่มี สิทธิการใช้งานที่ต้องการซึ่งกำหนดไว้สำหรับการติดตามแบบไดนามิก ก่อนที่จะเรียกใช้งานคำสั่ง **probevue** สคริปต์ต่อไปนี้คือ ตัวอย่างของวิธีการที่ใช้ในการจัดเตรียมการให้สิทธิแก่ผู้ใช้ ″joe″ เพื่อเปิดใช้งานพื้นที่ของผู้ใช้และการเรียกระบบโพรบสำหรับการประมวลผลที่เริ่มต้นโดย ″joe″

```
 mkrole authorizations=
  "aix.ras.probevue.trace.user.self,aix.ras.probevue.trace.syscall.self"
  apptrace
chuser roles=apptrace joe
setkst -t roleTR
```

คำสั่ง ng:

```
swrole apptrace
```

หมายเหตุ: ตัวจัดการช่วงเวลาโพรบไม่มีสิทธิการใช้งานเฉพาะที่เชื่อมโยง กับตัวจัดการโพรบ คุณสามารถเปิดใช้งานช่วงเวลาของจุดโพรบได้ หากคุณมีสิทธิการใช้งาน **aix.ras.probevue.trace***

ProbeVue privileges

privileges ที่พร้อมใช้งานสำหรับ ProbeVue จะแสดงในตารางต่อไปนี้ คำอธิบายของแต่ละ privilege และการให้สิทธิ ที่แม็ฑกับ privilege นั้นจะถูกจัดเตรียมไว้ Privileges จากลำดับชั้นที่ parent privilege มีสิทธิทั้งหมดที่เชื่อมโยงกับ privileges ของ children แต่สามารถสอดแทรก privileges เพิ่มเติมได้

ตารางที่ 19. สิทธิพิเศษ ProbeVue.

สิทธิพิเศษ	คำอธิบาย	การให้สิทธิ์	คำสั่งที่เชื่อมโยง
PV_PROBEVUE_ TRC_USER_SELF	อนุญาตให้การประมวลผลเปิดใช้งานจุดโพรบสำหรับพื้นที่ผู้ใช้แบบไดนามิก บนการประมวลผลอื่นด้วย ID ผู้ใช้จริงตัวเดียวกัน	aix.ras.probevue.trace.user.self aix.ras.probevue.trace.user aix.ras.probevue.trace aix.ras.probevue	probevue
PV_PROBEVUE_ TRC_USER	อนุญาตให้การประมวลผลเพื่อเปิดใช้งานจุดโพรบ สำหรับพื้นที่ผู้ใช้แบบไดนามิก ซึ่งประกอบด้วย PV_PROBEVUE_ TRC_USER_SELF privilege	aix.ras.probevue.trace.user aix.ras.probevue.trace aix.ras.probevue	probevue
PV_PROBEVUE_ TRC_SYSCALL_SELF	อนุญาตให้การประมวลผลเปิดใช้งานจุดโพรบสำหรับการเรียกของระบบบนการประมวลผลอื่นด้วย ID ผู้ใช้จริงตัวเดียวกัน	aix.ras.probevue.trace.syscall.self aix.ras.probevue.trace.syscall aix.ras.probevue.trace aix.ras.probevue	probevue
PV_PROBEVUE_ TRC_SYSCALL	อนุญาตให้การประมวลผลเปิดใช้งานจุดโพรบ สำหรับพื้นที่การเรียกของระบบแบบไดนามิก ซึ่งประกอบด้วย PV_PROBEVUE_ TRC_SYSCALL_ SELF privilege	aix.ras.probevue.trace.syscall aix.ras.probevue.trace aix.ras.probevue	probevue
PV_PROBEVUE _TRC_KERNEL	อนุญาตให้การประมวลผลเข้าถึงข้อมูลเคอร์เนล เมื่อติดตามแบบไดนามิก	aix.ras.probevue.trace aix.ras.probevue	probevue
PV_PROBEVUE_ MANAGE	อนุญาตให้การประมวลผลดูแล ProbeVue	aix.ras.probevue.manage aix.ras.probevue	probevctrl
PV_PROBEVUE_ RASE	สิทธิ์ในการใช้ฟังก์ชัน "RAS events" แบบจำกัด	aix.ras.probevue.rase aix.ras.probevue	probevue

ตารางที่ 19. สิทธิพิเศษ ProbeVue (ต่อ).

สิทธิพิเศษ	คำอธิบาย	การให้สิทธิ์	คำสั่งที่เชื่อมโยง
PV_PROBEVUE_	เทียบเท่ากับการรวม privileges (PV_PROBEVUE_*) ก่อนหน้านี้ทั้งหมด	aix.ras.probevue	probevue probevctrl

ProbeVue พารามิเตอร์

AIX จัดเตรียมชุดของพารามิเตอร์ที่คุณสามารถใช้เพื่อปรับ ProbeVue หรือกรอบงาน ProbeVue พารามิเตอร์อนุญาตให้คุณระบุทั้งข้อจำกัดแบบโกลบอลเกี่ยวกับการใช้รีซอร์สโดย กรอบงาน ProbeVue และเพื่อระบุการใช้รีซอร์สสำหรับผู้ใช้แต่ละราย

หมายเหตุ: ตัวจัดการโพรบ ไม่ได้อยู่ภายในกรอบงาน ProbeVue และข้อจำกัดเหล่านี้ไม่ได้ใช้กับตัวจัดการโพรบ

พารามิเตอร์ ProbeVue สามารถแก้ไขได้ผ่านอินเตอร์เฟส SMIT (ใช้วิธีลัด "smit probevue") หรือผ่านคำสั่ง **probevctrl** ได้โดยตรง ProbeVue สามารถหยุดได้หากไม่มีเซสชันการติดตามแบบไดนามิกที่แอ็คทีฟ และสามารถเริ่มต้นได้หลังจากที่หยุดทำงานโดยไม่ต้องรีบูต ProbeVue อาจล้มเหลวในการหยุดทำงาน ถ้าเซสชันใดๆ ที่ใช้ตัวแปรเธรดแบบโลคัล ได้แอ็คทีฟแล้วก่อนหน้า

ตารางต่อไปนี้สรุปพารามิเตอร์ที่กำหนดไว้สำหรับเซสชันการติดตาม แบบไดนามิก สำหรับคำอธิบาย ผู้ใช้ที่มี privilege จะอ้างอิงกับ superuser หรือผู้ใช้ที่มีสิทธิ **aix.ras.probevue.trace** และผู้ใช้ที่ไม่มี privilege คือผู้ใช้ที่ไม่มีสิทธิการใช้นี้

ตารางที่ 20. พารามิเตอร์สำหรับเซสชันการติดตามแบบไดนามิก.

คำอธิบายที่เหมือนกับใน SMIT	ค่าสูงสุด	ค่าคอนฟิกูเรชันสูงของการกำหนดค่าเริ่มต้น	ค่าคอนฟิกูเรชันต่ำของการกำหนดค่าเริ่มต้น	ค่าต่ำสุด	คำสั่งที่เชื่อมโยง
หน่วยความจำ MAX ที่ตรึงไว้สำหรับกรอบงาน ProbeVue	64 GB	10% ของหน่วยความจำที่มีอยู่ของค่าสูงสุดไม่ว่าจะเป็นค่าที่เล็กกว่าก็ตาม	16 MB	3 MB	หน่วยความจำสูงสุดที่ตรึงไว้ในหน่วย MB ซึ่งได้จัดสรรไว้สำหรับโครงสร้างข้อมูล ProbeVue รวมถึงสแต็กต่อ CPU และส่วนของตารางโลคัลต่อ CPU และเซสชันการติดตามแบบไดนามิกทั้งหมด โดยไม่ได้รวมหน่วยความจำใดๆ ที่จัดสรรไว้แล้วโดยตัวจัดการโพรบ **หมายเหตุ:** แม้ว่าพารามิเตอร์นี้สามารถปรับเปลี่ยนได้ตลอดเวลา ค่าจะมีมีผลบังคับใช้ในครั้งถัดไปที่เริ่มต้นทำงานกับ ProbeVue เท่านั้น

ตารางที่ 20. พารามิเตอร์สำหรับเซสชันการติดตามแบบไดนามิก (ต่อ).

คำอธิบายที่เหมือนกับใน SMIT	ค่าสูงสุด	ค่าคอนฟิกูเรชันสูงของการกำหนดค่าเริ่มต้น	ค่าคอนฟิกูเรชันต่ำของการกำหนดค่าเริ่มต้น	ค่าต่ำสุด	คำสั่งที่เชื่อมโยง
ดีฟอลต์ขนาดบัฟเฟอร์การติดตามต่อ CPU	256 MB	128 KB	8 KB	4 KB	ขนาดดีฟอลต์ในหน่วย KB ของบัฟเฟอร์การติดตามต่อ CPU บัฟเฟอร์การติดตามสองตัวถูกจัดสรรไว้ต่อ CPU สำหรับแต่ละเซสชันการติดตามแบบไดนามิกโดย ProbeVue หนึ่งบัฟเฟอร์การติดตามจะแอ็คทีฟและใช้โดยตัวเขียนหรือโปรแกรม Vue เมื่อดักจับข้อมูลการติดตาม และอีกหนึ่งบัฟเฟอร์การติดตามจะไม่แอ็คทีฟและใช้โดยตัวอ่านหรือผู้ใช้การติดตามตัวอย่างเช่น 8 ทิศทาง พร้อมกับขนาดบัฟเฟอร์การติดตามต่อ CPU ที่ตั้งค่าเป็น 16 KB หน่วยความจำทั้งหมดที่ใช้โดยบัฟเฟอร์การติดตาม สำหรับเซสชัน ProbeVue คือ 256 KB คุณสามารถระบุขนาดบัฟเฟอร์ที่แตกต่างกันได้ (ใหญ่กว่าหรือเล็กกว่า) เมื่อคุณเริ่มต้นคำสั่ง **probevue** จนกระทั่งอยู่ภายในขีดจำกัดของหน่วยความจำเซสชัน

ตารางที่ 20. พารามิเตอร์สำหรับเซสชันการติดตามแบบไดนามิก (ต่อ).

คำอธิบายที่เหมือนกับใน SMIT	ค่าสูงสุด	ค่าคอนฟิกูเรชันสูงของการกำหนดค่าเริ่มต้น	ค่าคอนฟิกูเรชันต่ำของการกำหนดค่าเริ่มต้น	ค่าต่ำสุด	คำสั่งที่เชื่อมโยง
หน่วยความจำ MAX ที่ตรึงไว้สำหรับเซสชันของผู้ใช้ปกติ	64 GB	2 MB	2 MB	0 MB	หน่วยความจำสูงสุดที่ตรึงไว้ที่จัดสรรไว้สำหรับผู้ใช้ที่ไม่มีสิทธิพิเศษในเซสชัน ProbeVue โดยรวมถึงหน่วยความจำสำหรับบัฟเฟอร์การติดตามต่อ CPU ค่า 0 จะปิดใช้งานผู้ใช้ที่ไม่มีสิทธิพิเศษทั้งหมด ผู้ใช้ที่มีสิทธิพิเศษจะไม่มีข้อจำกัดเกี่ยวกับหน่วยความจำที่ใช้โดยเซสชัน ProbeVue อย่างไรก็ตาม ผู้ใช้เหล่านั้น ยังคงถูกจำกัดโดยหน่วยความจำสูงสุดที่ตรึงไว้ ซึ่งอนุญาตให้ใช้สำหรับกรอบงาน ProbeVue
อัตราการอ่านบัฟเฟอร์การติดตาม MIN สำหรับผู้ใช้ปกติ	5000 ms	100 ms	100 ms	10 ms	ระยะเวลาต่ำสุดในหน่วยมิลลิวินาทีที่ผู้ใช้ที่ไม่มีสิทธิพิเศษสามารถร้องขอ ผู้ใช้การติดตามเพื่อตรวจสอบข้อมูลการติดตาม ค่านี้จะถูกปัดเศษไปเป็นค่าถัดไปที่สูงที่สุดที่มากกว่า 10 มิลลิวินาที ผู้ใช้ที่มีสิทธิพิเศษจะไม่ถูกจำกัดโดยพารามิเตอร์นี้ แต่อัตราการอ่านที่เร็วที่สุด ซึ่งผู้ใช้เหล่านี้สามารถระบุได้ คือ 10 มิลลิวินาที
อัตราการอ่านบัฟเฟอร์การติดตามที่เป็นค่าดีฟอลต์	5000 ms	100 ms	100 ms	10 ms	ระยะเวลาดีฟอลต์ในหน่วยมิลลิวินาทีที่ตรวจสอบบัฟเฟอร์การติดตามในหน่วยความจำสำหรับข้อมูลการติดตามโดยผู้ใช้การติดตาม คุณสามารถระบุอัตราการอ่านที่แตกต่างกัน (ใหญ่กว่าหรือเล็กกว่า) เมื่อเริ่มต้นคำสั่ง **probevue** จนกระทั่งมากกว่าอัตราการอ่านบัฟเฟอต์ ต่ำสุด

ตารางที่ 20. พารามิเตอร์สำหรับเซสชันการติดตามแบบไดนามิก (ต่อ).

คำอธิบายที่เหมือนกับใน SMIT	ค่าสูงสุด	ค่าคอนฟิกูเรชันสูงของการกำหนดค่าเริ่มต้น	ค่าคอนฟิกูเรชันต่ำของการกำหนดค่าเริ่มต้น	ค่าต่ำสุด	คำสั่งที่เชื่อมโยง
เซสชันการใช้งานพร้อมกันสูงสุดสำหรับผู้ใช้ปกติ	8	1	1	0	จำนวนของเซสชัน ProbeVue ที่เกิดขึ้นพร้อมกันอนุญาตให้ใช้สำหรับผู้ใช้ที่ไม่มีสิทธิพิเศษ ค่าศูนย์จะปิดใช้งานผู้ใช้ที่ไม่มี สิทธิพิเศษทั้งหมด
ขนาดของสแต็กการคำนวณต่อ CPU	256 KB	20 KB	12 KB	8 KB	ขนาดของสแต็กการคำนวณต่อ CPU ที่ใช้โดย ProbeVue เมื่อใช้สคริปต์ Vue ค่า จะถูกปัดเศษไปเป็นค่าสูงสุดถัดไปที่มากกว่า 8 KB ซึ่ง ProbeVue จัดสรรสแต็กเดี่ยวต่อ CPU ไว้สำหรับทุกเซสชัน ProbeVue หน่วยความจำ จะถูกใช้สำหรับสแต็กที่ไม่ได้รวมอยู่ในข้อจำกัดต่อเซสชัน **หมายเหตุ:** แม้ว่าพารามิเตอร์นี้ สามารถปรับเปลี่ยนได้ตลอดเวลา ค่าจะมีผลเฉพาะหลังจากบู๊ตอิมเมจเคอร์เนล AIX ถูกสร้างใหม่ และรีบูตเท่านั้น คุณต้องกำหนดค่าสแต็ก ProbeVue เพื่อใช้หน่วยความจำเสมือน 96K เพื่อดูรายการไดเร็กทอรีปัจจุบัน

ตารางที่ 20. พารามิเตอร์สำหรับเซสชันการติดตามแบบไดนามิก (ต่อ).

คำอธิบายที่เหมือนกับใน SMIT	ค่าสูงสุด	ค่าคอนฟิกูเรชันสูงของการกำหนดค่าเริ่มต้น	ค่าคอนฟิกูเรชันต่ำของการกำหนดค่าเริ่มต้น	ค่าต่ำสุด	คำสั่งที่เชื่อมโยง
ขนาดของตารางโลคัลต่อ CPU	256 KB	32 KB	4 KB	4 KB	ขนาดของตารางโลคัลต่อ CPU ที่ใช้โดย ProbeVue เพื่อบันทึกตัวแปรของคลาสแบบอัตโนมัติ และเพื่อบันทึกตัวแปรชั่วคราว ProbeVue ใช้ครึ่งของพื้นที่นี่สำหรับตัวแปรอัตโนมัติ และครึ่งที่เหลือ สำหรับการบันทึกตัวแปรชั่วคราว ค่าจะถูกปัดเศษไปเป็นค่าสูงสุดถัดไปที่ มากกว่า 4 KB ProbeVue จัดสรรตารางโลคัลตารางเดียวและตารางชั่วคราวตารางเดียวต่อ CPU ที่ใช้โดยเซสชัน ProbeVue ทั้งหมด หน่วยความจำที่ใช้สำหรับตารางโลคัลไม่ได้รวมอยู่ในข้อจำกัดต่อเซสชัน **หมายเหตุ:** แม้ว่าพารามิเตอร์นี้สามารถปรับเปลี่ยนได้ตลอดเวลา ค่าจะมีมีผลบังคับใช้ในครั้งถัดไปที่เริ่มต้นทำงานกับ ProbeVue เท่านั้น
ช่วงเวลา MIN ที่ใช้ได้ในโพรบช่วงเวลา	N/A	1		1	ช่วงเวลาต่ำสุดของตัวจับเวลาในหน่วยมิลลิวินาทีอนุญาตให้ใช้สำหรับผู้ใช้ root แบบโกลบอลที่อยู่ในช่วงเวลาโพรบ

ตารางที่ 20. พารามิเตอร์สำหรับเซสชันการติดตามแบบไดนามิก (ต่อ).

คำอธิบายที่เหมือนกับใน SMIT	ค่าสูงสุด	ค่าคอนฟิกูเรชันสูงของการกำหนดค่าเริ่มต้น	ค่าคอนฟิกูเรชันต่ำของการกำหนดค่าเริ่มต้น	ค่าต่ำสุด	คำสั่งที่เชื่อมโยง
จำนวนของเธรดที่ถูกติดตาม	N/A	32	32	1	จำนวนสูงสุดของเธรดที่เซสชัน ProbeVue สามารถสนับสนุนเมื่อมีตัวแปร thread-local เฟรมเวิร์ก ProbeVue จะจัดสรรตัวแปร thread-local เป็นจำนวนสูงสุดของเธรดระบุไว้โดยแอ็ตทริบิวต์นี้ ที่ตอนต้นของ เซสชัน ถ้าเกินจำนวนของเธรดที่ระบุไว้ที่ตรงกับโพรบที่มีตัวแปร thread-local แล้ว เซสชัน ProbeVue จะหยุดทำงานในทันที
จำนวนของข้อบกพร่องของเพจที่ต้องถูกจัดการ	1024	0	0	0	จำนวนของบริบทข้อบกพร่องของเพจที่จัดการกับข้อบกพร่องของเพจสำหรับกรอบงานทั้งหมด บริบทข้อบกพร่องของเพจรวมถึง สแต็ก และตารางโลคัลสำหรับการบันทึกตัวแปรคลาสแบบอัตโนมัติและตัวแปรชั่วคราว ซึ่งจำเป็นต้องใช้บริบทข้อบกพร่องของเพจสำหรับการเข้าถึงข้อมูล เพจเอาต์ ถ้าไม่มีบริบทข้อบกพร่องของเพจว่างขณะที่เกิด ข้อบกพร่องของเพจ ดังนั้น ProbeVue จะไม่ดึงข้อมูลเพจเอาต์
เวลาสูงสุดที่เรียกทำงานโพรบสำหรับโพรบ systrace เมื่อใช้ในบริบทอินเตอร์รัปต์	N/A	0	0	0	จำนวนนี้จำกัดเวลาสูงสุดในหน่วยมิลลิวินาที การเรียกโพรบ systrace ในบริบทอินเตอร์รัปต์สามารถใช้ได้ ตามค่าดีฟอลต์แล้ว ค่าคือศูนย์ ซึ่งหมายความว่า โพรบ systrace สามารถ เป็นเวลาใดๆ

ตารางที่ 20. พารามิเตอร์สำหรับเซสชันการติดตามแบบไดนามิก (ต่อ).

คำอธิบายที่เหมือนกับใน SMIT	ค่าสูงสุด	ค่าคอนฟิกูเรชันสูงของการกำหนดค่าเริ่มต้น	ค่าคอนฟิกูเรชันต่ำของการกำหนดค่าเริ่มต้น	ค่าต่ำสุด	คำสั่งที่เชื่อมโยง
เวลาสูงสุดที่เรียกทำงานโพรบสำหรับโพรบ io เมื่อใช้ในบริบทอินเตอร์รัปต์	N/A	0	0	0	จำนวนนี้จำกัดเวลาสูงสุดในหน่วยมิลลิวินาทีการเรียกโพรบ io ในบริบทอินเตอร์รัปต์สามารถใช้ได้ ตามค่าดีฟอลต์แล้ว ค่าคือศูนย์ ซึ่งหมายความว่าสามารถเป็นเวลาใดๆ ก็ได้
เวลาสูงสุดที่เรียกทำงานโพรบสำหรับโพรบ sysproc เมื่อใช้ในบริบทอินเตอร์รัปต์	N/A	0	0	0	จำนวนนี้จำกัดเวลาสูงสุดในหน่วยมิลลิวินาที ซึ่งการเรียกโพรบ sysproc ในบริบทอินเตอร์รัปต์ สามารถใช้ได้ตามค่าดีฟอลต์แล้ว ค่าคือศูนย์ ซึ่งหมายความว่าสามารถเป็นเวลาใดๆ ก็ได้
เวลาสูงสุดที่เรียกทำงานโพรบสำหรับโพรบ network เมื่อใช้ในบริบทอินเตอร์รัปต์	N/A	0	0	0	จำนวนนี้จำกัดเวลาสูงสุดในหน่วยมิลลิวินาที ซึ่งการเรียกโพรบ network ในบริบทอินเตอร์รัปต์ สามารถใช้ได้ตามค่าดีฟอลต์แล้ว ค่าคือศูนย์ ซึ่งหมายความว่าสามารถเป็นเวลาใดๆ ก็ได้
ขนาดบัฟเฟอร์เครือข่ายสูงสุด	64 KB	64 ไบต์	96 ไบต์	96 ไบต์	ค่านี้คือขนาดบัฟเฟอร์ที่จัดสรรไว้ก่อน (ในหน่วยไบต์) ที่ใช้โดยตัวจัดการโพรบ network สำหรับจุดโพรบ bpf ค่านี้ถูกจัดสรรไว้แล้วเมื่อเปิดใช้งานโพรบ bpf ในครั้งแรกและมีอยู่แล้วในระบบ จนกว่าจะปิดใช้งานโพรบ bpf อันดับสุดท้าย เมื่อปิดใช้งานชนิดโพรบ bpf อันดับสุดท้ายแล้ว บัฟเฟอร์นี้ จะถูกรีลีส บัฟเฟอร์นี้จะถูกใช้เพื่อคัดลอกข้อมูลเมื่อขยายข้อมูลแพ็กเก็ตระหว่างบัฟเฟอร์แพ็กเก็ตจำนวนมาก

ตารางที่ 20. พารามิเตอร์สำหรับเซสชันการติดตามแบบไดนามิก (ต่อ).

คำอธิบายที่เหมือนกับใน SMIT	ค่าสูงสุด	ค่าคอนฟิกูเรชันสูงของการกำหนดค่าเริ่มต้น	ค่าคอนฟิกูเรชันต่ำของการกำหนดค่าเริ่มต้น	ค่าต่ำสุด	คำสั่งที่เชื่อมโยง
ช่วงเวลาการดึงข้อมูลสถิติอะซิงโครนัส หน่วยเป็นมิลลิวินาที	NA	1000 มิลลิวินาที (1 วินาที)	1000 มิลลิวินาที (1 วินาที)	100 มิลลิวินาที	ช่วงเวลาหน่วยเป็นมิลลิวินาทีเพื่อดึงข้อมูลสถิติอะซิงโครนัส ค่านี้เป็นค่าโกลบอลและใช้ได้กับเซสชัน ProbeVue ทั้งหมด
ดึงข้อมูลสถิติในโหมดอะซิงโครนัสเท่านั้น	NA	ไม่	ไม่	NA	ระบุ ProbeVue ที่สถิติต้องถูกดึงข้อมูลในโหมดอะซิงโครนัส แม้ว่าโหมดซิงโครนัสจะพร้อมใช้งาน

การทำโปรไฟล์เซสชัน ProbeVue

กรอบงาน ProbeVue จัดเตรียมสิ่งอำนวยความสะดวกในการทำโปรไฟล์ที่สามารถเปิดหรือปิดได้ เพื่อประเมินค่าผลกระทบของโพรบที่เปิดใช้งานบนแอ็พพลิเคชัน สิ่งอำนวยความสะดวกนี้สะสมเวลาที่ใช้โดยแอ็คชันโพรบเมื่อแอ็คชันเหล่านั้นเริ่มต้นทำงานและรายงานเมื่อร้องขอหรือเมื่อจบเซสชัน

รายงานการทำโปรไฟล์แสดงสตริงโพรบและเวลาที่ใช้โดยแอ็คชัน ที่สอดคล้องกับสตริงโพรบนั้น เวลาที่ใช้โดยแอ็คชันโพรบจะถูกเก็บไว้เป็นรายการ โดยที่ข้อมูลที่รวบรวมไว้คือ เวลาทั้งหมด เวลาต่ำสุด เวลาสูงสุด และเวลาเฉลี่ยที่ใช้โดยแอ็คชันโพรบ ข้อมูลการทำโปรไฟล์ยังแสดงจำนวนครั้งที่แอ็คชันโพรบหมดเวลาใช้งาน เมื่อคุณกำลังมองหาโปรไฟล์ สำหรับฟังก์ชันจำนวนมากผ่านทางสตริงโพรบ (โดยใช้นิพจน์ปกติหรือ * ในตำแหน่งของชื่อฟังก์ชัน) ข้อมูลการทำโปรไฟล์จะจัดเตรียมข้อมูลสะสมของโพรบที่เริ่มต้นทำงาน สำหรับทุกฟังก์ชัน ซึ่งไม่ได้จัดเตรียมรายละเอียดของเวลาสำหรับฟังก์ชันที่ถูกโพรบแยกต่างหาก แต่จะจัดเตรียมต่อแอ็คชันโพรบเท่านั้น

แอ็คชันโพรบ `BEGIN` และ `END` ไม่ได้จัดทำโปรไฟล์ ด้วยสิ่งอำนวยความสะดวกนี้ รายละเอียดการทำโปรไฟล์เหล่านี้เป็นรายละเอียดที่ระบุเฉพาะเซสชัน คุณสามารถเปิดใช้งานการทำโปรไฟล์เซสชัน probevue พร้อมกับเริ่มต้นเซสชันโดยใช้คำสั่ง **probevue** หรือคำสั่ง **probevctrl**

สำหรับข้อมูลเพิ่มเติม โปรดดูคำสั่ง **probevue** และ **probevctrl**

ตัวอย่างโปรแกรม

ตัวอย่างที่ 1

โปรแกรม1 "Hello World" ต่อไปนี้จะพิมพ์ "Hello World" ลงในบัฟเฟอร์การติดตามและออก:

```
#!/usr/bin/probevue

 /* Hello World in probevue */
 /* Program name: hello.e */

 @@BEGIN
```

```
{
 printf("Hello World\n");
 exit();
}
```

ตัวอย่างที่ 2

โปรแกรม "Hello World" ต่อไปนี้จะพิมพ์ "Hello World" เมื่อคุณพิมพ์ Ctrl-C บนคีย์บอร์ด:

```
#!/usr/bin/probevue

 /* Hello World 2 in probevue */
 /* Program name: hello2.e */

 @@END
 {
  printf("Hello World\n");
 }
```

ตัวอย่างที่ 3

โปรแกรมต่อไปนี้แสดงวิธีการใช้ตัวแปรเธรดแบบโลคัล ซึ่งสคริปต์ Vue จะนับจำนวนไบต์ที่เขียนลงในไฟล์เฉพาะ สมมิตว่าการประมวลผลคือเธรดเดี่ยวหรือเธรดที่เปิดไฟล์ ซึ่งเป็นไฟล์เดียวกันกับที่เขียน และสมมุติว่า การดำเนินการเขียนทั้งหมดเป็นผลสำเร็จ สคริปต์สามารถยกเลิกได้ทุกเวลา และคุณสามารถขอรับจำนวนไบต์ปัจจุบันที่เขียนด้วยการพิมพ์ Ctrl-C บนเทอร์มินัล

```
#!/usr/bin/probevue

 /* Program name: countbytes.e */
 int open( char * Path, int OFlag, int mode );
 int write( int fd, char * buf, int sz);
 int done;

 @@syscall:*:open:entry
  when (done != 0 )
 {
  if (get_userstring(__arg1, -1) == "/tmp/foo") {
   thread:trace = 1;
   done = 1;
  }
 }

 @@syscall:*:open:exit
  when (thread:trace)
 {
  thread:fd = __rv;
 }

 @@syscall:*:write:entry
  when (thread:trace && __arg1 == thread:fd)
 {
  bytes += __arg3; /* number of bytes is third arg */
 }
```

```
@@END
{
 printf("Bytes written = %d\n", bytes);
}
```

ตัวอย่างที่ 4

โปรแกรมการติดตามที่ยังพัฒนาอยู่จะแสดงวิธีการติดตามอาร์กิวเมนต์ที่ส่งผ่านไปยังการเรียกระบบให้อ่าน ถ้าส่งคืนค่าศูนย์ไบต์เมื่ออ่านไฟล์ **foo.data** :

```
#!/usr/bin/probevue
 /* File: ttrace.e */
 /* Example of tentative tracing */
 /* Capture parameters to read system call only if read fails */
 int open ( char* Path, int OFlag , int mode );
 int read ( int fd, char * buf, int sz);

 @@syscall:*:open:entry
 {
  filename = get_userstring(__arg1, -1);
  if (filename == "foo.data") {
   thread:open = 1;
   start_tentative("read");
   printf("File foo.data opened\n");
  }
 }

 @@syscall:*:open:exit
  when (thread:open == 1)
 {
    thread:fd = __rv;
    start_tentative("read");
    printf("fd = %d\n", thread:fd);
    thread:open = 0;
 }

 @@syscall:*:read:entry
  when (__arg1 == thread:fd)
 {
  start_tentative("read");
  printf("Read fd = %d, input buffer = 0x%08x, bytes = %d,",
   __arg1, __arg2, __arg3);
  end_tentative("read");
  thread:read = 1;
 }

 @@syscall:*:read:exit
  when (thread:read == 1)
 {
  if (__rv < 0) {
   /* The printf below, even though non-tentative, is only
    * executed in error cases and merges with the
    * previously printed tentative data
    */
   printf(" errno = %d\n", __errno);
```

```
   commit_tentative("read");
  }
  else
   discard_tentative("read");
  thread:read = 0;
 }
```

เอาต์พุตที่เป็นไปได้หากการอ่านเกิดความล้มเหลว เนื่องจากแอดเดรสไม่ถูกต้อง (บอกว่า 0x1000) ส่งผ่านเป็นตัวชี้บัฟเฟอร์อินพุตอาจคล้ายกับที่แสดงด้านล่างนี้:

```
#probevue ttrace.e
File foo.data opened
fd = 4
Read fd = 4, input buffer = 0x00001000, bytes = 256, errno = 14
```

ตัวอย่างที่ 5

สคริปต์ Vue ต่อไปนี้จะพิมพ์ค่าของตัวแปรเคอร์เนลบางตัว และออกโดยทันที ให้สนใจที่ฟังก์ชัน **exit** ในโพรบ **@@BEGIN** :

```
/* File: kernel.e */
/* Example of accessing kernel variables */
/* System configuration structure from /usr/include/sys/systemcfg.h */
struct system_configuration {
 int architecture; /* processor architecture */
 int implementation; /* processor implementation */
 int version;  /* processor version */
 int width;  /* width (32 || 64) */
 int ncpus;  /* 1 = UP, n = n-way MP */
 int cache_attrib; /* L1 cache attributes (bit flags) */
    /* bit  0/1 meaning  */
    /* -----------------------------------*/
    /* 31  no cache / cache present */
    /* 30  separate I and D / combined    */
 int icache_size; /* size of L1 instruction cache */
 int dcache_size; /* size of L1 data cache */
 int icache_asc;  /* L1 instruction cache associativity */
 int dcache_asc;  /* L1 data cache associativity */
 int icache_block; /* L1 instruction cache block size */
 int dcache_block; /* L1 data cache block size */
 int icache_line; /* L1 instruction cache line size */
 int dcache_line; /* L1 data cache line size */
 int L2_cache_size; /* size of L2 cache, 0 = No L2 cache */
 int L2_cache_asc; /* L2 cache associativity */
 int tlb_attrib;  /* TLB attributes (bit flags)  */
    /* bit  0/1 meaning  */
    /* -----------------------------------*/
    /* 31  no TLB / TLB present  */
    /* 30  separate I and D / combined    */
 int itlb_size;  /* entries in instruction TLB */
 int dtlb_size;  /* entries in data TLB */
 int itlb_asc;  /* instruction tlb associativity */
 int dtlb_asc;  /* data tlb associativity */
 int resv_size;  /* size of reservation */
 int priv_lck_cnt; /* spin lock count in supevisor mode */
 int prob_lck_cnt; /* spin lock count in problem state */
```

```
int rtc_type;  /* RTC type */
int virt_alias;  /* 1 if hardware aliasing is supported */
int cach_cong;  /* number of page bits for cache synonym */
int model_arch;  /* used by system for model determination */
int model_impl;  /* used by system for model determination */
int Xint;  /* used by system for time base conversion */
int Xfrac;  /* used by system for time base conversion */
int kernel;  /* kernel attributes       */
   /* bit  0/1 meaning      */
   /* ---------------------------------------*/
   /* 31 32-bit kernel / 64-bit kernel     */
                              /* 30   non-LPAR      / LPAR                */
                              /* 29   old 64bit ABI / 64bit Large ABI     */
                              /* 28   non-NUMA      / NUMA                */
                              /* 27   UP            / MP                  */
                              /* 26   no DR CPU add / DR CPU add support  */
                              /* 25   no DR CPU rm  / DR CPU rm  support  */
                              /* 24   no DR MEM add / DR MEM add support  */
                              /* 23   no DR MEM rm  / DR MEM rm  support  */
                              /* 22   kernel keys disabled / enabled     */
                              /* 21   no recovery   / recovery enabled    */
                              /* 20   non-MLS    / MLS enabled     */
long long physmem; /* bytes of OS available memory      */
int slb_attr;  /* SLB attributes       */
   /* bit  0/1 meaning      */
   /* ---------------------------------------*/
   /* 31  Software Managed     */
int slb_size;  /* size of slb (0 = no slb)      */
int original_ncpus; /* original number of CPUs      */
int max_ncpus;  /* max cpus supported by this AIX image     */
long long maxrealaddr; /* max supported real memory address +1     */
long long original_entitled_capacity;
   /* configured entitled processor capacity   */
   /* at boot required by cross-partition LPAR */
   /* tools.        */
long long entitled_capacity; /* entitled processor capacity     */
long long dispatch_wheel; /* Dispatch wheel time period (TB units)  */
int capacity_increment; /* delta by which capacity can change     */
int variable_capacity_weight; /* priority weight for idle capacity*/
    /* distribution       */
int splpar_status; /* State of SPLPAR enablement      */
   /* 0x1 => 1=SPLPAR capable; 0=not     */
   /* 0x2 => SPLPAR enabled 0=dedicated;   */
   /*        1=shared       */
int smt_status;  /* State of SMT enablement                  */
   /*    0x1 = SMT Capable  0=no/1=yes         */
   /*    0x2 = SMT Enabled  0=no/1=yes         */
   /*    0x4 = SMT threads bound true 0=no/1=yes */
int smt_threads; /* Number of SMT Threads per Physical CPU   */
       int vmx_version;        /* RPA defined VMX version, 0=none/disabled */
long long sys_lmbsize; /* Size of an LMB on this system. */
int num_xcpus;  /* Number of exclusive cpus on line */
signed char errchecklevel;/* Kernel error checking level */
char pad[3];  /* pad to word boundary  */
       int dfp_version;        /* RPA defined DFP version, 0=none/disabled */
```

```
    /* if MSbit is set, DFP is emulated         */
};

__kernel struct system_configuration _system_configuration;

@@BEGIN
{
 String s[40];
 int j;
 __kernel int max_sdl; /* Atomic RAD system decomposition level */
 __kernel long lbolt; /* Ticks since boot  */

 printf("No. of online CPUs\t\t= %d\n", _system_configuration.ncpus);

 /* Print SMT status */
 printf("SMT status\t\t\t=");
 if (_system_configuration.smt_status == 0)
  printf(" None");
 else {
  if (_system_configuration.smt_status & 0x01)
   printf(" Capable");
  if (_system_configuration.smt_status & 0x02)
   printf(" Enabled");
  if (_system_configuration.smt_status & 0x04)
   printf(" BoundThreads");
 }
 printf("\n");

 /* Print error checking level */
 if (_system_configuration.errchecklevel == 1)
  s = "Minimal";
 else if (_system_configuration.errchecklevel == 3)
  s = "Normal";
 else if (_system_configuration.errchecklevel == 7)
  s = "Detail";
 else if (_system_configuration.errchecklevel == 9)
  s = "Maximal";
 printf("Error checking level\t\t= %s\n",s);

 printf("Atomic RAD system detail level\t= %d\n", max_sdl);

 /* Long in the kernel is 64-bit, so we use %lld below */
 printf("Number of ticks since boot\t= %lld\n", lbolt);

 exit();
}
```

เอาต์พุตต่อไปนี้เป็นเอาต์พุตที่เป็นไปได้เมื่อคุณรันสคริปต์ ก่อนหน้าบนพาร์ติชันเฉพาะ Power 5 ด้วยดีฟอลต์เคอร์เนลแอ็ตทริบิวต์:

```
# probevue kernel.e
No. of online CPUs              = 4
SMT status                      = Capable Enabled BoundThreads
Error checking level            = Normal
Atomic RAD system detail level  = 2
Number of ticks since boot      = 34855934
```

ตัวจัดการโพรบ

ตัวจัดการโพรบไม่ใช่ส่วนหนึ่งของกรอบงานพื้นฐาน ProbeVue แต่เป็นคอมโพเนนต์การติดตามแบบไดนามิก ตัวจัดการโพรบคือผู้ให้บริการจุดโพรบที่สามารถสร้างวิธีการโดย ProbeVue

ตัวจัดการโพรบจะสนับสนุนชุดของจุดโพรบที่เป็นของโดเมนทั่วไป และแบ่งใช้คุณลักษณะหรือแอ็ตทริบิวต์ที่แยกออกจากจุดโพรบอื่น ตัวจัดการโพรบจะมีประโยชน์ที่จุดที่ควบคุมสายงานการเปลี่ยนแปลงที่สำคัญ จุดของการเปลี่ยนสถานะ และจุดของสิ่งที่น่าสนใจที่สำคัญอื่นๆ ตัวจัดการโพรบจะมีความระมัดระวัง ในการเลือกจุดโพรบที่อยู่ในตำแหน่งที่ปลอดภัย ต่ออุปกรณ์

ตัวจัดการโพรบสามารถเลือกเพื่อกำหนดกฎที่แยกความแตกต่างของตัวเองสำหรับ ข้อกำหนดคุณสมบัติภายในลักษณะทั่วไปที่ต้องปฏิบัติตาม สำหรับข้อกำหนดคุณสมบัติโพรบทั้งหมด

ProbeVue สนับสนุนตัวจัดการโพรบต่อไปนี้:

- การเรียกระบบตัวจัดการโพรบ
- ฟังก์ชันของผู้ใช้ตัวจัดการโพรบ
- ช่วงเวลาของตัวจัดการโพรบ
- การติดตามระบบตัวจัดการโพรบ
- ตัวจัดการโพรบการเรียกระบบที่ขยายเพิ่ม
- โปรแกรมจัดการโพรบ I/O
- โปรแกรมจัดการเครือข่ายโพรบ
- โปรแกรมจัดการโพรบ Sysproc

การเรียกระบบตัวจัดการโพรบ

ตัวจัดการโพรบ syscall สนับสนุนโพรบที่ทางเข้าและทางออก ของการเรียกระบบ AIX แบบอิงการนิยามหรืออิงเอกสาร ซึ่งเป็นการเรียกระบบ ที่มีอินเตอร์เฟสเดียวกันกับที่ **libc.a** (หรือ ไลบรารี C) entry point และในเคอร์เนล entry point การเรียกระบบ คือ pass-through (ไลบรารี C ที่อิมพอร์ตสัญลักษณ์จากเคอร์เนล และเอ็กซ์พอร์ตโดยไม่ใช่โค้ดในไลบรารี) หรือคือโค้ดที่ไม่สำคัญสำหรับอินเตอร์เฟสภายในไลบรารี

ตัวจัดการโพรบ syscall ยอมรับข้อกำหนดคุณสมบัติโพรบ 4-tuple ในหนึ่งในรูปแบบต่อไปนี้:

- ```
  syscall:*:<system_call_name>:entry
  ```
- ```
  syscall:*:<system_call_name>:exit
  ```

โดยที่ ฟิลด์ *system_call_name* จะถูกแทนที่ด้วยชื่อการเรียกระบบจริง ซึ่งจะบ่งชี้ว่า โพรบถูกวางอยู่ที่ทางเข้า และทางออกของการเรียกระบบ การกำหนด * ลงในฟิลด์ที่สองจะบ่งชี้ว่า โพรบจะถูกเรียกใช้งานสำหรับการประมวลผลทั้งหมด

หมายเหตุ: privilege ที่ต่างกันจำเป็นสำหรับ การเปิดใช้งานการเรียกระบบโพรบ การโพรบทุกๆ การประมวลผลในระบบต้องการ privilege ที่สูงกว่าการโพรบการประมวลผลที่คุณเป็นเจ้าของ

นอกจากนี้ ตัวจัดการโพรบ syscall ยังยอมรับข้อกำหนดคุณสมบัติโพรบ 4-tuple ในหนึ่งในรูปแบบต่อไปนี้:

- ```
  syscall:<process_ID>:<system_call_name>:entry
  ```
- ```
  syscall:<process_ID>:<system_call_name>:exit
  ```

โดยที่ ID การประมวลผลสามารถระบุในฟิลด์ที่สองของข้อกำหนดคุณสมบัติโพรบ เพื่อสนับสนุนการโพรบของการประมวลผลที่ระบุเฉพาะ

ชื่อการเรียกระบบจะถูกยอมรับโดยตัวจัดการโพรบ syscall ซึ่งเป็นชื่อของอินเตอร์เฟส *libc.a* และไม่ใช่ชื่อการเรียกระบบภายในของเคอร์เนล ตัวอย่างเช่น รูทีนย่อย **read** จะถูกเอ็กซ์พอร์ตโดย *libc.a* แต่ชื่อการเรียกระบบจริงหรือเคอร์เนลl entry point คือ **kread** ตัวจัดการโพรบ syscall จะแปลอินเตอร์เฟส *libc* ไปเป็น entry point เคอร์เนลและเปิดใช้งานโพรบที่ entry ในรูทีนเคอร์เนล **kread** ด้วยเหตุนี้ หากอินเตอร์เฟสไลบรารี C จำนวนมากเรียกใช้รูทีน **kread** จุดโพรบจะใช้สำหรับ อินเตอร์เฟสเหล่านั้นด้วยเช่นกัน โดยทั่วไป นี่ไม่ใช่ปัญหาเนื่องจากการเรียกระบบส่วนใหญ่จะสนับสนุนโดยตัวจัดการโพรบ syscall ซึ่งมีการแม็พแบบ 1 ต่อ 1 ระหว่างอินเตอร์เฟส *libc* และรูทีนเคอร์เนล

สำหรับโพรบ syscall แต่ละตัว มีจุดโพรบที่เทียบเท่ากันในโค้ดไลบรารี ที่จัดเตรียมโดยตัวจัดการโพรบ uft ตัวจัดการโพรบ uft ไม่สนับสนุนอินเตอร์เฟสไลบรารีทั้งหมด (ยกเว้นอินเตอร์เฟสแบบ passthrough และไม่มีโค้ดสำหรับการเรียกหรือการอ้างอิงกับไลบรารี ทั้งหมด) ซึ่งประกอบด้วยไลบรารีที่ไม่ได้รับการสนับสนุนโดยตัวจัดการโพรบ syscall อย่างไรก็ตาม ตัวจัดการโพรบ syscall มีข้อดีอยู่สองข้อ:

- ตัวจัดการโพรบ syscall สามารถโพรบการประมวลผลในระบบ โดยระบุเครื่องหมายดอกจันในฟิลด์ที่สอง
- ตัวจัดการโพรบ syscall จะมีประสิทธิภาพมากกว่าตัวจัดการโพรบ uft เนื่องจากไม่ต้องการสับเปลี่ยนจากโหมดผู้ใช้ไปเป็นโหมดเคอร์เนล และกลับสู่การรันการดำเนินการโพรบ

สำหรับข้อมูลเพิ่มเติมเกี่ยวกับรายการการเรียกใช้คำสั่งระบบทั้งหมดที่สนับสนุน โดย syscall probe manager โปรดดูที่ ProbeVue

ตัวจัดการโพรบ UFT

uft หรือตัวจัดการโพรบสำหรับการติดตามฟังก์ชันของผู้ใช้สนับสนุนการโพรบฟังก์ชันพื้นที่ผู้ใช้ ที่มองเห็นได้ในตารางสัญลักษณ์ XCOFF ของการประมวลผล ตัวจัดการโพรบ uft สนับสนุนจุดโพรบที่อยู่ที่จุดเข้าและออกของฟังก์ชันที่ซอร์สเป็นไฟล์ข้อความภาษา C หรือ FORTRAN แม้ว่าตารางสัญลักษณ์สามารถมีซอร์สที่มาจากภาษาอื่นที่ไม่ใช่ภาษา C หรือ FORTRAN

การติดตามแอ็พพลิเคชัน Java™ ในวิธีที่เหมือนกับกลไกการติดตามที่มีอยู่จากมุมมองของผู้ใช้ และ JVM เป็นการติดตามที่ทำงานจริงๆ ส่วนใหญ่แทน Probevue

สำหรับการโพรบแอ็พพลิเคชัน java โปรดดูที่ "Java Applications Probe Manager" ด้านล่าง

ตัวจัดการโพรบ uft ยอมรับข้อกำหนดคุณลักษณ์ 5-tuple โพรบ ในรูปแบบต่อไปนี้:

```
uft:<processID>:*:<function_name>:<entry|exit>
```

หมายเหตุ: ตัวจัดการโพรบ uft ต้องการ ID กระบวนการเพื่อให้สามารถติดตามกระบวนการและชื่อของฟังก์ชันที่สมบูรณ์ที่เป็นจุดเข้าหรือออกที่จะใช้โพรบ

เมื่อฟิลด์ที่สามถูกตั้งค่าเป็น * ตัวจัดการโพรบ UFT จะค้นหาฟังก์ชันในโมดูลทั้หมดที่ถูกโหลดลงในพื้นที่แอดเดรสของกระบวนการรวมถึงไฟล์ที่สามารถเรียกทำงานหลักและโมดูลแบบแบ่งใช้ ซึ่งหมายความว่า ถ้าโปรแกรมมีฟังก์ชัน C มากกว่าหนึ่งฟังก์ชัน ที่มีชื่อนี้ (ตัวอย่างเช่น ฟังก์ชันที่มีคลาสแบบสแตติกที่อยู่ในโมดูลอ็อบเจ็กต์อื่น) โพรบจะถูกนำมาใช้กับ entry point ของหนึ่งในฟังก์ชันเหล่านี้ทุกฟังก์ชัน

ถ้าต้องการโพรบชื่อฟังก์ชันในโมดูลที่ระบุ ต้องระบุชื่อโมดูลในฟิลด์ที่สาม ไวยากรณ์สำหรับโพรบสำหรับระบุชื่อโมดูลไลบรารีจะแสดงด้านล่าง:

```
# Function foo in any module
@@uft:<pid>:*:foo:entry
# Function foo in any module in any archive named libc.a
@@uft:<pid>:libc.a:foo:entry
# Function foo in the shr.o module in any archive named libc.a
@@uft:<pid>:libc.a(shr.o):foo:entry
```

ชื่อฟังก์ชันใน tuple ที่สี่สามารถระบุเป็น Extended Regular Expression (ERE) ERE ควรอยู่ระหว่าง ″/ และ /″ เช่น ″/<**ERE**>/″

เมื่อระบุชื่อฟังก์ชันเป็น ERE ฟังก์ชันทั้งหมดที่ตรงกับนิพจน์ธรรมดาที่ระบุในโมดูลที่ระบุจะถูกโพรบ

```
 /* Probe entry of all libc.a functions starting with “malloc” word */
@@uft:$__CPID:libc.a: “/^malloc.*/”:entry
/* Probe exit of all functions in the executable a.out */
@@uft:$__CPID:a.out:”/.*/”:exit
```

ในโพรบเข้า ที่ชื่อฟังก์ชันถูกระบุเป็นนิพจน์ธรรมดา จะไม่สามารถเข้าถึงแต่ละอาร์กิวเมนต์ได้ อย่างไรก็ตาม สามารถใช้ฟังก์ชัน probevue function **print_args** เพื่อพิมพ์ชื่อฟังก์ชันและอาร์กิวเมนต์ของฟังก์ชันได้ ค่าของอาร์กิวเมนต์จะถูกพิมพ์ตามข้อมูลชนิดของอาร์กิวเมนต์ที่มีในตาราง traceback ของงฟังก์ชัน

ในโพรบออก ที่ชื่อฟังก์ชันถูกระบุเป็นนิพจน์ธรรมดา จะไม่สามารถเข้าถึงค่าที่ส่งคืนได้

Probevue สนับสนุนการเปิดใช้งานโพรบในมากกว่าหนึ่งกระบวนการในเวลาเดียวกัน อย่างไรก็ตาม คุณอาจต้องการ privilege สำหรับการโพรบ การประมวลผลที่เป็นของคุณ

Probevue จะบังคับใช้ข้อจำกัดที่ป้องกันไม่ให้กระบวนการที่มีโพรบของพื้นที่ผู้ใช้ไม่ให้ถูกดีบักโดยใช้ API ที่ใช้ **ptrace** หรือ **procfs**

ตามที่ได้กล่าวไว้ข้างต้น ตัวจัดการโพรบ uft สนับสนุนโพรบที่แบ่งใช้โมดูล ซึ่งเหมือนกับโมดูลไลบรารีที่แบ่งใช้ ต่อไปนี้คือสคริปต์ ที่แสดงตัวอย่างที่กิจกรรมการติดตาม mutex โดยเปิดใช้งานโพรบ ในล็อก mutex ของไลบรารีเธรดและปลดล็อกรูทีนย่อย

```
/* pthreadlocks.e */
/* Trace pthread mutex activity for a given multithreaded process */
/* The following defines are from /usr/include/sys/types.h */
```

```
typedef long long pid_t;
typedef long long thread_t;

typedef struct {
 int __pt_mutexattr_status;
 int __pt_mutexattr_pshared;
 int __pt_mutexattr_type;
} pthread_mutexattr_t;

typedef struct __thrq_elt thrq_elt_t;

struct __thrq_elt {
 thrq_elt_t *__thrq_next;
 thrq_elt_t *__thrq_prev;
};

typedef volatile unsigned char _simplelock_t;

typedef struct __lwp_mutex {
 char  __wanted;
 _simplelock_t __lock;
} lwp_mutex_t;

typedef struct {
 lwp_mutex_t  __m_lmutex;
 lwp_mutex_t  __m_sync_lock;
 int   __m_type;
 thrq_elt_t  __m_sleepq;
 int   __filler[2];
} mutex_t;

typedef struct {
 mutex_t   __pt_mutex_mutex;
 pid_t   __pt_mutex_pid;
 thread_t  __pt_mutex_owner;
 int   __pt_mutex_depth;
 pthread_mutexattr_t __pt_mutex_attr;
} pthread_mutex_t;

int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);

@@uft:$__CPID:*:pthread_mutex_lock:entry
{
 printf("thread %d: mutex 0x%08x locked\n", __tid, __arg1);
}

@@uft:$__CPID:*:pthread_mutex_unlock:entry
{
 printf("thread %d: mutex 0x%08x unlocked\n", __tid, __arg1);
}
```

ข้อกำหนดคุณลักษณะของโพรบ การเข้าถึงอาร์กิวเมนต์ และการใช้ฟังก์ชัน ProbeVue ในการดำเนินการโพรบสำหรับโพรบฟังก์ชัน Fortran จะเหมือนกับโพรบ uft โดยมีข้อแตดต่างต่อไปนี้:

- ผู้ใช้ต้องแม็พชนิดของมูล Fortran กับชนิดข้อมูล ProbeVue และใช้ที่เหมือนกันในสคริปต์ การแม็พชนิดข้อมูล Fortran พื้นฐานกับชนิดข้อมูล ProbeVue จะแสดงรายการในตารางด้านล่าง

ตารางที่ 21. การแม็พชนิดข้อมูล Fortran กับ ProveVue.

ชนิดข้อมูล Fortran	ชนิดข้อมูล ProbeVue
INTEGER * 2	short
INTEGER * 4	int/long
INTEGER * 8	long long
REAL	float
DOUBLE PRECISION	double

COMPLEX	ไม่มีชนิดข้อมูลพื้นฐานที่ใช้แทนได้ ซึ่งจำเป็นต้องถูกแม็พกับโครงสร้างดังแสดงด้านล่าง:
	`typedef struct complex {`
	`float a;`
	`float b;`
	`} COMPLEX;`

LOGICAL	int (มาตรฐาน Fortran ต้องการให้ตัวแปรแบบโลจิคัลให้มีขนาดเดียวกันกับตัวแปร INTEGER/REAL)
CHARACTER	char
BYTE	signed char

- Fortran จะผ่านอาร์กิวเมนต์ IN scalar ของโพรซีเดอร์ภายในโดยใช้ค่า และอาร์กิวเมนต์อื่นโดยใช้การอ้างอิง อาร์กิวเมนต์ที่ถูกผ่านโดยการอ้างอิงควรสามารถเข้าถึงได้ด้วย copy_userdata() ข้อมูลเพิ่มเติมเกี่ยวกับการเชื่อมโยงอาร์กิวเมนต์ใน fortran สามารถค้นหาได้ในหัวข้อ การเชื่อมโยงอาร์กิวเมนต์
- ชื่อรูทีนในโปรแกรม Fortran จะคำนึงถึงขนาดตัวพิมพ์ แต่ขณะที่ระบุในสคริปต์ ProbeVue ชื่อรูทีนควรเป็นตัวพิมพ์เล็ก

ตัวอย่างสคริปต์ต่อไปนี้แสดงวิธีแม็พชนิดข้อมูล Fortran กับชนิดข้อมูล ProbeVue:

```
/* cmp_calc.e */
/* Trace fortran routines
cmp_calc(COMPLEX, INTEGER) and
cmplxd(void) */

typedef struct complex{
        float a;
        float b;
        } COMPLEX;

typedef int INTEGER;

/* arguments are indicated to be of pointer type as they are passed by reference */
void cmp_calc(COMPLEX *, INTEGER *);
void cmplxd();

@@uft:$__CPID:*:cmplxd:entry
{
printf("In cmplxd entry \n");
}

@@uft:$__CPID:*:cmp_calc:entry
{
```

```
COMPLEX c;
int i;
copy_userdata(__arg1, c);
copy_userdata(__arg2, i);
printf("%10.7f+j%9.7f  %d \n", c.a,c.b,i);
}
```

- Fortran จะเก็บอาร์เรย์ในรูปแบบ column-major โดยที่ ProbeVue จะเก็บในรูปแบบ และสคริปต์ด้านล่างจะแสดงวิธีที่ผู้ใช้สามารถ ดึงอิลิเมนต์ของอาร์เรย์

```
/* array.e*/
/* ProbeVue script to probe fortran program array.f */

void displayarray(int **, int, int);
@@uft:$__CPID:*:displayarray:entry
{
int a[5][4];  /* row and column sizes are interchanged */
copy_userdata(__arg1, a);
/* to print the first row */
printf("%d %d %d \n", a[0][0], a[1][0], a[2][0]);
/* to print the second row */
printf("%d %d %d\n", a[0][1], a[1][1], a[2][1]);
}

/* Fortran program array.f */

PROGRAM ARRAY_PGM
IMPLICIT NONE
INTEGER, DIMENSION(1:4,1:5) :: Array
INTEGER :: RowSize, ColumnSize
CALL ReadArray(Array, RowSize, ColumnSize)
CALL DisplayArray(Array, RowSize, ColumnSize)
CONTAINS
SUBROUTINE ReadArray(Array, Rows, Columns)
IMPLICIT NONE
INTEGER, DIMENSION(1:,1:), INTENT(OUT) :: Array
INTEGER, INTENT(OUT) :: Rows, Columns
INTEGER :: i, j
READ(*,*) Rows, Columns
DO i = 1, Rows
READ(*,*) (Array(i,j), j=1, Columns)
END DO
END SUBROUTINE ReadArray
SUBROUTINE DisplayArray(Array, Rows, Columns)
IMPLICIT NONE
INTEGER, DIMENSION(1:,1:), INTENT(IN) :: Array
INTEGER, INTENT(IN) :: Rows, Columns
INTEGER :: i, j
DO i = 1, Rows
WRITE(*,*) (Array(i,j), j=1, Columns )
END DO
END SUBROUTINE DisplayArray
END PROGRAM ARRAY_PGM
```

- ฟังก์ชัน Intrinsic หรือ built-in สามารถมารถโพรบด้วย ProbeVue รูทีน FORTRAN ทั้งหมดที่แสดงรายการในตารางสัญลักษณ์ XCOFF ของไลบรารี executable/linked สามารถโพรบได้ ProbeVue จะใช้ตารางสัญลักษณ์ XCOFF เพื่อระบุ

ตำแหน่งของรูทีนเหล่านี้ อย่างไรก็ตาม ผู้ใช้ต้องจัดเตรียมต้นแบบสำหรับรูทีนและ ProbeVue พยายามเข้าถึงอาร์กิวเมนต์ตามที่เต้นแบบจัดเตรียมให้ สำหรับรูทีนที่คอมไพลเลอร์ตัดชื่อรูทีน ควรมีการระบุชื่อที่ถูกตัด เนื่องจาก Vue เป็บภาษาในลักษณะของภาษา C ผู้ใช้ควรแน่ใจว่าต้นแบบฟังก์ชัน/รูทีนย่อยภาษา FORTRAN ถูกแม็พกับต้นแบบฟังก์ชันที่เป็นลักษณะภาษา C อย่างเหมาะสม โปรดอ้างถึงหลักการของการลิงก์สำหรับการส่งผ่านอาร์กิวเมนต์และค่าส่งคืนของฟังก์ชันในหัวข้อ การส่งผ่าน ข้อมูลจากภาษาหนึ่งไปยังอีกภาษาหนึ่ง ตัวอย่างด้านล่างจะแสดงสิ่งที่กล่าวถึงนี้:

```
/* Fortran program ext_op.f */
/* Operator "*" is extended for rational multiplication */
MODULE rational_arithmetic
IMPLICIT NONE
        TYPE RATNUM
                INTEGER :: num, den
        END TYPE RATNUM
        INTERFACE OPERATOR (*)
                MODULE PROCEDURE rat_rat, int_rat, rat_int
        END INTERFACE
        CONTAINS
        FUNCTION rat_rat(l,r)      ! rat * rat
                TYPE(RATNUM), INTENT(IN) :: l,r
                TYPE(RATNUM) :: val,rat_rat
                val.num=l.num*r.num
                val.den=l.den*r.den
                rat_rat=val
        END FUNCTION rat_rat
        FUNCTION int_rat(l,r)      ! int * rat
                INTEGER, INTENT(IN)      :: l
                TYPE(RATNUM), INTENT(IN) :: r
                TYPE(RATNUM) :: val,int_rat
                val.num=l*r.num
                val.den=r.den
                int_rat=val
        END FUNCTION int_rat
        FUNCTION rat_int(l,r)      ! rat * int
                TYPE(RATNUM), INTENT(IN) :: l
                INTEGER, INTENT(IN)      :: r
                TYPE(RATNUM) :: val,rat_int
                val.num=l.num*r
                val.den=l.den
                rat_int=val
        END FUNCTION rat_int
END MODULE rational_arithmetic
PROGRAM Main1
Use rational_arithmetic
IMPLICIT NONE
 TYPE(RATNUM) :: l,r,l1
 l.num=10
        l.den=11
        r.num=3
        r.den=4
        L1=l*r
END PROGRAM Main1

/* ext_op.e */
/* ProbeVue script to probe routine that gets called when "*"
    is used to multiply rational numbers in ext_op.f */

struct rat
{
        int num;
```

```
        int den;
};
struct rat rat;
void __rational_arithmetic_NMOD_rat_rat(struct rat*,
 struct rat*,struct rat*);
/* Note that the mangled function name is provided. */
/* Also, the structure to be returned is sent in the buffer whose address is provided as the first argument. */
/* The first explicit parameter is in the second argument. */
@@BEGIN
{
        struct rat* rat3;
}
@@uft:$__CPID:*:__rational_arithmetic_NMOD_rat_rat:entry
{
 struct rat rat1,rat2;
        copy_userdata((struct rat *)__arg2,rat1);
        copy_userdata((struct rat *)__arg3,rat2);
        rat3=__arg1;
 /* The address of the buffer where the returned structure will be stored is saved at the function entry */
        printf("Argument Passed rat_rat = %d:%d,%d:%d\n",rat1.num,rat1.den,rat2.num,rat2.den);
}
@@uft:$__CPID:*:__rational_arithmetic_NMOD_rat_rat:exit
{
        struct rat rrat;
        copy_userdata((struct rat *)rat3,rrat);
        /* The saved buffer address is used to fetch the returned structure */
        printf("Return from rat_rat = %d:%d\n",rrat.num,rrat.den);
        exit();
}
```

- ProbeVue ไม่สนับสนุนการรวมไฟล์ส่วนหัวของ Fortran โดยตรงในสคริปต์ อย่างไรก็ตาม การแม็พชนิดข้อมูล Fortran กับชนิดข้อมูล ProbeVue สามารถถูกจัดเตรียมในไฟล์ส่วนหัว ProbeVue และถูกรวมกับอ็อพชัน "-I''

ตัวจัดการโพรบแอ็พพลิเคชัน C++

ตัวจัดการโพรบ C++ สนับสนุนการโพรบของแอ็พพลิเคชัน C++ ในวิธีที่เหมือนกับตัวจัดการโพรบ C สนับสนุนโพรบ entry/exit แบบ "uft" บนฟังก์ชัน C++ รวมถึงฟังก์ชัน member, overloaded, operator และ template ในไฟล์ที่สามารถเรียกทำงานได้ ที่สำคัญ ฟังก์ชันโพรบ entry/exit ใน C++ ต้องใช้ตัวจัดการโพรบ **@@uftxlc++**

tuples ทั้งหมดในข้อกำหนดคุณลักษณะโพรบลักษณะ **@@uftxlc++** จะมีการใช้งานที่เหมือนกัน และมีรูปแบบเป็นสตริงโพรบลักษณะ **@@uft** โดยมีข้อยกเว้นคือชื่อฟังก์ชัน เนื่องจาก C++ อนุญาตให้ชื่อฟังก์ชันเดียวสามารถโอเวอร์โหลดได้ ชื่อฟังก์ชันที่ระบุในสตริงโพรบอาจต้องมีชนิดอาร์กิวเมนต์ของฟังก์ชันเพื่อระบุฟังก์ชันที่จะถูกโพรบโดยเฉพาะ

ตัวอย่างเช่น:

```
@@uftxlc++:12345:*:"foobar(int, char *)":entry
```

หมายเหตุ: ชนิดที่ส่งคืนจะหายไปากสตริงโพรบข้างบนเนื่องจากไม่ได้เป็นส่วนในชื่ออัลกอริทึม mangling สำหรับฟังก์ชันธรรมดา ในกรณีของฟังก์ชันเท็มเพลต ผู้ใช้ต้องระบุอินสแตนท์เท็มเพลตที่ต้องการโพรบอย่างชัดเจน และยังต้องระบุชนิดที่ส่งคืนของอินสแตนท์เท็มเพลต:

```
@@uftxlc++:12345:*:void foobar<int>(int, char *):entry
```

หมายเหตุ: สตริงโพรบต้องใช้เครื่องหมายคำพูดล้อมรอบชื่อฟังก์ชัน ดังที่ระบุในตัวอย่างสองตัวอย่างข้างบน และคำสั่ง probevue จะแสดงข้อผิดพลาด ถ้าเครื่องหมายคำพูดหายไป เครื่องหมายคำพูดจะไม่ใช่เนื่องจากโคลอน ":" เท่านั้นแต่ยังเนื่อง

จากเครื่องหมายคอมมา ″,″ ตัวดำเนินการเครื่องหมายคอมมาจะถูกใช้เพื่อแยกโพรบหลายโพรบบนบรรทัดเดียวกันและโดยไม่มีเครื่องหมายคำพูดนำหน้า ซึ่งจะแสดงข้อความแสดงข้อผิดพลาดที่แปลงสำหรับผู้ใช้

เมื่อโพรบฟังก์ชันสมาชิกคลาส หรือฟังก์ชันที่กำหนดในเนมสเปซ ต้องใช้ชื่อฟังก์ชันที่ได้รับการรับรองในสตริงโพรบ เพื่อหลีกเลี่ยงความสับสนระหว่างเครื่องหมายโคลอนเดี่ยว (:) ตัวแยก tuple ในสตริงโพรบและตัวดำเนินการแก้ไขขอบเขตเครื่องหมายโคลอนคู่ (::) ในชื่อ C++ ที่ได้รับการรับรอง ชื่อขอบเขตทั้งหมดที่ tuple ในสตริงโพรบต้องอยู่ในเครื่องหมายคำพูด

```
@@uftxlc++:12345:*:"Foo::bar(int)":entry
```

Limitations:

1. Access to data fields that are inherited from a virtual base class is not supported.
2. Template classes are not supported and must not be included in the C++ header.
3. Pointers to members are not supported.
4. To probe a class with the class definition, an object of the class is instantiated in the header file either as a global object or in a dummy function.

ตัวอย่าง:

ด้านล่างเป็นแอ็พพลิเคชัน c++

```
#include "header.cc"
main()
{
int             i = 10;
incr_num(i);
float           a = 3.14;
incr_num(a);
char            ch = 'A';
incr_num(ch);
double          d = 1.11;
incr_num(d);
}
```

Content of the ″header.cc″

```
# cat header.cc
#include <iostream.h>
template <class T>
T incr_num( T a)
{
return (++a);
}
int dummy()
{
int  i=10,j=20;
incr_num(i);
float a=3.14;
incr_num(a);
char  ch ='A',dh='Z';
incr_num(ch);
```

```
double d=1.1,e=1.11;
incr_num(d);
return  0;
}
```

เนื่อหาของสคริปต์ Vue vue_cpp.e

```
##C++
#include "header.cc"
##Vue
@@uftxlc++:$__CPID:*:"incr_num<int>(int)":entry
{
printf("Hello1_%d\n",__arg1 );
}
@@uftxlc++:$__CPID:*:"incr_num <  float   > (float)" :entry
{
printf("Hello2_%f\n",__arg1 );
}
@@uftxlc++:$__CPID:*:"incr_num <  char    > ( char )":entry
{
printf("Hello3_%c\n",__arg1 );
}
@@uftxlc++:$__CPID:*:"incr_num <  double    > ( double )":entry
{
printf("Hello4_%lf\n",__arg1 );
exit();
}
```

Execution :

```
/usr/vacpp/bin/xlC  app.c++
#  probevue  -X ./a.out  vue_cpp.e
Hello1_10
Hello2_3.140000
Hello3_A
Hello4_1.110000
```

ต้นแบบของฟังก์ชันใน tuple ที่สี่สามารถระบุเป็น Extended Regular Expression (ERE) ERE ควรอยู่ระหว่าง '"/' และ '/"' เช่น "/<**ERE**>/" เมื่อระบุต้นแบบของฟังก์ชันเป็น ERE ฟังก์ชันทั้งหมดที่ตรงกับนิพจน์ธรรมดาที่ระบุในโมดูลที่ระบุจะถูกโพรบ

```
/* Probe entry of all the C++ functions in the executable a.out */
@@uftxlc++:$__CPID:a.out:"/.*/":entry
/* Probe exit of all the C++ functions with 'foo' word in it */
@@uftxlc++:$__CPID:*:"/foo/":exit
```

ในโพรบเข้า ที่ชื่อฟังก์ชันถูกระบุเป็นนิพจน์ธรรมดา จะไม่สามารถเข้าถึงแต่ละอาร์กิวเมนต์ได้ อย่างไรก็ตาม สามารถใช้ฟังก์ชัน probevue **print_args ( )** เพื่อพิมพ์ชื่อฟังก์ชันและอาร์กิวเมนต์ของฟังก์ชันได้ ค่าของอาร์กิวเมนต์จะถูกพิมพ์ตามข้อมูลชนิดของอาร์กิวเมนต์ที่มีในตาราง traceback ของงฟังก์ชัน

ในโพรบออก ที่ชื่อฟังก์ชันถูกระบุเป็นนิพจน์ธรรมดา จะไม่สามารถเข้าถึงค่าที่ส่งคืนได้

ตัวจัดการโพรบแอ็พพลิเคชัน Java

Java Probe Manager (JPM) สนับสนุนการโพรบแอ็พพลิเคชัน Java ในวิธีเดียวกับตัวจัดการการโพรบ C และ C++ สคริปต์ Vue สคริปต์เดียวควรสามารถติดตามแอ็พพลิเคชัน java ได้หลายแอ็พพลิเคชันพร้อมกัน โดยใช้ ID กระบวนการที่ต่างกันของ JVMs สคริปต์เดียวกันสามารถใช้เพื่อโพรบ syscalls หรือแอ็พพลิเคชัน C/C++ และแอ็พพลิเคชัน Java และสามารถใช้ตัวจัดการโพรบอื่นได้

เช่นนเดียวกับ uft (การติดตามฟังก์ชันผู้ใช้) ตัวจัดการโพรบ ตัวจัดการโพรบ java ยังยอมรับข้อกำหนดคุณลักษณะโพรบ 5-tuple ในรูปแบบต่อไปนี้:

```
uftjava :< process_ID> :*:< _qualified_function_name >: entry
```

โดยที่ tuple ที่สองเป็น ID กระบวนการของกระบวนการ JVM ที่สอดคล้องกับแอ็พพลิเคชัน Java ที่ถูกโพรบ

ฟิลด์ที่สาม: ถูกสงวนไว้สำหรับในอนาคต

ฟิลด์ที่สี่: เป็นฟิลด์ที่ต้องระบุเมธอด java

ชื่อนี้เป็นชื่อที่ได้รับการรับรองที่ใช้ในแอ็พพลิเคชัน java เช่น Mypackage.Myclass.Mymethod

ข้อกำจัดบางรายการที่อาจนำมาใช้คือ

- เฉพาะเมธอด pure java เท่านั้นที่สามารถโพรบได้ โค้ดแบบดั้งเดิม (การเรียกใช้ไลบรารีแบบแบ่งใช้) หรือโค้ดที่เข้ารหัสไม่สามารถติดตามได้
- สนับสนุนเฉพาะโพรบ entry
- สามารถสนับสนุนเฉพาะ JVM v 1.5 และสูงกว่าที่สนับสนุนอินเตอร์เฟส JVMTI
- ขณะเวลาใดเวลาหนึ่ง จะไม่มีเซสชัน Probevue สองเซสชันที่สามารถโพรบแอ็พพลิเคชัน Java เดียวกันที่มี @@uftjava
- ไม่สนับสนุนเมธอด Polymorphic/Overloaded
- ไม่สนับสนุนการติดตาม/การเข้าถึงตัวแปรภายนอกที่มีชื่อเดียวกันกับคีย์เวิร์ด Probevue ใดๆ หรือชื่อแบบ built-in ซึ่งต้องมีการเปลี่ยนชื่อสัญลักษณ์ภายนอกเหล่านั้น (ชื่อตัวแปรแอ็พพลิเคชัน Java)
- การเข้าถึงอาร์เรย์ของแอ็พพลิเคชัน java ไม่ได้รับการสนับสนุนใน รีลีสนี้
- การเข้าถึงอาร์เรย์ของแอ็พพลิเคชัน java ไม่ได้รับการสนับสนุนใน รีลีสนี้
- ไม่สนับสนุน get_function () แบบ built-in สำหรับภาษา java ในรีลีสนี้

หมายเหตุ: ในกรณีของการติดตามเมธอดแบบไม่ใช่สแตติก หมายเลขอาร์กิวเมนต์จะเริ่มต้นด้วย __arg2 เช่นเดียวกับเมธอดแบบไม่ใช้สแตติกของ C++ __arg1 ถูกใช้เพื่ออ้างถึงตัวเอง (ตัวชี้นี้)

การเข้าถึงข้อมูล: บล็อกการดำเนินการของโพรบ java สามารถเข้าถึงข้อมูลที่มเหมือนกับลักษณะการทำงานที่มีอยู่ต่อไปนี้

- บล็อกแอ็คชันสามารถเข้าถึงตัวแปรสคริปต์แบบ global, local และ kernel
- บล็อกแอ็คชันสามารถเข้าถึงอาร์กิวเมนต์เมธอด (ตัวแปรคลาส Entry) ของชนิดดั้งเดิม
- บล็อกการดำเนินการสามารถเข้าถึงตัวแปรแบบ built-in
- บล็อกแอ็คชันสามารถเข้าถึงตัวแปรแอ็พพลิเคชัน Java โดยใช้ชื่อที่ได้รับการรับรอง สแตติกเท่านั้น (สมาชิกคลาส)

```
x = some_package.app.class.var_x;    //Access static/class member.
```

- สนับสนุนการเข้าถึงตัวแปรชนิดดั้งเดิมของแอ็พพลิเคชัน java ซึ่งต้องเป็นแบบ converted/promoted/casted แบบไม่ชัดเจนโดยค่าไม่สูญหายกับชนิดที่ใช้แทนได้ในภาษา Vue แต่การใช้หน่วยความจำจริงๆ (ขนาด) อาจแตกต่างจากการใช้ในภาษา Java

ฟังก์ชันที่สนับสนุนในบริบทของตัวจัดการโพรบ Java ถูกแสดงอยู่ในตารางต่อไปนี้:

ตารางที่ 22. ฟังก์ชันที่สนับสนุนโดยตัวจัดการโพรบ Java.

ฟังก์ชัน	คำอธิบาย
stktrace()	จัดเตรียมการติดตามสแต็กของแอ็พพลิเคชัน Java (เธรดที่รันอยู่) ที่ถูกติดตาม
copy_userdata()	คัดลอกข้อมูลจากแอ็พพลิเคชัน java ไปยังตัวแปรสคริปต์
get_probe()	ส่งคืนสตริงโพรบ
get_stktrace	คืนค่าการติดตามรันไทม์สแต็ก
get_location_point()	ส่งคืนตำแหน่งโพรบปัจจุบัน
get_userstring()	คัดลอกข้อมูลสตริงจากแอ็พพลิเคชัน java
exit()	ออกจากเซสชันการติดตาม probevue

การเปลี่ยนแปลงกับคำสั่ง Probevue:

ตารางที่ 23. การเปลี่ยนคำสั่ง probevue.

คำสั่ง	คำอธิบาย
อ็อพชัน -X	อ็อพชันนี้สามารถใช้ (พร้อมกับอ็อพชัน -A) เพื่อเรียกทำงานแอ็พพลิเคชัน Java ในรีลีสปัจจุบันที่ผู้ใช้ต้องส่งผ่านสตริงตัวเลือกเพิ่มเติมแบบแมนวล agentlib:probevuejava พร้อมกับอ็อพชันอื่นๆ ทั้งหมดที่จำเป็น ต้องรันแอ็พพลิเคชัน java

ตัวอย่างเช่น:

```
probevue -X /usr/java5/bin/java -A  -agentlib:probevuejava myjavaapp  myscript.e
```

เมื่อรัน JVM 64 เราต้องใช้ "agentlib:probevuejava64" เช่นเดียวกับใน:

```
probevue -X /usr/java5_64/bin/java -A  -agentlib:probevuejava64 myjavaapp  myscript.e
โดยที่ myjavaapp เป็นคลาส java ของแอ็พพลิเคชัน myjavaapp.java
```

ตัวอย่างซอร์ส ExtendedClass.java:

```
class BaseClass
{
        static int i=10;

        public static void test(int x)
        {
                i += x;
        }
}

public class ExtendedClass extends BaseClass
{
        public static void test(int x, String msg)
        {
                i += x;
                System.out.print("Java: " + msg + "\n\n");
                BaseClass.test(x);
        }
```

```
        public static void main(String[] args)
        {
                BaseClass.test(5);
                ExtendedClass.test(10, "hello");
        }
}
```

ตัวอย่างสคริปต์ test.e สำหรับแอ็พพลิเคชัน Java ด้านบน:

```
@@uftjava:$__CPID:*:"BaseClass.test":entry
{
        printf("BaseClass.i: %d\n", BaseClass.i);
        printf("BaseClass.test: %d\n", __arg1);
        stktrace(0, -1);
        printf("\n");
}

@@uftjava:$__CPID:*:"ExtendedClass.test":entry
{
        printf("BaseClass.i: %d\n", BaseClass.i);
        printf("ExtendedClass.test: %d, %s\n", __arg1, __arg2);
        stktrace(0, -1);
        printf("\n");
}
```

ตัวอย่างเซสชัน ProbeVue ที่มีสคริปต์ด้านบน:

```
# probevue -X /usr/java5/jre/bin/java \
-A "-agentlib:probevuejava ExtendedClass" test.e
Java: hello

BaseClass.i: 10
BaseClass.test: 5
BaseClass.test()+0
ExtendedClass.main()+1

BaseClass.i: 15
ExtendedClass.test: 10, hello
ExtendedClass.test()+0
ExtendedClass.main()+8

BaseClass.i: 25
BaseClass.test: 10
BaseClass.test()+0
ExtendedClass.test()+39
ExtendedClass.main()+8
```

ช่วงเวลาของตัวจัดการโพรบ

ตัวจัดการช่วงเวลาของโพรบจะจัดเตรียมจุดโพรบที่เรียกใช้ที่ช่วงเวลา ที่กำหนดโดยผู้ใช้ จุดโพรบไม่ได้อยู่ในเคอร์เนล หรือโค้ดแอ็พพลิเคชัน แต่จะอ้างอิงตามช่วงเวลาของนาฬิกา แบบอิงเหตุการณ์โพรบแทน

ตัวจัดการช่วงเวลาโพรบมีประโยชน์สำหรับการสรุปสถิติที่เก็บรวบรวมผ่าน ช่วงเวลา ซึ่งจะยอมรับข้อกำหนดคุณสมบัติแบบ 4-tuple โพรบ ในรูปแบบดังต่อไปนี้:

```
@@interval:*:clock:<# milliseconds>
```

ตัวจัดการโพรบ interval จะกรองเหตุการณ์โดยใช้ ID กระบวนการ ถ้ามีการระบุในฟิลด์ที่สอง การกำหนด * ลงในฟิลด์ที่สองจะบ่งชี้ว่า โพรบจะถูกเรียกใช้งานสำหรับการประมวลผลทั้งหมด ยิ่งไปกว่านั้น เฉพาะค่าที่สนับสนุนโดยตัวจัดการช่วงเวลาของโพรบสำหรับฟิลด์ที่สาม คือคีย์เวิร์ดนาฬิกาที่ระบุข้อกำหนดคุณสมบัติโพรบ ที่เป็นอยู่สำหรับโพรบนาฬิกา ฟิลด์ที่สี่หรือฟิลด์สุดท้าย คือ ฟิลด์ <# milliseconds> จะระบุจำนวนมิลลิวินาที ระหว่าง firings ของโพรบ ตัวจัดการโพรบ interval ต้องการให้ค่าสำหรับฟิลด์ประกอบด้วยเฉพาะตัวเลข 0-9 สำหรับโพรบ interval ที่ไม่มี id กระบวนการ intervals ควรถูกหารด้วย 100 ดังนั้น เหตุการณ์โพรบที่แยกเป็น 100ms, 200ms, 300ms และอื่นๆ อนุญาตให้ใช้ในโพรบ interval แบบ non-profiling สำหรับโพรบ interval ที่มี ID กระบวนการ intervals ควรมากกว่าหรือเท่ากับ interval ที่น้อยที่สุดที่อนุญาตสำหรับผู้ใช้รูปแบบโกลบอล หรือหารด้วย 100 สำหรับผู้ใช้อื่น ดังนั้น เหตุการณ์โพรบที่แยกเป็น 10ms, 20ms, 30ms และอื่นๆ จะได้รับอนุญาตสำหรับผู้ใช้ธรรมดาในโพรบ interval แบบ profiling จะมีเพียงโพรบ interval แบบ profiling เดียวเท่านั้นที่แอ็คทีฟสำหรับกระบวนการ

หมายเหตุ: ตัวจัดการช่วงของโพรบไม่ได้รับประกันว่า โพรบจะเรียกใช้จำนวนมิลลิวินาทีโดยไม่พิจารณาการบ่งชี้ ค่าของฟิลด์ที่สี่ ลำดับความสำคัญที่สูงกว่าของการอินเตอร์รัปต์ และโค้ดที่รันหลังจากปิดใช้งานอินเตอร์รัปต์ทั้งหมดที่สามารถเป็นสาเหตุที่โพรบ เรียกใช้ในภายหลังข้อกำหนดคุณสมบัติ

ตัวจัดการช่วงของโพรบต้องการ privilege ในการติดตามแบบไดนามิกเท่านั้น ตัวจัดการช่วงของโพรบจะบังคับให้จำกัดจำนวนของโพรบต่อไปนี้ ที่สนับสนุนเพื่อป้องกันผู้ใช้ที่ประสงค์ร้ายจากการรันเคอร์เนล โดยที่หน่วยความจำไม่เพียงพอด้วยการสร้างจำนวนของช่วงของโพรบขนาดใหญ่

ตารางที่ 24. จำกัดข้อมูลที่ระบุโดยตัวจัดการโพรบ interval.

ช่วงเวลา	จำนวน
จำนวนสูงสุดของช่วงของโพรบต่อผู้ใช้	32
จำนวนสูงสุดของช่วงของโพรบในระบบ	1024

ตัวจัดการช่วงของโพรบไม่ได้สนับสนุนฟังก์ชันต่อไปนี้ ถ้าใช้ภายในจุดโพรบของตัวจัดการช่วง ฟังก์ชันเหล่านี้จะสร้างสตริงว่างหรือศูนย์ที่เป็นเอาต์พุต

- **get_function**
- **get_probe**
- **get_location_point**

เมื่อไม่ระบุ ID กระบวนการ โพรบ interval สามารถทริกเกอร์ในบริบทของกระบวนการใดๆ ขึ้นอยู่กับเมื่อโพรบถูกใช้เนื่องจากเหตุการณ์โพรบจะอ้างอิงตามเวลา ด้วยเหตุนี้ กรอบงาน ProbeVue จะไม่อนุญาตให้ใช้ฟังก์ชันต่อไปนี้ ภายในบล็อกการดำเนินการของตัวจัดการช่วงของโพรบ เพื่อป้องกันการเข้าถึงข้อมูลภายในของการประมวลผล ซึ่งไม่ได้รับอนุญาต การละเมิดนี้จะถูกจัดในเคอร์เนลเท่านั้น สคริปต์ Vue จะคอมไพล์เป็นผลสำเร็จ แต่เซสชันจะล้มเหลวในการกำหนดค่าเริ่มต้น

- **stktrace**
- **get_userstring**

ฟังก์ชันเหล่านี้ไม่มีค่า เมื่อใช้จากตัวจัดการโพรบ หากคุณไม่ใช่ผู้ใช้ root คุณจะไม่สามารถเรียกฟังก์ชันเหล่านี้ ภายในตัวจัดการช่วงของโพรบได้

When the process ID is specified, the interval probe is triggered for all the threads within the process at the specified time interval. As the probe is fired in the context of the process, **stktrace( )** function and **__pname** built-in is allowed inside the interval probe manager's action block, unlike when process ID is not specified.

ตัวจัดการโพรบสำหรับการติดตามระบบ

ตัวจัดการโพรบสำหรับการติดตามระบบจัดเตรียมจุดโพรบที่การติดตามระบบที่มีอยู่ hook กับช่องสัญญาณการติดตามที่มีค่าศูนย์ (ช่องสัญญาณเหตุการณ์ของระบบ) เกิดขึ้น ทั้งภายในเคอร์เนลและภายในแอ็พพลิเคชัน เมื่อต้องการใช้ตัวจัดการโพรบนี้ คุณต้องมีสิทธิ์เข้าถึงเคอร์เนล และไม่รันใน WPAR

ตัวจัดการโพรบสำหรับการติดตามระบบจะยอมรับข้อกำหนดคุณสมบัติโพรบ 3-tuple ในรูปแบบดังนี้:

```
@@systrace:*:<hookid>
```

โดยที่อาร์กิวเมนต์ *hookid* จะระบุ ID สำหรับ hook การติดตามระบบที่ระบุเฉพาะ อาร์กิวเมนต์ *hookid* ประกอบด้วย 4 หลักของเลขฐานสิบหกในรูปของ hhh0 ตัวอย่างเช่น หากต้องการระบุอาร์กิวเมนต์ *hookid* สำหรับการเรียกระบบ **fork** ให้ระบุ 1390 โปรดดูไฟล์ **/usr/include/sys/trchkid.h** สำหรับตัวอย่าง เช่น HKWD_SYSC_FORK๑ รายการที่อยู่ในไฟล์นี้คือ hook words โดยที่ค่า *hookid* อยู่ในตำแหน่งครึ่งคำด้านบน เนื่องจาก hook words สามารถสุ่มเลือกได้ ไม่มีการตรวจสอบความถูกต้องของอาร์กิวเมนต์ *hookid* ซึ่งใกล้กับ การตรวจสอบว่าเป็นสตริงฐานสิบหกที่มีมากสุด 4 หลักของเลขฐานสิบหกจะถูกดำเนินการ ซึ่งไม่ใช่ข้อผิดพลาดที่ระบุค่า *hookid* ที่ไม่เคยเกิดขึ้น

เพื่อความสะดวก คุณสามารถระบุอาร์กิวเมนต์ *hookid* ที่มีเลขฐานหกที่ต่ำกว่า 4 ชุด ในกรณี ศูนย์ที่ตามมาซึ่งเป็นตัวแรกจะถูกยอมรับ จากนั้นศูนย์นำหน้าเพิ่มเติมตามความจำเป็นเพื่อกำหนดหลัก 4 หลักที่ต้องการ ตัวอย่างเช่น คุณสามารถใช้ 139 เป็นอักษรย่อของ 1390 ซึ่งคล้ายกับ 0100, 010 และ 10 ทั้งหมดนี้ระบุค่า *hookid* เดียวกันที่ใช้จาก KWD_USER1

คุณสามารถระบุอาร์กิวเมนต์ *hookid* ด้วยอักขระ wildcard * ซึ่งจะโพรบการติดตามระบบทั้งหมด ที่มีความหมายโดยนัยเกี่ยวกับผลการทำงานที่ไม่สามารถยอมรับได้ ดังนั้น ข้อกำหนดคุณสมบัติต้องถูกนำมาใช้เมื่อมีความจำเป็นที่แน่นอนเท่านั้น

tuple สำรองจะถูกจองไว้ และต้องถูกระบุเป็นเครื่องหมายดอกจัน ตามที่แสดง

เหตุการณ์สำหรับการติดตามระบบเท่านั้นที่เกิดขึ้น และบันทึกโพรบที่ทริกเกอร์ข้อมูลการติดตาม โดยเฉพาะอย่างยิ่ง โพรบการติดตามระบบ สามารถเกิดขึ้นได้เมื่อการติดตามระบบนั้นแอ็คทีฟ ตัวจัดการโพรบ systrace คือตัวจัดการโพรบแบบอิงเหตุการณ์ ดังนั้น ชื่อโพรบ ชื่อฟังก์ชัน และจุดตำแหน่งจะไม่พร้อมใช้งาน เนื่องจาก hookword ถูกส่งผ่านไปยังสคริปต์ จึงไม่ใช่ข้อจำกัดที่มีความสำคัญ

ผู้ใช้ที่ไม่ใช่ผู้ใช้ root จะถูกจำกัดจำนวนโพรบ systrace ที่มี 64 ตัวที่เปิดใช้งานอย่างพร้อมเพียงกัน ซึ่งไม่เกิน 128 โพรบt systrace ที่สามารถเปิดใช้งานในระบบแบบกว้างๆ ได้

ตัวแปรรีจิสเตอร์ในตัวของ ProbeVue อนุญาตให้เข้าถึงข้อมูลการติดตามนี้ คุณไม่สามารถใช้ ตัวแปร __arg* สำหรับวัตถุประสงค์นี้ ซึ่งมีลักษณะอยู่ด้วยการสองลักษณะ สำหรับการติดตามระบบ

ลักษณะต่อไปนี้ใช้สำหรับ trchook(64)/utrchook(64) (หรือ เทียบเท่ากับแมโคร TRCHKLx ใน C) hooks:

- __r3 contains the 16 bit hookid.
- __r4 contains the subhookid.
- __r5 contains traced data word D1.

- __r6 contains traced data word D2.
- __r7 contains traced data word D3.
- __r8 contains traced data word D4.
- __r9 contains traced data word D5.

ไม่ใช่ hook การติดตามทั้งหมดที่มีคำบรรจุข้อมูลทั้ง 5 คำ คำบรรจุข้อมูลที่ไม่ได้กำหนดไว้ จาก hook การติดตามที่กำหนดไว้จะปรากฏเป็นศูนย์ Vue clause สำหรับ ID hook ที่กำหนดไว้ต้องรู้ว่าข้อมูลและจำนวนของข้อมูลที่ ID hook ติดตาม

ถ้าเร็กคอร์ดการติดตามถูกสร้างโดยหนึ่งในฟังก์ชันที่อยู่ในตระกูล trcgen หรือ trcgent ให้ใช้ลักษณะต่อไปนี้:

- __r3 contains the 16 bit hookid.
- __r4 contains the subhookid.
- __r5 contains traced data word D1.
- __r6 contains the length of the traced data.
- __r7 contains the address of the traced data.

สคริปต์ต่อไปนี้จะแสดงตัวอย่างแบบง่ายๆ ของตัวจัดการโพรบ systrace :

```
@@systrace:*:1390
{
 if (__r4 == 0) { /* normal fork is traced with subhookid zero */
  printf("HKWD_SYSC_FORK: %d forks child %d\n", __pid, __r5);
  exit();
 }
}
```

Systrace probe manager ไม่ขึ้นอยู่กับฟังก์ชันการติดตามระบบ และสามารถติดตาม ตำแหน่ง hook ได้แม้ว่าฟังก์ชันการติดตามระบบไม่อยู่ในสถานะที่แอ็คทีฟ คุณสามารถเปิดใช้งาน ฟังก์ชันการติดตามระบบเมื่อเซสชัน ProbeVue แอ็คทีฟ

Systrace probe manager ใช้เคอร์เนลระบบปฏิบัติการ AIX เพื่ออำนวยความสะดวกการสร้างบางอย่าง การติดตาม hooks ที่ใช้โดย ProbeVue ในปัจจุบันจะมีผลกระทบกับความน่าเชื่อถือของ AIX ดังนั้น การสร้าง ProbeVue บางอย่างจึงไม่สามารถใช้ได้เมื่อคุณติดตาม hook ID ดังกล่าว ตารางต่อไปนี้แสดงข้อยกเว้นของการสร้าง:

หมายเหตุ: การสร้างต่อไปนี้จะถูกข้าม หากคุณต้องการติดตาม hook ทั้งหมดโดยใช้ข้อมูลจำเพาะ `@@systrace:*:*` คุณอาจไม่สามารถแสดงการติดตามสแต็กได้ หากเคอร์เนล AIX ห้ามการสร้างข้อยกเว้นใน สภาวะแวดล้อมที่มี hook ความสามารถของ ProbeVue ในการแสดงสแต็กการติดตาม จะถูกกำหนดขณะรันไทม์

ตารางที่ 25. การสร้าง Trace-hook

ตัวเลข	Trace-Hook	การสร้าง
1	HKWD_KERN_HCALL	ALL
2	HKWD_KERN_SLIH	Associative array, ช่วง, ตัวแปร stktrace, __stat
3	HKWD_KERN_LOCK	Associative array, ช่วง, ตัวแปร stktrace, __stat
4	HKWD_KERN_UNLOCK	Associative array, ช่วง, ตัวแปร stktrace, __stat
5	HKWD_KERN_DISABLEMENT	ALL

ตารางที่ 25. การสร้าง Trace-hook (ต่อ)

ตัวเลข	Trace-Hook	การสร้าง
6	HKWD_KERN_DISPATCH	__ublock, stktrace, get_stktrace, __pname, __execname, __errno
7	HKWD_KERN_DISPATCH_SRAD	__ublock, stktrace, get_stktrace, __pname, __execname, __errno
8	HKWD_KERN_DISP_AFFIN	__ublock, stktrace, get_stktrace, __pname, __execname, __errno
9	HKWD_KERN_UNDISP	__ublock, stktrace, get_stktrace, __pname, __execname, __errno
10	HKWD_KERN_IDLE	__ublock, stktrace, get_stktrace, __pname, __execname, __errno
11	HKWD_KERN_FLIH	Associative array, ช่วง, ตัวแปร stktrace, __stat
12	HKWD_KERN_RESUME	Associative array, ช่วง, ตัวแปร stktrace, __stat
13	HKWD_KERN_VPM	Associative array, ช่วง, ตัวแปร stktrace, __stat
14	HKWD_PM_NOTIFY	Associative array, ช่วง, ตัวแปร stktrace, __stat

โดยการใช้สิทธิ์ที่เหมาะสม สคริปต์ Vue สามารถสร้างเร็กคอร์ดการติดตามระบบโดยใช้ฟังก์ชัน Vue ของเหตุการณ์ (RAS) ที่พร้อมใช้งาน และสามารถใช้บริการได้ อย่างไรก็ตาม Systrace probe manager จะไม่ตรวจหาเร็กคอร์ดการติดตามที่สร้างจากสคริปต์ Vue

Extended system call probe manager (syscallx)

ตัวจัดการโพรบ syscallx หรืออีกนัยหนึ่ง อนุญาตให้สามารถ ติดตามการเรียกใช้ระบบฐานทั้งหมด การเรียกใช้ระบบฐานเป็นชุดของการเรียกใช้ระบบ ที่ถูกเอ็กซ์พอร์ตโดยเคอร์เนลและส่วนขยายเคอร์เนลฐาน ซึ่งพร้อมใช้งานทันทีหลังจากบูต ไม่สนับสนุนการเรียกใช้ระบบที่เอ็กซ์พอร์ต จากส่วนขยายเคอร์เนลที่อาถูกโหลดภายหลัง ทั้ง การเรียกใช้ระบบเฉพาะ หรือการเรียกใช้ระบบทั้งหมดสามารถระบุโดยใช้ probe point tuple อย่างไรก็ตาม ไม่เหมือนกับตัวจัดการโพรบ syscall ฟิลด์ที่สามของ probe point tuple สำหรับ syscallx ต้องระบุ ฟังก์ชันจุดเข้าของเคอร์เนลที่แท้จริง ตัวจัดการโพรบ syscallx ยังจำกัดโพรบที่จะใช้ในกระบวนการเฉพาะ ถ้าระบุ ID กระบวนการ เช่นเดียวกับฟิลด์ที่สองของ probe point tuple

ต่อไปนี้เป็นตัวอย่างบางส่วน:

```
/* Probe point tuple to probe the read system call entry for all processes */
@@syscallx:*:kread:entry

/* Probe point tuple to probe the fork system call exit for process with ID 434 */
@@syscallx:434:kfork:exit

/* Probe point tuple to probe entry for all base system calls */
@@syscallx:*:*:entry

/* Probe point tuple to probe exit for all base system calls for process 744 */
@@syscallx:744:*:exit
```

การเรียกระบบที่สนับสนุนโดยตัวจัดการโพรบ syscall

ตารางต่อไปนี้จะแสดงการเรียกระบบที่สนับสนุนโดยตัวจัดการโพรบ syscall พร้อมกับชื่อ entry ที่เกิดขึ้นจริงในเคอร์เนล

หมายเหตุ: ชื่อ entry ของเคอร์เนลจะถูกจัดเตรียมไว้ที่นี่เท่านั้น สำหรับวัตถุประสงค์ในการจัดทำเอกสาร ชื่อ entry ของเคอร์เนลสามารถเปลี่ยนได้ระหว่างรีลีส หรือหลังเซอร์วิสอัพเดต

ตารางที่ 26. การเรียกระบบที่สนับสนุนโดยตัวจัดการโพรบ syscall.

ชื่อการเรียกของระบบ	ชื่อรายการของเคอร์เนล
absinterval	absinterval
accept	accept1
bind	bind
close	close
creat	creat
execve	execve
exit	_exit
fork	kfork
getgidx	getgidx
getgroups	getgroups
getinterval	getinterval
getpeername	getpeername
getpid	_getpid
getppid	_getppid
getpri	_getpri
getpriority	_getpriority
getsockname	getsockname
getsockopt	getsockopt
getuidx	getuidx
incinterval	incinterval
kill	kill
listen	listen
lseek	klseek
mknod	mknod
mmap	mmap
mq_close	mq_close
mq_getattr	mq_getattr
mq_notify	mq_notify
mq_open	mq_open
mq_receive	mq_receive
mq_send	mq_send

ตารางที่ 26. การเรียกระบบที่สนับสนุนโดยตัวจัดการโพรบ syscall (ต่อ).

ชื่อการเรียกของระบบ	ชื่อรายการของเคอร์เนล
mq_setattr	mq_setattr
mq_unlink	mq_unlink
msgctl	msgctl
msgget	msgget
msgrcv	__msgrcv
msgsnd	__msgsnd
nsleep	_nsleep
open	kopen
pause	_pause
pipe	pipe
plock	plock
poll	_poll
read	kread
reboot	reboot
recv	_erecv
recvfrom	_enrecvfrom
recvmsg	_erecvmsg
select	_select
sem_close	_sem_close
sem_destroy	sem_destroy
sem_getvalue	sem_getvalue
sem_init	sem_init
sem_open	_sem_open
sem_post	sem_post
sem_unlink	sem_unlink
sem_wait	_sem_wait
semctl	semctl
semget	semget
semop	__semop
semtimedop	__semtimedop
send	_esend

ตารางที่ 26. การเรียกระบบที่สนับสนุนโดยตัวจัดการโพรบ syscall (ต่อ).

ชื่อการเรียกของระบบ	ชื่อรายการของเคอร์เนล
sendmsg	_esendmsg
sendto	_esendto
setpri	_setpri
setpriority	_setpriority
setsockopt	setsockopt
setuidx	setuidx
shmat	shmat
shmctl	shmctl
shmdt	shmdt
shmget	shmget
shutdown	shutdown
sigaction	_sigaction
sigpending	_sigpending
sigprocmask	sigprocmask
sigsuspend	_sigsuspend
socket	socket
socketpair	socketpair
stat	statx
waitpid	kwaitpid
write	kwrite

การรัน WPAR

เวิร์กโหลดพาร์ติชันหรือ WPARs คือสภาวะแวดล้อมของระบบปฏิบัติการที่ทำเวอร์ชวลไลซ์ ภายในอินสแตนซ์เดียวของระบบปฏิบัติการ AIX สภาวะแวดล้อม WPAR จะมีความแตกต่างจากสภาวะแวดล้อมของระบบปฏิบัติการ AIX มาตรฐาน

การติดตามแบบไดนามิกได้รับการสนับสนุนในสภาวะแวดล้อม WPAR ตามค่าดีฟอลต์ ขณะที่สร้าง WPAR เฉพาะ privilege **PV_PROBEVUE_TRC_USER_SELF** และ **PV_PROBEVUE_TRC_USER** เท่านั้นที่ถูกกำหนดให้กับ WPAR และ superuser (root) บนระบบ WPAR ซึ่งจะให้สิทธิ privilege เหล่านี้ ผู้ใช้ admin จากโกลบอลพาร์ติชันสามารถเปลี่ยนค่าของชุด privilege WPAR ที่เป็นค่าดีฟอลต์ หรือสามารถกำหนด privilege เพิ่มเติมได้ ขณะที่สร้าง WPAR

privilege บน WPAR มีความหมายเดียวกับบน โกลบอลพาร์ติชัน โปรดระวัง ขณะที่กำหนด **PV_PROBEVUE_TRC_KERNEL** หรือ the **PV_PROBEVUE_TRC_MANAGE** ให้กับ WPAR ผู้ใช้ใดๆ ที่มี privilege **PV_PROBEVUE_TRC_KERNEL** จะสามารถเข้าถึงตัวแปรเคอร์เนลแบบโกลบอลได้ ขณะที่ผู้ใช้ที่มี privilege

PV_PROBEVUE_TRC_MANAGE สามารถเปลี่ยนค่าพารามิเตอร์ ProbeVue หรือปิดระบบ ProbeVue การเปลี่ยนแปลงเหล่านี้จะกระทบกับผู้ใช้ทั้งหมดแม้ว่าผู้ใช้เหล่านั้นจะอยู่ในพาร์ติชันอื่นก็ตาม

เมื่อคุณออกคำสั่ง **probevue** ใน WPAR การประมวลผลที่รันอยู่ใน WPAR ตัวอื่น หรือในโกลบอลพาร์ติชัน จะไม่สามารถมองเห็นได้ ด้วยเหตุนี้ คุณจึงสามารถโพรบการประมวลผลใน WPAR ของคุณที่เหมือนกันได้เท่านั้น คำสั่ง **probevue** จะล้มเหลวหากข้อกำหนดคุณสมบัติโพรบไม่มี ID การประมวลผลที่อยู่ภายนอก พาร์ติชัน privilege **PV_PROBEVUE_TRC_USER** และ **PV_PROBEVUE_TRC_SYSCALL** ใน WPAR เท่านั้น จะอนุญาตให้คุณโพรบฟังก์ชันพื้นที่ผู้ใช้ หรือการเรียกระบบของการประมวลผล ที่อยู่ใน WPAR ของคุณ ขณะที่โพรบการเรียกระบบ ฟิลด์สำรองของข้อกำหนดคุณสมบัติโพรบ syscall ต้องถูกตั้งค่าเป็น ID การประมวลผลที่สามารถมองเห็น WPAR ที่ถูกต้องได้ การกำหนดค่า * ให้กับฟิลด์สำรองไม่ได้รับการสนับสนุน

เมื่อเซสชัน ProbeVue ไม่ได้เริ่มต้นใน WPAR ที่เคลื่อนที่ได้ เซสชันจะสับเปลี่ยน WPAR ไปเป็นสถานะที่ไม่ใช่จุดตรวจสอบ หลังจากที่ยกเลิกเซสชัน ProbeVue WPAR คือจุดตรวจสอบอีกครั้ง

โปรแกรมจัดการโพรบ I/O

โปรแกรมจัดการโพรบ I/O จัดเตรียมความสามารถ ในการติดตามเหตุการณ์การดำเนินการ I/O ในเลเยอร์ต่างๆ ของสแต็ก I/O ของ AIX ใช้โปรแกรมจัดการโพรบ syscall เพื่อติดตามคำร้องขอแอ็พพลิเคชัน I/O ที่ถูกทริกเกอร์โดยเขียน/อ่าน การเรียกระบบ ใช้ปรแกรมจัดการโพรบ I/O เพื่อโพรบไปยังเลเยอร์ syscall เพิ่มเติม

ใช้โปรแกรมจัดการโพรบ I/O เพื่อวิเคราะห์เวลาตอบกลับของการดำเนินการ I/O ของอุปกรณ์บล็อกที่แยกเวลาให้บริการกับการหน่วงการจัดคิว

เลเยอร์ต่อไปนี้ได้รับการสนับสนุน:

- Logical File System (LFS)
- Virtual File System (VFS)
- Enhanced Journaled File Systems (JFS2)
- Logical Volume Manager (LVM)
- ไดร์เวอร์ดิสก์ Small Computer System Interface (SCSI)
- อุปกรณ์บล็อกทั่วไป

กรณีการใช้งานหลักสำหรับโปรแกรมจัดการโพรบ I/O มีดังต่อไปนี้:

- ระบุแพตเทิร์นต่อไปนี้ของการใช้งาน I/O ของอุปกรณ์ อุปกรณ์ที่ถูกต้องสามารถเป็นดิสก์โลจิคัลวอลุ่ม หรือกลุ่มวอลุ่ม หรือระบบไฟล์ (ชนิดหรือเมาท์พาธ) ในระยะเวลาที่ระบุเฉพาะ:
  - จำนวนการดำเนินการ I/O
  - ขนาดของการดำเนินการ I/O
  - ชนิดการดำเนินการ I/O (อ่าน/เขียน)
  - ลักษณะของ I/O แบบลำดับหรือแบบสุ่ม
- รับข้อมูลกระบวนการหรือการใช้ thread-wise ของระบบไฟล์ (ชนิดหรือเมาท์พาธ) โลจิคัลวอลุ่ม กลุ่มวอลุ่ม หรือดิสก์
- รับการแม็พแบบ end-to-end ของโฟลว์ I/O ระหว่างเลเยอร์ต่างๆ (เมื่อเป็นไปได้)
- มอนิเตอร์การใช้รีซอร์ส I/O ที่ระบุเฉพาะ ตัวอย่างเช่น:
  - ติดตามการดำเนินการเขียนใดๆ ของไฟล์ `/etc/password`

- ติดตามการดำเนินการอ่านบนบล็อก 0 ของอุปกรณ์ hdisk0
- ติดตามเมื่อเปิดโลจิคัลวอลุ่มใหม่ในกลุ่มวอลุ่ม root (rootvg)

- สำหรับดิสก์ Multipath I/O (MPIO) รับข้อมูลที่ระบุเฉพาะพาธตามแอ็คชันต่อไปนี้:
  - รับการใช้งาน path-wise และข้อมูลเวลาตอบกลับ
  - ระบุการสลับพาธหรือพาธที่ล้มเหลว
- สำหรับข้อผิดพลาด I/O รับรายละเอียดเพิ่มเติมเกี่ยวกับข้อผิดพลาดในเลเยอร์ดิสก์ไดร์เวอร์

ข้อกำหนดคุณสมบัติโพรบ

โพรบ I/O ต้องถูกระบุไว้ในรูปแบบต่อไปนี้ในสคริปต์ Vue :

`@@io:`*`sub_type`*`:`*`io_event`*`:`*`operation_type`*`:`*`filter`*`[|`*`filter ...`*`]`

ข้อมูลจำเพาะนี้ ประกอบด้วยห้า tuple ที่คั่นด้วยเครื่องหมายโคลอน (:) tuple ตัวแรกจะเป็น `@@io` เสมอ

ชนิดย่อยของโพรบ

tuple ตัวที่สอง หมายถึงชนิดย่อยของโพรบที่บ่งชี้ถึงเลเยอร์ของสแต็ก I/O ของ AIX ที่มีโพรบ tuple นี้สามารถมีหนึ่งในค่าต่อไปนี้:

ตารางที่ 27. tuple ตัวที่สองสำหรับโพรบ.

tuple ตัวที่สอง (ชนิดย่อย)	**คำอธิบาย**
ดิสก์	โพรบนี้เริ่มต้นสำหรับเหตุการณ์ไดร์เวอร์ของดิสก์ ณ ปัจจุบัน โปรแกรมจัดการโพรบ I/O สนับสนุนเฉพาะไดร์เวอร์ scsidisk เท่านั้น
lvm	โพรบนี้เริ่มต้นสำหรับเหตุการณ์ Logical Volume Manager (LVM)
bdev	โพรบนี้เริ่มต้นสำหรับอุปกรณ์ I/O สำหรับบล็อกใดๆ ดิสก์ ซีดีรอม แผ่นดิสก์ คือตัวอย่างของอุปกรณ์ บล็อก ชนิดย่อยนี้ถูกใช้ก็ต่อเมื่อไม่มีชนิดย่อยอื่นใดที่สามารถเรียกทำงานได้ ตัวอย่างเช่น ถ้าอุปกรณ์บล็อกไม่ใช่ดิสก์ กลุ่มวอลุ่ม หรือโลจิคัลวอลุ่ม ชนิดย่อยนี้จะสามารถเรียกทำงานได้
jfs2	โพรบนี้เริ่มต้นสำหรับเหตุการณ์ระบบไฟล์ JFS2
vfs	โพรบนี้เริ่มต้นสำหรับการดำเนินการอ่าน/เขียนใดๆ บนไฟล์

หมายเหตุ: tuple ตัวที่สองไม่สามารถมีค่าเครื่องหมายดอกจัน (*) ได้

สำหรับชนิดของดิสก์ของ tuple ตัวที่สอง tuple ตัวที่สามสามารถมีค่าดังต่อไปนี้:

ตารางที่ 28. tuple ตัวที่สองของดิสก์: ค่า tuple ตัวที่สาม.

ชนิดย่อย (tuple ตัวที่สอง)	เหตุการณ์ I/O (Tuple ตัวที่สาม)	คำอธิบาย
ดิสก์	entry	โพรบนี้เริ่มต้นเมื่อใดก็ตามที่ดิสก์ไดร์เวอร์ได้รับคำร้องขอ I/O เพื่อประมวลผล
	iostart	โพรบนี้เริ่มต้นเมื่อไดร์เวอร์ของดิสก์เลือกคำร้องขอ I/O จากคิวที่พร้อมใช้งานและส่ง ไปยังเลเยอร์ที่อยู่ต่ำกว่า (ตัวอย่างเช่น ไดร์เวอร์อะแด็ปเตอร์) I/O เดี่ยวต้นทางที่ร้องขอไปยังไดร์เวอร์ของดิสก์ สามารถส่งคำร้องขอคำสั่งจำนวนมาก (บางคำร้องขออาจเป็นคำร้องขอคำสั่งการจัดการกับภารกิจที่เกี่ยวข้องกับไดร์เวอร์) ไปยังเลเยอร์ที่อยู่ต่ำกว่า อย่างไรก็ตาม ในบางครั้ง ไดร์เวอร์สามารถรวมคำร้องขอต้นทางจำนวนมากและส่ง คำร้องขอเดี่ยวไปยังเลเยอร์ที่อยู่ต่ำกว่า
	iodone	โพรบนี้เริ่มต้นเมื่อเลเยอร์ที่อยู่ต่ำกว่า (ตัวอย่างเช่น ไดร์เวอร์อะแด็ปเตอร์) ส่งคืนคำร้องขอ I/O (สำเร็จหรือล้มเหลว) ไปยังไดร์เวอร์ของดิสก์
	exit	โพรบนี้เริ่มต้นเมื่อไดร์เวอร์ของดิสก์ส่งคืนคำร้องขอ I/O (สำเร็จหรือล้มเหลว) ไปยัง เลเยอร์ที่อยู่บนสุด

หมายเหตุ: สมาชิกของค่า built-in ต่อไปนี้พร้อมใช้งานใน โพรบที่กล่าวถึงสำหรับชนิดย่อยของโพรบ: `__iobuf`, `__diskinfo`, `__diskcmd` (เฉพาะใน)`disk:iostart` และ `disk:iodone`), และ `__iopath` (เฉพาะใน `disk:iostart` และ `disk:iodone`)

สำหรับทุกๆ รายการ โพรบ exit ที่สอดคล้องกันถูกนิยามไว้ว่า มีค่า `__iobuf->bufid` ที่เหมือนกันซึ่งมีที่จุดโพรบทั้งสองจุด เหตุการณ์ของการเข้า สามารถตามด้วยเหตุการณ์ `iostart` แต่อย่างน้อยหนึ่งในเหตุการณ์เหล่านี้ ต้องมีค่า `__iobuf->bufid` ที่เหมือนกัน ทุกๆ เหตุการณ์ `iostart` มีการจับคู่เหตุการณ์ `iodone` ที่มีค่า `__iobuf->child_bufid` ที่เหมือนกัน

สำหรับชนิด LVM ของ tuple ตัวที่สอง tuple ตัวที่สามสามารถมีค่า ต่อไปนี้:

ตารางที่ 29. tuple ตัวที่สองของ LVM: ค่า tuple ตัวที่สาม.

ชนิดย่อย (tuple ตัวที่สอง)	เหตุการณ์ I/O (tuple ตัวที่สาม)	คำอธิบาย
lvm	entry	โพรบนี้เริ่มต้นเมื่อใดก็ตามที่เลเยอร์ LVM รับคำร้องขอ I/O เพื่อประมวลผล
	iostart	โพรบนี้เริ่มต้นเมื่อ LVM เลือกคำร้องขอ I/O จากคิวที่พร้อมใช้งานและส่งไปยัง เลเยอร์ที่อยู่ต่ำกว่า (โดยปกติแล้วคือ ไดร์เวอร์ของดิสก์)
	iodone	โพรบนี้เริ่มต้นเมื่อเลยอร์ที่อยู่ต่ำกว่า (ตัวอย่างเช่น ไดร์เวอร์ของดิสก์) ส่งคืนคำร้องขอ I/O (สำเร็จหรือล้มเหลว) ไปยัง LVM
	exit	โพรบนี้เริ่มต้นเมื่อ LVM ส่งคืนคำร้องขอ I/O (สำเร็จหรือล้มเหลว) ไปยังเลเยอร์ที่สูงกว่า

หมายเหตุ: สมาชิกของค่า built-ins ต่อไปนี้พร้อมใช้งาน ในโพรบที่กล่าวถึงสำหรับ LVM: `__iobuf`, `__lvol` และ `__volgrp` ทุกๆ รายการมีโพรบ exit ที่สอดคล้องกัน ซึ่งมีค่า `__iobuf->bufid` ที่เหมือนกันซึ่งมีที่จุดโพรบทั้งสองจุด

เหตุการณ์ของการเข้าสามารถตามด้วยเหตุการณ์ `iostart` จำนวนมาก แต่มีอย่างน้อยหนึ่งเหตุการณ์ที่มีค่า `__iobuf->bufid` ที่เหมือนกัน ทุกๆ เหตุการณ์ `iostart` มีการจับคู่เหตุการณ์ `iodone` ที่มีค่า `__iobuf->child_bufid` ที่เหมือนกัน

สำหรับโพรบอุปกรณ์บล็อกทั่วไป tuple ตัวที่สามสามารถมีค่า ต่อไปนี้:

ตารางที่ 30. tuple ตัวที่สองของอุปกรณ์บล็อกทั่วไป: ค่า tuple ตัวที่สาม.

ชนิดย่อย (tuple ตัวที่สอง)	เหตุการณ์ I/O (tuple ตัวที่สาม)	คำอธิบาย
bdev	iostart	โพรบนี้รับการใช้งานเมื่ออุปกรณ์บล็อก I/O ใดๆ (ตัวอย่างเช่น ดิสก์โลจิคัลวอลุ่ม ซีดีรอม) เริ่มต้นทำงาน ซึ่งเกิดขึ้นเมื่อเรียกเซอร์วิสเคอร์เนล AIX `devstrat` ตามโค้ดใดๆ
	iodone	โพรบนี้รับการใช้งานเมื่อคำร้องขอบล็อก I/O ที่สมบูรณ์เกิดขึ้น เมื่อเซอร์ซิสเคอร์เนล AIX `iodone` ถูกเรียกตามโค้ดใดๆ

หมายเหตุ: สมาชิกของค่า built-in ต่อไปนี้พร้อมใช้งาน ในโพรบที่กล่าวถึงใน bdev: `__iobuf` ทุกๆ เหตุการณ์ `iostart` มีการจับคู่เหตุการณ์ `iodone` ที่มีค่า `__iobuf->bufid` ที่เหมือนกัน

สำหรับโพรบระบบไฟล์ JFS2 tuple ตัวที่สามสามารถมีค่า ต่อไปนี้:

ตารางที่ 31. tuple ตัวที่สองของ JFS2: ค่า tuple ตัวที่สาม.

ชนิดย่อย (tuple ตัวที่สอง)	เหตุการณ์ I/O (tuple ตัวที่สาม)	คำอธิบาย
jfs2	buf_map	โพรบนี้เริ่มต้นเมื่อส่วนขยายโลจิคัลไฟล์รับการแม็พไปยังบัฟเฟอร์ I/O และถูกส่งต่อไปยังโลจิคัลวอลุ่มที่มีความสำคัญ

หมายเหตุ: สมาชิกของค่า built-in ต่อไปนี้พร้อมใช้งาน ในโพรบที่กล่าวถึงสำหรับโพรบระบบไฟล์ JFS2: `__j2info`

สำหรับโพรบระบบไฟล์เสมือน (VFS) tuple ตัวที่สามสามารถมีค่า ต่อไปนี้:

ตารางที่ 32. tuple ตัวที่สองของ VFS: ค่า tuple ตัวที่สาม.

ชนิดย่อย (tuple ตัวที่สอง)	เหตุการณ์ I/O (tuple ตัวที่สาม)	คำอธิบาย
vfs	entry	โพรบนี้เริ่มต้นสำหรับการดำเนินการอ่าน/เขียนบนไฟล์ที่เริ่มต้นแล้ว
	exit	โพรบนี้เริ่มต้นเมื่อการดำเนินการอ่าน/เขียนบนไฟล์เสร็จสิ้นแล้ว (ไม่ว่าจะสำเร็จหรือล้มเหลว)

หมายเหตุ: สมาชิกของบิวด์อินต่อไปนี้พร้อมใช้งานในโพรบที่ถูกกล่าวถึงในโพรบ VFS: `__file`

สำหรับเธรดเดียวกัน ทุกๆ การเข้าถูกตามด้วยเหตุการณ์ออก ซึ่งมีค่า `__file->inode_id` ที่เหมือนกัน

ชนิดของการดำเนินการโพรบ

tuple ตัวที่สี่บ่งชี้ชนิดของการดำเนินการ I/O ที่ถูกระบุไว้โดยโพรบ tuple ตัวที่สี่สามารถมีหนึ่งในค่าต่อไปนี้:

ตารางที่ 33. tuple ตัวที่สี่สำหรับการดำเนินการ I/O.

tuple ตัวที่สี่	คำอธิบาย
การอ่าน	โพรบเริ่มต้นสำหรับการดำเนินการอ่านเท่านั้น
การเขียน	โพรบเริ่มต้นสำหรับการดำเนินการเขียนเท่านั้น
*	โพรบเริ่มต้นสำหรับการดำเนินการอ่านและเขียน

ตัวกรองโพรบ

tuple ตัวที่ห้าเป็น tuple ตัวกรองที่ช่วยในการกรอง โพรบที่ระบุเฉพาะเพิ่มเติมตามข้อกำหนด ค่าที่เป็นไปได้คือการพึ่งพาชนิดย่อย ค่าจำนวนมากสามารถระบุได้โดยคั่นด้วยอักขระ | และโพรบเริ่มต้นหากตรงกับ ตัวกรองเหล่านั้น ถ้าค่าของ tuple ตัวที่ห้าเป็น * จะไม่มีการกรองเกิดขึ้น และโพรบจะเริ่มต้น หาก tuple อื่นตรงกัน ถ้าระบุตัวเลือกจำนวนมากไว้ และมีหนึ่งในตัวเลือกเหล่านั้นเป็น * นั่นคือเทียบเท่ากับ ค่า tuple ทั้งหมดของ *

สำหรับโพรบ disk tuple ตัวที่ห้าสามารถมี ค่าต่อไปนี้:

ตารางที่ 34. tuple ตัวกรองดิสก์.

ตัวกรอง (tuple ตัวที่ห้า)	คำอธิบาย
ชื่อดิสก์ ตัวอย่างเช่น `hdisk0`	แอ็คชันโพรบจะรันสำหรับดิสก์เฉพาะเท่านั้น
ชนิดของดิสก์ สัญลักษณ์ที่อนุญาตให้ใช้: FC, ISCSI, VSCSI, SAS	แอ็คชันโพรบจะรันสำหรับดิสก์ที่มีชนิดที่ตรงกันเท่านั้น ความหมายของสัญลักษณ์มีดังต่อไปนี้: • FC: ดิสก์แบบไฟเบอร์แชนแนล • ISCSI: ดิสก์ iSCSI • VSCSI: ดิสก์ SCSI เสมือน (บนไคลเอ็นต์ VIOS) • SAS: ดิสก์ SCSI ที่พ่วงต่อแบบอนุกรม

หมายเหตุ: ชื่อดิสก์และชนิดของดิสก์สามารถรวมเป็นตัวกรองได้ ตัวอย่าเช่น โพรบต่อไปนี้ เริ่มต้นสำหรับ `hdisk0` หรือดิสก์ FC อื่นๆ (ที่เหตุการณ์การเข้าสู่ดิสก์ สำหรับชนิดการดำเนินการอ่าน/เขียน)

```
@@io:disk:entry:*:hdisk0|FC
```

สำหรับโพรบ Logical Volume Manager (LVM) tuple ตัวที่ห้าสามารถมีค่าต่อไปนี้:

ตารางที่ 35. tuple ตัวกรอง LVM.

ตัวกรอง (Tuple ตัวที่ห้า)	คำอธิบาย
ชื่อโลจิคัลวอลุ่ม ตัวอย่างเช่น `hd5, lg_dumplv`	แอ็คชันโพรบจะรันสำหรับโลจิคัลวอลุ่มเฉพาะ
ชื่อกลุ่มวอลุ่ม ตัวอย่างเช่น `rootvg`	แอ็คชันโพรบจะรันสำหรับโลจิคัลวอลุ่มเหล่านั้นที่เป็นของ กลุ่มวอลุ่มเฉพาะ

โพรบต่อไปนี้เริ่มต้นสำหรับโลจิคัลวอลุ่มที่เป็นของกลุ่มวอลุ่ม root (rootvg) หรือกลุ่มวอลุ่มการทดสอบ (testvg) (ที่เหตุการณ์ iostart สำหรับการดำเนินการเขียน) :

```
@@io:lvm:iostart:write:rootvg|testvg
```

สำหรับโพรบอุปกรณ์บล็อกทั่วไป tuple ตัวที่ห้าสามารถมีค่าต่อไปนี้:

ตารางที่ 36. tuple ตัวกรองอุปกรณ์บล็อกทั่วไป.

ตัวกรอง (tuple ตัวที่ห้า)	คำอธิบาย
ชื่ออุปกรณ์บล็อก ตัวอย่างเช่น: `hdisk0, hd5, cd0`	แอ็คชันโพรบจะรันสำหรับอุปกรณ์บล็อกเฉพาะเท่านั้น

ให้พิจารณาตัวอย่างต่อไปนี้สำหรับโพรบ อุปกรณ์บล็อกทั่วไป:

```
@@io:bdev:iostart:*:cd0
```

```
@@io:bdev:iodone:read:hdisk3|hdisk5
```

สำหรับโพรบระบบไฟล์ JFS2 tuple ตัวที่ห้าสามารถมีค่าต่อไปนี้:

ตารางที่ 37. tuple ตัวกรอง JFS2.

ตัวกรอง (tuple ตัวที่ห้า)	คำอธิบาย
เมาท์พาธของระบบไฟล์ ตัวอย่างเช่น: `/usr`	แอ็คชันโพรบจะรันสำหรับระบบไฟล์ที่มีเมาท์พาธเฉพาะ ซึ่งต้องเป็นระบบไฟล์ JFS2 มิฉะนั้น ProbeVue จะปฏิเสธข้อมูลจำเพาะโพรบนั้น

ให้พิจารณาตัวอย่างต่อไปนี้สำหรับโพรบระบบไฟล์ JFS2:

```
@@io:jfs2:buf_map:*:/usr|/tmp
```

สำหรับโพรบระบบไฟล์เสมือน (VFS) tuple ตัวที่ห้าสามารถมีค่าต่อไปนี้:

ตารางที่ 38. tuple ตัวกรอง VFS.

ตัวกรอง (Tuple ตัวที่ห้า)	คำอธิบาย
เมาท์พาธระบบไฟล์ ตัวอย่างเช่น `/tmp`	แอ็คชันโพรบจะรันสำหรับไฟล์ที่เป็นของระบบไฟล์
ชนิดของระบบไฟล์ สัญลักษณ์ที่อนุญาตให้ใช้คือ `JFS2, NAMEFS, NFS, JFS, CDROM, PROCFS, SFS, CACHEFS, NFS3, AUTOFS, POOLFS, VXFS, VXODM, UDF, NFS4, RFS4, CIFS, PMEMFS, AHAFS, STNFS, ASMFS`	แอ็คชันโพรบจะรันสำหรับไฟล์ของระบบไฟล์เฉพาะ สัญลักษณ์สอดคล้องกับระบบไฟล์ AIX ที่นิยามไว้ในไฟล์ส่วนหัวที่เอ็กซ์พอร์ต `sys/vmount.h`

ให้พิจารณาตัวอย่างต่อไปนี้สำหรับโพรบระบบไฟล์เสมือน (VFS) :

```
@@io:vfs:entry:read:JFS2
```

```
@@io:vfs:exit:*:/usr|JFS
```

ตัวแปร built-in ที่เกี่ยวกับโพรบ I/O สำหรับสคริปต์ Vue

ตัวแปร built-in `__iobuf`

คุณสามารถใช้ตัวแปร built-in __iobuf พิเศษเพื่อเข้าถึงข้อมูลต่างๆ เกี่ยวกับบัฟเฟอร์ I/O ที่นำมาใช้ใน การดำเนินการ I/O ปัจจุบัน ซึ่งสามารถเข้าถึงได้ในโพรบ ของชนิดย่อย: disk, lvm และ bdev อิลิเมนต์ สมาชิกสามารถเข้าถึงได้โดยใช้ไวยากรณ์ __iobuf->member

หมายเหตุ: เมื่อใดก็ตามที่ไม่สามารถขอรับค่าจริงได้ ค่าที่ทำเครื่องหมายเป็น Invalid Value จะถูกส่งคืน ค่านี้ถูกส่งคืน เนื่องจากเหตุผลต่อไปนี้:

- บริบทข้อบกพร่องของเพจจำเป็นต้องมี แต่ค่าที่สามารถปรับได้ probevctrl ปัจจุบัน num_pagefaults มีค่า 0 หรือไม่เพียงพอ
- ตำแหน่งของหน่วยความจำที่มีค่าที่เพจเอาต์
- ข้อผิดพลาดระบบที่รุนแรง เช่น ตัวชี้ที่ไม่ถูกต้องหรือหน่วยความจำล้มเหลว

ตัวแปร built-in __iobuf มีสมาชิกต่อไปนี้:

ตารางที่ 39. สมาชิกตัวแปรบิวต์อิน __iobuf.

ชื่อของสมาชิก	ชนิด	คำอธิบาย	ค่าที่ไม่ถูกต้อง
blknum	unsigned long long	เริ่มต้นจากจำนวนบล็อกของคำร้องขอ I/O	0xFFFFFFFFFFFFFFFF
bcount	unsigned long long	จำนวนไบต์ที่ร้องขอในการดำเนินการ I/O	0xFFFFFFFFFFFFFFFF
bflags	unsigned long long	แฟล็กที่ถูกเชื่อมโยงกับการดำเนินการ I/O สัญลักษณ์ต่อไปนี้พร้อมใช้งาน: B_READ, B_ASYNC, B_ERROR สัญลักษณ์ต่างๆ สามารถใช้พร้อมกับค่า bflags เพื่อดูว่าได้ตั้งค่าไว้หรือไม่ ตัวอย่างเช่น หาก (__iobuf->bflags & B_READ) เป็น true ดังนั้นจึงเป็นการดำเนินอ่าน **หมายเหตุ:** ไม่มีแฟล็ก B_WRITE ถ้าไม่ได้ตั้งค่าแฟล็ก B_READ อาจต้องพิจารณาเป็นการดำเนินการเขียน	0
devnum	unsigned long long	จำนวนอุปกรณ์ของอุปกรณ์เป้าหมายที่เชื่อมโยงกับการดำเนินการ I/O ซึ่งเป็น จำนวนหลักของอุปกรณ์และจำนวนรองที่ฝังไว้	0
major_num	int	จำนวนหลักของอุปกรณ์เป้าหมายของการดำเนินการ I/O	-1
minor_num	int	จำนวนรองของอุปกรณ์เป้าหมายของการดำเนินการ I/O	-1
error	int	ในกรณีของข้อผิดพลาดใดๆ ในการดำเนินการ I/O ค่านี้จึงเป็นหมายเลขข้อผิดพลาด ค่านี้ ถูกนิยามไว้ในไฟล์ส่วนหัว errno.h ที่เอ็กซ์พอร์ต	-1

ตารางที่ 39. สมาชิกตัวแปรบิวด์อิน __iobuf (ต่อ).

ชื่อของสมาชิก	ชนิด	คำอธิบาย	ค่าที่ไม่ถูกต้อง
residue	unsigned long long	จำนวนไบต์ที่เหลืออยู่จากคำร้องขอต้นทางที่อาจไม่ได้ถูกอ่านหรือเขียนสำหรับเหตุการณ์ความสมบูรณ์ของ I/O ค่านี้จะเป็นศูนย์ แต่สำหรับการดำเนินการอ่าน ค่าที่ไม่ใช่ศูนย์อาจหมายความว่า คุณกำลังพยายามอ่านมากกว่าที่จะมีอยู่ ซึ่งสามารถยอมรับได้ ค่านี้ ถูกพิจารณาเมื่อค่าข้อผิดพลาดไม่ใช่ค่าศูนย์	0xFFFFFFFFFFFFFFFF
bufid	unsigned long long	หมายเลขเฉพาะที่ถูกเชื่อมโยงกับคำร้องขอ I/O ขณะที่ I/O กำลังดำเนินการอยู่ ค่า bufid จะระบุคำร้องขอ I/O เฉพาะในเหตุการณ์ทั้งหมดของชนิดย่อยเฉพาะ ตัวอย่างเช่น ใน `disk: entry`, `disk: iostart`, `disk: iodone` และ `disk:exit` ถ้า `__iobuf->bufid` ตรงกัน สิ่งนี้จะเป็นคำร้องขอ I/O ที่เหมือนกันที่สเตจต่างๆ)	0
parent_bufid	unsigned long long	ถ้าค่าไม่ใช่ 0 ค่านี้จัดเตรียม `bufid` ของบัฟเฟอร์เลเยอร์ที่อยู่สูงกว่า ซึ่งเชื่อมโยงกับคำร้องขอ I/O นี้ คุณสามารถลิงก์การดำเนินการ I/O ปัจจุบันด้วยคำร้องขอ I/O ในเลเยอร์ที่สูงกว่า ตัวอย่างเช่น ในคำร้องขอดิสก์ I/O ที่สอดคล้องกับ LVM I/O สามารถกำหนดได้ **หมายเหตุ:** ฟิลด์ parent_bufid ไม่ได้ถูกตั้งค่าไว้ในพาธโค้ดทั้งหมด และไม่มีประโยชน์ใช้ฟิลด์ child_bufid เพื่อลิงก์คำร้องขอ I/O ระหว่างสองเลเยอร์ที่อยู่ติดกัน	0
child_bufid	unsigned long long	ถ้าค่าไม่ใช่ 0 ค่านี้จะจัดเตรียม `bufid` ของคำร้องขอ I/O ใหม่ ที่ส่งไปยังเลเยอร์ที่อยู่ต่ำกว่า เหตุการณ์ที่ดีที่สุดที่ต้องบันทึกไว้คือ `disk:iostart`, `lvm:iostart` และ `bdev:iostart` คุณสามารถระบุ I/O ในเลเยอร์ต่ำกว่า ที่อยู่ติดกันโดยจับคู่ค่า `__iobuf->bufid` กับค่า child_bufid นี้ ตัวอย่างเช่น ใน `lvm:iostart` คุณสามารถบันทึกค่า `__iobuf->child_buf` จากนั้น ใน `disk:entry` คุณสามารถจับคู่กับ `__iobuf->bufid` เพื่อระบุคำร้องขอ I/O ที่สอดคล้องกัน	0

ตัวแปร built-in__file

คุณสามารถใช้ตัวแปร built-in พิเศษ __file เพื่อดูข้อมูลต่างๆ เกี่ยวกับการดำเนินการกับไฟล์ ซึ่งพร้อมใช้งานในโพรบของชนิดย่อย VFS อิลิเมนต์สมาชิกสามารถเข้าถึงได้โดยใช้ไวยากรณ์ __file->member

หมายเหตุ: เมื่อใดก็ตามที่ไม่สามารถขอรับค่าจริงได้ ค่าที่ถูกทำเครื่องหมายว่าไม่ถูกต้องจะถูกส่งคืน ค่าที่ไม่ถูกต้องถูกส่งคืนเนื่องจากหนึ่งในเหตุผลต่อไปนี้:

- บริบทข้อบกพร่องของเพจจำเป็นต้องมี แต่ค่าที่สามารถปรับได้ probevctrl ปัจจุบัน num_pagefaults มีค่า 0 หรือไม่เพียงพอ
- ตำแหน่งหน่วยความจำ ซึ่งมีค่าที่ถูกเพจเอาต์
- ข้อผิดพลาดระบบที่รุนแรงอื่นใด เช่น ตัวชี้ไม่ถูกต้อง หรือหน่วยความจำล้มเหลว

ตัวแปรบิวด์อิน __file มีสมาชิกต่อไปนี้:

ตารางที่ 40. สมาชิกตัวแปรบิวด์อิน __file.

ชื่อของสมาชิก	ชนิด	คำอธิบาย	ค่าที่ไม่ถูกต้อง
f_type	int	ระบุชนิดของไฟล์ ซึ่งสามารถตรงกับหนึ่งในค่าคงที่ built-in ต่อไปนี้: • F_REG (ไฟล์ปกติ) • F_DIR (ไดเร็กทอรี) • F_BLK (ไฟล์อุปกรณ์บล็อก) • F_CHR (ไฟล์อุปกรณ์ตัวอักขระ) • F_LNK (ลิงก์ไฟล์) • F_SOCK (ซ็อกเก็ต) **หมายเหตุ:** ค่าอาจไม่ตรงกับค่าคงที่ built-in ใดๆ เนื่องจาก รายการไม่รวมชนิดไฟล์ทุกชนิดที่เป็นไปได้ แต่มีเฉพาะชนิด ที่มีประโยชน์ที่สุด	-1

ตารางที่ 40. สมาชิกตัวแปรบิวด์อิน__file (ต่อ).

ชื่อของสมาชิก	ชนิด	คำอธิบาย	ค่าที่ไม่ถูกต้อง
fs_type	int	ระบุชนิดของระบบไฟล์ที่ต้องเป็นเจ้าของไฟล์ซึ่งสามารถตรงกับหนึ่งในค่าคงที่ built-in ต่อไปนี้: • FS_JFS2 • FS_NAMEFS • FS_NFS • FS_JFS • FS_CDROM • FS_PROCFS • FS_SFS • FS_CACHEFS • FS_NFS3 • FS_AUTOFS • FS_POOLFS • FS_VXFS • FS_VXODM • FS_UDF • FS_NFS4 • FS_RFS4 • FS_CIFS • FS_PMEMFS • FS_AHAFS • FS_STNFS • FS_ASMFS ค่าคงที่ built-in สอดคล้องกับชนิดระบบไฟล์ AIX ที่กำหนดไว้ในไฟล์ส่วนหัว sys/vmount.h ที่ถูกเอ็กซ์พอร์ต	-1
mount_path	char *	ระบุพาธที่ระบบไฟล์ที่เชื่อมโยงไว้ถูกเมาท์	null string
devnum	unsigned long long	ระบุจำนวนอุปกรณ์ของอุปกรณ์บล็อกที่เชื่อมโยงของไฟล์ ทั้งจำนวนหลักและจำนวนรอง ถูกฝั่งไว้ ถ้าไม่มีอุปกรณ์บล็อกที่เชื่อมโยงไว้ ค่าจะเป็น 0	0
major_num	int	ระบุจำนวนหลักของอุปกรณ์บล็อกที่เชื่อมโยงของไฟล์	-1
minor_num	int	ระบุจำนวนรองของอุปกรณ์บล็อกที่เชื่อมโยงของไฟล์	-1

ตารางที่ 40. สมาชิกตัวแปรบิวด์อิน __file (ต่อ).

ชื่อของสมาชิก	ชนิด	คำอธิบาย	ค่าที่ไม่ถูกต้อง
offset	unsigned long long	ระบุออฟเซ็ตไบต์อ่าน/เขียนปัจจุบันของไฟล์	0xFFFFFFFFFFFFFFFF
rw_mode	int	ระบุโหมดอ่าน/เขียนของไฟล์ซึ่งตรงกับหนึ่งใน ค่าของค่าคงที่ built-in: F_READ หรือ F_WRITE	-1
byte_count	unsigned long long	ที่เหตุการณ์การเข้า vfs: byte_count จัดเตรียมจำนวนไบต์ของคำร้องขอ การอ่านหรือการเขียน ณ เหตุการณ์การออก vfs: จะจัดเตรียมจำนวนไบต์ที่ยังคง เติมไม่เต็ม ตัวอย่างเช่น ความแตกต่างของค่านี้ระหว่างเหตุการณ์สองเหตุการณ์เหล่านี้ กำหนดจำนวนไบต์ที่ประมวลผลในการดำเนินการ	0xFFFFFFFFFFFFFFFF
fname	char *	ระบุชื่อของไฟล์ (เฉพาะชื่อหลัก ไม่ใช่พาธ)	null string
inode_id	unsigned long long	ระบุหมายเลขเฉพาะสำหรับทุกระบบที่เชื่อมโยงกับไฟล์ **หมายเหตุ:** ซึ่งแตกต่างจาก หมายเลข inode ของไฟล์	0
path	path_t (ชนิดข้อมูลแบบใหม่ใน VUE)	ระบุพาธไฟล์ที่สมบูรณ์ซึ่งสามารถพิมพ์ได้โดยใช้ printf() และตัวระบุรูปแบบ %p	null string เป็นพาธไฟล์
error	int	ถ้าการดำเนินการอ่าน/เขียนล้มเหลว นั่นเป็นหมายเลขข้อผิดพลาดที่นิยามไว้ในไฟล์ส่วนหัว errno.h ที่เอ็กซ์พอร์ต ถ้าไม่มีข้อผิดพลาด ค่าจะเป็น 0	-1

ตัวแปร built-in __lvol

คุณสามารถใช้ตัวแปร built-in พิเศษ __lvol เพื่อดู ข้อมูลต่างๆ เกี่ยวกับโลจิคัลวอลุ่มในการดำเนินการ LVM ซึ่งพร้อมใช้งานในโพรบของชนิดย่อย lvm อิลิเมนต์สมาชิก สามารถเข้าถึงได้โดยใช้ไวยากรณ์ __lvol->member

หมายเหตุ: เมื่อใดก็ตามที่ไม่ได้ขอรับค่าจริงไว้ ค่าซึ่งทำเรื่องหมายเป็น Invalid Value จะถูกส่งคืน ซึ่งอาจเป็นเพราะเหตุผลต่อไปนี้สำหรับการขอรับค่าที่ไม่ถูกต้องนี้:

- บริบทข้อบกพร่องของเพจจำเป็นต้องมี แต่ค่าที่สามารถปรับได้ probevctrl ปัจจุบัน num_pagefaults มีค่า 0 หรือไม่เพียงพอ
- ตำแหน่งของหน่วยความจำที่มีค่าที่เพจเอาต์
- ข้อผิดพลาดระบบที่รุนแรง เช่น ตัวชี้ที่ไม่ถูกต้องหรือหน่วยความจำล้มเหลว

ตัวแปรบิวด์อิน __lvol มีสมาชิกต่อไปนี้:

ตารางที่ 41. สมาชิกตัวแปรบิวด์อิน __lvol.

ชื่อของสมาชิก	ชนิด	คำอธิบาย	ค่าที่ไม่ถูกต้อง
ชื่อ	char *	ชื่อของโลจิคัลวอลุ่ม	null string
devnum	unsigned long long	หมายเลขอุปกรณ์ของโลจิคัลวอลุ่ม ซึ่งมีทั้งหมายเลขหลักและหมายเลขรอง ที่ฝังไว้	0
major_num	int	หมายเลขหลักของโลจิคัลวอลุ่ม	-1
minor_num	int	หมายเลขรองของโลจิคัลวอลุ่ม	-1
lv_options	unsigned int	อ็อพชันที่เกี่ยวข้องกับโลจิคัลวอลุ่ม ค่าต่อไปนี้ ถูกกำหนดเป็นค่าคงที่ built-in: • LV_RDONLY (โลจิคัลวอลุ่มแบบอ่านอย่างเดียว) • LV_NOMWC (ไม่มีการตรวจสอบความสอดคล้องกันของการเขียนมิเรอร์) • LV_ACTIVE_MWC (ความสอดคล้องกันของการเขียนมิเรอร์ที่แอ็คทีฟ) • LV_PASSIVE_MWC (ความสอดคล้องกันของการเขียนมิเรอร์ที่พาสซีฟ) • LV_SERIALIZE_IO (I/O ที่ทำเป็นอนุกรม) • LV_DMPDEV (LV นี้เป็นอุปกรณ์ดัมพ์) คุณสามารถตรวจสอบได้ว่าหนึ่งในค่าเหล่านี้ถูกตั้งค่าไว้โดยให้มีเงื่อนไข เช่น `__lvol->lv_options & LV_RDONLY` **หมายเหตุ:** ค่าที่อาจเป็นไปได้ทั้งหมด ไม่ได้ถูกนิยามไว้ และอ็อพชันอื่นอาจพร้อมใช้งานในค่า	0xFFFFFFFF

ตัวแปร built-in__volgrp

คุณ สามารถใช้ตัวแปร built-in พิเศษ `__volgrp` เพื่อดู ข้อมูลต่างๆ เกี่ยวกับกลุ่มวอลุ่มในการดำเนินการ LVM ซึ่งพร้อมใช้งานในโพรบของชนิดย่อย `lvm` อิลิเมนต์สมาชิก สามารถเข้าถึงได้โดยใช้ไวยากรณ์ `__volgrp->member`

หมายเหตุ: เมื่อใดก็ตามที่ไม่สามารถขอรับค่าจริงได้ ค่าที่ทำเครื่องหมายเป็น `Invalid Value` จะถูกส่งคืน ค่าอาจเป็นค่าที่ไม่ถูกต้องเนื่องจากเหตุผลต่อไปนี้:

- บริบทข้อบกพร่องของเพจจำเป็นต้องมี แต่ค่าที่สามารถปรับได้ `probevctrl` ปัจจุบัน `num_pagefaults` มีค่า 0 หรือไม่เพียงพอ

- ตำแหน่งของหน่วยความจำที่มีค่าที่เพจเอาต์
- ข้อผิดพลาดระบบที่รุนแรง เช่น ตัวชี้ที่ไม่ถูกต้องหรือหน่วยความจำล้มเหลว

ตัวแปรบิวด์อิน `__volgrp` มีสมาชิกต่อไปนี้:

ตารางที่ 42. สมาชิกตัวแปรบิวด์อิน __volgrp.

ชื่อของสมาชิก	ชนิด	คำอธิบาย	ค่าที่ไม่ถูกต้อง
ชื่อ	char *	ชื่อของกลุ่มวอลุ่ม	null string
devnum	unsigned long long	หมายเลขอุปกรณ์ของกลุ่มวอลุ่ม ซึ่งมีหมายเลขหลักและหมายเลขรองที่ฝังไว้	0
major_num	int	หมายเลขหลักของกลุ่มวอลุ่ม	-1
minor_num	int	หมายเลขรองของกลุ่มวอลุ่ม **หมายเหตุ:** สำหรับกลุ่มวอลุ่ม AIX ยังกำหนดค่า 0 เป็นหมายเลขรอง	-1
num_open_lvs	int	จำนวนของโลจิคัลวอลุ่มที่เป็นของกลุ่มวอลุ่มนี้	-1

ตัวแปร built-in `__diskinfo`

คุณ สามารถใช้ตัวแปร built-in พิเศษ `__diskinfo` เพื่อดู ข้อมูลเกี่ยวกับดิสก์ในการดำเนินการกับดิสก์ I/O ซึ่งพร้อมใช้งานในโพรบของดิสก์ชนิดย่อย อิลิเมนต์สมาชิกสามารถเข้าถึงได้ โดยใช้ไวยากรณ์ `__diskinfo->member`

หมายเหตุ: เมื่อใดก็ตามที่ไม่สามารถขอรับค่าจริงได้ ค่าที่ถูกทำงานเครื่องหมายเป็น “Invalid Value” จะถูกส่งคืน ซึ่งอาจเป็นเพราะเหตุผลต่อไปนี้ สำหรับการขอรับค่านี้:

- บริบทข้อบกพร่องของเพจจำเป็นต้องมี แต่ค่าที่สามารถปรับได้ `probevctrl` ปัจจุบัน `num_pagefaults` มีค่า 0 หรือไม่เพียงพอ
- ตำแหน่งของหน่วยความจำที่มีค่าที่เพจเอาต์
- ข้อผิดพลาดระบบที่รุนแรง เช่น ตัวชี้ที่ไม่ถูกต้องหรือหน่วยความจำล้มเหลว

ตัวแปรบิวด์อิน `__diskinfo` มีสมาชิกต่อไปนี้:

ตารางที่ 43. สมาชิกตัวแปรบิวด์อิน __diskinfo.

ชื่อของสมาชิก	ชนิด	คำอธิบาย	ค่าที่ไม่ถูกต้อง
ชื่อ	char *	ชื่อของดิสก์	null string.
devnum	unsigned long long	หมายเลขอุปกรณ์ของดิสก์ ซึ่งมีหมายเลขหลักและหมายเลขรอง ที่ฝังไว้	0
major_num	int	หมายเลขหลักของดิสก์	-1
minor_num	int	หมายเลขรองของดิสก์	-1
lun_id	unsigned long long	Logical Unit Number (LUN) สำหรับดิสก์	0xFFFFFFFFFFFFFFFF

ตารางที่ 43. สมาชิกตัวแปรบิวด์อิน__diskinfo (ต่อ).

ชื่อของสมาชิก	ชนิด	คำอธิบาย	ค่าที่ไม่ถูกต้อง
transport_type	int	ชนิดการขนส่งของดิสก์ซึ่งสามารถตรงกับหนึ่งในค่าคงที่ built-in ต่อไปนี้: • T_FC (ไฟเบอร์แชนแนล) • T_ISCSI (iSCSI) • T_VSCSI (SCSI เสมือน) • T_SAS (SCSI ที่พ่วงต่อแบบอนุกรม)	-1
queue_depth	int	ความลึกของคิวของดิสก์ซึ่งบ่งชี้จำนวนสูงสุดของคำร้องขอ I/O แบบพร้อมเพียงกัน ที่ไดร์เวอร์ของดิสก์สามารถส่งผ่านไปยังเลเยอร์ที่ต่ำกว่า (ตัวอย่างเช่น อะแด็ปเตอร์) ถ้าหมายเลขของคำร้องขอ I/O ขาเข้ามากกว่า queue_depth คำร้องขอจะถูกจัดการแตกต่างกัน คำร้องขอพิเศษจะถูกจัดการโดยไดร์เวอร์ของดิสก์ในคิวที่รอจนกว่าเลเยอร์ที่อยู่ต่ำกว่าจะตอบสนองกับคำร้องขอ I/O อย่างน้อยหนึ่งคำร้องขอ	-1
cmds_out	int	จำนวนของคำสั่ง I/O ที่ค้างอยู่ที่ร้องขอไปยังเลเยอร์ที่อยู่ต่ำกว่า (ตัวอย่างเช่น อะแด็ปเตอร์)	-1
path_count	int	จำนวนของพาธ MPIO ของดิสก์ (หากดิสก์มีความสามารถ MPIO หรือมีค่า 0)	-1
reserve_policy	int	นโยบายการจอง SCSI ของดิสก์ซึ่งสามารถตรงกับหนึ่งในค่าของค่าคงที่ built-in ต่อไปนี้: • DK_NO_RESERVE (no_reserve) • DK_SINGLE_PATH (single_path) • DK_PR_EXCLUSIVE (PR_exclusive) • DK_PR_SHARED (PR_shared) โปรดอ้างถึงเอกสารคู่มือ AIX MPIO เพื่อศึกษาเพิ่มเติมเกี่ยวกับ นโยบายการจอง	-1

ตารางที่ 43. สมาชิกตัวแปรบิวด์อิน __diskinfo (ต่อ).

ชื่อของสมาชิก	ชนิด	คำอธิบาย	ค่าที่ไม่ถูกต้อง
scsi_flags	int	แฟล็ก SCSI ของดิสก์ ค่าแฟล็ก built-in ต่อไปนี้ ถูกกำหนดไว้: • SC_AUTOSENSE_ENABLED (มีข้อผิดพลาด เป้าหมายส่งข้อมูลที่สำคัญในการตอบสนอง Initiator ไม่จำเป็นต้องส่งคำสั่งคำร้องขอ ที่สำคัญ) • SC_NACA_1_ENABLED (ACA ปกติถูกเปิดใช้งานและมีเป้าหมายไปยังสถานะ ACA หากกำลังส่งคืน เงื่อนไขการตรวจสอบ) • SC_64BIT_IDS (64 บิต SCSI ID และ logical unit number (LUN) • SC_LUN_RESET_ENABLED (คำสั่งรีเซ็ต LUN สามารถส่งออกได้) • SC_PRIORITY_SUP (อุปกรณ์สนับสนุนลำดับความสำคัญ I/O) • SC_CACHE_HINT_SUP (อุปกรณ์สนับสนุนคำแนะนำเกี่ยวกับแคช) • SC_QUEUE_UNTAGGED (อุปกรณ์สนับสนุนการจัดคิวของคำสั่งที่ไม่ได้แท็กไว้) **หมายเหตุ:** ค่าแฟล็กทั้งหมดไม่ได้นิยามไว้ การมีอยู่ของแฟล็กอื่นๆ อาจพร้อมใช้งานใน ค่า	0

ตัวแปร built-in__diskcmd

คุณ สามารถใช้ตัวแปร built-in พิเศษ __diskcmd เพื่อดูข้อมูลต่างๆ เกี่ยวกับคำสั่ง SCSI I/O สำหรับการดำเนินการ ปัจจุบัน ซึ่งพร้อมใช้งานในโพรบของดิสก์ชนิดย่อย (แต่เฉพาะเหตุการณ์ iostart และ iodone เท่านั้น) อิลิเมนต์สมาชิกสามารถเข้าถึงได้โดยใช้ไวยากรณ์ __diskcmd->member

หมายเหตุ: เมื่อใดก็ตามที่ไม่สามารถขอรับค่าจริงได้ ค่าที่ถูกทำงานเครื่องหมายเป็น “Invalid Value” จะถูกส่งคืน ซึ่งอาจเป็นเพราะเหตุผลต่อไปนี้ สำหรับการขอรับค่า:

- บริบทข้อบกพร่องของเพจจำเป็นต้องมี แต่ค่าที่สามารถปรับได้ **probevctrl** ปัจจุบัน num_pagefaults มีค่า 0 หรือไม่เพียงพอ
- ตำแหน่งของหน่วยความจำที่มีค่าที่เพจเอาต์

- ข้อผิดพลาดระบบที่รุนแรง เช่น ตัวชี้ที่ไม่ถูกต้องหรือหน่วยความจำล้มเหลว

ตัวแปร built-in__diskcmd มีสมาชิกต่อไปนี้:

ตารางที่ 44. สมาชิกตัวแปรบิวด์อิน __diskcmd.

ชื่อของสมาชิก	ชนิด	คำอธิบาย
cmd_type	int	ชนิดของคำสั่ง SCSI (ทั้งชนิดและชนิดย่อยจะถูกผสานเข้าด้วยกัน) ค่าของค่าคงที่ built-in ต่อไปนี้พร้อมใช้งานเป็น ชนิดของคำสั่ง: • DK_BUF (การอ่าน/เขียน I/O ปกติ) • DK_IOCTL (ioctl) • DK_REQSNS (ร้องขอข้อมูลสำคัญ) • DK_TGT_LUN_RST (เป้าหมายหรือรีเซ็ต LUN) • DK_TUR (หน่วยการทดสอบเท่านั้น) • DK_INQUIRY (คำถาม) • DK_RESERVE (SCSI-2 RESERVE เวอร์ชัน 6 ไบต์) • DK_RELEASE (SCSI-2 RELEASE เวอร์ชัน 6 ไบต์) • DK_RESERVE_10 (SCSI-2 RESERVE เวอร์ชัน 10 ไบต์) • DK_RELEASE_10 (SCSI-2 RELEASE เวอร์ชัน 10 ไบต์) • DK_PR_RESERVE (SCSI-3 Persistent Reserve, RESERVE) • DK_PR_RELEASE (SCSI-3 Persistent Reserve, RELEASE) • DK_PR_CLEAR (SCSI-3 Persistent Reserve, CLEAR) • DK_PR_PREEMPT (SCSI-3 Persistent Reserve, PREEMPT) • DK_PR_PREEMPT_ABORT (SCSI-3 Persistent Reserve, PREEMPT AND ABORT) • DK_READCAP (READ CAPACITY เวอร์ชัน 10 ไบต์) • DK_READCAP16 (READ CAPACITY เวอร์ชัน 16 ไบต์) **หมายเหตุ:** ค่าคงที่ built-in เป็นค่าตำแหน่งบิต และ ดังนั้น การมีอยู่ต้องถูกตรวจสอบโดยใช้ตัวดำเนินการ '&' (ต้องไม่ใช้ตัวดำเนินการ '==') ตัวอย่างเช่น: `__diskcmd->cmd_type & DK_IOCTL`

ตารางที่ 44. สมาชิกตัวแปรบิวด์อิน__diskcmd (ต่อ).

ชื่อของสมาชิก	ชนิด	คำอธิบาย
retry_count	int	บ่งชี้ว่าให้ลองคำสั่ง I/O อีกครั้งหลังจากที่เกิดความล้มเหลว **หมายเหตุ:** ค่า 1 หมายถึงค่านั้นเป็นความพยายามในครั้งแรก ค่าที่สูงกว่าใดๆ บ่งชี้การทดลองใหม่อีกครั้ง
path_switch_count	int	บ่งชี้จำนวนครั้งที่พาธถูกเปลี่ยนสำหรับการดำเนินการ I/O นี้โดยเฉพาะ (โดยปกติแล้ว จะบ่งชี้ความล้มเหลวของพาธ I/O บางส่วน ชั่วคราวหรือถาวร)
status_validity	int	ในกรณีที่มีข้อผิดพลาดใดๆ ค่านี้จะบ่งชี้ว่ามีข้อผิดพลาด SCSI หรือข้อผิดพลาดเกี่ยวกับอะแด็ปเตอร์ ซึ่งสามารถตรงกับหนึ่งในค่าของค่าคงที่ built-in ต่อไปนี้: SC_SCSI_ERROR or SC_ADAPTER_ERROR ถ้าไม่มีข้อผิดพลาด ค่านั้นจะเป็น 0
scsi_status	int	ถ้าฟิลด์ status_validity ถูกตั้งค่าเป็น SC_SCSI_ERROR ฟิลด์นี้จะกำหนดรายละเอียดเพิ่มเติมเกี่ยวกับ ข้อผิดพลาด ซึ่งสามารถตรงกับหนึ่งใน ค่าของค่าคงที่ built-in: • SC_GOOD_STATUS (ภารกิจเสร็จสิ้นอย่างสมบูรณ์) • SC_CHECK_CONDITION (ข้อผิดพลาดบางส่วน ข้อมูลที่สำคัญจัดเตรียมข้อมูลเพิ่มเติมไว้) • SC_BUSY_STATUS (LUN ไม่ว่าง ไม่สามารถยอมรับคำสั่งได้) • SC_RESERVATION_CONFLICT (การฝ่าฝืนการจอง SCSI ที่มีอยู่) • SC_COMMAND_TERMINATED (อุปกรณ์จบคำสั่ง) • SC_QUEUE_FULL (คิวอุปกรณ์เต็ม) • SC_ACA_ACTIVE (อุปกรณ์อยู่ในสเตจ Auto Contingent Allegiance) • SC_TASK_ABORTED (อุปกรณ์หยุดคำสั่ง) **หมายเหตุ:** ค่าที่เป็นไปได้ไม่ได้ถูกนิยามไว้ ดังนั้น SC_SCSI_ERROR สามารถมีค่าที่อาจไม่ตรงกับค่า built-in ใดๆ คุณสามารถมองหาโค้ดตอบกลับคำสั่ง SCSI ที่สอดคล้องกันได้

ตารางที่ 44. สมาชิกตัวแปรบิวด์อิน __diskcmd (ต่อ).

ชื่อของสมาชิก	ชนิด	คำอธิบาย
adapter_status	int	ถ้าฟิลด์ status_validity ถูกตั้งค่าเป็น SC_ADAPTER_ERROR ฟิลด์นี้จัดเตรียมข้อมูลเพิ่มเติม เกี่ยวกับข้อผิดพลาด ซึ่งสามารถตรงกับหนึ่งในค่าคงที่ built-in ต่อไปนี้: • ADAP_HOST_IO_BUS_ERR (ข้อผิดพลาดเกี่ยวกับโฮสต์ I/O) • ADAP_TRANSPORT_FAULT (ข้อผิดพลาดเกี่ยวกับเลเยอร์การขนส่ง) • ADAP_CMD_TIMEOUT (คำสั่ง I/O หมดเวลาใช้งาน) • ADAP_NO_DEVICE_RESPONSE (ไม่มีการตอบกลับจากอุปกรณ์) • ADAP_HDW_FAILURE (ความล้มเหลวของฮาร์ดแวร์อะแด็ปเตอร์) • ADAP_SFW_FAILURE (ความล้มเหลวของไมโครโค้ดอะแด็ปเตอร์) • ADAP_TRANSPORT_RESET (อะแด็ปเตอร์ที่ตรวจพบการรีเซ็ตบัส SCSI ภายนอก) • ADAP_TRANSPORT_BUSY (เลเยอร์การขนส่งไม่ว่าง) • ADAP_TRANSPORT_DEAD (เลเยอร์การขนส่งไม่ดำเนินการ) • ADAP_TRANSPORT_MIGRATED (เลเยอร์การขนส่งถูกโอนย้าย) • ADAP_FUSE_OR_TERMINAL_PWR (อะแด็ปเตอร์ที่ใช้ฟิวส์หรือมีการยกเลิกการใช้ไฟฟ้าที่ล้มเหลว)

ตัวแปร built-in__iopath

คุณ สามารถใช้ตัวแปร built-in พิเศษ __iopath เพื่อดูข้อมูลต่างๆ เกี่ยวกับพาธ I/O สำหรับการดำเนินการปัจจุบัน ซึ่งพร้อมใช้งานในโพรบของดิสก์ชนิดย่อยสำหรับเหตุการณ์ iostart และ iodone เท่านั้น อิลิเมนต์สมาชิกสามารถเข้าถึงได้โดยใช้ไวยากรณ์ __iopath->member

หมายเหตุ: เมื่อใดก็ตามที่ไม่ได้ขอรับค่าจริงไว้ ค่าซึ่งทำเรื่องหมายเป็น Invalid Value จะถูกส่งคืน ซึ่งอาจเป็นเพราะเหตุผลต่อไปนี้ สำหรับการขอรับค่านี้:

- บริบทข้อบกพร่องของเพจจำเป็นต้องมี แต่ค่าที่สามารถปรับได้ **probevctrl** ปัจจุบัน num_pagefaults มีค่า 0 หรือไม่เพียงพอ
- ตำแหน่งของหน่วยความจำที่มีค่าที่เพจเอาต์
- ข้อผิดพลาดระบบที่รุนแรง เช่น ตัวชี้ที่ไม่ถูกต้องหรือหน่วยความจำล้มเหลว

__iopath มีสมาชิกต่อไปนี้:

ตารางที่ 45. สมาชิกตัวแปรแบบบิวด์อิน __iopath.

ชื่อของสมาชิก	ชนิด	คำอธิบาย	ค่าที่ไม่ถูกต้อง
path_id	int	ID ของพาธปัจจุบัน (เริ่มต้นจาก 0)	-1
scsi_id	unsigned long long	SCSI ID ของเป้าหมายบนพาธนี้	0xFFFFFFFFFFFFFFFF
lun_id	unsigned long long	Logical Unit Number (LUN) บนพาธนี้	0xFFFFFFFFFFFFFFFF
ww_name	unsigned long long	ชื่อสากลของพอร์ตเป้าหมายบนพาธนี้	0
cmds_out	int	จำนวนของคำสั่ง I/O ที่ค้างอยู่บนพาธนี้	-1

ตัวแปร built-in__j2info

__j2info เป็นตัวแปร built-in พิเศษ ที่คุณสามารถใช้เพื่อดูข้อมูลต่างๆ เกี่ยวกับการดำเนินการของระบบไฟล์ JFS2 ซึ่งพร้อมใช้งานในโพรบของชนิดย่อย jfs2 อิลิเมนต์สมาชิกสามารถเข้าถึงได้โดยใช้ไวยากรณ์ __j2info->member

หมายเหตุ: เมื่อใดก็ตามที่ค่าจริงไม่สามารถ ขอรับได้ ค่าซึ่งทำเครื่องหมายเป็น Invalid Value จะถูกส่งคืน ซึ่งอาจเป็นเพราะเหตุผลต่อไปนี้ สำหรับการขอรับค่านี้:

- บริบทข้อบกพร่องของเพจจำเป็นต้องมี แต่ค่าที่สามารถปรับได้ probevctrl ปัจจุบัน num_pagefaults มีค่า 0 หรือไม่เพียงพอ
- ตำแหน่งของหน่วยความจำที่มีค่าที่เพจเอาต์
- ข้อผิดพลาดระบบที่รุนแรง เช่น ตัวชี้ที่ไม่ถูกต้องหรือหน่วยความจำล้มเหลว

__j2info มีสมาชิกต่อไปนี้:

ตารางที่ 46. สมาชิกตัวแปรบิวด์อิน __j2info.

ชื่อของสมาชิก	ชนิด	คำอธิบาย	ค่าที่ไม่ถูกต้อง
inode_id	unsigned long long	หมายเลขเฉพาะสากลที่เชื่อมโยงกับไฟล์ของการดำเนินการปัจจุบัน **หมายเหตุ:** ซึ่งแตกต่างจาก หมายเลข inode ของไฟล์	0
f_type	int	ประเภทของไฟล์ คำอธิบาย __file->f_type จัดเตรียมค่า ที่เป็นไปได้	-1
mount_path	char *	พาธที่เมาท์ระบบไฟล์	null string.
devnum	unsigned long long	หมายเลขอุปกรณ์ของอุปกรณ์บล็อกของระบบไฟล์ ซึ่งมีทั้งหมายเลขหลักและหมายเลขรองที่ผังไว้	0
major_num	int	หมายเลขหลักของอุปกรณ์บล็อกของระบบไฟล์	-1

ตารางที่ 46. สมาชิกตัวแปรบิวต์อิน __j2info (ต่อ).

ชื่อของสมาชิก	ชนิด	คำอธิบาย	ค่าที่ไม่ถูกต้อง
minor_num	int	หมายเลขรองของอุปกรณ์บล็อกของระบบไฟล์	-1
l_blknum	unsigned long long	หมายเลขบล็อกแบบโลจิคัลสำหรับการดำเนินการกับไฟล์นี้	0xFFFFFFFFFFFFFFFF
l_bcount	unsigned long long	จำนวนไบต์ที่ร้องขอระหว่างบล็อกโลจิคัลในการดำเนินการนี้	0xFFFFFFFFFFFFFFFF
child_bufid	unsigned long long	bufid ของบัฟเฟอร์คำร้องขอ I/O ที่ส่งลงไปยังเลเยอร์ที่ต่ำกว่า (ตัวอย่างเช่น LVM) ในเลเยอร์นั้น จะปรากฏเป็น __iobuf->bufid	0
child_blknum	unsigned long long	หมายเลขบล็อกของบัฟเฟอร์คำร้องขอ I/O ที่ส่งลงไปยังเลเยอร์ที่ต่ำกว่า (ตัวอย่างเช่น LVM) ในเลเยอร์นั้น จะปรากฏขึ้นเป็น __iobuf->blknum	0xFFFFFFFFFFFFFFFF
child_bcount	unsigned long long	จำนวนไบต์ของบัฟเฟอร์คำร้องขอ I/O ที่ส่งลงไปยังเลเยอร์ที่ต่ำกว่า (ตัวอย่างเช่น LVM) ในเลเยอร์นั้น จะปรากฏขึ้นเป็น __iobuf->bcount	0xFFFFFFFFFFFFFFFF
child_bflags	unsigned long long	แฟล็กของบัฟเฟอร์คำร้องขอ I/O ที่ส่งลงไปยังเลเยอร์ที่ต่ำกว่า (ตัวอย่างเช่น LVM) ในเลเยอร์นั้น จะปรากฏขึ้นเป็น __iobuf->bflags	0

ตัวอย่างสคริปต์สำหรับโปรแกรมจัดการโพรบ I/O

1. สคริปต์เพื่อติดตามการดำเนินการเขียนใดๆ ไปยังไฟล์ /etc/passwd:

```
int write(int, char *, int);
@@BEGIN  {
        target_inodeid = fpath_inodeid("/etc/passwd");
}
@@syscall:*:write:entry {
        if (fd_inodeid(__arg1) == target_inodeid) {
                printf("write on /etc/passwd: timestamp=%A, pid=%lld, pname=[%s], uid=%lld\n",
                        timestamp(), __pid, __pname, __uid);
        }
}
```

ถ้าสคริปต์อยู่ในไฟล์ VUE ที่ชื่อ etc_passwd.e สคริปต์สามารถรันดังนี้:

```
# probevue etc_passwd.e
```

ในเทอร์มินัลอื่น หากผู้ใช้ (root) รัน:

```
# mkuser user1
```

ดังนั้น probevue จะแสดงเอาต์พุตที่คล้ายกับตัวอย่างต่อไปนี้:

```
write on /etc/passwd: timestamp=Mar/03/15 16:10:07, pid=14221508, pname=[mkuser], uid=0
```

2. สคริปต์เพื่อค้นหาเวลาการดำเนินการ I/O สูงสุดและต่ำสุดสำหรับดิสก์ (ตัวอย่างเช่น hdisk0) ใช่วงระยะเวลา และค้นหาหมายเลขบล็อก จำนวนไบต์ที่ร้องขอ เวลาของการดำเนินการ และชนิดของการดำเนินการ (อ่านหรือเขียน) ที่สอดคล้องกับเวลาสูงสุดหรือต่ำสุด

```
long long min_time, max_time;
@@BEGIN {
        min_time = max_time = 0;
}
@@io:disk:entry:*:hdisk0 {
        ts_entry[__iobuf->bufid] = (long long)timestamp();
}
@@io:disk:exit:*:hdisk0 {
        if (ts_entry[__iobuf->bufid]) { /* only if we recorded entry time */
                ts_now = timestamp();
                op_type = (__iobuf->bflags & B_READ) ? "READ" : "WRITE";
                dt = (long long)diff_time(ts_entry[__iobuf->bufid], ts_now, MICROSECONDS);
                if (min_time == 0 || dt < min_time) {
                        min_time = dt;
                        min_blknum = __iobuf->blknum;
                        min_bcount = __iobuf->bcount;
                        min_ts = ts_now;
                        min_optype = op_type;
                }
                if (max_time == 0 || dt > max_time) {
                        max_time = dt;
                        max_blknum = __iobuf->blknum;
                        max_bcount = __iobuf->bcount;
                        max_ts = ts_now;
                        max_optype = op_type;
                }
                ts_entry[__iobuf->bufid] = 0;
        }
}
@@END {
        printf("Maximum and minimum IO operation time for [hdisk0]:\n");
        printf("Max: %lld usec, block=%lld, byte count=%lld, operation=%s, time of operation=[%A]\n",
                max_time, max_blknum, max_bcount, max_optype, max_ts);
        printf("Min: %lld usec, block=%lld, byte count=%lld, operation=%s, time of operation=[%A]\n",
                min_time, min_blknum, min_bcount, min_optype, min_ts);
}
```

อนุญาตให้สคริปต์นี้ต้องอยู่ในไฟล์ VUE ที่ชื่อ disk_min_max_time.e ซึ่งสามารถเรียกทำงานได้ดังนี้:

```
# probevue disk_min_max_time.e
```

อนุญาตให้เป็นกิจกรรม IO บางอย่างบน hdisk0 (คำสั่ง dd ที่สามารถใช้ได้)
หลังจากเวลาผ่านไปไม่นาน หากคำสั่งข้างต้นถูกยกเลิก (โดยกดปุ่ม CTRL-C) ระบบจะพิมพ์เอาต์พุตที่คล้ายกับที่แสดงดังนี้:

```
^CMaximum and minimum IO operation time for [hdisk0]:
Max: 48174 usec, block=6927976, byte count=4096, operation=READ, time of operation=[Mar/04/15 03:31:07]
Min: 133 usec, block=6843288, byte count=4096, operation=READ, time of operation=[Mar/04/15 03:31:03]
```

โปรแกรมจัดการโพรบเครือข่าย

โปรแกรมจัดการโพรบเครือข่ายแทร็กแพ็กเก็ต เครือข่ายขาเข้าและขาออกในระบบ (ข้อมูลแพ็กเก็ตถูกตีความโดยโมดูล bpf ใน AIX) ข้อมูลจำเพาะโพรบ อนุญาตให้ผู้ใช้ระบุตัวกรอง Berkeley Packet Filter (BPF) เช่นเดียวกับ นิพจน์ตัวกรอง tcpdump สำหรับการติดตามขนาดเล็ก

คุณสามารถ ใช้ตัวแปร built-in เพื่อรวบรวมส่วนหัวแพ็กเก็ต และข้อมูลเพย์โหลด สำหรับอินเตอร์เน็ตโปรโตคอล ตัวอย่างเช่น โปรโตคอล Ethernet, Internet Protocol Version 4/Version 6 (IPv4/v6), Transmission Control Protocol (TCP), User Datagram Protocol (UDP), Internet Control Message Protocol (ICMP), Internet Group Message Protocol (IGMP), and Address Resolution Protocol (ARP)

โปรแกรมจัดการเครือข่ายรายงานเหตุการณ์ที่ระบุเฉพาะโปรโตคอลที่รุนแรง (การเปลี่ยนสถานะ TCP เวลาแบบไปกลับ การส่งใหม่โอเวอร์โฟลว์บัฟเฟอร์ UDP)

โปรแกรมจัดการโพรบเครือข่ายกำหนดกรณีการใช้งานหลัก ดังต่อไปนี้:

- จัดเตรียมข้อมูลที่ระบุเฉพาะแพ็กเก็ตตามโมดูล bpf อ้างอิง IP address และพอร์ต:
  - แท็กไบต์ขาเข้าและขาออกสำหรับการเชื่อมต่อ
  - ใช้ built-in เพื่อรวบรวมส่วนหัวของโปรโตคอลและ ข้อมูลเพย์โหลด
    - แฟล็ก TCP (SYN, FIN), ลำดับของ TCP และหมายเลขการตอบรับ
    - IPv4/IPv6 (IP แอดเดรส, ชนิดโปรโตคอล: tcp, udp, icmp, igmp และอื่นๆ)
    - ICMP (ชนิดแพ็กเก็ต: ECHO REQUEST, ECHO RESPONSE และอื่นๆ)
- จัดเตรียมการเข้าถึงเพื่อทำให้แพ็กเก็ตเครือข่าย RAW สมบูรณ์สำหรับการประมวลผลสคริปต์โพรบ
- รายงานเหตุการณ์ที่เกี่ยวข้องกับโปรโตคอล:
  - แท็กเหตุการณ์บัฟเฟอร์ผู้ส่งและผู้รับ TCP ที่เต็ม
  - สถานะการเชื่อมต่อ TCP เปลี่ยนแปลงจากสถานะ SYN-SENT ไปเป็นสถานะ ESTABLISHED หรือจากสถานะ ESTABLISHED ไปเป็น CLOSE
  - มอนิเตอร์เวลาที่เปลี่ยนแปลงระหว่างการเปลี่ยนสถานะ (ตัวอย่างเช่น เวลาที่ใช้จากสถานะ SYN-SENT ไปเป็นสถานะ ESTABLISHED)
  - ระบุ listener (ข้อมูลการเชื่อมต่อ) ที่ละเว้นการเชื่อมต่อเนื่องจากคิวของ listener เต็ม
  - ระบุการส่งใหม่อีกครั้ง (การส่งใหม่ตัวที่สองและเพิ่มเติมสำหรับแพ็กเก็ต) สำหรับการเชื่อมต่อ TCP
  - ระบุซ็อกเก็ต UDP ที่ปล่อยแพ็กเก็ตเนื่องจากบัฟเฟอร์การรับไม่เพียงพอ

ข้อกำหนดคุณสมบัติโพรบ

ข้อมูลจำเพาะของโพรบสำหรับโปรแกรมโพรบเครือข่าย มีสามถึงห้า tuple ที่คั่นด้วย : (โคลอน) tuple ตัวแรกเป็น `@@net` เสมอ

โปรแกรมจัดการโพรบเครือข่ายสนับสนุนสองหมวดหมู่หลักของข้อมูลจำเพาะ: หนึ่งหมวดหมู่จะรวบรวมข้อมูลที่ระบุเฉพาะแพ็กเก็ตและอีกหนึ่งหมวดหมู่รวบรวมข้อมูล ที่ระบุเฉพาะโปรโตคอล

- จัดรูปแบบเพื่อรวบรวมข้อมูลที่ระบุเฉพาะแพ็กเก็ต:

  ```
  @@net:bpf:<interface1>|<interface 2>|.....:<protocol>:<Filter>
  ```

- จัดรูปแบบเพื่อรวบรวมข้อมูลที่ระบุเฉพาะโปรโตคอล

  ```
  @@net:tcp:<event_name>
  @@net:udp:<event_name>
  ```

ชนิดย่อยของโพรบ

tuple ตัวที่สองแสดงหมายถึงชนิดย่อยของโพรบ ที่บ่งชี้ว่าเลเยอร์ของสแต็กเครือข่าย AIX มีโพรบ tuple นี้สามารถมีหนึ่งในค่าต่อไปนี้ (ซึ่งไม่สามารถเป็น *):

ตารางที่ 47. ข้อมูลจำเพาะของ tuple ตัวที่สองสำหรับชนิดย่อยโพรบ.

Tuple ตัวที่สอง (ชนิดย่อย)	**คำอธิบาย**
bpf	โพรบนี้เริ่มต้นทำงานที่เลเยอร์อินเตอร์เฟสเครือข่ายเมื่อแพ็กเก็ตตรงกับตัวกรองที่ระบุเฉพาะ
tcp	โพรบนี้เริ่มต้นสำหรับเหตุการณ์ที่ระบุเฉพาะโปรโตคอล TCP
udp	โพรบนี้เริ่มต้นสำหรับเหตุการณ์ที่ระบุเฉพาะโปรโตคอล UDP

โพรบเหตุการณ์เครือข่ายโพรบหรือรวบรวมข้อมูลแพ็กเก็ตเครือข่าย

tuple ตัวที่สาม ระบุเฉพาะชนิดย่อย (ระบุใน tuple ตัวที่สอง) ซึ่งไม่สามารถมีค่า *

โพรบแบบอิง bpf

ข้อมูลจำเพาะมี 5 tuple สำหรับโพรบแบบอิง bpf ที่กล่าวถึงในตารางต่อไปนี้:

ตารางที่ 48. โพรบแบบอิง bpf: ข้อมูลจำเพาะของ Tuple.

tuple ตัวที่สอง (ชนิดย่อย)	**tuples ลำดับถัดมา**	**คำอธิบาย**
bpf	tuple ตัวที่สาม: ชื่ออินเตอร์เฟส	tuple นี้ระบุอินเตอร์เฟสหรือรายการของอินเตอร์เฟสที่ข้อมูลแพ็กเก็ต สามารถดักจับได้ ค่าที่เป็นไปได้ คือ `enX` (เช่น) `en0`,`en1`) และ `lo0` ค่า * ไม่ได้สนับสนุน tuple นี้ คุณสามารถระบุอินเตอร์เฟสได้มากกว่าหนึ่งในแต่ละครั้งโดยใช้	เป็นตัวคั่น
	tuple ตัวที่สี่: โปรโตคอล	tuple นี้ระบุโปรโตคอลเครือข่ายเพื่อเริ่มต้นโพรบ ค่าที่สามารถใช้งานได้คือ `ether`, `arp`, `rarp`, `ipv4`, `ipv6`, `tcp`, `udp`, `icmp4`, `icmp6` และ `igmp` built-in เฉพาะโปรโตคอลถูกเติมสำหรับการเข้าถึงในสคริปต์ Vue เช่น ค่าโปรโตคอล ของ ipv4 จะเติม built-in `__ip4hdr` ค่า * สำหรับ tuple นี้บ่งชี้ว่า โพรบเริ่มต้นสำหรับโปรโตคอลทุกชนิดที่ตรงกับตัวกรองที่ระบุไว้ เมื่อโปรโตคอล คือ * จะไม่มี ค่า built-in ที่สนับสนุนโดย network probe manager ที่พร้อมใช้งาน สำหรับสคริปต์ Vue คุณสามารถเข้าถึง ข้อมูลแพ็กเก็ตดิบโดยใช้ฟังก์ชัน Vue `copy_kdata()` และแม็พกับส่วนหัวโปรโตคอลที่สอดคล้องกัน **หมายเหตุ:** การระบุ * เป็นค่าที่สามารถให้ประสิทธิภาพการทำงานเพื่อให้โพรบเริ่มต้นสำหรับแพ็กเก็ตขาเข้าและขาออกบนอินเตอร์เฟสที่ระบุไว้ซึ่งตรงกับตัวกรอง ซึ่งยังมีสำเนาที่เกี่ยวข้องเมื่อข้อมูลแพ็กเก็ตถูกขยายระหว่างบัฟเฟอร์แพ็กเก็ตจำนวนมาก
	tuple ที่ห้า: สตริงตัวกรอง bpf	tuple นี้ระบุนิพจน์ตัวกรอง bpf (นิพจน์ตัวกรองตามที่ระบุอยู่ในคำสั่ง **tcpdump**) นิพจน์ตัวกรองต้องถูกจัดเตรียมไว้ในเครื่องหมายอัญประกาศ คู่นิพจน์ตัวกรองและโปรโตคอลที่ถูกระบุใน tuple ตัวที่สี่ต้องทำงานเข้ากันได้ ค่า * ไม่ได้รับการสนับสนุนใน tuple นี้ โปรดอ้างถึงเอกสารคู่มือ **tcpdump** สำหรับข้อมูลโดยละเอียด เกี่ยวกับนิพจนตัวกรอง

ตัวอย่าง

1. รูปแบบข้อมูลจำเพาะเพื่อเข้าถึงตัวแปร built-in ที่สัมพันธ์กับข้อมูล ส่วนหัวอีเทอร์เน็ต (`__etherhdr`), IP header (`__ip4hdr`) หรือ (`__ip6hdr`) และส่วนหัว TCP (`__tcphdr`) จากสคริปต์ Vue เมื่ออินเตอร์เฟส en0 ได้รับหรือส่งแพ็กเก็ต บนพอร์ต 23 (สตริงตัวกรอง " port 23"):

   ```
   @@net:bpf:en0:tcp:"port 23"
   ```

2. รูปแบบข้อมูลจำเพาะเพื่อเข้าถึงตัวแปร built-in ที่สัมพันธ์กับ ข้อมูลส่วนหัวอีเทอร์เน็ต (`__etherhdr`), IP header (`__ip4hdr` หรือ `__ip6hdr`), และส่วนหัว UDP (`__udphdr`) จากสคริปต์ Vue เมื่อระบบได้รับหรือส่งแพ็กเก็ตจากโฮสต์ example.com (สตริงตัวกรอง "example.com") บนอินเตอร์เฟส en0 และ en1:

   ```
   @@net:bpf:en0|en1:udp:"host example.com"
   ```

3. รูปแบบข้อมูลจำเพาะเพื่อเข้าถึงข้อมูลแพ็กเก็ตดิบเมื่อระบบรับและส่งแพ็กเก็ตจาก หรือไปยัง "host example.com":

```
@@net:bpf:en1:*:"host example.com"
```

หมายเหตุ: แต่ละข้อมูลจำเพาะของโพรบ bpf ใช้อุปกรณ์ bpf อุปกรณ์เหล่านี้ ถูกแบ่งใช้โดย ProbeVue, **tcpdump** และแอ็พพลิเคชันอื่นๆ ที่ใช้เซอร์วิส libpcap หรือ bpf สำหรับการดักจับและการอัดฉีด จำนวนของโพรบ bpf ขึ้นอยู่กับ จำนวนของอุปกรณ์ bpf ที่พร้อมใช้งานในระบบ

เมื่อเริ่มต้นโพรบ bpf แล้ว ตัวแปร *__mdata* มีข้อมูลแพ็กเก็ตดิบ คุณสามารถเข้าถึงข้อมูลดิบของขนาดที่ร้องขอ โดยใช้ฟังก์ชัน Vue copy_kdata () และแม็พกับ ether_header, ip header และอื่นๆ ใช้โครงสร้างต่อไปนี้เพื่อค้นหาส่วนหัวและข้อมูล เพย์โหลด

ตัวอย่าง

สคริปต์ VUE ที่ต้องการเข้าถึงข้อมูลแพ็กเก็ตดิบเมื่อระบุ "*" เป็นโปรโตคอล

```
 /* Define the ether header structure */
struct  ether_header {
        char  ether_dhost[6];
        char  ether_shost[6];
        short ether_type;
};

/* ProbeVue script to access and interpret the data from RAW packet */

@@net:bpf:en0:*:"port 23"
{
        /* define the script local variables */
        __auto struct ether_header eth;
        __auto char *mb;

        /* __mdata contains the address of packet data */
        mb =(char *) __mdata;
        printf("Network probevue\n");

       /*
        * Use already available "copy_kdata(...)" VUE function to copy data of
        * requested size (size of ether_header) from mbuf data pointer to eth
        * (ether_header) variable.
        */
        copy_kdata (mb, eth);
        printf("Ether Type  from raw data :%x\n",eth.ether_type);

}
```

โพรบ TCP

ข้อมูลจำเพาะมีสาม tuple สำหรับโพรบ TCP ตามที่กล่าวไว้ใน ตารางต่อไปนี้:

ตารางที่ 49. โพรบ TCP: ข้อมูลจำเพาะ Tuple.

tuple ตัวที่สอง (ชนิดย่อย)	เหตุการณ์ (tuple ตัวที่สาม) ค่า * ไม่ได้รับการสนับสนุนใน tuple นี้	คำอธิบาย
tcp	state_change	โพรบนี้เริ่มต้นขึ้นเมื่อใดก็ตามที่สถานะ TCP เปลี่ยนไป
	send_buf_full	โพรบนี้เริ่มต้นทำงานเมื่อก็ตามที่เกิดเหตุการณ์บัฟเฟอร์การส่งเต็ม
	recv_buf_full	โพรบนี้เริ่มต้นทำงานเมื่อใดก็ตามที่เกิดเหตุการณ์บัฟเฟอร์การรับเต็ม
	retransmit	โพรบนี้เริ่มต้นขึ้นเมื่อใดก็ตามการส่งใหม่ของแพ็กเก็ตที่เกิดขึ้นสำหรับการเชื่อมต่อ TCP
	listen_q_full	โพรบนี้เริ่มต้นทำงานเมื่อใดก็ตามที่เซิร์ฟเวอร์ (listener ซ็อกเก็ต) ละเว้นคำร้องขอการเชื่อมต่อใหม่ เนื่องจากคิวของ listener เต็ม

ตัวแปร built-in **__proto_info** จัดเตรียมข้อมูลการเชื่อมต่อ TCP (4 tuple) (IP โลคัล, IP รีโมต, โลคัลพอร์ต และ รีโมตพอร์ต) เมื่อเกิดเหตุการณ์ที่เกี่ยวข้องกับ TCP พอร์ตแบบรีโมตและ IP address มีค่า NULL สำหรับเหตุการณ์ listen_q_full

ตัวอย่าง

ข้อมูลจำเพาะของโพรบสำหรับการเปลี่ยนสถานะโปรโตคอล TCP:

```
@@net:tcp:state_change
```

โพรบ udp

สำหรับโพรบ udp ที่มีข้อมูลจำเพาะที่มีสาม tuple ตามที่กล่าวไว้ใน ตารางต่อไปนี้:

ตารางที่ 50. tuple ตัวที่สองของ udp: ค่า tuple ตัวที่สาม.

tuple ตัวที่สอง (ชนิดย่อย)	เหตุการณ์ (tuple ตัวที่สาม) ค่า * ไม่ได้รับการสนับสนุนใน tuple นี้	คำอธิบาย
udp	sock_recv_buf_overflow	โพรบนี้เริ่มต้นขึ้นเมื่อใดก็ตามที่เดต้าแกรมหรือโอเวอร์โฟลว์บัฟเฟอร์รับซ็อกเก็ต UDP

ตัวแปร built-in __proto_info จัดเตรียม ข้อมูลที่เกี่ยวข้องกับโปรโตคอล UDP (IP แอดเดรสต้นทางและ IP แอดเดรสปลายทาง, หมายเลขพอร์ตต้นทางและปลายทาง) เมื่อซ็อกเก็ตได้รับเหตุการณ์ บัฟเฟอร์โอเวอร์โฟลว์

```
@@net:udp:sock_recv_buf_overflow
```

ตัวอย่าง

ข้อมูลจำเพาะของโพรบ สำหรับโอเวอร์โฟลว์บัฟเฟอร์การรับซ็อกเก็ต UDP:

```
@@net:udp:sock_recv_buf_overflow
```

ตัวแปร built-in ที่เกี่ยวข้องกับโพรบเครือข่ายสำหรับสคริปต์ Vue

เหตุการณ์ที่เกี่ยวข้องกับเครือข่ายสามารถโพรบ ได้โดยใช้ตัวแปรบิวด์อินต่อไปนี้

ตัวแปร built-in__etherhdr

ตัวแปร __etherhdr เป็นตัวแปร built-in พิเศษเพื่อดูข้อมูลส่วนหัวอีเทอร์เน็ตจากแพ็กเก็ตที่ถูกกรอง ตัวแปร built-in นี้ พร้อมใช้งานเมื่อคุณโพรบข้อมูลแพ็กเก็ตที่เลเยอร์อินเตอร์เฟส ที่มีหนึ่งในโปรโตคอลเหล่านี้: “ether”, “ipv4”, “ipv6”, “tcp”, “udp”, “icmp4”, icmp6”, “igmp”, “arp” และ “rarp” ตัวแปรนี้พร้อมใช้งานในโพรบของชนิดย่อย bpf อิลิเมนต์สมาชิกสามารถเข้าถึงได้โดยใช้ไวยากรณ์ __etherhdr->member

ค่าบิวด์อิน __etherhdr มีสมาชิกต่อไปนี้:

ตารางที่ 51. สมาชิกตัวแปรบิวด์อิน __etherhdr.

ชื่อของสมาชิก	ชนิด	คำอธิบาย
src_addr	mac_addr_t	MAC address ต้นทาง ชนิดข้อมูล mac_addr_t ถูกใช้เพื่อเก็บแอดเดรส MAC address ให้ใช้ตัวระบุรูปแบบ “M” เพื่อพิมพ์ MAC address
dst_addr	mac_addr_t	MAC address ปลายทาง ชนิดข้อมูล mac_addr_t ถูกใช้เพื่อเก็บแอดเดรส MAC address ให้ใช้ตัวระบุรูปแบบ “M” เพื่อพิมพ์ MAC address
ether_type	unsigned short	ชื่อนี้บ่งชี้โปรโตคอลที่อยู่ในเพย์โหลดของกรอบอีเทอร์เน็ต โปรโตคอล สามารถเป็น IPv4, IPv6, ARP และ REVARP ซึ่งสามารถตรงกับ หนึ่งในค่าของค่าคงที่ built-in ต่อไปนี้สำหรับ ether_type: • ETHERTYPE_IP • ETHERTYPE_IPV6 • ETHERTYPE_ARP • ETHERTYPE_REVARP โปรดดูที่ไฟล์ส่วนหัว /usr/include/netinet/if_ether.h และ/usr/include/netinet/if_ether6.h สำหรับค่า ether_type

หมายเหตุ: ตัวแปรบิวด์อิน __etherhdr สามารถเรียกทำงานได้สำหรับอินเตอร์เฟสอีเทอร์เน็ต และไม่ใช่ อินเตอร์เฟสแบบลูปแบ็ก

ตัวแปร built-in __ip4hdr

ตัวแปร __ip4hdr เป็นตัวแปร built-in พิเศษ เพื่อดูข้อมูลส่วนหัว IPv4 จากแพ็กเก็ตที่ถูกกรอง ตัวแปรนี้พร้อมใช้งาน เมื่อคุณโพรบข้อมูลแพ็กเก็ตที่เลเยอร์อินเตอร์เฟสด้วยหนึ่งในโปรโตคอลใดๆ เหล่านี้: “ipv4”,“tcp”, “udp”, “icmp4” และ “igmp″ และมีข้อมูลที่ถูกต้องเมื่อเวอร์ชันของ IP คือ IPv4 ตัวแปรนี้พร้อมใช้งานในโพรบของชนิดย่อย bpf อิลิเมนต์สมาชิกสามารถเข้าถึงได้โดยใช้ไวยากรณ์ __ip4hdr->member

ตัวแปร built-in นี้มีสมาชิกต่อไปนี้:

ตารางที่ 52. สมาชิกตัวแปรบิวด์อิน __ip4hdr.

ชื่อของสมาชิก	ชนิด	คำอธิบาย
src_addr	ip_addr_t	IP address ต้นทาง ชนิดข้อมูล ip_addr_t ถูกใช้เพื่อเก็บ IP address ใช้ตัวระบุรูปแบบ “I” เพื่อพิมพ์ IP address ในรูปแบบทศนิยมแบบจุดและใช้ตัวระบุรูปแบบ “H” เพื่อพิมพ์ชื่อโฮสต์ การพิมพ์ชื่อโฮสต์เป็นการดำเนินการที่มีต้นทุน
dst_addr	ip_addr_t	IP address ปลายทาง ชนิดข้อมูล ip_addr_t ถูกใช้เพื่อเก็บ IP address ใช้ตัวระบุรูปแบบ “I” เพื่อพิมพ์ IP address ในรูปแบบทศนิยมแบบจุดและใช้ตัวระบุรูปแบบ “H” เพื่อพิมพ์ชื่อโฮสต์ การพิมพ์ชื่อโฮสต์เป็นการดำเนินการที่มีต้นทุน

โปรโตคอล	unsigned short	ชื่อสมาชิกบ่งชี้โปรโตคอลที่ใช้ในส่วนของข้อมูลของเดต้าแกรม IP โปรโตคอลสามารถเป็น TCP, UDP, ICMP, IGMP, FRAGMENTED และอื่นๆ
		
		ซึ่งสามารถตรงกับหนึ่งในค่าของค่าคงที่ built-in ต่อไปนี้สำหรับโปรโตคอลโปรโตคอล
		`IPPROTO_HOPOPTS,`
		`IPPROTO_ICMP,`
		`IPPROTO_IGMP,`
		`IPPROTO_TCP,`
		`IPPROTO_UDP,`
		`IPPROTO_ROUTING,`
		`IPPROTO_FRAGMENT,`
		`IPPROTO_NONE,`
		`IPPROTO_LOCAL`
		
		อ้างถึงไฟล์ส่วนหัว `/usr/include/netinet/in.h` สำหรับค่าโปรโตคอล

ttl	unsigned short	เวลาที่ใช้จริงหรือขีดจำกัดฮ็อป
cksum	unsigned short	เช็กซัมส่วนหัว IP
id	unsigned short	หมายเลขการระบุ สมาชิกนี้ถูกใช้สำหรับการระบุเฉพาะของแฟรกเมนต์ของเดต้าแกรม IP เดี่ยว
total_len	unsigned short	ความยาวทั้งหมด ค่านี้เป็นขนาดแพ็กเก็ตทั้งหมด (แพ็กเก็ต) ซึ่งประกอบด้วยส่ว IP และข้อมูล ในหน่วยไบต์
hdr_len	unsigned short	ขนาดของส่วนหัว IP

ตารางที่ 52. สมาชิกตัวแปรบิวด์อิน __ip4hdr (ต่อ).

ชื่อของสมาชิก	ชนิด	คำอธิบาย
tos	unsigned short	ชนิดเซอร์วิส
frag_offset	unsigned short	อ็อฟเซ็ตแฟรกเมนต์ ค่านี้ระบุอ็อฟเซ็ตของแฟรกเมนต์โดยเฉพาะ ซึ่งเกี่ยวข้องกับ จุดเริ่มต้นของเดต้าแกรม IP ที่ไม่ได้แฟรกเมนต์ที่เป็นต้นฉบับ แฟรกเมนต์แรกมีอ็อฟเซ็ต ศูนย์ ซึ่งสามารถตรงกับหนึ่งใน ค่าแฟล็ก `frag_offset` ของค่าคงที่ built-in ค่าแฟล็ก ต้องเป็น bitwise และมีค่าแฟล็กค่าคงที่ built-in เพื่อตรวจสอบการมีอยู่ของแฟล็กเฉพาะเจาะจง: • IP_DF (ไม่มีแฟล็กแฟรกเมนต์) • IP_MF (แฟล็กแฟรกเมนต์เพิ่มเติม) อ้างถึงไฟล์ส่วนหัว `/usr/include/netinet/ip.h` สำหรับ ค่าแฟล็ก

ตัวแปร built-in __ip6hdr

ตัวแปร __ip6hdr เป็นตัวแปร built-in พิเศษเพื่อดูข้อมูลส่วนหัว IPv6 จากแพ็กเก็ตที่ถูกกรอง ตัวแปรนี้พร้อมใช้งานเมื่อ ผู้ใช้โพรบข้อมูลแพ็กเก็ตที่เลเยอร์อินเตอร์เฟส ตัวแปรนี้มีหนึ่งในโปรโตคอลใดๆ (“ipv6”, “tcp”, “udp” และ “icmp6”) มีข้อมูลที่ถูกต้องเมื่อเวอร์ชัน IP เป็น IPv6 ตัวแปรนี้พร้อมใช้งานในโพรบของชนิดย่อย `bpf` อิลิเมนต์สมาชิกสามารถเข้าถึงได้โดยใช้ไวยากรณ์ `__ip6hdr->member`

ตัวแปร built-in นี้มีสมาชิกต่อไปนี้:

ตารางที่ 53. สมาชิกตัวแปรบิวด์อิน __ip6hdr.

ชื่อของสมาชิก	ชนิด	คำอธิบาย
src_addr	ip_addr_t	IP address ต้นทาง ชนิดข้อมูล ip_addr_t ถูกใช้เพื่อเก็บ IP address ใช้ตัวระบุรูปแบบ “I” เพื่อพิมพ์ IP address และใช้ตัวระบุรูปแบบ “H” เพื่อพิมพ์ชื่อโฮสต์ การพิมพ์ชื่อโฮสต์เป็นการดำเนินการที่มีต้นทุน
dst_addr	ip_addr_t	IP address ปลายทาง ชนิดข้อมูล ip_addr_t ถูกใช้เพื่อเก็บ IP address ใช้ตัวระบุรูปแบบ “I” เพื่อพิมพ์ IP address และใช้ตัวระบุรูปแบบ “H” เพื่อพิมพ์ชื่อโฮสต์ การพิมพ์ชื่อโฮสต์ เป็นการดำเนินการที่มีต้นทุน

ตารางที่ 53. สมาชิกตัวแปรบิวด์อิน __ip6hdr (ต่อ).

ชื่อของสมาชิก	ชนิด	คำอธิบาย
โปรโตคอล	unsigned short	ค่านี้บ่งชี้โปรโตคอลที่ถูกใช้ในส่วนของข้อมูลของเดต้าแกรม IP โปรโตคอลสามารถเป็น TCP, UDP และ ICMPV6 และอื่นๆ ซึ่งสามารถตรงกับ หนึ่งในค่าของค่าคงที่ built-in ต่อไปนี้สำหรับ protocol: `IPPROTO_TCP`, `IPPROTO_UDP`, `IPPROTO_ROUTING`, `IPPROTO_ICMPV6`, `IPPROTO_NONE`, `IPPROTO_DSTOPTS`, `IPPROTO_LOCAL` อ้างถึงไฟล์ส่วนหัว `/usr/include/netinet/in.h` สำหรับค่าโปรโตคอล
hop_limit	unsigned short	ขีดจำกัดฮ็อป (เวลาที่ใช้จริง)
total_len	unsigned short	ความยาวทั้งหมด (ความยาวของเพย์โหลด) ขนาดของเพย์โหลดที่สอดแทรกส่วนหัวที่ขยายเพิ่ม
next_hdr	unsigned short	ระบุชนิดของส่วนหัวถัดไปโดยปกติแล้ว ฟิลด์นี้ระบุโปรโตคอลเลเยอร์การขนส่ง ที่ถูกใช้โดยเพย์โหลดของแพ็กเก็ต เมื่อส่วนหัวที่ขยายเพิ่มแสดงอยู่ในแพ็กเก็ต ฟิลด์นี้จะบ่งชี้ถึงส่วนหัวที่ขยายเพิ่มที่ตามมา ค่าจะถูกแบ่งใช้กับส่วนหัวเหล่านั้นสำหรับฟิลด์โปรโตคอล IPv4
flow_label	unsigned int	เลเบลโฟลว์
traffic_class	unsigned int	คลาสทราฟฟิก

ตัวแปร built-in __tcphdr

ตัวแปร __tcphdr เป็นตัวแปร built-inพิเศษเพื่อดูข้อมูลส่วนหัว tcp จาก แพ็กเก็ตที่ถูกกรอง ตัวแปรนี้พร้อมใช้งานเมื่อคุณโพรบข้อมูลแพ็กเก็ตที่เลเยอร์อินเตอร์เฟส ด้วยโปรโตคอล tcp ซึ่งพร้อมใช้งานในโพรบของชนิดย่อย bpf อิลิเมนต์สมาชิกสามารถเข้าถึงได้โดยใช้ไวยากรณ์ `__tcphdr->member`

ตัวแปรบิวด์อิน __tcphdr มีสมาชิก ต่อไปนี้:

ตารางที่ 54. สมาชิกตัวแปรบิวด์อิน __tcphdr.

ชื่อของสมาชิก	ชนิด	คำอธิบาย
src_port	unsigned short	พอร์ตต้นทางของแพ็กเก็ต
dst_port	unsigned short	พอร์ตปลายทางของแพ็กเก็ต

ตารางที่ 54. สมาชิกตัวแปรบิวด์อิน__tcphdr (ต่อ).

ชื่อของสมาชิก	ชนิด	คำอธิบาย
flag	unsigned short	ค่าเหล่านี้เป็นบิตการควบคุมและตั้งค่าเพื่อบ่งชี้การสื่อสารของ ข้อมูลการควบคุม 1 บิตสำหรับแต่ละแฟล็ก ซึ่งสามารถตรงกับหนึ่งใน ค่าแฟล็กของค่าคงที่ built-in ค่าแฟล็ก ต้องเป็น bitwise และมีค่าแฟล็กของค่าคงที่ built-in เพื่อตรวจสอบการมีอยู่ของแฟล็กเฉพาะเจาะจง • TH_FIN (ไม่มีข้อมูลเพิ่มเติมจากผู้ส่ง) • TH_SYN (คำร้องขอเพื่อสร้างการเชื่อมต่อ) • TH_RST (รีเซ็ตการเชื่อมต่อ) • TH_PUSH (ฟังก์ชัน Push ถามเพื่อผลักดันข้อมูลที่บัฟเฟอร์ไปยังแอ็พพลิเคชันที่รับ) • TH_ACK (บ่งชี้ว่าแพ็กเก็ตนี้มีการตอบรับ) • TH_URG (บ่งชี้ว่าฟิลด์ตัวชี้จุดเร่งด่วนมีนัยยะสำคัญ) อ้างถึงเอกสารคู่มือ TCP สำหรับข้อมูลโดยละเอียดเกี่ยวกับแฟล็กนี้และอ้างถึงไฟล์ส่วนหัว `/usr/include/netinet/tcp.h` สำหรับค่าแฟล็ก
seq_num	unsigned int	หมายเลขลำดับ
ack_num	unsigned int	หมายเลขการตอบรับ
hdr_len	unsigned int	ข้อมูลความยาวส่วนหัว TCP
cksum	unsigned short	เช็กซัม
window	unsigned short	ขนาดของหน้าต่าง
urg_ptr	unsigned short	ตัวชี้จุดเร่งด่วน

ตัวแปร built-in __udphdr

__udphdr เป็นตัวแปร built-in พิเศษที่ใช้เพื่อดูข้อมูลส่วนหัว `udp` จากแพ็กเก็ตที่ถูกกรอง built-in นี้พร้อมใช้งานเมื่อ ผู้ใช้โพรบข้อมูลแพ็กเก็ตที่เลเยอร์อินเตอร์เฟสที่มี udp เป็นโปรโตคอล ซึ่งพร้อมใช้งาน ในโพรบของชนิดย่อย `bpf` อิลิเมนต์สมาชิกสามารถเข้าถึงได้โดยใช้ไวยากรณ์ `__udphdr->member`

ตัวแปร built-in __udphdr มีสมาชิกต่อไปนี้:

ตารางที่ 55. สมาชิกตัวแปรบิวด์อิน __udphdr.

ชื่อของสมาชิก	ชนิด	คำอธิบาย
src_port	unsigned short	พอร์ตต้นทางของแพ็กเก็ต
dst_port	unsigned short	พอร์ตปลายทางของแพ็กเก็ต
length	unsigned short	ข้อมูลส่วนหัวและความยาวข้อมูล UDP
cksum	unsigned short	เช็กซัม

ตัวแปร built-in __icmp

__icmp เป็นตัวแปร built-in พิเศษที่ใช้ เพื่อดูข้อมูลส่วนหัว icmp จากแพ็กเก็ตที่ถูกกรอง built-in นี้พร้อมใช้งานเมื่อผู้ใช้โพรบ ข้อมูลแพ็กเก็ตที่เลเยอร์อินเตอร์เฟสที่มีโปรโตคอล icmp ซึ่งพร้อมใช้งานในโพรบของชนิดย่อย bpf อิลิเมนต์สมาชิกสามารถเข้าถึงได้โดยใช้ไวยากรณ์ `__icmp->member`

ตัวแปร built-in นี้มีสมาชิกต่อไปนี้:

ตารางที่ 56. สมาชิกตัวแปรบิวด์อิน __icmp.

ชื่อของสมาชิก	ชนิด	คำอธิบาย

type	unsigned short	ชนิดของข้อความ ICMP
		
		ตัวอย่างเช่น: 0 - การตอบกลับแบบ echo 8 - คำร้องขอ echo 3 - ไม่สามารถ เข้าถึงปลายทางได้ ให้มองหาสำหรับชนิดทั้งหมด สำหรับข้อมูลเพิ่มเติม โปรดอ้างถึงเอกสารคู่มือ เครือข่ายมาตรฐาน
		
		ซึ่งสามารถ ตรงกับหนึ่งในค่าของค่าคงที่ built-in ต่อไปนี้สำหรับชนิดข้อความ ICMP :
		
		`ICMP_ECHOREPLY,`
		`ICMP_UNREACH`
		`ICMP_SOURCEQUENCH,`
		`ICMP_REDIRECT,`
		`ICMP_ECHO,`
		`ICMP_TIMXCEED,`
		`ICMP_PARAMPROB,`
		`ICMP_TSTAMP,`
		`ICMP_TSTAMPREPLY,`
		`ICMP_IREQ,`
		`ICMP_IREQREPLY,`
		`ICMP_MASKREQ,`
		`ICMP_MASKREPLY`
		
		อ้างถึงไฟล์ส่วนหัว `/usr/include/netinet/ip_icmp.h` สำหรับค่าโปรโตคอล
		หมายเหตุ: ไม่ใช่ค่าชนิดข้อความ ที่เป็นไปได้ทั้งหมดจะถูกนิยามไว้ แต่อาจมีอ็อพชันอื่นแสดงใน ค่า

ตารางที่ 56. สมาชิกตัวแปรบิวด์อิน __icmp (ต่อ).

ชื่อของสมาชิก	ชนิด	คำอธิบาย

code	unsigned short	ชนิดย่อยของข้อความ ICMP
		
		สำหรับชนิดข้อความแต่ละชนิด โค้ดอื่นๆ และชนิดย่อย ถูกนิยามไว้ ตัวอย่างเช่น ไม่มีเราต์ไปยังปลายทาง การสื่อสารกับปลายทางที่เป็นข้อห้ามในทางการจัดการดูแล ไม่ใช่เพื่อนบ้าน ไม่สามารถเข้าถึงแอดเดรส ไม่สามารถเข้าถึงพอร์ต สำหรับข้อมูลเพิ่มเติม โปรดอ้างถึงเอกสารคู่มือเครือข่ายมาตรฐาน
		
		ซึ่งสามารถ ตรงกับหนึ่งในค่าของค่าคงที่ built-in ต่อไปนี้สำหรับชนิดย่อย ICMP:
		
		ICMP_UNREACH_NET ICMP_UNREACH_HOST
		ICMP_UNREACH_PROTOCOL ICMP_UNREACH_PORT
		ICMP_UNREACH_NEEDFRAG ICMP_UNREACH_SRCFAIL
		ICMP_UNREACH_NET_ADMIN_PROHIBITED
		ICMP_UNREACH_HOST_ADMIN_PROHIBITED
		
		ค่าชนิดย่อย สำหรับชนิดที่ 4
		
		ค่าชนิดย่อยสำหรับชนิดที่ 4 มีดังต่อไปนี้:
		
		`ICMP_REDIRECT_NET`
		`ICMP_REDIRECT_HOST`
		`ICMP_REDIRECT_TOSNET`
		`ICMP_REDIRECT_TOSHOST`
		
		ค่าชนิดย่อย สำหรับชนิด 6
		
		ค่าชนิดย่อยสำหรับชนิด 6 มีดังต่อไปนี้:
		
		`ICMP_TIMXCEED_INTRANS`
		`ICMP_TIMXCEED_REASS`
		
		ค่าชนิดย่อยสำหรับชนิด 7
		
		ค่าชนิดย่อยสำหรับชนิด 7 มีดังต่อไปนี้:
		
		ICMP_PARAMPROB_PTR
		
		ICMP_PARAMPROB_MISSING
		
		อ้างถึงไฟล์ส่วนหัว /usr/include/netinet/ip_icmp.h สำหรับค่าชนิดย่อยข้อความ
		
		หมายเหตุ: ไม่ใช่ค่าชนิดย่อยข้อความที่เป็นไปได้ทั้งหมด จะถูกนิยามไว้ แต่อาจมีอ็อพชันอื่นที่แสดงในค่าชนิดย่อย ข้อความ

cksum	unsigned short	เช็กซัม

ตัวแปร built-in __icmp6

__icmp เป็นตัวแปร built-in พิเศษที่ใช้ เพื่อดูข้อมูลส่วนหัว icmpv6 จากแพ็กเก็ตที่ถูกกรอง ซึ่งพร้อมใช้งานเมื่อผู้ใช้โพรบข้อมูลแพ็กเก็ตที่เลเยอร์อินเตอร์เฟสด้วยโปรโตคอล `icmp6` ซึ่งพร้อมใช้งาน ในโพรบของชนิดย่อย `bpf` อิลิเมนต์สมาชิก ของตัวแปร built-in นี้สามารถเข้าถึงโดยใช้ไวยากรณ์ “__icmp6->member”

__icmp6 มีสมาชิกต่อไปนี้:

ตารางที่ 57. สมาชิกตัวแปรบิวด์อิน __icmp6.

ชื่อของสมาชิก	ชนิด	คำอธิบาย

type	unsigned short	ชนิดของข้อความ ICMPV6
		
		ระบุชนิดของข้อความที่กำหนดรูปแบบของ ข้อมูลที่เหลืออยู่
		
		ซึ่งสามารถตรงกับ หนึ่งในค่าของค่าคงที่ built-in ต่อไปนี้สำหรับชนิด ICMPV6
		`ICMP6_DST_UNREACH`
		`ICMP6_PACKET_TOO_BIG`
		`ICMP6_TIME_EXCEEDED`
		`ICMP6_PARAM_PROB`
		`ICMP6_INFOMSG_MASK`
		`ICMP6_ECHO_REQUEST`
		`ICMP6_ECHO_REPLY`
		
		อ้างถึง ไฟล์ส่วนหัว `/usr/include/netinet/icmp6.h` สำหรับค่าโปรโตคอล
		หมายเหตุ: ไม่ใช่ค่าชนิดข้อความ ที่เป็นไปได้ทั้งหมดจะถูกนิยามไว้ แต่อาจมีอ็อพชันอื่นแสดงในค่า

code	unsigned short	ชนิดย่อยของข้อความ ICMPV6
		
		ค่านี้ขึ้นอยู่กับชนิดข้อความ ซึ่งจัดเตรียมระดับพิเศษ ของข้อความที่มีขนาดเล็ก
		
		ซึ่งสามารถตรงกับ หนึ่งในค่าของค่าคงที่ built-in ต่อไปนี้สำหรับชนิดย่อย ICMPV6
		`ICMP6_DST_UNREACH_NOROUTE`
		`ICMP6_DST_UNREACH_ADMIN`
		`ICMP6_DST_UNREACH_ADDR`
		`ICMP6_DST_UNREACH_BEYONDSCOPE`
		`ICMP6_DST_UNREACH_NOPORT`
		
		อ้างถึงไฟล์ส่วนหัว `/usr/include/netinet/icmp6.h` สำหรับค่าชนิดย่อย ข้อความ
		หมายเหตุ: ไม่ใช่ค่าชนิดย่อยของข้อความที่เป็นไปได้ทั้งหมดจะถูกนิยามไว้ แต่อาจมีอ็อพชันอื่นที่แสดงในค่า

cksum	unsigned short	เช็กซัม

ตัวแปร built-in __igmp

__udphdr เป็นตัวแปร built-in พิเศษที่ใช้เพื่อดูข้อมูลส่วนหัว `igmp` จากแพ็กเก็ตที่ถูกกรอง ซึ่งพร้อมใช้งานเมื่อ ผู้ใช้โพรบข้อมูลแพ็กเก็ตที่เลเยอร์อินเตอร์เฟสด้วยโปรโตคอล `igmp` ซึ่งพร้อมใช้งาน ในโพรบของชนิดย่อย `bpf` อิลิเมนต์สมาชิกสามารถเข้าถึงได้โดยใช้ไวยากรณ์ “__igmp->member”

built-in __igmp มีสมาชิกต่อไปนี้:

ตารางที่ 58. สมาชิกตัวแปรบิวด์อิน__igmp.

ชื่อของสมาชิก	ชนิด	คำอธิบาย

type	unsigned short	ชนิดของข้อความ IGMP
		
		ตัวอย่างเช่น: `Membership Query (0x11)`, `Membership Report (IGMPv1: 0x12, IGMPv2: 0x16, IGMPv3: 0x22)`, `Leave Group (0x17)` สำหรับข้อมูลเพิ่มเติม โปรดอ้างถึงเอกสารคู่มือมาตรฐาน หรือเครือข่าย
		
		ซึ่งสามารถตรงกับ หนึ่งในค่าของค่าคงที่ built-in ต่อไปนี้สำหรับชนิดข้อความ IGMP
		`IGMP_HOST_MEMBERSHIP_QUERY`
		`IGMP_HOST_MEMBERSHIP_REPORT`
		`IGMP_DVMRP`
		`IGMP_HOST_NEW_MEMBERSHIP_REPORT`
		`IGMP_HOST_LEAVE_MESSAGE`
		`IGMP_HOST_V3_MEMBERSHIP_REPORT`
		`IGMP_MTRACE`
		`IGMP_MTRACE_RESP`
		`IGMP_MAX_HOST_REPORT_DELAY`
		
		อ้างถึงไฟล์ส่วนหัว `/usr/include/netinet/igmp.h` สำหรับค่าโปรโตคอล
		
		หมายเหตุ: ไม่ใช่ค่าชนิดข้อความที่เป็นไปได้ทั้งหมดจะถูกนิยามไว้ แต่สามารถมีอ็อพชันอื่น ที่แสดงในค่า

code	unsigned short	ชนิดย่อยของชนิด IGMP
		
		ซึ่งสามารถตรงกับ หนึ่งในค่าของค่าคงที่ built-in ต่อไปนี้สำหรับชนิดย่อยข้อความ IGMP
		
		ค่าชนิดย่อยสำหรับชนิดหมายเลข 3
		`DVMPP_PROBE 1`
		`DVMRP_REPORT 2`
		`DVMRP_ASK_NEIGHBORS 3`
		`DVMRP_ASK_NEIGHBORS2 4`
		`DVMRP_NEIGHBORS 5`
		`DVMRP_NEIGHBORS2 6`
		`DVMRP_PRUNE 7`
		`DVMRP_GRAFT 8`
		`DVMRP_GRAFT_ACK 9`
		`DVMRP_INFO_REQUEST 10`
		`DVMRP_INFO_REPLY 11`
		
		หมายเหตุ: ไม่ใช่ค่าชนิดย่อยของข้อความที่เป็นไปได้ทั้งหมดจะถูกนิยามไว้ แต่สามารถมีอ็อพชันอื่น ที่แสดงในค่า

cksum	unsigned short	ค่า IGMP Checksum

ตารางที่ 58. สมาชิกตัวแปรบิวด์อิน __igmp (ต่อ).

ชื่อของสมาชิก	ชนิด	คำอธิบาย
group_addr	ip_addr_t	แอดเดรสกลุ่มที่รายงานหรือที่เคียวรี แอดเดรสนี้เป็นแอดเดรสมัลติคาสท์ที่ถูกเคียวรีเมื่อคุณกำลังส่ง Group-Specific หรือ Group-and-Source-Specific Query ฟิลด์มีค่าศูนย์เมื่อคุณกำลังส่ง General Query ชนิดข้อมูล ip_addr_t ถูกใช้เพื่อเก็บกลุ่ม IP address ให้ใช้ตัวระบุรูปแบบ "I" เพื่อพิมพ์ IP address

ตัวแปร built-in__arphdr

ตัวแปร __arphdr เป็นตัวแปร built-in พิเศษที่ใช้เพื่อดูข้อมูลส่วนหัว `arphdr` จากแพ็กเก็ตที่ถูกกรอง ตัวแปรนี้พร้อมใช้งานมื่อผู้ใช้โพรบข้อมูลแพ็กเก็ตที่เลเยอร์อินเตอร์เฟสด้วยโปรโตคอล `arp` หรือ `rarp` ซึ่งพร้อมใช้งานในโพรบของชนิดย่อย `bpf` อิลิเมนต์สมาชิก __arphdr สามารถเข้าถึงได้โดยใช้ไวยากรณ์ `__arphdr->member`

ตัวแปรบิวด์อิน __arphdr มีสมาชิกต่อไปนี้:

ตารางที่ 59. สมาชิกตัวแปรบิวด์อิน __arphdr.

ชื่อของสมาชิก	ชนิด	คำอธิบาย

hw_addr_type	unsigned short	จัดรูปแบบของชนิดฮาร์ดแวร์แอดเดรส ฟิลด์นี้ระบุโปรโตคอล data-link ที่ระบุเฉพาะ ซึ่งถูกใช้
		
		ซึ่งสามารถตรงกับ หนึ่งในค่าของค่าคงที่ built-in ต่อไปนี้สำหรับโปรโตคอล data link:
		`ARPHRD_ETHER,`
		`ARPHRD_802_5,`
		`ARPHRD_802_3, and`
		`ARPHRD_FDDI`
		
		อ้างถึงไฟล์ส่วนหัว `/usr/include/net/if_arp.h` สำหรับค่าโปรโตคอล

protocol_type	unsigned short	รูปแบบของชนิดแอดเดรสโปรโตคอล ฟิลด์นี้ระบุโปรโตคอลเครือข่ายที่ระบุเฉพาะ ซึ่งถูกใช้
		
		ซึ่งสามารถตรงกับ หนึ่งในค่าของค่าคงที่ built-in ต่อไปนี้สำหรับโปรโตคอลเครือข่าย:
		`SNAP_TYPE_IP,`
		`SNAP_TYPE_AP,`
		`SNAP_TYPE_ARP,`
		`VLAN_TAG_TYPE`
		
		อ้างถึงไฟล์ส่วนหัว `/usr/include/net/nd_lan.h` สำหรับค่าโปรโตคอล

hdr_len	unsigned short	ความยาว Mac address หรือฮาร์ดแวร์แอดเดรส

ตารางที่ 59. สมาชิกตัวแปรบิวด์อิน __arphdr (ต่อ).

ชื่อของสมาชิก	ชนิด	คำอธิบาย
proto_len	unsigned short	ความยาวโปรโตคอลหรือ IP address
operation	unsigned short	ระบุการดำเนินการที่ผู้ส่งดำเนินการ: 1 สำหรับคำร้องขอ 2 สำหรับการตอบกลับ ซึ่งสามารถตรงกับ หนึ่งในค่าของค่าคงที่ built-in ต่อไปนี้สำหรับโปรโตคอลเครือข่าย: `ARPOP_REQUEST, ARPOP_REPLY` อ้างถึงไฟล์ส่วนหัว `/usr/include/net/if_arp.h` สำหรับค่าโปรโตคอล
src_mac_addr	mac_addr_t	ผู้ส่งหรือ MAC address ต้นทาง ฮาร์ดแวร์ผู้ส่งหรือ mac address ถูกเก็บอยู่ในชนิดข้อมูล mac_addr_t ตัวระบุรูปแบบ “%M” ถูกใช้เพื่อพิมพ์ MAC ผู้ส่งหรือแอดเดรสฮาร์ดแวร์
dst_mac_addr	mac_addr_t	MAC แอดเดรสเป้าหมายหรือปลายทาง ฮาร์ดแวร์เป้าหมายหรือ MAC address ถูกเก็บอยู่ในชนิดข้อมูล mac_addr_t ตัวระบุรูปแบบ “%M” ถูกใช้เพื่อพิมพ์ MAC เป้าหมายหรือแอดเดรสฮาร์ดแวร์
src_ip	ip_addr_t	IP address ต้นทางหรือผู้ส่ง IP address ผู้ส่งถูกเก็บอยู่ในชนิดข้อมูล `ip_addr_t` ตัวระบุรูปแบบ “%I” ถูกใช้เพื่อพิมพ์ IP address ผู้ส่ง
dst_ip	ip_addr_t	IP แอดเดรสเป้าหมายหรือปลายทาง IP address เป้าหมายถูกเก็บอยู่ในชนิดข้อมูล `ip_addr_t` ตัวระบุรูปแบบ “%I” ถูกใช้เพื่อพิมพ์ IP address เป้าหมาย

ตัวอย่าง

สคริปต์ Vue เพื่อโพรบข้อมูลแพ็กเก็ตส่วนหัวสำหรับแพ็กเก็ตที่ได้รับหรือส่งผ่านพอร์ต 23 จัดเตรียมข้อมูลโหนดต้นทางและปลายทางและยังแสดงข้อมูลความยาวส่วนหัว tcp

```
@@net:bpf:en0:tcp:"port 23"
{
        printf("src_addr:%I and dst_addr:%I\n",__ip4hdr->src_addr,__ip4hdr->dst_addr);
        printf("src port:%d\n",__tcphdr->src_port);
        printf("dst port:%d\n",__tcphdr->dst_port);
        printf("tcp hdr_len:%d\n",__tcphdr->hdr_len);
}
```

เอาต์พุต:
```
# probevue bpf_tcp.e
src_addr:10.10.10.12 and dst_addr:10.10.18.231
src port:48401
dst port:23
tcp hdr_len:20
..................
..................
```

ตัวแปร built-in **__proto_info**

ตัวแปร __proto_info เป็นตัวแปร built-in พิเศษ ที่ใช้เพื่อดูข้อมูลโปรโตคอล (IP แอดเดรสและพอร์ตต้นทาง และปลายทาง) สำหรับเหตุการณ์ TCP หรือ UDP ตัวแปร __proto_info พร้อมใช้งานในโพรบของชนิดย่อย `tcp` หรือ `udp` อิลิเมนต์สมาชิกสามารถเข้าถึงได้โดยใช้ไวยากรณ์ `__proto_info->member`

ตัวแปรบิวด์อิน __proto_info มีสมาชิก ต่อไปนี้:

ตารางที่ 60. สมาชิกตัวแปรบิวด์อิน __proto_info.

ชื่อของสมาชิก	ชนิด	คำอธิบาย
local_port	unsigned short	โลคัลพอร์ต
remote_port	unsigned short	พอร์ตแบบรีโมต
local_addr	ip_addr_t	แอดเดรสโลคัล
remote_addr	ip_addr_t	แอดเดรสรีโมต

ข้อมูลเพิ่มเติมสำหรับเหตุการณ์ที่ระบุเฉพาะ TCP

เหตุการณ์การเปลี่ยนสถานะ TCP ถูกกล่าวถึงในตารางต่อไปนี้:

ตารางที่ 61. เหตุการณ์การเปลี่ยนสถานะ TCP.

ชื่อ	ชนิด	คำอธิบาย
__prev_state	short	ข้อมูลสถานะก่อนหน้านี้สำหรับการเชื่อมต่อ
__cur_state	short	แสดงข้อมูลสถานะสำหรับการเชื่อมต่อ

ตารางที่ 61. เหตุการณ์การเปลี่ยนสถานะ TCP (ต่อ).

ชื่อ	ชนิด	คำอธิบาย
		ซึ่งสามารถใช้หนึ่งในค่าของค่าคงที่ built-in ต่อไปนี้ สำหรับสถานะ TCP: • TCPS_ESTABLISHED (การเชื่อมต่อสร้างขึ้นแล้ว) • TPCS_CLOSED (การเชื่อมต่อปิดแล้ว) • TPCS_LISTEN (Listen สำหรับการเชื่อมต่อ) • TPCS_SYN_SENT (ส่ง SYN ไปยังส่วนปลายของรีโมต) • TCPS_SYN_RECEIVED (รับ SYN จากส่วนปลายของรีโมต) • TCPS_CLOSE_WAIT (รับ Fin รอสำหรับปิด) • TCPS_FIN_WAIT_1 (เปิดแล้ว ส่ง fin) • TCPS_CLOSING (ปิดการแลกเปลี่ยน FIN รอ FIN ACK) • TCPS_LAST_ACK (มี Fin และปิด รอ FIN ACK) • TCPS_FIN_WAIT_2 (ปิดแล้ว Fin คือ Acked) • TCPS_TIME_WAIT (ในการรอแบบไม่โต้ตอบ 2*msl หลังจากที่ปิด) ค่าต่างๆ ถูกนิยามไว้ในไฟล์ส่วนหัวที่เอ็กซ์พอร์ต `/usr/include/netinet/tcp_fsm.h`

ตัวอย่าง:

สคริปต์ Vue ต่อไปนี้จัดเตรียมข้อมูลการเปลี่ยนสถานะสำหรับ การเชื่อมต่อ โดยเฉพาะ:

```
@@net:tcp:state_change
when(__proto_info->local_addr ==″10.10.10.1″ and __proto_info->remote_addr == 10.10.10.2″
 and __proto_info->local_port =″8000″ and  __proto_info->remote_port =″9000″)
{
 printf(″Previous state:%d and current_state:%d\n″,__prev_state,__cur_state);
}
```

เหตุการณ์การส่ง TCP อีกครั้ง

ตารางที่ 62. เหตุการณ์การส่ง TCP อีกครั้ง.

ชื่อ	ชนิด	คำอธิบาย
__nth_retransmit	unsigned short	การส่ง Nth

ตัวอย่าง

1. ตัวอย่างต่อไปนี้ระบุ listener ที่ละเว้น การเชื่อมต่อเนื่องจากคิวของ listener เต็ม

```
@@net:tcp:listen_q_full
{
 printf("Listener IP address:%I and  Port number is:%d\n",__proto_info->local_addr, __proto_info->local_port);
}
```

2. ตัวอย่างต่อไปนี้ระบุการเชื่อมต่อที่ดร็อปแพ็กเก็ต เนื่องจากบัฟเฟอร์ซ็อกเก็ตโอเวอร์โฟลว์

```
@@net:udp:sock_recv_buf_overflow
{
        printf("Connection information which drops packet due to socket buffer overflows:\n");
        printf("Local IP address:%I and Remote IP address:%I\n",__proto_info->local_addr,__proto_info->remote_addr);
        printf("local port :%d and remote port:%d\n",__proto_info->local_port, __proto_info->remote_port);
 }
```

3. ระบุการส่งข้อมูลอีกครั้ง (การส่งข้อมูลเพิ่มเติม & ครั้งที่สองสำหรับแพ็กเก็ต) สำหรับการเชื่อมต่อ TCP สำหรับการเชื่อมต่อเฉพาะเจาะจง

```
@@net:tcp:retransmit
when (__proto_info->local_addr == "10.10.10.1" &&
  __proto_info->remote_addr == "10.10.10.2" &&
__proto_info->local_port == "4000" &&
  __proto_info->remote_port == "5000")
{
        printf(" %d th re-transmition for this connection\n", _nth_retransmit);
}
```

4. ระบุข้อมูลการเชื่อมต่อเมื่อเกิดเหตุการณ์ บัฟเฟอร์ผู้ส่งเต็ม

```
@@net:tcp:send_buf_full
{
        printf("Connection information whenever send buffer full event occurs:\n");
        printf("Local IP address:%I and Remote IP address:%I\n",__proto_info->local_addr,__proto_info->remote_addr);
        printf("local port :%d and remote port:%d\n",__proto_info->local_port, __proto_info->remote_port);
 }
```

โปรแกรมจัดการโพรบ Sysproc

ภาพรวม

โปรแกรมจัดการโพรบ sysproc จัดเตรียมโครงสร้างพื้นฐานให้กับผู้ใช้และผู้ดูแลระบบเพื่อติดตามกระบวนการ หรือเธรดแบบไดนามิกโดยเกี่ยวข้องกับข้อมูลที่ไม่รู้จักระบบย่อยภายในของ sysproc

ลักษณะของ ระบบย่อย sysproc สำหรับผู้ใช้หรือผู้ดูแลระบบถูกแบ่งออกเป็น หมวดหมู่หลักๆ ดังนี้:

- การสร้างหรือการยกเลิกกระบวนการ (หรือเธรด)
- การสร้างสัญญาณและการส่ง

- เหตุการณ์ตัวจัดตารางเวลาและตัวแจกจ่าย
- เหตุการณ์โยง DR และ CPU

การสร้างหรือการยกเลิกกระบวนการ (หรือเธรด)

ข้อมูลที่เกี่ยวข้องกับ วิธีสร้างและทำลายกระบวนการหรือเธรอดที่จำเป็นต่อผู้ดูแลระบบเพื่อดูแลจัดการ รีซอร์สของระบบ โปรแกรมจัดการโพรบ sysproc กำหนดกรณีการใช้งานที่สำคัญไว้ดังต่อไปนี้:

- กระบวนการออกตามปกติหรือเกิดข้อผิดพลาด?
- เมื่อใดที่กระบวนการหรือเธรดถูกสร้างหรือยกเลิกหรือมีมากเกิน?
- กระบวนการใช้เวลาในการรันนานเท่าไหร่?
- แทร็กเหตุการณ์เมื่อเธรดได้รับหรือส่งคืนจากข้อยกเว้น

การสร้างสัญญาณและการส่ง

สัญญาณเป็นตัวตัดสินสถานะปัจจุบัน ของเธรดตัวประมวลผลในระบบ เมื่อต้องการทำความเข้าใจเกี่ยวกับกระบวนการเธรดที่มีลักษณะการทำงานผิดปกติ ผู้ดูแลระบบต้องใช้สถานะของสัญญาณและสถานะปัจจุบันของกระบวนการจากสัญญาณเหล่านี้ กรณีใช้งานที่สำคัญ ภายใต้หมวดหมู่การสร้างและการส่งสัญญาณ (แต่ไม่จำกัด) ถูกกำหนดไว้โดยโปรแกรมจัดการโพรบนี้ดังต่อไปนี้:

- แหล่งที่มาของสัญญาณและข้อมูลเกี่ยวกับสัญญาณสำหรับเป้าหมายที่ระบุเฉพาะ
- การส่งสัญญาณของสัญญาณแบบอะซิงโครนัส
- การติดตามสัญญาณที่เคลียร์แล้ว
- การติดตามเหตุการณ์เมื่อตัวจัดการสัญญาณที่ติดตั้งไว้เป็นตัวจัดการที่ไม่ใช่ดีฟอลต์
- เป้าหมายของสัญญาณและข้อมูลสัญญาณสำหรับแหล่งที่มาที่ระบุเฉพาะ
- การเข้าและการออกของตัวจัดการสัญญาณ

เหตุการณ์ตัวจัดตารางเวลาและตัวแจกจ่าย

ตัวจัดการตารางเลาและตัวแจกจ่ายคาดการณ์ถึงวิธีการรันกระบวนการหรือเธรด ในระบบ ผู้ดูแลระบบจะวิเคราะห์ประสิทธิภาพการทำงานของระบบโดยใช้ระบบย่อย ตัวจัดตารางเวลาหรือตัวแจกจ่ายการติดตามแบบไดนามิก

ระบบย่อยตัวจัดตารางเวลาหรือตัวแจกจ่ายแบบไดนามิก ช่วยค้นหาเหตุกผลสำหรับการเก็บรักษาเธรด

ต่อไปนี้คือกรณีการใช้งานที่สำคัญ ภายใต้หมวดหมู่การสร้างและการส่งสัญญาณ (แต่ไม่จำกัด) ถูกกำหนดไว้โดยโปรแกรมจัดการโพรบ sysproc

- ติดตามเธรดที่อยู่ในคิวหรือออกจากคิวจากคิวที่รัน
- ติดตามเหตุการณ์เมื่อเธรดใดๆ ในระบบที่ถูกครอบครอง
- ติดตามเมื่อเธรดกำลังเข้าสู่โหมดพักเหนือเหตุการณ์
- ติดตามเมื่อเธรดที่พักอยู่กำลังเริ่มกลับมาทำงาน
- ติดตามการแจกจ่ายเวลาแฝงของเธรด
- แทร็กเหตุการณ์ที่พับเก็บตัวประมวลผลเสมือน
- ติดตามการเปลี่ยนแปลงในลำดับความสำคัญของเธรดเคอร์เนล

เหตุการณ์ Dynamic Reconfiguration (DR) และการโยง CPU

คลาสนี้ของโพรบ นำเสนอความสามารถในการติดตามแบบไดนามิกให้กับผู้ใช้ผู้ที่แทร็กการโยงกับกระบวนการ

กรณีการใช้งานที่สำคัญ บางกรณี (แต่ไม่จำกัด) ภายใต้หมวดหมู่นี้ที่ถูกกำหนดไว้โดยโปรแกรมจัดการโพรบเหตุการณ์ DR และการโยง CPU มีดังต่อไปนี้:

- แทร็กเมื่อการโยงเธรดเปลี่ยนแปลงจาก CPU หนึ่งไปยังอีก CPU หนึ่ง
- แทร็กเมื่อรีซอร์สถูกพ่วงต่อหรือถอดออกจากกระบวนการ
- แทร็กเหตุการณ์การโยง CPU
- แทร็กจุดเริ่มต้นหรือสิ้นสุดของเหตุการณ์ DR

ข้อกำหนดคุณสมบัติโพรบ

รูปแบบต่อไปนี้ต้องใช้ในสคริปต์ Vue เพื่อโพรบเหตุการณ์ sysproc

```
@@sysproc:<sysproc_event>:<pid/tid/*>
```

tuple ตัวแรก `@@sysproc` บ่งชี้ว่าโพรบระบุเฉพาะเหตุการณ์ sysproc

tuple ตัวที่สองระบุเฉพาะเหตุการณ์ที่ต้องโพรบ

tuple ตัวที่สามทำหน้าที่เป็นตัวกรอง เพื่อแยกเหตุการณ์ที่ระบุผ่าน tuple ตัวที่สองโดยอ้างถึงกระบวนการหรือ id เธรดเคอร์เนล

หมายเหตุ: การใช้กระบวนการหรือเธรดเคอร์เนลเป็นตัวกรองในโพรบ sysproc ไม่ได้เป็นการรับประกันเหตุการณ์ เพื่อให้เกิดขึ้นในกระบวนการหรือบริบทเธรด โปรแกรมจัดการโพรบ Sysproc ใช้กระบวนการหรือ id เธรด เป็นตัวกรองเท่านั้น เหตุการณ์เหล่านี้อาจมีประโยชน์จากเปอร์สเปคทีฟกระบวนการหรือเธรดโดยไม่คำนึงถึง บริบทการเรียกทำงานของเหตุการณ์โพรบ

สัญญาณส่งเหตุการณ์โดยที่กระบวนการที่กำลังส่งสัญญาณ หรือกระบวนการที่รับสัญญาณสามารถได้รับประโยชน์ ข้อมูลต่อไปนี้ระบุตัวกรองที่เหมาะสม สำหรับเหตุการณ์โพรบ

จุดโพรบ (เหตุการณ์ที่น่าสนใจ)

คำอธิบายแบบย่อ ของเหตุการณ์ทั้งหมดที่สามารถโพรบได้ผ่านโปรแกรมจัดการโพรบ sysproc ถูกกล่าวถึงใน ตารางต่อไปนี้:

ตารางที่ 63. เหตุการณ์โพรบ Sysproc.

โพรบ (sysproc_event)	คำอธิบาย
forkfail	แทร็กความล้มเหลวในอินเตอร์เฟส fork
execfail	แทร็กความล้มเหลวในอินเตอร์เฟส exec
execpass	แทร็ก exec ที่สำเร็จ
exit	แทร็กการออกของกระบวนการ
threadcreate	แทร็กการสร้างของเธรดเคอร์เนล

ตารางที่ 63. เหตุการณ์โพรบ Sysproc (ต่อ).

โพรบ (sysproc_event)	คำอธิบาย
threadterminate	แทร็กการยกเลิกของเธรดเคอร์เนล
threadexcept	แทร็กข้อยกเว้นของกระบวนการ
sendsig	แทร็กสัญญาณที่ส่งไปยังกระบวนการโดยแหล่งที่มาภายนอก
sigqueue	แทร็กสัญญาณที่อยู่ในคิวกระบวนการ
sigdispose	แทร็กการปล่อยสัญญาณ
sigaction	แทรก็กการติดตั้งและการติดตั้งใหม่ของตัวจัดการสัญญาณ
sighandlestart	แทร็กเมื่อตัวจัดการสัญญาณเกี่ยวข้องกับการเรียก
sighandlefinish	แทร็กเมื่อตัวจัดการสัญญาณเสร็จสมบูรณ์
changepriority	แทร็กเมื่อลำดับความสำคัญของกระบวนการเปลี่ยนแปลง
onreadyq	แทร็กเมื่อเธรดเคอร์เนลอยู่บนคิวที่พร้อมใช้งาน
offreadyq	แทร็กเมื่อเธรดเคอร์เนลถูกย้ายออกจากคิวที่พร้อมใช้งาน
dispatch	แทร็กเมื่อตัวกระจายระบบถูกเรียกเพื่อกำหนดตารางเวลาเธรด
oncpu	แทร็กเมื่อเธรดเคอร์เนลได้รับ CPU
offcpu	แทร็กเมื่อเธรดเคอร์เนลปล่อย CPU
blockthread	แทร็กเมื่อเธรดถูกบล็อกจากการขอรับ CPU
foldcpu	แทร็คเมื่อ fold คอร์ CPU
bindprocessor	แทร็กเหตุการณ์เมื่อกระบวนการ/เธรดถูกโยงกับ CPU
changecpu	แทร็กเมื่อเธรดเคอร์เนลเปลี่ยน CPU ชั่วคราว
resourceattach	แทร็กเหตุการณ์เมื่อรีซอร์สพ่วงต่อกับรีซอร์สอื่น
resourcedetach	แทร็กเหตุการณ์เมื่อรีซอร์สถูกถอดออกจากรีซอร์สอื่น
drphasestart	แทร็กเมื่อ drphase เริ่มต้นทำงาน
drphasefinish	แทร็กเมื่อ drphase เสร็จสิ้น

เมธอดเพื่อเข้าถึงข้อมูลที่จุดโพรบ

ProbeVue อนุญาตให้เข้าถึงข้อมูล ผ่านตัวแปร built-in

ค่า built-in มีสามชนิด ที่จัดหมวดหมู่ตามความสามารถในการเข้าถึง:

1. ความสามารถในการเข้าถึงที่จุดโพรบใดๆ โดยไม่คำนึงถึงโปรแกรมจัดการโพรบ ตัวอย่างเช่น: `__curthread`
2. ความสามารถในการเข้าถึงโพรบของโปรแกรมจัดการโพรบที่ระบุเฉพาะ
3. ความสามารถในการเข้าถึงโพรบที่นิยามไว้เท่านั้น (เหตุการณ์ที่น่าสนใจ)

ตัวจัดการโพรบ sysproc อนุญาตให้เข้าถึงข้อมูลผ่าน built-ins ชนิด (1) และ (3) ตารางต่อไปนี้ใช้เพื่อระบุความสามารถในการเข้าถึงของ built-in ชนิด (1) บิวด์อินพิเศษ ที่จัดเตรียมไว้สำหรับโปรแกรมจัดการโพรบ sysproc เป็นชนิด long long

ต่อไปนี้คือรายการของบิวด์อิน ชนิด (1)

- __trcid
- __errno__kernelmode
- __arg1 to __arg7
- __curthread
- __curproc
- __m
- __tid
- __pid
- __ppid
- __pgid
- __uid
- __euid
- __ublock
- __execname
- __pname

ตัวแปร built-in ยังแบ่งเป็นแบบเฉพาะบริบท และไม่ขึ้นกับบริบท built-in เฉพาะบริบทจัดเตรียมข้อมูล ตามบริบทการเรียกทำงานของโพรบ

เคอร์เนล AIX ดำเนินการในเธรดหรือบริบทอินเตอร์รัปต์โพรบที่ระบุเฉพาะบริบทสร้างผลลัพธ์ที่ถูกต้องเมื่อโพรบเริ่มต้นทำงานที่เธรดหรือบริบทกระบวนการ

ผลลัพธ์ที่ได้รับจาก built-in เฉพาะบริบทในบริบทการเรียกทำงานอินเตอร์รัปต์อาจ ไม่ได้คาดการณ์ไว้ built-in ที่ไม่ขึ้นกับบริบทไม่ขึ้นอยู่กับบริบทการเรียกทำงาน และสามารถเข้าถึงได้อย่างปลอดภัยโดยไม่คำนึงถึงสภาวะแวดล้อม การเรียกทำงานโพรบ

ตารางที่ 64. ตัวแปรบิวด์อินที่ระบุเฉพาะบริบทและเป็นอิสระจากบริบท.

ตัวแปรบิวด์อินที่ระบุเฉพาะบริบท	ตัวแปรบิวด์อินที่เป็นอิสระจากบริบท
__curthread	__trcid
__curproc	__errno
__tid	__kernelmode
__pid	__arg1 to __arg7
__ppid	__m
__pgid	
__uid	
__euid	
__ublock	

ตารางที่ 64. ตัวแปรบิวด์อินที่ระบุเฉพาะบริบทและเป็นอิสระจากบริบท (ต่อ).

ตัวแปรบิวด์อินที่ระบุเฉพาะบริบท	ตัวแปรบิวด์อินที่เป็นอิสระจากบริบท
__pname	
__execname	

จุดโพรบ

จุดโพรบเป็นเหตุการณ์ที่ระบุเฉพาะที่โพรบ ถูกใช้ ต่อไปนี้คือรายของจุดโพรบ

forkfail

โพรบ `forkfail` เริ่มต้นเมื่อ fork ล้มเหลว โพรบนี้กำหนดเหตุผลของความล้มเหลว ของ fork

ไวยากรณ์: `@@sysproc:forkfail:<pid/tid/*>`

built-in พิเศษที่สนับสนุน

```
__forkfailinfo
{
fail_reason;
}
```

ตัวแปร `fail_reason`อมีหนึ่งในค่าต่อไปนี้:

ตารางที่ 65. โพรบ fail_reason: เหตุผลของความล้มเหลว.

เหตุผล	คำอธิบาย	
FAILED_RLIMIT	ล้มเหลวเนื่องจากข้อจำกัดเกี่ยวกับ rlimit	
FAILED_ALLOCATIONS	ล้มเหลวเนื่องจากการจัดสรรรีซอร์สภายใน	
FAILED_LOADER	ล้มเหลวที่สเตจของโหลดเดอร์	
FAILED_PROCDUP	ล้มเหลวที่ procdup	

built-in อื่นที่สนับสนุน

`__errno__kernelmode`, `__arg1` to `__arg7`, `__curthread`, `__curproc`, `__mst`, `__tid`, `__pid`, `__ppid`, `__pgid`, `__uid`, `__euid`, `__ublock`, `__execname`, `__pname`

สภาวะแวดล้อม การเรียกทำงาน

รันในสภาวะแวดล้อมกระบวนการ

ตัวอย่าง

ตัวอย่างต่อไปนี้แสดงวิธีการมอนิเตอร์ความล้มเหลวทั้งหมด เนื่องจาก rlimit ในระบบ

```
@@BEGIN
{
         x = 0;
}

@@sysproc:forkfail:*
        when (__forkfailinfo->fail_reason == FAILED_RLIMIT)
{
                printf ("process %s with pid %llu failed to fork a child\n",__pname,__pid);
                x++;
}

@@END
{

        printf ("Found %d failures during this vue session\n",x);
}
```

execfail

โพรบ execfail **เริ่มต้นเมื่อการเรียกฟังก์ชัน** exec **ล้มเหลว ใช้โพรบ** execfail **เพื่อกำหนดเหตุผลสำหรับ ความล้มเหลว**

ไวยากรณ์: @@sysproc:execfail:<pid/tid/*>

ตารางที่ 66. โพรบ execfail: เหตุผลของความล้มเหลว.

เหตุผล	คำอธิบาย	
FAILED_PRIVILEGES	กระบวนการใหม่ล้มเหลวในการขอรับหรือสืบทอดสิทธิพิเศษ	
FAILED_COPYINSTR	กระบวนการใหม่ล้มเหลวในการคัดลอกคำสั่ง	
FAILED_V_USERACC	กระบวนการใหม่ล้มเหลวในการละเว้นส่วนของ v_useracc	
FAILED_CLEARDATA	ล้มเหลวในระหว่างการเคลียร์ข้อมูลสำหรับกระบวนการใหม่	
FAILED_PROCSEG	ล้มเหลวในการสร้างเซ็กเมนต์ไพรเวตของกระบวนการ	
FAILED_CH64	ล้มเหลวในการแปลงเป็นกระบวนการแบบ 64 บิต	
FAILED_MEMATT	ล้มเหลวในการพ่วงต่อกับชุดรีซอร์สหน่วยความจำ	
FAILED_SRAD	ล้มเหลวในการพ่วงต่อกับ srad	
FAILED_MSGBUF	ข้อความแสดงข้อผิดพลาดเกี่ยวกับความยาวบัฟเฟอร์เป็นศูนย์	
FAILED_ERRBUF	ล้มเหลวในการจัดสรรบัฟเฟอร์ข้อความแสดงข้อผิดพลาด	
FAILED_ENVAR	ล้มเหลวในการจัดสรรตัวแปรสภาวะแวดล้อม	
FAILED_CPYSTR	คัดลอกข้อผิดพลาดสตริง	
FAILED_ERRBUFCPY	ล้มเหลวในการคัดลอกข้อความแสดงข้อผิดพลาดจาก errmsg_buf	
FAILED_TOOLNGENV	Env ยาวเกินไปสำหรับหน่วยความจำที่จัดสรรแล้ว	
FAILED_USRSTK	ล้มเหลวในการตั้งค่าสแต็กผู้ใช้	
FAILED_CPYARG	ล้มเหลวในการคัดลอก arglist ไปยังสแต็ก	

ตารางที่ 66. โพรบ execfail: เหตุผลของความล้มเหลว (ต่อ).

เหตุผล	คำอธิบาย	
FAILED_INITPTRACE	ล้มเหลวในการ init ptrace	

หมายเหตุ: 64 ถูกเพิ่มไปยังค่าข้อผิดพลาดหากไม่พบข้อผิดพลาดของโหลดเดอร์

built-in อื่นที่สนับสนุน

`__errno__kernelmode, __arg1 to __arg7, __curthread, __curproc, __mst, __tid, __pid, __ppid, __pgid, __uid, __euid, __ublock, __execname, __pname`

สภาวะแวดล้อม การเรียกทำงาน

รันในสภาวะแวดล้อมกระบวนการ

exit

โพรบนี้เริ่มต้น เมื่อกระบวนการออก การออกยังเป็นโปรแกรมจัดการการเรียกระบบและถูกติดตามผ่านโปรแกรมจัดการโพรบ การเรียกระบบ การโพรบการเรียกระบบ exit ผ่านตัวจัดการโพรบ `sysproc` อธิบายถึงลักษณะและ เหตุผลของการออก และยังอธิบายถึงเหตุผลสำหรับการยกเลิกเธรดของผู้ใช้ในพื้นที่เคอร์เนล และไม่ส่งคืนกลับไปยังพื้นที่ผู้ใช้

ไวยากรณ์: `@@sysproc:forkfail:<pid/tid/*>`

โปรแกรมสามารถออกได้เนื่องจากเหตุผล ต่อไปนี้:

- เมื่อเข้าสู่เงื่อนไขการยกเลิกเมื่อโปรแกรมพื้นที่ผู้ใช้ไม่สามารถดำเนินการต่อไปได้
- เมื่อรับสัญญาณการยกเลิก

built-in พิเศษที่สนับสนุน

```
__exitinfo{
 signo;
 returnval;
 iscore;
}
```

โดยที่ ค่า signo หมายถึงหมายเลขสัญญาณที่เป็นสาเหตุทำให้เกิดการยกเลิกกระบวนการ returnval คือค่าที่ ส่งคืนโดย exit ซึ่ง signo ที่ไม่ใช่ค่าศูนย์เป็นค่าที่ถูกต้องหากโปรแกรมหยุดทำงานโดยสัญญาณ

ตัวแปร `iscore` ถูกตั้งค่าไว้เมื่อคอร์ถูสร้างขึ้นเป็นผลลัพธ์ของกระบวนการ exit

built-in อื่นที่สนับสนุน

`__errno__kernelmode, __arg1 to __arg7, __curthread, __curproc, __mst, __tid, __pid, __ppid, __pgid, __uid, __euid, __ublock, __execname, __pname`

สภาวะแวดล้อม การเรียกทำงาน

รันในสภาวะแวดล้อมกระบวนการ

ตัวอย่าง

ตัวอย่างต่อไปนี้อธิบายถึงวิธีการโพรบเหตุการณ์ exit

```
echo '@@sysproc:exit:* { printf (" %s    %llu    %llu\n", __pname, __pid,__exitinfo->returnval);}' | probevue
```

ซึ่งจะสร้างเอาต์พุตที่คล้ายกับที่แสดงต่อไปนี้

```
 ksh    5833042    0
 telnetd    7405958    1
 dumpctrl    7405960    0
 setmaps    7275006    0
 termdef    7274752    0
 hostname    7274754    0
 id    8257976    0
 id    8257978    0
 uname    8257980    0
 expr    8257982    1
```

threadcreate

โพรบ threadcreate เริ่มต้นเมื่อเธรดถูกสร้างเป็นผลสำเร็จ

ไวยากรณ์: @@sysproc:threadcreate:<pid/tid/*>

หมายเหตุ: pid หรือ tid ที่ระบุต้องเป็นกระบวนการหรือ ID เธรดของกระบวนการ หรือเธรดที่สร้างเธรด

built-in พิเศษที่สนับสนุน

```
__threadcreateinfo
{
 tid;
 pri;
 policy;
}
```

โดยที่ tid บ่งชี้ id เธรดของเธรดใหม่ที่สร้างขึ้น และลำดับความสำคัญคือ ลำดับความสำคัญของเธรด นโยบายแสดงนโยบายการกำหนดตารางเวลาเธรดของเธรด

ตารางที่ 67. ค่านโยบายสำหรับโพรบ threadcreate.

นโยบาย	คำอธิบาย
SCHED_OTHER	นโยบายการกำหนดตารางเวลาดีฟอลต์ของ AIX
SCHED_FIFO	นโยบายการกำหนดตารางเวลาแบบเข้าก่อนออกก่อน
SCHED_RR	นโยบายการกำหนดตารางเวลาแบบวนรอบ
SCHED_LOCAL	นโยบายการกำหนดตารางเวลาขอขอบเขตเธรดโลคัล
SCHED_GLOBAL	นโยบายการกำหนดตารางเวลาแบบขอบเขตเธรดโกลบอล
SCHED_FIFO2	FIFO พร้อมกับ RQHEAD หลังพักในระยะสั้น

ตารางที่ 67. ค่านโยบายสำหรับโพรบ threadcreate (ต่อ).

นโยบาย	คำอธิบาย
SCHED_FIFO3	FIFO พร้อมกับ RQHEAD ทุกเวลา
SCHED_FIFO4	FIFO พร้อมกับการครอบครองจุดอ่อน

built-in อื่นที่สนับสนุน

__errno__kernelmode, __arg1 to __arg7, __curthread, __curproc, __mst, __tid, __pid, __ppid, __pgid, __uid, __euid, __ublock, __execname, __pname

สภาวะแวดล้อม การเรียกทำงาน

รันใน สภาวะแวดล้อมกระบวนการ (ผู้ใช้หรือ kproc)

ตัวอย่าง

เมื่อต้องการพิมพ์กระบวนการทั้งหมด อย่างต่อเนื่องในระบบ ให้สร้างชื่อกระบวนการพิมพ์เธรด สร้าง id กระบวนการ id ของเธรดที่สร้างขึ้นใหม่ และ สร้างเวลาประทับ

```
echo '@@sysproc:threadcreate:*
{ printf ("%s %llu %llu %A\n",__pname,__pid,__threadcreateinfo->tid,timestamp());}' | probevue
```

เอาต์พุต ที่คล้ายกับตัวอย่างต่อไปนี้ จะแสดงขึ้น

```
nfssync_kproc 5439964 23921151 Feb/22/15 09:22:38
nfssync_kproc 5439964 24052201 Feb/22/15 09:22:38
nfssync_kproc 5439964 23920897 Feb/22/15 09:22:38
nfssync_kproc 5439964 22479285 Feb/22/15 09:22:55
nfssync_kproc 5439964 23920899 Feb/22/15 09:22:55
nfssync_kproc 5439964 22479287 Feb/22/15 09:22:55
```

threadterminate

โพรบ เริ่มต้นสำหรับเธรดที่ถูกยกเลิก

ไวยากรณ์: `@@sysproc:threadterminate:<pid/tid/*>`

หมายเหตุ: ID กระบวนการที่ระบุไว้หรือ ID เธรดต้องสอดคล้องกับกระบวนการหรือเธรดที่ขอให้หยุดทำงานในปัจจุบัน

built-ins พิเศษที่สนับสนุน

ไม่มี

built-in อื่นที่สนับสนุน

__errno__kernelmode, __arg1 to __arg7, __curthread, __curproc, __mst, __tid, __pid, __ppid, __pgid, __uid, __euid

สภาวะแวดล้อม การเรียกทำงาน

รันใน สภาวะแวดล้อมกระบวนการ (ผู้ใช้หรือ kproc)

ตัวอย่าง

เมื่อต้องการพิมพ์กระบวนการทั้งหมดอย่างต่อเนื่อง ในการยกเลิกระบบ ชื่อกระบวนการพิมพ์ การสร้าง ID กระบวนการ ID ของเธรดที่สร้างขึ้นใหม่ และการสร้าง เวลาประทับ

```
# echo '@@sysproc:threadterminate:* { printf ("%s %llu %llu %A\n",__pname,__pid,__tid,timestamp());}' | probevue

A output similar to one shown below can be observed.
nfssync_kproc 5439964 23855555 Feb/22/15 09:59:30
nfssync_kproc 5439964 21758249 Feb/22/15 09:59:30
nfssync_kproc 5439964 23855557 Feb/22/15 09:59:30
```

threadexcept

โพรบนี้เริ่มต้นเมื่อเกิดข้อยกเว้นของโปรแกรม ข้อยกเว้นของโปรแกรมถูกสร้างขึ้นเมื่อ ระบบตรวจพบเงื่อนไขที่โปรแกรมไม่สามารถดำเนินการต่ออย่างเป็นปกติได้ ข้อยกเว้นบางอย่างเป็นข้อยกเว้นที่รุนแรง (คำสั่งที่ผิดกฎเกณฑ์) ขณะที่ข้อยกเว้นบางอย่างสามารถกู้คืนได้ (การเปลี่ยนพื้นที่แอดเดรส)

ไวยากรณ์: `@@sysproc:threadexcept:<pid/tid/*>`

built-ins พิเศษ ที่สนับสนุน

```
__threadexceptinfo
{
 pid;
 tid;
 exception;
 excpt_address
}
```

โดยที่ `pid` แสดงถึง ID กระบวนการที่รับข้อยกเว้น **tid** คือ ID เธรดของเธรดเคอร์เนลที่รับข้อยกเว้น `excpt_address` คือแอดเดรสที่เป็นสาเหตุทำให้เกิดข้อยกเว้นขณะที่ข้อยกเว้นสามารถเป็นหนึ่งในค่าที่แสดงใน ตาราง

ตารางที่ 68. ค่าข้อยกเว้นสำหรับโพรบ threadexcept.

ข้อยกเว้น	คำอธิบาย	
EXCEPT_FLOAT	ข้อยกเว้นเกี่ยวกับ Floating point	
EXCEPT_INV_OP	op-code ไม่ถูกต้อง	
EXCEPT_PRIV_OP	op พิเศษในโหมดผู้ใช้	
EXCEPT_TRAP	คำสั่ง Trap	
EXCEPT_ALIGN	โค้ดหรือการจัดวางข้อมูล	
EXCEPT_INV_ADDR	แอดเดรสไม่ถูกต้อง	
EXCEPT_PROT	การปกป้อง	
EXCEPT_IO	ซิงโครนัส I/O	

ตารางที่ 68. ค่าข้อยกเว้นสำหรับโพรบ threadexcept (ต่อ).

ข้อยกเว้น	คำอธิบาย	
EXCEPT_IO_IOCC	ข้อยกเว้น I/O จาก IOCC	
EXCEPT_IO_SGA	ข้อยกเว้น I/O จาก SGA	
EXCEPT_IO_SLA	ข้อยกเว้น I/O จาก SLA	
EXCEPT_IO_SCU	ข้อยกเว้น I/O จาก SCU	
EXCEPT_EOF	อ้างอิงจุดที่อยู่ไกลจากส่วนท้ายของไฟล์ (mmap)	
EXCEPT_FLOAT_IMPRECISE	ข้อยกเว้นเกี่ยวกับ floating point ที่เชื่อถือไม่ได้	
EXCEPT_ESTALE_I	ข้อยกเว้นเกี่ยวกับเซ็กเมนต์ข้อความสเตล	
EXCEPT_ESTALE_D	ข้อยกเว้นเกี่ยวกับเซ็กเมนต์ข้อความสเตล	
EXCEPT_PT_WATCHP	ตรงกับจุดดู ptrace	

built-in อื่นที่สนับสนุน

`__errno__kernelmode, __arg1 to __arg7, __curthread, __curproc, __mst, __tid, __pid, __ppid, __pgid, __uid, __euid`

สภาวะแวดล้อม ข้อยกเว้น

รันในกระบวนการหรือสภาวะแวดล้อมแบบอินเตอร์รัปต์

หมายเหตุ: เนื่องจากโพรบนี้สามารถเริ่มทำงานในบริบทอินเตอร์รัปต์ ตัวแปร built-in เช่น `__pid`, `__tid` ที่ขึ้นอยู่กับ บริบทการเรียกทำงานอาจไม่ระบุ รก กระบวนการ หรือเธรด สมาชิก built-in พิเศษสำหรับโพรบนี้รับประกัน id กระบวนการหรือเธรดที่ถูกต้อง ที่มีเจตนาสำหรับกระบวนการหรือเธรด

ตัวอย่าง

ตัวอย่างต่อไปนี้แสดงข้อยกเว้นโปรแกรมการติดตามที่สร้างขึ้นโดย เหตุการณ์โพรบที่กำลังถูกติดตามโดย โปรแกรมดีบัก

```
# cat threadexcept.e
@@sysproc:threadexcept:*
{
 printf ("PID = %llu TID= %llu EXCEPTION=%llu ADDRESS = %llu\n ",__threadexceptinfo->pid,__threadexceptinfo->tid,__threadexceptinfo-
>exception,__threadexceptinfo->excpt_address);
}
```

รันเซสชันการดีบักบนโปรแกรมที่คอมไพล์กับส่วนสนับสนุนการดีบัก

```
# dbx a.out
Type 'help' for help.
Core file "core" is older than current program (ignored)
reading symbolic information . . .
(dbx) stop in main
[1] stop in main
(dbx) r
[1] stopped in main at line 5
    5           int a=5;
```

```
A output similar to one shown below can be observed.

PID = 6816134 TID= 24052015 EXCEPTION=131 ADDRESS = 268436372
```

sendsig

โพรบนี้ เริ่มต้นเมื่อสัญญาณถูกส่งไปยังกระบวนการผ่านแหล่งที่มาภายนอก (กระบวนการอื่น กระบวนการจากพื้นที่ผู้ใช้ จากสตรีมเคอร์เนล หรือ บริบทอินเตอร์รัปต์)

ไวยากรณ์:`@@sysproc:sendsig:<pid/*>`

```
__dispatchinfo{
 cpuid;  <- cpu id

 oldpid;            <- pid of the thread currently running
 oldtid;  <- thread id of the thread currently running
 oldpriority; <- priority of the thread currenly running
 newpid; <- pid of the new process process selected for running
 newtid; <- thread id of the thread selected for running
 newpriority; <-priority of the thread selected for running
}
```

โดยที่ pid id เป็นตัวระบุกระบวนการของกระบวนการเป้าหมายที่รับสัญญาณ โพรบนี้ไม่อนุญาตให้ระบุตัวระบุเธรด เพื่อกรองผลลัพธ์ที่ระบุเฉพาะเธรด

built-in พิเศษ

```
_sigsendinfo{
 tpid;               ← target pid
 spid;   ← source pid
 signo;        ← signal sent
}
```

โดยที่ tpid เป็นตัวระบุกระบวนการต้นทางเป้าหมาย spid ระบุแหล่งที่มาของสัญญาณ spid เป็นค่าที่ไม่ใช่ศูนย์เมื่อสัญญาณถูกส่งจากพื้นที่ผู้ใช้หรือ บริบทกระบวนการ ตัวระบุกระบวนการต้นทางเป็น 0 หากสัญญาณถูกส่งจากบริบทที่ยกเว้น หรืออินเตอร์รัปต์ ข้อมูลหมายเลขสัญญาณจะอยู่ใน signo

built-in อื่นที่สนับสนุน

__errno__kernelmode, __arg1 to __arg7, __curthread, __curproc, __mst, __tid, __pid, __ppid, __pgid, __uid, __euid

สภาวะแวดล้อม การเรียกทำงาน

รันในกระบวนการหรือสภาวะแวดล้อมแบบอินเตอร์รัปต์

หมายเหตุ: เนื่องจาก โพรบนี้สามารถเริ่มทำงานในบริบทอินเตอร์รัปต์ จึงเป็นไปได้ที่ built-in เช่น __pid, __tid ซึ่ง ขึ้นอยู่กับบริบทการเรียกทำงานเธรดอาจไม่ระบุ id กระบวนการหรือเธรด ที่สนใจ สมาชิก built-in พิเศษสำหรับโพรบนี้รับประกัน id กระบวนการหรือเธรดที่ถูกต้อง ที่มีเจตนาสำหรับกระบวนการ หรือเธรด

เมื่อโพรบนี้เริ่มทำงานในบริบทกระบวนการ สมาชิก built-in ที่ขึ้นอยู่กับบริบทการเรียกทำงานจะชี้ไปยังกระบวนการต้นทาง สมาชิก built-in เช่น __pid, __tid และ __curthread จัดเตรียม ข้อมูลที่เกี่ยวกับกระบวนการต้นทาง .

ตัวอย่าง

เมื่อต้องการพิมพ์เป้าหมายสัญญาณต้นทาง เป้าหมายสัญญาณ และหมายเลขสัญญาณของสัญญาณทั้งหมดอย่างต่อเนื่อง

```
echo '@@sysproc:sendsig:* {printf ("Source=%llu Target=%llu sig=%llu\n",__sigsendinfo->spid,__sigsendinfo->tpid,__sigsendinfo->signo);}' |
probevue

A output similar to one shown below can be observed.

Source=0 Target=6619618 sig=14
Source=0 Target=8257944 sig=20
Source=0 Target=8257944 sig=20
```

sigqueue

โพรบนี้เริ่มต้น เมื่อสัญญาณที่อยู่ในคิวกำลังถูกส่งไปยัง กระบวนการ

ไวยากรณ์:`@@sysproc:sigqueue:<pid/*>`

built-in พิเศษ

```
_sigsendinfo{
 tpid;               ← target pid
 spid;   ← source pid  preprocess.cp
 signo;        ← signal sent
}
```

เนื่องจากสัญญาณ posix อยู่ในคิวของกระบวนการ การระบุตัวระบุเธรดจะไม่ได้รับอนุญาตให้ใช้ใน โพรบนี้

built-in อื่นที่สนับสนุน

`__errno__kernelmode, __arg1 to __arg7, __curthread, __curproc, __mst, __tid, __pid, __ppid, __pgid, __uid, __euid`

โพรบนี้เริ่มต้นในบริบทของกระบวนการ ส่ง ดังนั้น built-in ที่อิงบริบทจะอ้างถึงกระบวนการการส่งในเหตุการณ์โพรบนี้

สภาวะแวดล้อม การเรียกทำงาน

โพรบนี้รันในบริบท กระบวนการ

ตัวอย่าง

```
echo '@@sysproc:sigqueue:*{printf ("%llu   %llu   %llu\n",__sigsendinfo->spid,__sigsendinfo->tpid,__sigsendinfo->signo);}' | probevue

A output similar to one shown below can be observed.

8258004   6095294   31

sigdispose

ไวยากรณ์ : @@sysproc:sigdispose:<pid/tid/*>
```

โพรบ เริ่มทำงานเมื่อสัญญาณถูกปล่อยไปยังกระบวนการเป้าหมาย ระบุ ID กระบวนการของกระบวนการ ที่รับสัญญาณนี้ในข้อมูลจำเพาะ sysprobe เพื่อกรองโพรบนี้

built-in พิเศษ

```
__sigdisposeinfo{
 tpid;         ← target pid
 ttid;          ← target tid
 signo;      ← signal whose action is being taken.
 fatal;        ← will be set if the process is going to be killed as part of signal action
}
```

built-in อื่นที่สนับสนุน

__errno__kernelmode, __arg1 to __arg7, __curthread, __curproc, __mst, __tid, __pid, __ppid, __pgid, __uid, __euid

สภาวะแวดล้อม การเรียกทำงาน

โพรบนี้ สามารถเริ่มต้นจากกระบวนการหรือบริบทแบบอินเตอร์รัปต์ หากเริ่มทำงานจากบริบทอินเตอร์รัปต์ โพรบนี้อาจไม่จัดเตรียม ค่าที่ต้องการสำหรับ built-in ที่อิงบริบท

ตัวอย่าง

ตัวระบุกระบวนการพิมพ์อย่างต่อเนื่อง ตัวระบุเธรด หมายเลขสัญญาณ และการบ่งชี้ว่าการปล่อยสัญญาณ จะส่งผลทำให้เกิดการยกเลิกของกระบวนการสำหรับกระบวนการทั้งหมด ในระบบ

```
cat sigdispose.e

@@sysproc:sigdispose:*
{
 printf ("%llu  %llu %llu %llu\n",__sigdisposeinfo->tpid,__sigdisposeinfo->ttid, __sigdisposeinfo->signo,__sigdisposeinfo->fatal);
}
```

เอาต์พุตที่คล้ายกับที่แสดงด้านล่างสามารถสังเกตเห็นได้

```
5964064  20840935 14 0
1  65539 14 0
4719084  19530213 14 0
```

sigaction

ไวยากรณ์:@@sysproc:sigaction:<pid/tid/*>

โพรบนี้เริ่มต้นเมื่อตัวจัดการสัญญาณถูกติดตั้งไว้หรือถูกแทนที่

built-in พิเศษ

```
__sigactioninfo{
 old_sighandle;            ← old signal handler function address
 new_sighandle; ←new signal handler function address
 signo;              ← Signal number
 rpid;              ← requester's pid
}
```

old_sighandle จะเป็น 0 หากตัวจัดการสัญญาณถูกติดตั้งไว้ในครั้งแรก

built-in อื่นที่สนับสนุน

__errno__kernelmode, __arg1 to __arg7, __curthread, __curproc, __mst, __tid, __pid, __ppid, __pgid, __uid, __euid

สภาวะแวดล้อม การเรียกทำงาน

โพรบนี้เริ่มต้นในสภาวะแวดล้อมกระบวนการ

หมายเหตุ: เคอร์เนล AIX ตรวจสอบให้มั่นใจได้ว่า มีเพียงหนึ่งสัญญาณเท่านั้นที่ส่งไปยังกระบวนการหรือเธรดในแต่ละครั้ง สัญญาณอื่นๆ ไปยังกระบวนการหรือเธรดถูกส่งไปเมื่อการส่งสัญญาณ เสร็จสิ้น

ตัวอย่าง

เมื่อต้องการติดตามจุดเริ่มต้นและสิ้นสุดของสัญญาณทั้งหมดในระบบ :

```
@@sysproc:sighandlestart:*
{

  signal[__tid] = __sighandlestartinfo->signo;
  printf ("Signal handler at address 0x%x invoked for thread id %llu to handle signal %llu\n",__sighandlestartinfo-
>sighandle,__curthread->tid,__sighandlestartinfo->signo);
}


@@sysproc:sighandlefinish:*
{

  printf ("Signal handler completed for thread id %llu for  signal %llu\n",__curthread->tid,signal[__tid]);
  delete (signal,__tid);
}
```

เอาต์พุตที่คล้ายกับที่แสดงด้านล่างสามารถสังเกตเห็นได้

```
Signal handler at address 0x20001d58 invoked for thread id 19923365 to handle signal 20
Signal handler completed for thread id 19923365 for  signal 20
Signal handler at address 0x10003400 invoked for thread id 20840935 to handle signal 14
Signal handler completed for thread id 20840935 for  signal 14
Signal handler at address 0x10002930 invoked for thread id 19530213 to handle signal 14
Signal handler completed for thread id 19530213 for  signal 14
Signal handler at address 0x300275d8 invoked for thread id 22348227 to handle signal 14
Signal handler completed for thread id 22348227 for  signal 14
Signal handler at address 0x20001a3c invoked for thread id 65539 to handle signal 14
Signal handler completed for thread id 65539 for  signal 14
```

sighandlefinish

โพรบเริ่มต้นที่ ความสมบูรณ์ของตัวจัดการสัญญาณ

ไวยากรณ์: @@sysproc:sighandlestart:<pid/tid/*>

built-in พิเศษที่สนับสนุน: ไม่มี

built-in อื่นที่สนับสนุน

__errno__kernelmode, __arg1 to __arg7, __curthread, __curproc, __mst, __tid, __pid, __ppid, __pgid, __uid, __euid

สภาวะแวดล้อม การเรียกทำงาน

รันในสภาวะแวดล้อมกระบวนการ สวิตช์บริบทที่ปกป้องไว้ไม่ได้รับอนุญาตให้ใช้บน CPU ที่เรียกทำงาน

changepriority

โพรบนี้เริ่มทำงานเมื่อลำดับความสำคัญของกระบวนการเปลี่ยนไป เหตุการณ์นี้ไม่ใช่ตัวจัดตารางเวลา หรือตัวแจกจ่ายที่บังคับ

ไวยากรณ์: `@@sysproc:changepriority:<pid/tid/*>`

หมายเหตุ: การเปลี่ยนลำดับความสำคัญอาจยังไม่จำเป็น; ความสำเร็จของการเปลี่ยนลำดับความสำคัญไม่ได้รับประกันไว้

built-ins พิเศษที่สนับสนุน

```
__chpriorityinfo{
 pid;
 old_priority;   <- current priority
 new_priority; <-  new scheduling priority of the thread.
}
```

สภาวะแวดล้อมการเรียกทำงาน

โพรบนี้รันอยู่ในสภาวะแวดล้อมกระบวนการ

built-in อื่นที่สนับสนุน

`__errno__kernelmode, __arg1 to __arg7, __curthread, __curproc, __mst, __tid, __pid, __ppid, __pgid, __uid, __euid, __ublock, __execname, __pname`

ตัวอย่าง

เพื่อติดตามกระบวนการทั้งหมด ที่มีลำดับความสำคัญ ที่ต้องเปลี่ยน:

```
echo '@@sysproc:changepriority:* { printf ("%s priority changing from %llu to %llu\n",__pname,__chpriorityinfo->old_priority,__chpriorityinfo-
>new_priority);}' | probevue
```

เอาต์พุตที่คล้ายกับที่แสดงด้านล่างสามารถสังเกตเห็นได้

```
xmgc priority changing from 60 to 17
xmgc priority changing from 17 to 60
xmgc priority changing from 60 to 17
xmgc priority changing from 17 to 60
xmgc priority changing from 60 to 17
```

offreadyq

โพรบนี้ เริ่มต้นเมื่อเธรดถูกถอนออกจากระบบ ที่รันคิว

ไวยากรณ์: `@@sysproc:offreadyq:<pid/tid/*>`

built-in พิเศษที่สนับสนุน

```
__readyprocinfo{
 pid;  <- process id of thread becoming ready
 tid;  <- Thread id.
 priority; <- priority of the thread
}
```

built-in อื่นที่สนับสนุน

__errno__kernelmode, __arg1 to __arg7, __curthread, __curproc, __mst, __tid, __pid, __ppid, __pgid, __uid, __euid

สภาวะแวดล้อม การเรียกทำงาน

รันในกระบวนการหรือสภาวะแวดล้อมแบบอินเตอร์รัปต์

กรณีศึกษา: เวลาติดตามที่ใช้โดยเธรดที่กำลังดำเนินการ I/O เพื่อขอกลับไปยังคิว ที่พร้อมใช้งาน

```
@@BEGIN
{
 printf ("           Pid      Tid      Time           Delta\n");
}

@@sysproc:offreadyq :*
{
 ready[__tid] = timestamp();
 printf ("offreadyq: %llu %llu %W\n",__readyprocinfo->pid,__readyprocinfo->tid,ready[__tid]);
}

@@sysproc:onreadyq :*
{

 if (diff_time(ready[__tid],0,MICROSECONDS))
 {
  auto:diff = diff_time (ready[__tid],timestamp(),MICROSECONDS);
  printf ("onreadyq : %llu %llu %W      %llu\n",__readyprocinfo->pid,__readyprocinfo->tid,ready[__tid],diff);
  delete (ready,__tid);
 }
}
```

เอาต์พุตที่คล้ายกับที่แสดงไว้ด้านล่างอาจสังเกตเห็นได้

```
           Pid      Tid      Time           Delta
offreadyq: 7799280 20709717 5s 679697µs
onreadyq : 7799280 20709717 5s 679697µs      6
offreadyq: 7799280 20709717 5s 908716µs
onreadyq : 7799280 20709717 5s 908716µs      3
offreadyq: 7799280 20709717 6s 680186µs
onreadyq : 7799280 20709717 6s 680186µs      5
offreadyq: 7799280 20709717 6s 710720µs
onreadyq : 7799280 20709717 6s 710720µs      4
offreadyq: 7799280 20709717 6s 800720µs
onreadyq : 7799280 20709717 6s 800720µs      2
offreadyq: 7799280 20709717 6s 882231µs
onreadyq : 7799280 20709717 6s 882231µs      2
offreadyq: 7799280 20709717 6s 962313µs
onreadyq : 7799280 20709717 6s 962313µs      2
offreadyq: 7799280 20709717 6s 980311µs
onreadyq : 7799280 20709717 6s 980311µs      2
```

onreadyq

โพรบนี้ เริ่มต้นเมื่อเธรดวางอยู่ในคิวที่พร้อมใช้งานสำหรับระบบหรือตำแหน่งในคิวที่พร้อมใช้งาน ถูกปรับเปลี่ยน

ไวยากรณ์:@@sysproc:offreadyq:<pid/tid/*>

built-ins พิเศษ ที่สนับสนุน

```
__readyprocinfo{
 pid;  <- process id of thread becoming ready
 tid;  <- Thread id.
 priority; <- priority of the thread
}
```

built-in อื่นที่สนับสนุน

__errno__kernelmode, __arg1 to __arg7, __curthread, __curproc, __mst, __tid, __pid, __ppid, __pgid, __uid, __euid

สภาวะแวดล้อม ข้อยกเว้น

รันใน กระบวนการหรือสภาวะแวดล้อมแบบอินเตอร์รัปต์

แจกจ่าย

โพรบนี้เริ่มต้นเมื่อตัวแจกจ่ายของระบบ ถูกเรียกเพื่อเลือกเธรดที่รันบน CPU ที่ระบุเฉพาะ

ไวยากรณ์:@@sysproc:dispatch:<pid/tid/*>

built-in พิเศษที่สนับสนุน

```
__dispatchinfo{
 cpuid;  <- CPU where selected  thread will run.
 oldpid;            <- pid of the thread currently running
 oldtid;  <- thread id of the thread currently running
 oldpriority; <- priority of the thread currenly running
 newpid; <- pid of the new process process selected for running
 newtid; <- thread id of the thread selected for running
 newpriority; <-priority of the thread selected for running
}
```

built-in อื่นที่สนับสนุน

__errno__kernelmode, __arg1 to __arg7, __curthread, __curproc, __mst, __tid, __pid, __ppid, __pgid, __uid, __euid

สภาวะแวดล้อม ข้อยกเว้น

รันใน สภาวะแวดล้อมแบบอินเตอร์รัปต์เท่านั้น

ตัวอย่าง

id เธรดกระบวนการพิมพ์ของเก่าและเธรดที่เลือกไว้บน CPU '0' พร้อมกับเวลากระจายที่เกี่ยวข้องกับการเริ่มต้นสคริปต์

```
echo '@@sysproc:dispatch:* when (__cpuid == 0){printf ("%llu %llu %W\n",__dispatchinfo->oldtid,__dispatchinfo->newtid,timestamp());}' |
probevue
```

เอาต์พุตที่คล้ายกับที่แสดงด้านล่างสามารถสังเกตเห็นได้

```
24641983 20709717 0s 48126μs
20709717 23593357 0s 48164μs
23593357 20709717 0s 48185μs
20709717 23593357 0s 48214μs
```

```
23593357 20709717 0s 48230μs
20709717 23593357 0s 48288μs
23593357 261 0s 48303μs
261 20709717 0s 48399μs
```

ตัวอย่างที่ II

เวลาที่ใช้บน CPU '0' ตามเธรดในระหว่างเหตุการณ์การแจกจ่าย

```
@@BEGIN
{
 printf ("Thread cpu Time-Spent\n");
}

@@sysproc:dispatch:* when (__cpuid == $1)
{
 if (savetime[__cpuid] != 0)
  auto:diff = diff_time (savetime[__cpuid],timestamp(),MICROSECONDS);
 else
  diff = 0;
 savetime[__cpuid] = timestamp();
 printf ("%llu %llu %llu\n",__dispatchinfo->oldtid,__dispatchinfo->cpuid,diff);
}

# probevue cputime.e 6
Thread cpu Time-Spent
3146085 6 0
3146085 6 9995
3146085 6 10002
3146085 6 10008
3146085 6 99988
3146085 6 100006
3146085 6 99995
3146085 6 99989
3146085 6 100010
3146085 6 100001
3146085 6 100000
3146085 6 99998
```

เนื่องจากสามารถสังเกตเธรด 3146085 จะถูกกระจายบน CPU ที่ช่วงเวลา 1 วินาทีที่ไม่มีการทำเธรดให้สมบูรณ์สำหรับ CPU นี้

oncpu

โพรบนี้ เริ่มต้นเมื่อกระบวนการใหม่หรือเธรดได้ CPU มา

ไวยากรณ์:`@@sysproc:oncpu:<pid/tid/*>`

โดยที่ `pid` คือตัวระบุกระบวนการและ `tid` คือตัวระบุเธรดของกระบวนการ หรือเธรดที่ได้รับ CPU

built-ins พิเศษ ที่สนับสนุน

```
__dispatchinfo{
 cpuid;  <- CPU where selected  thread will run.
 newpid; <- pid of the new process process selected for running
 newtid; <- thread id of the thread selected for running
 newpriority; <-priority of the thread selected for running
}
```

built-in อื่นที่สนับสนุน

`__errno__kernelmode, __arg1 to __arg7, __curthread, __curproc, __mst, __tid, __pid, __ppid, __pgid, __uid, __euid`

สภาวะแวดล้อม การเรียกทำงาน

รันใน สภาวะแวดล้อมแบบอินเตอร์รัปต์เท่านั้น

ตัวอย่าง

เมื่อต้องการพิมพ์เวลาที่ใช้โดยเธรดของ sysncd บน CPU ทั้งหมด

```
#!/usr/bin/probevue

@@BEGIN

{

 printf ("PROCESSID THREADID CPU TIME\n");

}



@@sysproc:oncpu:$1

{

 savetime[__cpuid] = timestamp();

}



@@sysproc:offcpu:$1

{

 if (savetime[__cpuid] != 0)

  auto:diff = diff_time (savetime[__cpuid],timestamp(),MICROSECONDS);

 else

  diff = 0;

 printf ("%llu %llu %llu %llu\n",

  __dispatchinfo->oldpid,

  __dispatchinfo->oldtid,

  __dispatchinfo->cpuid,

  diff);

}


# cputime.e `ps aux|grep syncd| grep -v grep| cut -f 6 -d " "`
```

เอาต์พุตที่คล้ายกับที่แสดงด้านล่างสามารถสังเกตเห็นได้

```
3735998 18612541 0 2

3735998 15663427 0 1

3735998 15073557 0 1

3735998 18743617 0 1

3735998 18874693 0 1

3735998 18809155 0 15

3735998 18940231 0 20

3735998 18547003 0 1

3735998 19267921 0 1

3735998 19071307 0 17

3735998 18678079 0 1

3735998 18481465 0 1

3735998 19202383 0 15

3735998 19005769 0 1

3735998 19136845 0 19

3735998 6160689 0 190
```

offcpu

โพรบนี้ เริ่มต้นเมื่อกระบวนการหรือเธรดถูกแจกจ่ายจาก CPU

ไวยากรณ์:`@@sysproc:dispatch:<pid/tid/*>`

built-ins พิเศษ ที่สนับสนุน

```
__dispatchinfo{
 cpuid;  <- CPU where selected  thread will run.
 newpid; <- pid of the new process process selected for running
 newtid; <- thread id of the thread selected for running
 newpriority; <-priority of the thread selected for running
}
```

built-in อื่นที่สนับสนุน

`__errno__kernelmode, __arg1 to __arg7, __curthread, __curproc, __mst, __tid, __pid, __ppid, __pgid, __uid, __euid`

สภาวะแวดล้อม การเรียกทำงาน

รันใน สภาวะแวดล้อมแบบอินเตอร์รัปต์เท่านั้น

blockthread

โพรบนี้เริ่มต้นเมื่อเธรด บล็อกจากการรันบน CPU การบล็อกเป็นฟอร์มของการหยุดชั่วคราวเมื่อเธรดหยุดชั่วคราวโดยไม่ต้องพัก รีซอร์สใดๆ

ไวยากรณ์: `@@sysproc:blockthread:*`

built-ins พิเศษ ที่สนับสนุน

```
__sleepinfo{
 pid;
 tid;
 waitchan;   <-- wait channel of this sleep.
}
```

built-in อื่นที่สนับสนุน

`__errno__kernelmode, __arg1 to __arg7, __curthread, __curproc, __mst, __tid, __pid, __ppid, __pgid, __uid, __euid`

สภาวะแวดล้อม การเรียกทำงาน

รันใน สภาวะแวดล้อมแบบอินเตอร์รัปต์เท่านั้น

foldcpu

โพรบนี้เริ่มต้นเมื่อ CPU หลัก เกี่ยวข้องกับสิ่งที่ต้อง fold โพรบนี้ไม่ได้เกิดขึ้นในบริบทกระบวนการและไม่ต้องถูกกรอง ด้วย pid หรือ tid

ไวยากรณ์: `@@sysproc:foldcpu:*`

built-ins พิเศษ ที่สนับสนุน

```
__foldcpuinfo{
 cpuid;  <- logical cpu id which triggers core folding
 gpcores;         <- general purpose (unfolded, non-exclusive) cores available.
}
```

built-in อื่นที่สนับสนุน

`__errno__kernelmode, __arg1 to __arg7.`

ตัวอย่าง:

เมื่อต้องการติดตามเหตุการณ์ CPU folding ทั้งหมดในระบบ :

```
__foldcpuinfo{
 cpuid;  <- logical cpu id which triggers core folding
 gpcores;         <- general purpose (unfolded, non-exclusive) cores available.
}
```

bindprocessor

ไวยากรณ์: `@@sysproc:bindprocessor:<pid/tid/*>`

โพรบนี้เริ่มต้นเมื่อเธรด หรือกระบวนการถูกโยงกับ CPU Bindprocessor เป็นเหตุการณ์ถาวรและต้องไม่ยุ่งกับ สวิตช์ CPU แบบชั่วคราว

built-ins พิเศษ ที่สนับสนุน

```
__bindprocessorinfo{
 ispid       <- 1 if cpu is bound to process; 0 for a thread
 id;     <- thread or process id.

 cpuid;

};
```

built-in อื่นที่สนับสนุน

`__errno__kernelmode, __arg1 to __arg7, __curthread, __curproc, __mst, __tid, __pid, __ppid, __pgid, __uid, __euid`

สภาวะแวดล้อม การเรียกทำงาน

รันในสภาวะแวดล้อมกระบวนการ

changecpu

โพรบนี้เริ่มต้นเมื่อเธรดเปลี่ยน CPU ชั่วคราว เหตุการณ์นี้ มักถูกดักจับระหว่างเหตุการณ์ CPU funneling หรือการกระโดดแบบตั้งใจ ของเหตุการณ์ kproc บางเหตุการณ์เพื่อดำเนินการ built-in พิเศษของงานที่เกี่ยวข้องกับ CPU (กระบวนการ xmgc กระโดดไปยัง CPU ทั้งหมดเพื่อจัดการ เคอร์เนลฮีพ)

ไวยากรณ์: `@@sysproc:changecpu:*>`

built-ins พิเศษ ที่สนับสนุน

```
__changecpuinfo
{
 oldcpuid; <-source CPU
 newcpuid;  <- target CPU
 pid;
 tid;  <-Thread id
}
```

บิวด์อินที่สนับสนุนอื่นๆ

__errno__kernelmode, __arg1 to __arg7, __curthread, __curproc, __mst, __tid, __pid, __ppid, __pgid, __uid, __euid

สภาวะแวดล้อม การเรียกทำงาน

รันในสภาวะแวดล้อมกระบวนการ

ตัวอย่าง

```
@@sysproc:changecpu:*

{
printf ("changecpu PID=%llu TID=%llu old_cpuid=%d new_cpuid= %d \n",
__changecpuinfo->pid,__changecpuinfo->tid,__changecpuinfo->oldcpuid,__changecpuinfo->newcpuid);

}
```

เอาต์พุตที่คล้ายกับที่แสดงไว้ด้านล่างอาจสังเกตเห็นได้

```
changecpu PID=852254 TID=1769787 old_cpuid=26 new_cpuid= 27

changecpu PID=852254 TID=1769787 old_cpuid=-1 new_cpuid= 0

changecpu PID=852254 TID=1769787 old_cpuid=0 new_cpuid= 1

changecpu PID=852254 TID=1769787 old_cpuid=1 new_cpuid= 2
```

resourceattach

โพรบนี้ ถูกใช้เมื่อพ่วงต่อรีซอร์สกับรีซอร์สอื่นในระบบ

ไวยากรณ์:`@@sysproc:resourceattach:*>`

built-ins พิเศษ ที่สนับสนุน

```
__srcresourceinfo{
 type;
 subtype;
 id;  <- resource type identifier
 offset;   <-offset if a memory resource
 length;  <- length if a memory resource
 policy;
}
__tgtresourceinfo{
 type;
 subtype;
 id;  <- resource type identifier
 offset;  <-offset if a memory resource
 length;  <- length if a memory resource
 policy;
}
```

โดยที่ ชนิดและชนิดย่อยสามารถมีหนึ่งในค่าต่อไปนี้

ตารางที่ 69. โพรบ resourceattach: ค่าชนิดและชนิดย่อย.

ชนิดรีซอร์ส	คำอธิบาย
R_NADA	ไม่มีสิ่งใด – ข้อมูลจำเพาะไม่ถูกต้อง
R_PROCESS	กระบวนการ
R_RSET	ชุดรีซอร์ส
R_SUBRANGE	ช่วงของหน่วยความจำ
R_SHM	หน่วยความจำแบบแบ่งใช้
R_FILDES	ไฟล์ที่ระบุโดยไฟล์เปิด
R_THREAD	เธรด
R_SRADID	ตัวระบุ SRAD
R_PROCMEM	หน่วยความจำการประมวลผล

บิวด์อินที่สนับสนุนอื่นๆ

`__errno__kernelmode, __arg1 to __arg7, __mst.`

สภาวะแวดล้อมการเรียกทำงาน

รันใน สภาวะแวดล้อมกระบวนการ

resourcedetach

โพรบนี้ถูกใช้ เมื่อถอดรีซอร์สออกจากรีซอร์สอื่นในระบบ

ไวยากรณ์:`@@sysproc:resourcedetach:*>`

built-ins พิเศษ ที่สนับสนุน

```
__srcresourceinfo{
 type;
 subtype;
 id;  <- resource type identifier
 offset;   <-offset if a memory resource
 length;  <- length if a memory resource
 policy;
}

__tgtresourceinfo{
 type;
 subtype;
 id;  <- resource type identifier
 offset;  <-offset if a memory resource
 length;  <- length if a memory resource
 policy;
}
```

โดยที่ ชนิดและชนิดย่อยสามารถมีหนึ่งในค่าต่อไปนี้

ตารางที่ 70. โพรบ resourcedetach: ค่าชนิดและชนิดย่อย.

ชนิดรีซอร์ส	คำอธิบาย
R_NADA	ไม่มีสิ่งใด – ข้อมูลจำเพาะไม่ถูกต้อง
R_PROCESS	กระบวนการ
R_RSET	ชุดรีซอร์ส
R_SUBRANGE	ช่วงของหน่วยความจำ
R_SHM	หน่วยความจำแบบแบ่งใช้
R_FILDES	ไฟล์ที่ระบุโดยไฟล์เปิด
R_THREAD	เธรด
R_SRADID	ตัวระบุ SRAD
R_PROCMEM	หน่วยความจำการประมวลผล

บิวด์อินที่สนับสนุนอื่นๆ

`__errno__kernelmode, __arg1 to __arg7, __mst, __tid, __pname.`

สภาวะแวดล้อม การเรียกทำงาน

รันในสภาวะแวดล้อมกระบวนการ

drphasestart

โพรบนี้ถูกใช้เมื่อตัวจัดการ dr เกี่ยวข้องกับสิ่งที่ต้องเรียก

ไวยากรณ์: `@@sysproc:drphasestart:*`

built-ins พิเศษ ที่สนับสนุน

```
__drphaseinfo{
 dr_operation;   ← dr operation
 dr_flags;
 dr_phase;
 handler_rc;    ← always 0 in drphasestart
}
```

dr_operation สามารถมีหนึ่งในค่าต่อไปนี้:

- การดำเนินการ DR
- DR_RM_MEM_OPER
- DR_ADD_MEM_OPER
- DR_RM_CPU_OPER
- DR_ADD_CPU_OPER

- DR_CPU_SPARE_OPER
- DR_RM_CAP_OPER
- DR_ADD_CAP_OPER
- DR_RM_RESMEM_OPER
- DR_PMIG_OPER
- DR_WMIG_OPER
- DR_WMIG_CHECKPOINT_OPER
- DR_WMIG_RESTART_OPER
- DR_SOFT_RES_CHANGES_OPER
- DR_ADD_MEM_CAP_OPER
- DR_RM_MEM_CAP_OPER
- DR_CPU_AFFINITY_REFRESH_OPER
- DR_AME_FACTOR_OPER
- DR_PHIB_OPER
- DR_ACC_OPER
- DR_CHLMB_OPER
- DR_ADD_RESMEM_OPER

แฟล็ก dr สามารถเป็นชุดของค่าต่อไปนี้:

- แฟล็ก
- DRP_FORCE
- DRP_RPDP
- DRP_DOIT_SUCCESS
- DRP_PRE_REGISTERED
- DRP_CPU DRP_MEM DRP_SPARE
- DRP_ENT_CAP
- DRP_VAR_WGT
- DRP_RESERVE
- DRP_PMIG DRP_WMIG
- DRP_WMIG_CHECKPOINT
- DRP_WMIG_RESTART
- DRP_SOFT_RES_CHANGES
- DRP_MEM_ENT_CAP
- DRP_MEM_VAR_WGT
- DRP_CPU_AFFINITY_REFRESH
- DRP_AME_FACTOR

- DRP_PHIB
- DRP_ACC_UPDATE
- DRP_CHLMB

บิวด์อินที่สนับสนุนอื่นๆ

__errno__kernelmode, __arg1 ถึง __arg7, __tid

สภาวะแวดล้อมการเรียกทำงาน

รันในกระบวนการหรือสภาวะแวดล้อมแบบอินเตอร์รัปต์

ตัวอย่าง

เชลล์สคริปต์สำหรับ ProbeVue

ส่วนนี้อธิบายเชลล์สคริปต์สำหรับการรันคำสั่ง ProbeVue

โปรแกรมผู้ช่วย shell สำหรับ ProbeVue

สคริปต์ shell ต่อไปนี้มีประโยชน์ขณะรัน ProbeVue:

sprobevue

สคริปต์ shell ที่ตัดอาร์กิวเมนต์ทั้งหมดในเครื่องหมายอัญประกาศคู่:

```
#!/usr/bin/ksh
#
# sprobevue:
#
# Simple helper function for probevue
# Wraps arguments to probevue in double quotes
#
# Usage: sprobevue <probevue flags> <script> <args>
#  Doesn't support the -c and -A flags of probevue
#

usage()
{
 echo "Usage: sprobevue <probevue flags> <script> <args>" >&2
 echo " Doesn't support the -c and -A flags of probevue" >&2
 exit 1
}

CMD=probevue
# Generate command to execute

while getopts 'c:A:I:s:o:t:X:' zargs
do
 case $zargs in
  I|s|o|t|X) CMD="$CMD -$zargs $OPTARG"       ;;
```

```
  ?) usage
 esac
done

shift $(($OPTIND -1))

if [ -n "$1" ]
then
 CMD="$CMD $1"
 shift
fi

for i
do
 CMD="$CMD \"$i\""
done

# Execute command
$CMD
```

prgrep

สคริปต์ shell ที่พิมพ์ ID กระบวนการที่กำหนดชื่อกระบวนการ:

```
#/usr/bin/ksh
#
# prgrep:
#
# Simple helper function for probevue
# Prints all process IDs with given process name
#
#  Need options to print only one process
#   to print process belong to a certain UID

#
# Usage: prgrep <process_name>
#        prgrep -p <processID>
#


usage()
{
 echo "Usage: prgrep <process_name>" >&2
 echo "       prgrep -p <process_ID>" >&2
 exit 1
}

[ -z "$1" ] && usage

if [ $1 = "-p" ]
then
 [ -z "$2" ] && usage
 pid=$2
 export pid
 ps -e | awk 'BEGIN {pid = ENVIRON["pid"]} {if ($1 == pid) print $4}'
```

```
else
 pname=$1
 export pname
 ps -e | awk 'BEGIN {pname = ENVIRON["pname"]} {if ($4 == pname) print $1}'
fi
```

ProbeVue error messages

ตามที่ได้กล่าวไว้ข้างต้น การรันคำสั่ง **probevue** อาจต้องการ privilege ถ้าผู้ใช้ดั้งเดิมพยายามรันคำสั่ง **probevue** กรอบงาน RBAC จะตรวจพบเหตุการณ์นี้ และเกิดความล้มเหลวในการประมวลผลของคำสั่งโดยทันที

```
$ probevue kernel.e
ksh: probevue: 0403-006 Execute permission denied.
```

ส่วน *การกำหนดสิทธิ์และสิทธิ์* ของ "การรัน ProbeVue" ในหน้า 340 อธิบายวิธีอนุญาตให้ผู้ใช้ที่ไม่ใช่ root ที่มีการกำหนดสิทธิ์และสิทธิ์เพื่อใช้คำสั่ง **probevue**

คอมไพลเลอร์ ProbeVue ซึ่งเป็นคอมไพลเลอร์ในตัวคำสั่ง **probevue** จะพิมพ์ข้อความแสดงความผิดพลาดโดยละเอียด ในระหว่างเฟสของการคอมไพล์ เมื่อคอมไพลเลอร์ตรวจพบข้อผิดพลาดทางไวยากรณ์ ความผิดพลาดทางความหมาย หรือข้อผิดพลาดทางความเข้ากันไม่ได้ของชนิด โปรดพิจารณาสคริปต์ต่อไปนี้:

```
/* Syntax error example:
 * syntaxbug.e
 */
@@BEGIN
{
        int i, j, k;

        i = 4;
        j = 22;

        k = i _ z;

        printf("k = %d\n", k);

        exit();
}
```

สคริปต์ที่นำหน้ามีข้อผิดพลาดทางไวยากรณ์บนบรรทัดที่ 11 คอลัมน์ที่ 15 นั่นคือ ข้อความสั่งกำหนดค่า แทนที่จะเป็นเครื่องหมายลบ (-) หรือสัญลักษณ์ขีดเส้นใต้ (_) ถูกพิมพ์ด้วยความผิดพลาด สำหรับการรันสคริปต์ คอมไพลเลอร์ ProbeVue จะจับข้อผิดพลาดนี้ และสร้างข้อความแสดงความผิดพลาด:

```
# probevue syntaxbug.e
syntaxbug.e: token between line 11: column 15 and line11: column 15: , expected
instead of this token
```

คอมไพล์เลอร์ ProbeVue ยังเรียกใช้งานการเรียกระบบภายใน เพื่อตรวจสอบว่าข้อกำหนดคุณสมบัติโพรบในสคริปต์ Vue ถูกต้อง ข้อผิดพลาดทั่วไปคือ การส่งผ่าน ID กระบวนการที่ไม่ถูกต้อง หรือ ID กระบวนการที่ออกไปแล้วในจุดโพรบ tuple ข้อผิดพลาดอื่นๆ โดยทั่วไปคือ การลืมส่งค่า ID กระบวนการเป็นอาร์กิวเมนต์บนบรรทัดรับคำสั่ง เมื่อสคริปต์ต้องการค่า ID นั้น โปรดพิจารณาสคริปต์ต่อไปนี้:

```
/* simpleprobe.e
 */
@@syscall:$1:read:entry
{
        printf("In read system call: thread ID = %d\n", __tid);
        exit();
}
```

สคริปต์ก่อนหน้านี้ต้องการ ID ของกระบวนการที่เป็นอาร์กิวเมนต์ เพื่อแทนที่ตัวแปร '$1' ในจุดโพรบ tuple ในบรรทัดที่ 3 เคอร์เนลจะส่งคืนข้อผิดพลาดหากคุณพยายามโพรบกระบวนการที่ออกแล้ว หรือไม่มีอยู่ และยังเกิดความล้มเหลวหาก ID ของกระบวนการบ่งชี้ถึงเคอร์เนลกระบวนการ หรือกระบวนการ init นอกจากนี้ คุณไม่สามารถโพรบกระบวนการที่ไม่ได้เป็นของคุณ ยกเว้นว่า คุณมี privilege ที่ต้องมีในการโพรบกระบวนการของผู้ใช้รายอื่น คุณสามารถใช้คำสั่ง **prgrep** ด้วยแฟล็ก **-p** เพื่อพิมพ์ชื่อกระบวนการที่กำหนด ID ของกระบวนการ

หมายเหตุ: คำสั่งนี้จะสร้างเอาต์พุตเปล่า หาก ID ของกระบวนการที่ระบุไม่มีอยู่

```
# probevue simpleprobe.e 233
probevue: The process does not exist.
ERR-19: Line:3 Column:3 Invalid probe string
# prgrep -p 232
#
# probevue simpleprobe.e 1
ERR-19: Line:3 Column:3 Invalid probe string
# prgrep -p 1
init
# probevue simpleprobe.e
ERR-19: Line:3 Column:3 Invalid probe string
```

คำสั่ง **probevue** ยังสามารถตรวจพบได้ หากผู้ใช้ที่ไม่มี privilege ที่ต้องการพยายามเข้าถึงตัวแปรเคอร์เนล โปรดพิจารณาสคริปต์ kernel.e จากส่วนของโปรแกรมตัวอย่าง เซสชันตัวอย่างต่อไปนี้แสดงสิ่งที่เกิดขึ้น ถ้าคุณพยายามรันเซสชันนี้ด้วยผู้ใช้ที่ไม่มี privilege ที่ต้องการ:

```
$ probevue kernel.e
ERR-56: Line:93 Column:39 No authority to access kernel variable
ERR-56: Line:99 Column:23 No authority to access kernel variable
ERR-56: Line:100 Column:24 No authority to access kernel variable
ERR-56: Line:101 Column:25 No authority to access kernel variable
ERR-56: Line:102 Column:24 No authority to access kernel variable
ERR-102: Line:140 Column:13 Operation not allowed
ERR-46: Line:140 Column:9 Invalid Assignment, Type mismatch
```

หลังจากคอมไพล์สคริปต์ Vue เป็นผลสำเร็จแล้ว คำสั่ง **probevue** จะเรียกใช้งานการเรียกระบบเพื่อเริ่มต้นเซสชัน ProbeVue ใหม่ที่ส่งโค้ดระดับกลาง ซึ่งสร้างโดยคอมไพลเลอร์ การเรียกระบบจะล้มเหลวหากกรอบงาน ProbeVue ล้มเหลวในการกำหนดค่าเริ่มต้นให้กับเซสชัน ProbeVue ใหม่ ซึ่งมีหลายเหตุผล สำหรับความล้มเหลวนี้ ตัวอย่างเช่น การเริ่มต้นเซสชันใหม่อาจเป็นสาเหตุทำให้รีซอร์สหน่วยความจำ สำหรับผู้ใช้มีค่าเกินกว่าข้อจำกัดที่ระบุโดยผู้ดูแลระบบ เซสชันอาจต้องการรีซอร์สหน่วยความจำเพิ่มเติม ซึ่งมากกว่าที่อนุญาตไว้สำหรับเซสชันเดี่ยว และอาจเป็นฟังก์ชันที่ไม่ได้รับอนุญาตให้ใช้ใน ตัวจัดการช่วงของโพรบ หนึ่งในกระบวนการที่กำลังถูกโพรบ สามารถออกได้ หลังจากทำการตรวจสอบเฟสของการคอมไพล์แล้ว เมื่อเซสชันไม่สามารถเริ่มต้นได้ เคอร์เนลจะเกิดความล้มเหลวในการเรียกระบบโดยส่งคืนข้อผิดพลาดเฉพาะแบบ 64 บิต

กรอบงาน ProbeVue สามารถยกเลิกเซสชัน ProbeVue ที่เริ่มต้นแล้วและแอ็คทีฟอยู่ได้เป็นผลสำเร็จ หากพบข้อผิดพลาดที่รุนแรงหรือไม่สามารถกู้คืนได้ขณะที่ออกคำสั่งดำเนินการโพรบ ข้อผิดพลาดที่อาจเกิดขึ้นได้รวมถึง เซสชันที่มีขนาดเกินหรือข้อ

จำกัดเกี่ยวกับหน่วยความจำของผู้ใช้ (ข้อกำหนดด้านหน่วยความจำสำหรับตัวแปรเธรดโลคัล และตัวแปรลิสต์สามารถโตขึ้นตามความคืบหน้าของเซสชัน) สตริงชั่วคราวมีขนาดเกินหรือพื้นที่สแต็กมีข้อจำกัด เข้าถึงดัชนีแบบไม่มีอาร์เรย์ พยายามหารด้วยศูนย์ และอื่นๆ ในกรณีทั้งหมดนี้ เคอร์เนลจะส่งคืนหมายเลขข้อผิดพลาดเฉพาะแบบ 64 บิต ขณะที่ยกเลิกเซสชัน

เมื่อเซสชันเกิดความล้มเหลวไม่ว่าจะ ณ ตอนเริ่มต้น หรือหลังจากที่เริ่มต้นเป็นผลสำเร็จแล้ว คำสั่ง **probevue** จะพิมพ์ข้อความแสดงความผิดพลาดทั่วไป ซึ่งรวมถึงหมายเลขข้อผิดพลาดเฉพาะแบบ 64 บิต ในรูปของเลขฐานสิบหก จากนั้นจึงออกจากเซสชัน แผนภูมิต่อไปนี้แสดงความหมายของข้อผิดพลาดทั่วไปแบบ 64 บิตบางส่วนที่สามารถส่งคืนโดยเคอร์เนลได้:

เกิดข้อผิดพลาดเกี่ยวกับเคอร์เนล	ความหมาย	เกิดขึ้นที่
0xEEEE00008C285034	หน่วยความจำไม่เพียงพอขณะจัดสรรบัฟเฟอร์ trace หลัก	เซสชันเริ่มต้นขึ้น
0xEEEE00008C285035	หน่วยความจำไม่เพียงพอขณะที่จัดสรรบัฟเฟอร์ trace สำรอง	เซสชันเริ่มต้นขึ้น
0xEEEE00008C52002B	หน่วยความจำไม่เพียงพอขณะจัดสรรหน่วยเก็บสำหรับสตริงข้อกำหนดคุณสมบัติของโพรบ	เซสชันเริ่มต้นขึ้น
0xEEEE000096284122	หน่วยความจำไม่เพียงพอขณะจัดสรรหน่วยเก็บสำหรับหน่วยเก็บเธรดโลคัล	เซสชันเริ่มต้นขึ้น
0xEEEE000081284049	ใช้ฟังก์ชันการเข้าถึงพื้นที่ของผู้ใช้ใน ตัวจัดการช่วงของโพรบ	เซสชันเริ่มต้นขึ้น
0xEEEE0000D3520022	จำนวนของเซสชันที่จำกัดไว้สำหรับผู้ใช้ปกติ	เซสชันเริ่มต้นขึ้น
0xEEEE000096284131	แอดเดรสที่ผิดกฎเกณฑ์ถูกส่งไปยังฟังก์ชัน **get_userstring**	การเรียกใช้งานการดำเนินการโพรบ
0xEEEE00008C520145	ตรงกับค่าสูงสุดของข้อจำกัดเธรดสำหรับตัวแปรเธรดโลคัล	การเรียกใช้งานการดำเนินการโพรบ

ฟังก์ชัน RAS event

ฟังก์ชัน "RAS events" เป็นชุดที่มีสิทธิ์พิเศษของฟังก์ชัน Vue ที่จัดเตรียมไว้สำหรับระบบที่มีความพิเศษมากหรือเพื่อการดีบักแอ็พพลิเคชัน แต่ไม่ได้มีวัตถุประสงค์สำหรับการใช้โดยทั่วไป และจะจัดเตรียมการ trace ระบบและตัวช่วยสร้างดัมพ์ ฟังก์ชันเหล่านี้คือฟังก์ชัน "pass through" ที่อนุญาตให้สคริปต์ Vue เรียกใช้งานเคอร์เนลเซอร์วิสได้โดยตรง เพราะฉะนั้นจึงมีความเสี่ยงสำหรับการใช้ฟังก์ชันเหล่านี้ คุณอาจต้องการ privilege พิเศษเพื่อเรียกใช้งานฟังก์ชันเหล่านี้ในสคริปต์ Vue ของคุณ: คุณต้องมีสิทธิแบบผู้ใช้ root หรือมีสิทธิใน **aix.ras.probevue.rase**

หากต้องการหลีกเลี่ยงความเสี่ยงของฟังก์ชันเหล่านี้ให้ส่งแฟล็ก **-K** ไปยังคำสั่ง **probevue** มิฉะนั้น ฟังก์ชันเหล่านี้จะไม่ปรากฏขึ้นจากภาษา Vue อย่างสมบูรณ์

การสร้าง trace recor

d

ฟังก์ชัน Vue สำหรับการสร้างเร็กคอร์ด trace ของระบบ (และ LMT trace) มีไวยากรณ์ที่คล้ายคลึงกับอินเตอร์เฟสเคอร์เนลที่ฟังก์ชันเหล่านั้น เรียกใช้งานภายในการครอบคลุม เพราะฉะนั้น การบันทึกเร็กคอร์ด trace จากสคริปต์ Vue ไม่มีความแตกต่างจากการทำเช่นนี้ และมีข้อจำกัดบางอย่างดังต่อไปนี้ :

- ถ้า trace ของระบบไม่ได้เริ่มต้นขึ้น หรือค่า *hookid* ไม่ได้ถูกดักจับโดย trace ของระบบ การดำเนินการเหล่านี้ไม่ได้สร้างเร็กคอร์ด trace ของระบบ (LMT trace ในบัฟเฟอร์ทั่วไปสำหรับ TRCHKLx trace จะยังคงมีความพยายามอยู่ แต่ LMT อาจถูกปิดใช้งาน)
- คุณไม่สามารถสร้างเร็กคอร์ด trace จากภายใน **@@systrace** Vue clause ได้ การเรียกฟังก์ชัน trace จะสร้างเร็กคอร์ด trace สำหรับบัฟเฟอร์ทั่วไปของ LMT สำหรับ TRCHKLx trace ในกรณีนี้ LMT ถูกเปิดใช้งานอยู่
- คุณไม่สามารถโพรบเหตุการณ์ trace ที่สร้างโดย ProbeVue เหล่านี้ได้ เฉพาะเคอร์เนลและแอ็พพลิเคชันที่สร้าง trace เท่านั้นที่สามารถโพรบได้
- คุณต้องมี privilege ซึ่งมีสิทธิแบบ root หรือสิทธิใน **aix.ras.probevue.rase**

ฟังก์ชัน Vue ต่อไปนี้มีอยู่สำหรับการบันทึกเร็กคอร์ด trace ของระบบ คำบรรจุข้อมูลทั้งหมดของ **long long integers** ชนิดนี้:

TRCHKL0(hookID)
trace โดยไม่มีคำบรรจุข้อมูล

TRCHKL1(hookID, D1)
trace ที่มี 1 คำบรรจุข้อมูล

TRCHKL2(hookID, D1,D2)
trace ที่มี 2 คำบรรจุข้อมูล

TRCHKL3(hookID, D1,D2,D3)
trace ที่มี 3 คำบรรจุข้อมูล

TRCHKL4(hookID, D1,D2,D3,D4)
trace ที่มี 4 คำบรรจุข้อมูล

TRCHKL5(hookID, D1,D2,D3,D4,D5)
trace ที่มี 5 คำบรรจุข้อมูล

void trcgenk(int channel, int hook_ID, unsigned long long data_word, int length, untyped buffer)
trace บัฟเฟอร์

ฟังก์ชัน trace เหล่านี้จะผนวกการประทับเวลาเข้ากับข้อมูลเหตุการณ์ พารามิเตอร์ *hookid* ในฟังก์ชันเหล่านี้ จะอยู่ในรูปของ `0xhhhh0000` ซึ่งไม่ได้หมายความว่า ค่า *hookid* จำเป็นต้องเป็นค่าคงที่ แต่เป็นเพียงการบ่งชี้วิธีการจัดรูปแบบค่า *hookid* เท่านั้น

หมายเหตุ: ค่า *hookid* แบบ 12 บิตที่ไม่ได้รับการสนับสนุน จะใช้หลัก hex สามหลักที่อยู่ทางซ้ายสุด และหลักที่ 4 จะมีค่าศูนย์

สำหรับเคอร์เนลเซอร์วิส **trcgenk** พารามิเตอร์ *buffer* คือตัวชี้ไปยังไบต์ความยาวของข้อมูล เพื่อ trace ซึ่งมีขนาดประมาณ 4096 ไบต์ พารามิเตอร์ *buffer* สามารถเป็นตัวแปรภายนอกได้ เช่น เคอร์เนล หรือตัวชี้แอ็พพลิเคชัน ไปยังข้อมูลที่มีเลขรหัสส่วนตัวแล้ว หรือตัวแปรสคริปต์ เช่น สตริงหรือโครงสร้างอินสแตนซ์ Vue ข้อกำหนดคุณสมบัติ "untyped" เป็นแบบย่อของสิ่งนี้

หมายเหตุ: เคอร์เนลเซอร์วิส **trcgenk** จะ trace ไปยัง trace ของระบบเท่านั้น ไม่ได้ trace ไปยังบัฟเฟอร์ trace ของ LMT

คุณสามารถใช้หมายเลขช่องสัญญาณที่ไม่ใช่ศูนย์ แต่คุณต้องมั่นใจว่า ช่องสัญญาณที่ระบุเปิดใช้งานสำหรับการ trace แล้ว ค่าที่ส่งคืนจากคำสั่ง **trace** ที่เริ่มต้น trace ของจุดสนใจสามารถส่งผ่านไปยังสคริปต์ Vue สำหรับวัตถุประสงค์นี้ได้ การใช้ช่องสัญญาณที่ปิดใช้งานจะส่งผลทำให้ไม่มี trace

ฟังก์ชัน trace เหล่านี้ไม่ได้ส่งคืนค่าใดๆ

สำหรับข้อมูลเพิ่มเติม โปรดดู เคอร์เนลเซอร์วิส **trcgenk** ใน *Technical Reference: Kernel and Subsystems, Volume 1* และ แมโครสำหรับการบันทึกเหตุการณ์ trace in *Performance management*

การหยุดทำงาน trace

หากต้องการตรึง trace ของระบบในทันทีหลังจากที่เกิดเหตุการณ์ที่ต้องการแล้ว คุณสามารถใช้ `void trcoff()` ได้ในสคริปต์ Vue ฟังก์ชันนี้ปิดใช้งานการ trace ช่องสัญญาณศูนย์โดยทันที คุณต้องหยุดทำงาน trace ด้วยวิธีปกติด้วยคำสั่งภายนอก **trcstop** ใน ProbeVue หากการประมวลผล trace จะเสร็จสิ้นแบบปกติ

คุณสามารถหยุดทำงาน LMT และคอมโพเนนต์ trace ได้ในทันที ดังนั้น การ trace แบบต่อเนื่องจะไม่ตัดข้อมูลของจุดสนใจ ฟังก์ชันที่ทำงานต่อ จำเป็นต้องมี เนื่องจากไม่มีบรรทัดรับคำสั่งที่เท่าเทียมกัน ซึ่งพร้อมใช้งานในการเริ่มต้น trace เหล่านี้ ซึ่งมีฟังก์ชัน Vue ใหม่ต่อไปนี้:

```
void mtrcsuspend()
void ctsuspend()
void mtrcresume()
void ctresume()
```

รูทีน **ctsuspend** จะหยุดการ trace คอมโพเนนต์ทั้งหมด คุณไม่สามารถใช้รูทีนนี้สำหรับการเลือก trace ที่ต้องการหยุดทำงานโดยคอมโพเนนต์ ซึ่งจะสนับสนุน trace คอมโพเนนต์เท่านั้น และไม่ต้องการ trace อื่นใดที่แมโคร CT_HOOKx จะสามารถร้องขอได้ เช่น ระบบและการบันทึก LMT

คุณต้องใช้ฟังก์ชันการควบคุม trace เหล่านี้ด้วยความระมัดระวัง เนื่องจากไม่มี serialization ของโค้ดการ trace เคอร์เนลที่ส่งผลกระทบ คุณต้องมั่นใจว่า เฉพาะหนึ่งสคริปต์หรือคำสั่งเท่านั้นที่จะมีผลกระทบต่อการ trace ในหนึ่งครั้ง

การหยุดระบบ

คุณสามารถยกเลิกระบบ และใช้ดัมพ์เต็มโดยใช้รูทีนต่อไปนี้ :

```
void abend(long long code, long long data_word, ...)
```

รูทีนนี้จะคล้ายกับเคอร์เนลเซอร์วิส **abend** ยกเว้นเฉพาะพารามิเตอร์ข้อมูลมากที่สุด 7 ค่า (ซึ่งจะถูกโหลดภายในรีจิสเตอร์ r3 ผ่านไปยัง r10) จะได้รับการยอมรับที่นี่

พารามิเตอร์ที่ไม่ได้กำหนดชนิดไว้

ในฟังก์ชันต้นแบบที่ต้องปฏิบัติตาม พารามิเตอร์บางตัวของฟังก์ชันเคอร์เนลที่เทียบเท่า จะถูกพิมพ์อย่างกำกวม โดยทั่งไป คอมไพลเลอร์ Vue จะดำเนินการตรวจสอบ ชนิดสำหรับพารามิเตอร์ทั้งหมดที่ผ่านไปยังฟังก์ชัน Vue แต่พารามิเตอร์ที่ถูกกำหนดให้เป็นชนิด "untyped" จะได้รับการยกเว้นจากการตรวจสอบ ตัวอย่างเช่น สตริงเผื่อเลือก อาจถูกส่งผ่านด้วย NULL ขณะที่เคอร์เนลเหล่านี้ให้เซอร์วิสโดยตรงในเคอร์เนล แต่ถ้าฟังก์ชัน Vue จะถูกกำหนดตามการใช้พารามิเตอร์ชนิด String NULL จะไม่สามารถยอมรับได้ หากต้องการหลีกเลี่ยง ความไม่สะดวกของการส่งผ่านค่าเปล่าแทน และอนุญาตให้ฟังก์ชัน Vue ใช้พารามิเตอร์ที่เหมือนกันกับอินเตอร์เฟสเคอร์เนลต่อไปนี้ ฟังก์ชันเหล่านี้ต้องถูกกำหนดเป็นพารามิเตอร์ที่ไม่มีชนิด

พารามิเตอร์แบบไม่มีชนิด จัดเตรียมความเป็นอิสระของการผ่าน NULL แทนการผาน สตริง Vue ที่แท้จริง แต่ ต้องระวังเมื่อระบุค่าสำหรับพารามิเตอร์ "untyped" เนื่องจาก คอมไพลเลอร์จะยอมรับชนิดใดๆ สำหรับพารามิเตอร์

หมายเหตุ: ไม่มีข้อกำหนดคุณสมบัติตัวแปร "untyped" จริงๆ ในภาษา Vue ซึ่งจะเป็นเพียงแค่การใช้ในช่วงสั้นๆ

การใช้ live dump

เซอร์วิส ProbeVue สำหรับความสามารถของ live dump ของเคอร์เนลได้ถูกจัดเตรียมไว้ และจะดูคล้ายกับเคอร์เนลเซอร์วิสที่สอดคล้องกัน สำหรับข้อมูลโดยละเอียดเกี่ยวกับ live dump ของเคอร์เนลเซอร์วิส โปรดดู **livedump** ของเคอร์เนลเซอร์วิสใน *Technical Reference: Kernel and Subsystems, Volume 1*

ข้อยกเว้นในความเหมือนกันทั่วไปนี้คือโครงสร้าง **ldmp_parms** ซึ่งจะไม่เปิดเผยที่ระดับของสคริปต์ ฟังก์ชัน **ldmp_setupparms** แบบในตัวที่เป็นอินสแตนซ์ไพรเวตของโครงสร้าง ซึ่งถูกจัดสรร และส่งคืนค่าไปยังตัวเรียกโดยอ้อมซึ่งเป็นคุกกี้แบบ 64 บิต ซึ่งต้องถูกส่งผ่านไปยังเซอร์วิส live dump ตามลำดับในตำแหน่งนั้น เฉพาะเซสชันเดียวเท่านั้น ที่สามารถใช้โครงสร้างตัวบุคคลในหนึ่งครั้ง คุณสามารถใช้เซอร์วิส live dump อื่นๆ เพื่อให้คล้ายคลึงกับไวยากรณ์ของชุดสำเนาของเคอร์เนล เนื่องจากการจัดสรรที่ซ่อนอยู่นี้ (และการจัดสรรที่ซ่อนอยู่ซึ่งถูกทำโดยเซอร์วิสของ live dump ของตัวเคอร์เนลเอง) จำเป็นต้องเรียกเคอร์เนลเซอร์วิส **ldmp_freeparms** หรือเคอร์เนลเซอร์วิส the **livedump** อย่างใดอย่างหนึ่ง หากเคอร์เนลเซอร์วิส **ldmp_setupparms** ถูกเรียกและส่งคืนเป็นผลสำเร็จ มิฉะนั้น เซสชันปัจจุบันจะดำเนินการกับโครงสร้างไพรเวตที่ตนเป็นเจ้าของ ซึ่งเป็นสาเหตุทำให้การเรียก **ldmp_setupparms** ทั้งหมดในอนาคต เกิดความล้มเหลว ถ้าโครงสร้างไพรเวตถูกรีลีส โครงสร้างนั้นอาจไม่สามารถใช้ได้โดยเจ้าของก่อนหน้านี้อีกต่อไป โดยไม่มีการเรียก **ldmp_setupparms** ตัวอื่น ห้ามใช้แฟล็ก LDT_POST ด้วยเคอร์เนลเซอร์วิส **ldmp_setupparms** ตามที่ได้กล่าวโดยนัยถึงการไม่สนับสนุน การอ้างถึงโครงสร้างที่ซ่อนไว้ในอนาคต

แอ็พพลิเคชัน live dump ทั่วไปต้องพักโครงสร้างที่ซ่อนไว้ สำหรับช่วงเวลาสั้นๆ เท่านั้นภายใน **probevue** clause เดี่ยว โครงสร้างที่ซ่อนไว้ที่เป็นเจ้าของโดยเซสชัน และสามารถนำมาใช้ได้โดย Vue clause ในเซสชันนั้น ตามความเป็นจริง กรอบงานจะรีลีสโครงสร้างไพรเวต และรีซอร์สเคอร์เนลอื่นๆ ด้วยเคอร์เนลเซอร์วิส **ldmp_freeparms** โดยอัตโนมัติเมื่อเซสชัน ProbeVue ยกเลิก

เนื่องจากอิลิเมนต์ของโครงสร้าง **ldmp_parms** ไม่สามารถมองเห็น ProbeVue ซึ่งต้องการหรืออนุญาตให้กำหนดค่าเริ่มต้นโดยตัวเรียก ที่ถูกตั้งค่าโดยใช้พารามิเตอร์พิเศษที่ส่งไปยัง ProbeVue ในเวอร์ชันของเคอร์เนลเซอร์วิส **ldmp_setupparms** แทน

```
long long ldmp_setupparms(String symptom,required symptom string
untyped title,        dump title string or NULL
untyped prefix,       dump file name prefix string or NULL
untyped func,         failing function name string or NULL
long long errcode,   error code
int flags,           dump characteristics
int prio )           dump priority
```

ฟังก์ชัน **ldmp_setupparms** Vue ที่นำหน้าคืออินเตอร์เฟสไปยังเคอร์เนลเซอร์วิสของชื่อที่เหมือนกัน ยกเว้นว่า โครงสร้าง **ldmp_parms** ไม่สามารถเรียกสคริปต์ Vue ได้ ค่าที่ส่งคืนต้องถูกส่งไปยังเซอร์วิส live dump อื่นซึ่งเป็นตัวแทนของตัวชี้ไปยังโครงสร้าง **ldmp_parms** แม้ว่าจะมีค่าเป็นเลขจำนวนเต็ม 64 บิต

สตริง *symptom* จำเป็นต้องมีตัวถูกดำเนินการ String ขณะที่สตริง *title*, *prefix* และ *func* คือตัวเลือก ส่งค่า String หรือ NULL สำหรับสามพารามิเตอร์เหล่านี้ ค่า String ทั้งหมดต้องเป็นค่าโลคัลในสคริปต์ Vue แฟล็ก *flags* และ *prio* สามารถเป็นค่าศูนย์หรือค่าจากไฟล์ส่วนหัวของเคอร์เนล **sys/livedump.h** ต้องใช้ค่าคงที่จำนวนเต็มที่เหมาะสมที่นี่ แม้ว่าจะมี ค่าอื่น

ค่าต่อไปนี้คือค่าที่มีประโยชน์สำหรับพารามิเตอร์ *flags* :

```
LDT_ONEPASS  0x02 limit dump to one pass
LDT_NOADDCOMPS 0x08 components can't be added by callbacks
LDT_NOLOG   0x10 no error is to be logged
LDT_FORCE   0x20 force this dump
```

เนื่องจากดัมพ์จะถูกใช้จาก ProbeVue' ที่ปิดใช้งานในสภาวะแวดล้อมภายใน ซึ่งต้องถูก serialize ซิงโครนัส ส่งผ่านค่าหนึ่งค่าให้กับดัมพ์

ค่าต่อไปนี้เป็นค่าที่ยอมรับได้สำหรับพารามิเตอร์ *prio* :

```
LDPP_INFO   1 informational dump
LDPP_CRITICAL 7 critical dump (this is the default)
```

ถ้าศูนย์ถูกระบุไว้สำหรับพารามิเตอร์ *prio* LDPP_CRITICAL คือค่าดีฟอลต์ตามเซอร์วิส **ldmp_setupparms** เฉพาะค่าที่ไม่ใช่ศูนย์เท่านั้นที่จะถูกเก็บไว้ในโครงสร้าง **ldmp_parms** ที่ซ่อนไว้เพื่อแทนที่ค่านี้

ค่าที่ส่งคืนตามความสำเร็จจะต้องเป็นคุกกี้ที่มีค่าบวกซึ่งแสดงถึงความเป็นเจ้าของโครงสร้าง **ldmp_parms** ที่ซ่อนไว้

สำหรับความล้มเหลวใดๆ ค่าที่ส่งคืนจะเป็นค่าลบดังนี้:

ค่า	คำอธิบาย
EINVAL_EVM_ COOKIE	บ่งชี้ถึงโครงสร้าง **ldmp_parms** ส่วนบุคคลที่ไม่พร้อมใช้งาน
EINVAL_EVM_ STRING	บ่งชี้ถึงพารามิเตอร์ที่มีค่า String ที่ไม่ถูกต้อง

ฟังก์ชัน Vue ที่อธิบายตามลำดับทั้งหมดจะส่งคืนการบ่งชี้ถึงความล้มเหลว ในรูปแบบที่เหมือนกัน ด้วยหมายเลขข้อผิดพลาดของเคอร์เนล ในทางลบ:

ค่า	คำอธิบาย
EINVAL_EVM_ COOKIE	บ่งชี้ว่า ตัวเลือกไม่ได้ระบุคุกกี้ที่ถูกต้อง ซึ่งแสดงความเป็นเจ้าของโครงสร้าง **ldmp_parms** ส่วนบุคคล
EINVAL_EVM_ STRING	บ่งชี้ถึงพารามิเตอร์ที่มีค่า String ที่ไม่ถูกต้อง
EINVAL_EVM_ EXTID	บ่งชี้ถึงพารามิเตอร์ *extid* ที่ไม่สนับสนุน และต้องเป็นศูนย์

หมายเลขข้อผิดพลาดของเคอร์เนลอื่นๆ สามารถส่งผ่านย้อนกลับได้ โดยเคอร์เนลเซอร์วิสต่อไปนี้:

long long ldmp_freeparms (long long cookie)

หลังจากที่เคอร์เนลเซอร์วิส **ldmp_setupparms** ได้ส่งคืนเป็นผลสำเร็จ โครงสร้าง **ldmp_parms** ภายในจะถูกจัดสรรไปยังสคริปต์ Vue ที่รันอยู่ คุณต้องล้างข้อมูลรีซอร์สนี้ให้ว่าง บวกกับรีซอร์สเคอร์เนลภายในที่จัดสรร โดยเซอร์วิสที่เพิ่มคอมโพเนนต์ให้กับดัมพ์ของคุณ โดยการใช้ดัมพ์ที่เรียกเคอร์เนลเซอร์วิส **livedump** หรือโดยการเรียกเคอร์เนลเซอร์วิส **ldmp_freeparms** ซึ่งจะรีลีสโครงสร้าง **ldmp_parms** ภายในสำหรับการใช้ในอนาคต

long long livedump (long long cookie)

หลังจากที่เคอร์เนลเซอร์วิส **ldmp_setupparms** และอย่างน้อยหนึ่งเซอร์วิสที่เพิ่มคอมโพเนนต์ (และคอมโพเนนต์จำลอง) ให้กับดัมพ์ได้ถูกเรียกแล้ว ดัมพ์จะถูกร้องขอโดยเซอร์วิส **livedump** เซอร์วิสนี้จะสร้าง live dump ที่เกิดขึ้นจริงในไฟล์ **/var/adm/ras/livedump** ตามข้อกำหนดคุณสมบัติผ่านเคอร์เนลเซอร์วิส **ldmp_setupparms** และเซอร์วิส live dump อื่นๆ ถูกเรียกใช้งาน พารามิเตอร์ *cookie* คือคุกกี้ที่ส่งคืนโดยการเรียกเซอร์วิสเคอร์เนล

ldmp_setupparms ในตอนต้น ค่าที่ส่งคืนจะมีค่าศูนย์หากดัมพ์ถูกใช้เป็นผลสำเร็จ ส่งคืนค่า EINVAL_EVM_COOKIE หากคุกกี้ไม่ถูกต้อง และส่งคืนค่าหมายเลขข้อผิดพลาดเคอร์เนลหากข้อผิดพลาดเกิดขึ้นในระหว่างการประมวลผลเคอร์เนล **livedump**

long long dmp_compspec(แฟล็ก long long, แฟล็ก DCF_xxx ที่กำหนดใน sys/dump.h untyped comp, คอมโพเนนต์ที่ต้องการเพิ่ม (ด้วย ras_block_t ชื่อ นามแฝง และอื่นๆ) long long cookie คุกกี้ที่ส่งคืนโดย ldmp_setupparms long long extid ไม่สนับสนุน – ต้องมีค่าศูนย์ untyped p1, พารามิเตอร์คอมโพเนนต์ตัวแรกที่เป็นไปได้ ...); พารามิเตอร์คอมโพเนนต์เพิ่มเติม

คุณสามารถเพิ่มคอมโพเนนต์ใดๆ ที่สนับสนุน live dump ให้กับดัมพ์โดยเรียกเซอร์วิสนี้ ซึ่งเป็นค่าเฉพาะในฟังก์ชันสำหรับเคอร์เนลเซอร์วิส พร้อมกับชื่อที่เหมือนกัน ซึ่งจะยกเว้นในสถานการณ์ต่อไปนี้:

- พารามิเตอร์ *extid* ซึ่งอนุญาตให้ dmp_extid_t (long) ส่งคืนค่าในสภาวะแวดล้อมการโปรแกรมมิงเคอร์เนล ไม่สนับสนุนและต้องมีค่าศูนย์ มิฉะนั้น EINVAL_EVM_EXTID จะถูกส่งคืน ไม่มีวิธีใด ที่ส่งคืนตัวชี้ไปยัง ProbeVue หน่วยความจำที่รับค่านี้ ซึ่งอาจถูกใช้ด้วยเคอร์เนลเซอร์วิส **dmp_compext** ซึ่งไม่ได้รับการสนับสนุนใน ProbeVue คุณสามารถเรียกเซอร์วิส **dmp_compspec** ในหลายๆ ครั้งแทน
- เคอร์เนลเซอร์วิสอนุญาตให้มีจำนวนของพารามิเตอร์ p1, p2 และอื่นๆ โดยที่พารามิเตอร์ NULL ที่เพิ่มเติมต้องตามหลังพารามิเตอร์จริงตัวล่าสุด เพื่อยกเลิกรายการพารามิเตอร์ ฟังก์ชัน Vue จะยอมรับพารามิเตอร์ได้มากสุดสี่ตัวคือ p1, p2 และอื่นๆ พารามิเตอร์ตัวล่าสุด ต้องเป็นศูนย์เพื่อแจ้งให้เคอร์เนลเซอร์วิสทราบถึงจำนวนของพารามิเตอร์เหล่านี้ที่มีอยู่ ดังนั้น ผลกระทบคือ คุณสามารถระบุได้มากสุด 3 ค่า อินเตอร์เฟสจะบังคับให้พารามิเตอร์ปฏิบัติตาม พารามิเตอร์ตัวแปร 3 ตัวล่าสุดด้วยค่าศูนย์โดยอัตโนมัติ เพื่อมั่นใจว่า เป็นไปตามกฎเหล่านี้
- พารามิเตอร์ *comp* สามารถเป็นความยาว เคอร์เนลสำหรับแอดเดรส ras_block_t หรือสตริงได้ตามความเหมาะสม ชนิดจะไม่ได้รับการตรวจสอบ
- ค่าแฟล็กสำหรับเคอร์เนล **#define** ไม่ใช่ส่วนหนึ่งของ ProbeVue

long long ras_block_lookup(String path)

ฟังก์ชันนี้มีอยู่ใน ras_block_t ที่สอดคล้องกับพารามิเตอร์ชื่อพาธคอมโพเนนต์ ซึ่งสามารถใช้ให้เกิดประโยชน์ได้สำหรับการเรียกเคอร์เนลเซอร์วิส **dmp_ct** ที่จำเป็นต้องใช้แอดเดรส เป็นต้น ถ้าคุณไม่สามารถค้นหาแอดเดรสเพิ่มเติมได้อย่างง่ายดายใน ตัวแปรเคอร์เนล

ค่าที่ส่งคืนจากฟังก์ชันนี้คือ เคอร์เนลแอดเดรสของ ras_block_t ที่ร้องขอ หรือ NULL ถ้าไม่สามารถค้นหา ras_block_t ได้

ฟังก์ชันต่อไปนี้คือฟังก์ชัน "pass through" แบบง่ายๆ ทั้งหมด ซึ่งอนุญาตให้สคริปต์ Vue เรียกใช้งานเคอร์เนลเซอร์วิสที่สอดคล้องกัน โดยตรง รายชื่อพารามิเตอร์บางตัวมีสมาชิกที่ไม่ได้ใช้สำหรับความเข้ากันได้กับเคอร์เนล ดังนั้น คุณจึงสามารถใช้เอกสารคู่มือเคอร์เนล ได้โดยตรง ค่า 0 ต้องถูกส่งผ่านสำหรับพารามิเตอร์ที่แสดงถึงการไม่ได้ใช้ คุณสามารถใช้เซอร์วิสเหล่านี้ด้วยวิธีเดียวกับการทำชุดสำเนาเคอร์เนล ยกเว้นว่า แอดเดรสของโครงสร้าง **ldmp_parms** จะถูกแทนที่ด้วยคุกกี้ที่ส่งคืนจากเคอร์เนลเซอร์วิส **ldmp_setupparms**

เช่นเดียวกัน ค่าส่งคืนที่ติดลบจะบ่งชี้ถึงข้อผิดพลาด ซึ่งสามารถเป็นหมายเลขข้อผิดพลาดของเคอร์เนล จากเคอร์เนลเซอร์วิสต่อไปนี้ หรือจากรูทีนอินเตอร์เฟส หากคุกกี้หรือสตริงไม่ถูกต้อง อินเตอร์เฟสต่อไปนี้ จะมีความยืดหยุ่นสูงซึ่งพร้อมใช้งานกับเคอร์เนล หรือส่วนขยายเคอร์เนลที่ผลักดันให้เกิด live dump

```
long long dmp_context (long long flags,DCF_xxx flags from dump.h
long long cookie,  cookie returned by ldmp_setupparms
long long name,  unused by this function
long long ctx_type, DMP_CTX_xxx flags from dump.h untyped p2)
parameter dependent on ctx_type (NULL, mst addr, cpuid, tid)
```

```
long long dmp_ct( long long flags, DCF_xxx flags from dump.h
long long cookie,  cookie returned by ldmp_setupparms
long long name,   unused by this function
untyped  rasb,    component's ras_block_t pointer
long long size)   amount of CT buffer to dump or 0 for all

long long dmp_eaddr( long long flags, DCF_xxx flags from dump.h
long long cookie,   cookie returned by ldmp_setupparms
String name,       cdt name
untyped  addr,       first address to dump
long long size)     number of bytes to dump

long long dmp_errbuf(long long flags, DCF_xxx flags from dump.h
long long cookie,    cookie returned by ldmp_setupparms
long long name,     unused by this function
long long erridx,   0 for global error log, or wpar id
long long p2)   unused

long long dmp_mtrc(long long flags, DCF_xxx flags from dump.h
long long cookie,  cookie returned by ldmp_setupparms
long long name,     unused by this function
long long com_size, amount of LMT common data to dump
long long rare_size)amount of LMT rare data to dump

long long dmp_pid( long long flags, DCF_xxx flags from dump.h
long long cookie,  cookie returned by ldmp_setupparms
long long name,  unused by this function
long long pid,   id of process to dump
long long p2)   unused

long long dmp_systrace (long long flags, DCF_xxx flags from dump.h
long long cookie,  cookie returned by ldmp_setupparms
long long name,  unused by this function
long long size,  amount to dump
long long p2)   unused

long long dmp_tid( long long flags, DCF_xxx flags from dump.h
long long cookie,  cookie returned by ldmp_setupparms
long long name,  unused by this function
long long tid,   id of thread to dump
long long p2)   unused
```

หมายเหตุ: คุณต้องเรียกเคอร์เนลเซอร์วิส **ldmp_freeparms** หลังจากเกิดความล้มเหลวใดๆ ในรูทีนก่อนหน้านี้ ซึ่งคุณต้องการปล่อยดัมพ์

สคริปต์ต่อไปนี้คือตัวอย่างที่ใช้ live dump ที่มีขนาดเล็ก และง่าย เคอร์เนลสัญลักษณ์ dc_data จะเอ็กซ์พอร์ตโครงสร้างจากเคอร์เนล ซึ่งมีรูปร่างจริงและเนื้อหาจริงที่ไม่มีความสำคัญกับตัวอย่างนี้

```
_kernel struct {int i1; int i2; int i3; int i4;} dc_data;

@@BEGIN
{
 long long ldmp_parms;
 long long rc;

 rc = ldmp_setupparms( "dc_data dump",
    "My Sample Dump",  /* dump title */
    "pvdump",  /* dump path prefix */
```

```
    NULL,   /* no function name */
    0x1122334455667788LL, /* error code */
    0x10,   /* LDT_NOLOG flag */
    0);   /* default dump prio */
 printf("ldmp_setupparms rc = %016llx\n", rc);
 if (rc < 0) {
  exit();
 }

 ldmp_parms = rc; /* cookie for other livedump functions */

 /*
  * Add 16 bytes of kernel data to sample dump.
  * Note that "dc_data" passes the structure's address.
  */
 rc = dmp_eaddr(0, ldmp_parms, "dc_data", dc_data, sizeof(dc_data));
 if (rc) {
  printf("dump_eaddr failed: %llx\n", rc);
  ldmp_freeparms(ldmp_parms);
  exit();
 }

 /*
  * Take the sample live dump.
  */
 rc = livedump(ldmp_parms);
 if (rc) {
  printf("livedump failed: %llx\n", rc);
 }

 exit();
}
```

การใช้สัญลักษณ์ #define สำหรับแฟล็ก live dump

ตัวอย่างสคริปต์ shell probe.dump ต่อไปนี้อาจมีประโยชน์ ถ้าคุณต้องการใช้สัญลักษณ์ที่กำหนดไว้จริงสำหรับแฟล็ก live dump แทนที่จะเป็นการแทนที่จากไฟล์ส่วนหัว ซึ่งจะดักจับนิยามที่เกี่ยวข้องจากไฟล์ **livedump.h** และ **dump.h** และใช้ตัวประมวลผลก่อนใน C เพื่อแทนที่ค่าให้คุณก่อนที่จะส่งสคริปต์ของคุณไปยัง ProbeVue สคริปต์ของคุณ ต้องทำตามกฎต่อไปนี้:

- ต้องไม่ขึ้นต้นด้วยข้อคิดเห็น #!/usr/bin/probevue
- ต้องไม่ใช้สัญลักษณ์ที่ขึ้นต้นด้วย LDPP_, LDT_, DCF_ หรือ DMP_ ซึ่งขัดแย้งกับนิยามที่อยู่ในไฟล์ส่วนหัว

ห้ามสร้างไฟล์ที่ชื่อ **pvdump.*** เนื่องจากสคริปต์ต่อไปนี้จะเขียนทับไฟล์เหล่านั้น

```
#!/bin/ksh
#
# Helper script for Vue scripts that need to pick up
# the values of the various flags used by livedump.
#
# The Vue script $1
# must not contain a "#!/usr/bin/probevue" comment because
# the C preprocessor doesn't like it.

sed -n      \
```

```
 -e '/(/d'    \
 -e '/^#define LDPP_/p'   \
 -e '/^#define LDT_/p'   \
 -e '/^#define DCF_/p'   \
 -e '/^#define DMP_CTX_/p'  \
      /usr/include/sys/dump.h   \
      /usr/include/sys/livedump.h  \
    > pvdump.h

echo "#include \"pvdump.h\"" > pvdump.c
cat $1 >> pvdump.c
cc -P pvdump.c
/usr/bin/probevue -K pvdump.i
rm pvdump.[cih]
```

โปรแกรมมิงแบบมัลติเธรด

ส่วนนี้ให้แนวทางสำหรับการเขียนโปรแกรมแบบ มัลติเธรดโดยใช้ไลบรารีเธรด (**libpthreads.a**)

ไลบรารีเธรด AIX จะอ้างอิงตาม X/Open Portability Guide Issue 5 มาตรฐาน ด้วยเหตุผลนี้ ข้อมูลต่อไปนี้จะแสดงไลบรารีเธรดตามที่ได้นำไปปฏิบัติสำหรับ AIX ของ XPG5 มาตรฐาน

โปรแกรมมิงแบบขนาน จะใช้ผลพลอยได้ของระบบแบบมัลติโพรเซสเซอร์ ขณะที่คงไว้ซึ่งความเข้ากันได้ของไบนารีแบบเต็มกับระบบแบบยูนิโพรเซสเซอร์ที่มีอยู่ ตัวช่วยโปรแกรมมิงแบบขนาน จะอ้างอิงตามแนวคิดของเธรด

ข้อดีของการใช้โปรแกรมมิงแบบขนาน แทนการใช้เทคนิคเกี่ยวกับโปรแกรมมิงแบบอนุกรมมีดังนี้:

- โปรแกรมมิงแบบขนานสามารถปรับปรุงผลการทำงานของโปรแกรม
- ในซอฟต์แวร์ทั่วๆ ไปบางรุ่นจะเหมาะกับเทคนิคเกี่ยวกับโปรแกรมมิงแบบขนาน

แต่เดิม กระบวนการเธรดเดี่ยวจำนวนมากได้ถูกนำมาใช้ เพื่อบรรลุการโปรแกรมมิงแบบขนาน แต่บางโปรแกรมจะได้ประโยชน์จากระดับของการทำงานแบบขนานที่ดีกว่า กระบวนการแบบมัลติเธรดมีการทำงานแบบขนานภายใน กระบวนการและแบ่งใช้แนวความคิดเกี่ยวกับโปรแกรมมิงกระบวนการ เธรดเดียวที่มีหลายกระบวนการ

ข้อมูลต่อไปนี้นำเสนอเธรด และตัวช่วยโปรแกรมมิงที่เชื่อมโยง ซึ่งยังกล่าวถึงหัวข้อทั่วไปที่เกี่ยวข้องกับโปรแกรมมิงแบบขนาน:

หมายเหตุ: ในคอลเล็กชันหัวข้อนี้ คำว่า *เธรด* ใช้เพียงลำพัง หมายถึง *เธรดผู้ใช้* นี้ยังใช้กับการอ้างถึงในการอ้างถึง สภาวะแวดล้อมโหมดผู้ใช้ แต่ไม่ใช้ในหัวข้อที่สัมพันธ์กับเคอร์เนลกำลัง

การทำความเข้าใจเธรดและกระบวนการ

เธรด คือการควบคุมสายงานอย่างเป็นอิสระที่ทำงานภายในพื้นที่แอดเดรสเดียวกันกับ การควบคุมสายงานอย่างเป็นอิสระอื่นๆ ภายในกระบวนการ

ตามหลักเกณฑ์แล้ว คุณสมบัติของเธรดและการประมวลผล จะถูกจัดกลุ่มภายใน entity เดี่ยวเรียกว่า *กระบวนการ* ในระบบปฏิบัติการอื่น เธรดอาจถูกเรียกว่า *กระบวนการแบบ lightweight* หรือในความหมายของคำว่า *เธรด* is อาจมีความแตกต่างกันเพียงเล็กน้อย

ส่วนต่อไปนี้จะกล่าวถึงความแตกต่างระหว่างเธรด และการประมวลผล

สำหรับระบบของการประมวลผลเธรดเดี่ยวแบบเก่า การประมวลผลจะมีชุดของคุณสมบัติ ในระบบมัลติเธรด คุณสมบัติเหล่านี้ ถูกแบ่งเป็นกระบวนการ และเธรด

เธรดมีข้อจำกัดบางอย่างและไม่สามารถใช้สำหรับวัตถุประสงค์พิเศษ ซึ่งต้องการโปรแกรมที่มีการประมวลผลหลายกระบวนการ

หลักการที่เกี่ยวข้อง:

"ขอบเขต Contention และระดับสภาวะพร้อมกัน" ในหน้า 505
ขอบเขต contention ของเธรดผู้ใช้จะกำหนดวิธีการ แม็พกับเคอร์เนลเธรด

"การทำกระบวนการซ้ำและการยกเลิก" ในหน้า 525
เนื่องจากกระบวนการทั้งหมดมีอย่างน้อยหนึ่งเธรด การสร้าง (นั่นคือ การทำซ้ำ) และการยกเลิกกระบวนการหมายความถึงการสร้าง และการยกเลิกเธรด

คุณสมบัติของกระบวนการ

กระบวนการในระบบมัลติเธรดคือเอนทิตีที่ เปลี่ยนแปลงได้

ซึ่งจะถูกพิจารณาตามกรอบของการประมวลผล และมีแอ็ตทริบิวต์การประมวลผลแบบดั้งเดิม เช่น:

- ID การประมวลผล ID กลุ่มการประมวลผล ID ผู้ใช้ และ ID กลุ่ม
- สภาพแวดล้อม
- ไดเร็กทอรีการทำงาน

การประมวลผลยังจัดเตรียมพื้นที่แอดเดรสทั่วไป และรีซอร์สของระบบทั่วไป ดังต่อไปนี้:

- File descriptors
- การดำเนินการกับสัญญาณ
- ไลบรารีที่แบ่งใช้
- เครื่องมือการสื่อสารระหว่างกระบวนการ (เช่น message queues ไพพ์ อุปกรณ์สัญญาณ และหน่วยความจำที่แบ่งใช้)

คุณสมบัติเธรด

เธรดคือความสามารถในการกำหนดตารางเวลา entity เธรดคือความสามารถในการกำหนดตารางเวลา entity

ซึ่งมีคุณสมบัติเหล่านั้น ที่จำเป็นต้องมีเพื่อมั่นใจว่ามีการควบคุมสายงานที่เป็นอิสระ และจะมีคุณสมบัติดังต่อไปนี้ :

- สแต็ก
- คุณสมบัติการกำหนดตารางเวลา (เช่น นโยบายหรือลำดับความสำคัญ)
- ชุดของสัญญาณที่คงค้างหรือถูกบล็อก
- ข้อมูลที่ระบุเฉพาะเธรดบางอย่าง

ตัวอย่างของข้อมูลที่ระบุเฉพาะเธรดคือ ตัวบ่งชี้ข้อผิดพลาด **errno** ในระบบมัลติเธรด **errno** ไม่ใช่ตัวแปรโกลบอลอีกต่อไป แต่โดยปกติรูทีนย่อยจะส่งคืนค่า **errno** เฉพาะเธรด ระบบอื่นๆ บางระบบ อาจจัดเตรียมการนำไปปฏิบัติของ **errno**

เธรดที่อยู่ภายในกระบวนการไม่ถูกพิจารณาเป็น กลุ่มของการประมวลผล เธรดทั้งหมดจะแบ่งใช้พื้นที่แอดเดรสเดียวกัน นั่นหมายความว่า ตัวชี้สองตัวมีค่าที่เหมือนกันในเธรดสองตัวที่อ้างถึงข้อมูลเดียวกัน และ หากเธรดใดๆ เปลี่ยนหนึ่งในรีซอร์สของระบบที่แบ่งใช้ เธรดทั้งหมดภายในกระบวนการจะได้รับผลกระทบ ตัวอย่างเช่น หากเธรดปิดไฟล์ ไฟล์นั้นจะถูกปิดสำหรับเธรดทั้งหมด

Initial thread

เมื่อการประมวลผลถูกสร้างขึ้น เธรดหนึ่งตัวจะถูกสร้างโดยอัตโนมัติ เธรดนี้จะเรียกว่า *initial thread*

ซึ่งช่วยให้แน่ใจว่ามีความเข้ากันได้ระหว่างกระบวนการที่มี เธรดโดยนัยแบบเฉพาะ และกระบวนการแบบมัลติเธรดใหม่ เธรดเริ่มต้นมีคุณสมบัติพิเศษบางประการ ที่โปรแกรมเมอร์ไม่เห็น ที่ช่วยให้แน่ใจว่ามีความเข้ากันได้แบบไบนารีระหว่างโปรแกรมที่มีเธรดเดียว แบบเก่า และระบบปฏิบัติการแบบมัลติเธรด รวมทั้งเป็น เธรดเริ่มต้นที่เรียกใช้งานรูทีน **main** ในโปรแกรมแบบมัลติเธรด

Modularity

บ่อยครั้งที่โปรแกรมจะจำลองจำนวนของชิ้นส่วนที่แตกต่างกัน ซึ่งโต้ตอบกับชิ้นส่วนอื่นเพื่อสร้างผลลัพธ์หรือเซอร์วิสที่ต้องการ

โปรแกรมสามารถนำไปปฏิบัติเป็น entity เดี่ยว entity ที่ซับซ้อนที่ดำเนินการกับฟังก์ชันจำนวนมาก ระหว่างชิ้นส่วนของโปรแกรมที่ต่างกัน โซลูชันแบบง่ายๆ ประกอบด้วยการนำ entity ทั้งหลายไปใช้งาน แต่ละ entity จะดำเนินการกับชิ้นส่วนของโปรแกรมและรีซอร์สที่แบ่งใช้ด้วย entity อื่น

โดยการใช้ entity จำนวนมาก โปรแกรมสามารถแยกออกได้ ตามกิจกรรมที่แตกต่างกัน ซึ่งแต่ละกิจกรรมจะมี entity ที่เชื่อมโยง entity เหล่านั้นไม่ต้องรู้ข้อมูลเกี่ยวกับชิ้นส่วนของโปรแกรมอื่น ยกเว้นเมื่อมีการแลกเปลี่ยนข้อมูล ในกรณีเหล่านี้ entity ต้องซิงโครไนซ์ กับ entity อื่นๆ เพื่อมั่นใจว่ามี data integrity

เธรดคือ entity ที่เหมาะสมสำหรับการโปรแกรมมิงแยกโมดูล เธรดจะจัดเตรียมการแบ่งใช้ข้อมูลแบบง่ายๆ (เธรดทั้งหมดที่อยู่ภายในการประมวลผล ที่แบ่งใช้พื้นที่แอดเดรสเดียวกัน) และตัวช่วยการประสานเวลาที่ไม่ประสิทธิภาพ เช่น mutexes (การล็อกที่ไม่เกิดร่วมกัน) และตัวแปรเงื่อนไข

โมเดลซอฟต์แวร์

ส่วนนี้อธิบายโมเดลซอฟต์แวร์แบบต่างๆ

แบบจำลองเหล่านี้ทั้งหมดจะนำไปสู่โปแกรมมอดูลาร์ แบบจำลอง ยังรวมเข้าด้วยกันเพื่อแก้ปัญหาภารกิจที่ซับซ้อนได้อย่างมีประสิทธิภาพ

โมเดลเหล่านี้สามารถใช้กับโซลูชันหลายกระบวนการแบบเดิม หรือกับโซลูชันมัลติเธรดกระบวนการเดียว บนระบบมัลติเธรด ในคำอธิบายต่อไปนี้ คำว่า *เอนทิตี* อ้างถึง *กระบวนการ* เธรดเดียว หรือ *เธรด* เดียวใน กระบวนการแบบมัลติเธรด

แบบจำลองซอฟต์แวร์ต่อไปนี้ สามารถนำไปปฏิบัติพร้อมกับเธรดได้โดยง่าย:

แบบจำลอง Master/Slave:

ในแบบจำลอง master/slave (บางครั้งเรียกว่า boss/worker) master entity จะได้รับคำร้องขอตั้งแต่หนึ่งคำร้องขอขึ้นไป จากนั้นสร้าง slave entities เพื่อเรียกใช้งานคำร้องเหล่านั้น โดยปกติแล้ว master จะควบคุมจำนวนของ slaves และสิ่งที่ slave แต่ละตัวดำเนินการ slave จะรันอย่างเป็นอิสระจาก slave อื่น

ตัวอย่างของแบบจำลองนี้คือ สพูลเลอร์งานพิมพ์ ที่ควบคุมชุดของเครื่องพิมพ์ บทบาทของสพูลเลอร์คือ ทำให้มั่นใจว่า คำร้องขอให้พิมพ์ที่ได้รับ ถูกจัดการในตามเวลาที่กำหนด เมื่อสพูลเลอร์ได้รับคำร้องขอ master entity จะเลือกเครื่องพิมพ์และเป็นสาเหตุทำให้ slave พิมพ์งานบนเครื่องพิมพ์ แต่ละ slave จะพิมพ์งานหนึ่งงานในหนึ่งครั้งบนเครื่องพิมพ์ ขณะที่ยังจัดการการควบคุมสายงาน และรายละเอียดการพิมพ์อื่นๆ สพูลเลอร์อาจสนับสนุนการยกเลิกงาน หรือคุณลักษณะอื่นๆ ที่ต้องการให้ master ยกเลิก slave entity หรือกำหนดงานใหม่

แบบจำลอง Divide-and-Conquer:

ใน divide-and-conquer (บางครั้งเรียกว่า *การคำนวณแบบพร้อมเพียงกัน* หรือแบบจำลอง *work crew*) ซึ่งมี entity ตั้งแต่หนึ่งตัวขึ้นไปดำเนินภารกิจเดียวกันแบบขนาน และไม่มี master entity ซึ่ง entity ทั้งหมดจะรันแบบขนานอย่างเป็นอิสระ

ตัวอย่างของแบบจำลอง divide-and-conquer คือการนำคำสั่ง **grep** ไปใช้งานแบบขนาน ซึ่งสามารถทำได้ดังนี้ เริ่มแรก คำสั่ง **grep** จะสร้างพูลของไฟล์ ที่ต้องการสแกน จากนั้น จะสร้างจำนวนของ entity แต่ละ entity จะใช้ไฟล์ที่แตกต่างกันจากพูลและค้นหารูปแบบ ส่งผลลัพธ์ไปยังอุปกรณ์เอาต์พุตทั่วไป เมื่อ entity เสร็จสิ้นการค้นหาไฟล์แล้ว entity ที่รับไฟล์จากพูล และหยุดการทำงานหากพูลนั้นว่าง

แบบจำลอง Producer/Consumer:

แบบจำลอง producer/consumer (บางครั้งเรียกว่า *การสร้างไพพ์ไลน์*) จะถูกทำเป็นตัวอย่างโดยสายการผลิต ไอเท็มจะดำเนินการจากคอมโพเนนต์ดิบ จนกระทั่งถึงไอเท็มสุดท้ายในชุดของลำดับขั้น

โดยปกติแล้ว worker เดี่ยวที่แต่ละขั้นตอนจะแก้ไขไอเท็มและส่งไปยังขั้นตอนถัดไป ในความหมายของซอฟต์แวร์ คำสั่ง pipe ของ AIX เช่น คำสั่ง **cpio** คือตัวอย่างของแบบจำลองนี้

ตัวอย่างเช่น ตัวอ่าน entity จะอ่านข้อมูลดิบจากอินพุตมาตรฐาน และส่งไปยังตัวประมวลผล entity ซึ่งจะประมวลผลข้อมูลและส่งผ่านไปยังตัวเขียน entity ซึ่งจะเขียนเอาต์พุตมาตรฐาน โปรแกรมมิงแบบขนาน อนุญาตให้กิจกรรมดำเนินการแบบพร้อมเพียงกัน ตัวเขียน entity อาจเอาต์พุตข้อมูลที่ประมวลผลแล้วบางตัว ขณะที่ตัวอ่าน entity ได้รับข้อมูลดิบเพิ่มเติม

เธรดของเคอร์เนลและเธรดของผู้ใช้

เธรดของเคอร์เนลคือ ความสามารถในการกำหนดตารางเวลาของ entity ซึ่งหมายความว่า ตัวกำหนดตารางเวลาของระบบจะจัดการเธรดของเคอร์เนล

เธรดเหล่านี้ซึ่งรู้จักโดย ตัวกำหนดตารางเวลาของระบบต้องการพึ่งพาการนำไปปฏิบัติ หากต้องการอำนวยความสะดวกในการเขียนโปรแกรมที่สามารถเคลื่อนย้ายได้ ไลบรารีจะจัดเตรียมเธรดของ *ผู้ใช้* ไว้

เธรดของเคอร์เนล คือเคอร์เนล entity ที่คล้ายกับ handler การประมวลผลและอินเตอร์รัปต์ ซึ่งเป็น entity ที่ถูกจัดการโดยตัวกำหนดตารางเวลาของระบบ เธรดของเคอร์เนลจะรันอยู่ภายในกระบวนการ แต่ไม่สามารถอ้างอิง ได้โดยเธรดอื่นในระบบ โปรแกรมเมอร์ไม่มีการควบคุมโดยตรงผ่านเธรดเหล่านี้ ยกเล้นคุณกำลังเขียนส่วนขยายเคอร์เนลหรือไดร์ฟเวอร์อุปกรณ์ สำหรับข้อมูลเพิ่มเตมเกี่ยวกับการโปรแกรมมิงเคอร์เนล โปรดดู *Kernel Extensions and Device Support Programming Concepts*

เธรดของผู้ใช้ คือ entity ที่ถูกใช้โดยโปรแกรมเมอร์เพื่อจัดการกับการควบคุมสายงานจำนวนมากภายในโปรแกรม API สำหรับการจัดการกับเธรดของผู้ใช้จะถูกจัดเตรียมไว้โดย *ไลบรารีเธรด* เธรดของผู้ใช้จะมีอยู่ภายในการประมวลผล เธรดของผู้ใช้ในการประมวลผล *A* ไม่สามารถอ้างอิงเธรดของผู้ใช้ในการประมวลผล *B* ได้ ไลบรารีใช้อินเตอร์เฟสที่เป็นเจ้าของเพื่อจัดการเ

ธรดของเคอร์เนล สำหรับการเรียกใช้งานเธรดของผู้ใช้ API เธรดของผู้ใช้ไม่เหมือนกับอินเตอร์เฟสของเธรดของเคอร์เนล ซึ่งเป็นส่วนหนึ่งของแบบจำลองโปรแกรมมิงที่สามารถเคลื่อนย้ายได้ตาม POSIX มาตรฐาน ดังนั้น โปรแกรม แบบมัลติเธรดที่พัฒนาบนระบบ AIX สามารถพอร์ตไปยังระบบอื่นได้โดยง่าย

สำหรับระบบอื่น เธรดของผู้ใช้จะเรียกว่า *เธรด* และ *lightweight process* จะอ้างถึงเธรดของเคอร์เนล

โมเดลเธรดและตัวประมวลผลเสมือน

เธรดของผู้ใช้จะถูกแม็พกับเธรดของเคอร์เนลโดยไลบรารีเธรด วิธีการแม็พที่ถูกทำนี้จะเรียกว่า *แบบจำลองเธรด*

ซึ่งมีแบบจำลองเธรดที่เป็นไปได้อยู่สามแบบโดยสอดคล้องกับวิธีการแม็พเธรดของผู้ใช้ กับเธรดของเคอร์เนลที่ต่างกันสามวิธี

- แบบจำลอง M:1
- แบบจำลอง 1:1
- แบบจำลอง M:N

การแม็พเธรดของผู้ใช้กับเธรดของเคอร์เนล จะถูกทำโดยใช้ *ตัวประมวลผลเสมือน* ตัวประมวลผลเสมือน (VP) คือไลบรารี entity ที่เป็น implicit เสมอ สำหรับเธรดของผู้ใช้ VP จะทำงานในลักษณะที่เหมือนกับ CPU ในไลบรารี VP คือเธรดของเคอร์เนลหรือโครงสร้างที่มีข้อจำกัด กับเธรดของเคอร์เนล

ในแบบจำลอง M:1 เธรดของผู้ใช้ทั้งหมดจะถูกแม็พกับเธรดของเคอร์เนล เธรดของผู้ใช้จะรันอยู่บนหนึ่ง VP การแม็พจะถูกจัดการโดย ตัวกำหนดตารางเวลาไลบรารี ตัวช่วยโปรแกรมมิงเธรดของผู้ใช้ทั้งหมดจะถูกจัดการโดยไลบรารี อย่างสมบูรณ์ แบบจำลองนี้สามารถใช้บนระบบอื่นๆ ได้โดยเฉพาะอย่างยิ่ง บนระบบเธรดเดี่ยวแบบเก่า

ในแบบจำลอง 1:1 เธรดของผู้ใช้แต่ละตัวจะถูกแม็พกับ เธรดของเคอร์เนล ซึ่งเธรดของผู้ใช้แต่ละตัวจะรันอยู่บน VP ตัวช่วยโปรแกรมมิงเธรดของผู้ใช้ส่วนใหญ่ จะถูกจัดการโดยเธรดของเคอร์เนล โมเดลนี้เป็นโมเดลดีฟอลต์

ในแบบจำลอง M:N เธรดของผู้ใช้ทั้งหมดจะถูกแม็พกับพูลของเธรดของเคอร์เนล เธรดของผู้ใช้ทั้งหมดจะรันอยู่บนพูลของหน่วยความจำแบบเสมือน เธรดของผู้ใช้อาจมีข้อจำกัดกับ VP ที่ระบุเฉพาะในแบบจำลอง 1:1 เธรดของผู้ใช้ทั้งหมด จะแบ่งใช้ VP ส่วนที่เหลือ ซึ่งจะมีประสิทธิภาพ และเป็นแบบจำลองเธรดที่มีความซับซ้อน ตัวช่วยโปรแกรมมิงเธรดของผู้ใช้จะแบ่งใช้ระหว่าง ไลบรารีเธรดและเธรดของเคอร์เนล โมเดลนี้สามารถตั้งค่าโดยการตั้งค่าตัวแปรสภาวะแวดล้อม `AIXTHREAD_SCOPE` เป็น **P**

API ไลบรารีเธรด

ส่วนนี้จัดเตรียมข้อมูลทั่วไปเกี่ยวกับ API ไลบรารี เธรด

แม้ว่าข้อมูลต่อไปนี้จะไม่จำเป็นสำหรับการเขียน โปรแกรมแบบมัลติเธรด แต่ก็ช่วยโปรแกรมเมอร์ได้เข้าใจ ถึง API เธรดไลบรารี

อินเตอร์เฟสเชิงวัตถุ

API ไลบรารีเธรดจะจัดเตรียม อินเตอร์เฟสเชิงวัตถุไว้ โปรแกรมเมอร์สามารถจัดการกับอ็อบเจ็กต์ opaque โดยใช้ตัวชี้ หรือ identifier สากล

ซึ่งช่วยให้แน่ใจในความ portability ของโปรแกรมแบบมัลติเธรดระหว่าง ระบบที่ใช้เธรดไลบรารี และยังอนุญาตการเปลี่ยนแปลง การประยุกต์ใช้ระหว่าง AIX สองรีลีส ทำให้จำเป็นต้อง คอมไพล์โปรแกรมนั้นใหม่เท่านั้น แม้ว่า นิยามบางส่งของชนิด

ข้อมูลอาจพบได้ในไฟล์ส่วนหัวของไลบรารีเธรด (**pthread.h**) โปรแกรมไม่ควรไว้วางใช้กับนิยามที่ต้องพึ่งพาการนำไปปฏิบัติ เพื่อจัดการกับเนื้องหาของโครงสร้างโดยตรง รูทีนย่อยไลบรารีเธรดปกติต้องถูกนำมาใช้ เพื่อจัดการกับอ็อบเจ็กต์

ไลบรารีเธรดจะใช้ชนิดของอ็อบเจ็กต์ต่อไปนี้ (ชนิดข้อมูล opaque): เธรด mutex, rwlocks และตัวแปรเงื่อนไข อ็อบเจ็กต์มีแอ็ตทริบิวต์ที่ระบุคุณสมบัติอ็อบเจ็กต์ เมื่อสร้างอ็อบเจ็กต์แล้ว แอ็ตทริบิวต์ต้องถูกระบุด้วย ในไลบรารีเธรด แอ็ตทรบิวต์การสร้างเหล่านี้คืออ็อบเจ็กต์ของตัวแอ็ตทริบิวต์เอง ซึ่งเรียกว่า*อ็อบเจ็กต์แอ็ตทริบิวต์เธรด*

คู่ต่อไปนี้ของอ็อบเจ็กต์จะถูกจัดการโดยไลบรารีเธรด:

- เธรดและอ็อบเจ็กต์แอ็ตทริบิวต์เธรด
- Mutexes และอ็อบเจ็กต์แอ็ตทริบิวต์ mutex
- ตัวแปรเงื่อนไขและอ็อบเจ็กต์แอ็ตทริบิวต์ตัวแปรเงื่อนไง
- การล็อกการเขียน-อ่าน

อ็อบเจ็กต์แอ็ตทริบิวต์จะถูกสร้าง ด้วยแอ็ตทริบิวต์ที่มีค่าดีฟอลต์ แอ็ตทริบิวต์แต่ละตัวสามารถแก้ไขได้โดยใช้รูทีนย่อย ทั้งนี้เพื่อให้แน่ใจว่าโปรแกรมแบบมัลติเธรดจะไม่ได้รับการจาก การมีแอ็ตทริบิวต์ใหม่ หรือโดยการเปลี่ยนแปลงในการประยุกต์ใช้ แอ็ตทริบิวต์ อ็อบเจ็กต์แอ็ตทริบิวต์ สามารถใช้เพื่อสร้างอ็อบเจ็กต์ต่างๆ ได้ จากนั้นทำลายโดยไม่กระทบกับอ็อบเจ็กต์ที่สร้างด้วยอ็อบเจ็กต์แอ็ตทริบิวต์

การใช้อ็อบเจ็กต์แอ็ตทริบิวต์จะอนุญาต ให้ใช้คลาสอ็อบเจ็กต์ หนึ่งอ็อบเจ็กต์แอ็ตทริบิวต์อาจกำหนดไว้สำหรับคลาสอ็อบเจ็กต์แต่ละคลาส การสร้างตัวอย่างของคลาสอ็อบเจ็กต์ จะถูกทำโดยการสร้างอ็อบเจ็กต์โดยใช้ คลาสอ็อบเจ็กต์แอ็ตทริบิวต์

หลักการตั้งชื่อสำหรับไลบรารีเธรด

identifier จะถูกใช้โดยไลบรารีเธรดโดยปฏิบัติตามหลักการตั้งชื่อ identifier ทั้งหมดของไลบรารีเธรดจะขึ้นต้นด้วย **pthread_**

โปรแกรมผู้ใช้ไม่ควรใช้คำนำหน้านี้สำหรับ identifier ส่วนบุคคล คำนำหน้านี้ จะตามด้วยชื่อคอมโพเนนต์ คอมโพเนนต์ต่อไปนี้จะถูกกำหนดในไลบรารีเธรด

คอมโพเนนต์	**คำอธิบาย**
pthread_	ตัวเธรดและรูทีนย่อยต่างๆ
pthread_attr	อ็อบเจ็กต์แอ็ตทริบิวต์เธรด
pthread_cond	ตัวแปรเงื่อนไข
pthread_condattr	อ็อบเจ็กต์แอ็ตทริบิวต์เงื่อนไข
pthread_key	คีย์ข้อมูลที่ระบุเฉพาะเธรด
pthread_mutex	Mutexes
pthread_mutexattr	อ็อบเจ็กต์แอ็ตทริบิวต์ Mutex

identifier ชนิดข้อมูลสิ้นสุดลงด้วย **_t** รูทีนย่อยของชื่อแมโคร จะสิ้นสุดด้วย _ (ขีดเส้นใต้) แล้วตามด้วย ชื่อที่บ่งชี้การดำเนินการที่ถูกดำเนินการโดยรูทีนย่อยหรือแมโคร ตัวอย่างเช่น **pthread_attr_init** คือ identifer ไลบรารีเธรด (**pthread_**) ที่เกี่ยวข้องกับแอ็ตทริบิวต์สัญญาณเธรด (**attr**) และเป็นรูทีนย่อยการกำหนดค่าเริ่มต้น (**_init**)

identifier แมโครจะอยู่ในรูปของตัวอักษรพิมพ์ใหญ่ รูทีนบางตัวอาจไม่ถูกนำมาใช้เป็นแมโคร แม้ว่าชื่อเหล่านั้นจะอยู่ในรูปตัวพิมพ์เล็ก

ไฟล์การนำ pthread ไปใช้

ส่วนนี้อธิบายไฟล์การนำ **pthread** ไปใช้งาน

ไฟล์ AIX ต่อไปนี้จัดเตรียมข้อมูลการนำไปปฏิบัติของ pthreads:

การประยุกต์ใช้	คำอธิบาย
/usr/include/pthread.h	ส่วนหัว C/C++ ที่มีนิยาม pthread เป็นส่วนใหญ่
/usr/include/sched.h	ส่วนหัว C/C++ ที่มีนิยามการกำหนดตารางเวลาบางอย่าง
/usr/include/unistd.h	ส่วนหัว C/C++ ด้วยนิยาม **pthread_atfork( )**
/usr/include/sys/limits.h	ส่วนหัว C/C++ ที่มีนิยามในบาง pthread
/usr/include/sys/pthdebug.h	ส่วนหัว C/C++ ที่มีนิยามการดีบัก pthread
/usr/include/sys/sched.h	ส่วนหัว C/C++ ที่มีนิยามการกำหนดตารางเวลาบางอย่าง
/usr/include/sys/signal.h	ส่วนหัวของ C/C++ พร้อมกับ **pthread_kill( )** และ **pthread_sigmask( )** นิยาม
/usr/include/sys/types.h	ส่วนหัว C/C++ ที่มีนิยามในบาง pthread
/usr/lib/libpthreads.a	ไลบรารีที่มี 32 บิต/64 บิตที่จัดเตรียม UNIX98 and POSIX 1003.1c pthreads
/usr/lib/libpthreads_compat.a	ไลบรารีที่มี 32 บิตเท่านั้นที่จัดเตรียม POSIX 1003.1c Draft 7 pthreads
/usr/lib/profiled/libpthreads.a	ไลบรารีที่มี 32 บิต/64 บิตที่จัดเตรียม UNIX98 and POSIX 1003.1c pthreads
/usr/lib/profiled/libpthreads_compat.a	ไลบรารีที่มีโปรไฟล์ 32 บิตเท่านั้นที่จัดเตรียม POSIX 1003.1c Draft 7 pthreads

ไลบรารี Threadsafe และที่ถูกเธรดใน AIX

ส่วนนี้อธิบายเธรดไลบรารีใน AIX

ตามค่าดีฟอลต์แล้ว แอ็พพลิเคชันทั้งหมดจะถูกพิจารณาว่า "มีผลกับการทำงานของเธรด" แม้ว่า แอ็พพลิเคชันส่วนใหญ่จะ "มีผลกับการทำงานของเธรดเดี่ยว" ไลบรารี threadsafe เหล่านี้ เป็นดังนี้:

ไลบรารี Threadsafe		
libbsd.a	libc.a	libm.a
libsvid.a	libtli.a	libxti.a
libnetsvc.a		

ไลบรารีเธรด POSIX

ไลบรารีเธรด POSIX ต่อไปนี้พร้อมใช้งาน:

ไลบรารีเธรด libpthreads.a POSIX

ไลบรารี **libpthreads.a** จะอ้างอิงตามมาตรฐานอุตสาหกรรม POSIX 1003.1c สำหรับ API สำหรับเธรดผู้ใช้แบบ portable โปรแกรมใดๆ ที่เขียนมาเพื่อใช้กับไลบรารีเธรด POSIX สามารถนำมาใช้กับไลบรารีเธรด POSIX ตัวอื่นได้ เฉพาะผลการทำงานและรูทีนย่อยเพียงเล็กน้อยของไลบรารีเธรดเท่านั้น ที่เป็นการนำไปปฏิบัติแบบพึ่งพา หากต้องการเพิ่มประสิทธิภาพในเคลื่อนย้ายของไลบรารีเธรด POSIX มาตรฐานต้องนำตัวช่วยโปรแกรมมิงต่างๆ ไปใช้งาน สำหรับข้อมูลเพิ่มเติมเกี่ยวกับ การตรวจสอบอ็อพชัน POSIX โปรดดูที่ อ็อพชัน ไลบรารีเธรด

ไลบรารีเธรด libpthreads_compat.a POSIX draft 7

AIX จัดเตรียมความเข้ากันได้แบบไบนารีสำหรับแอ็พพลิเคชันที่มีหลายเธรด ซึ่งจะถูกโค้ดเป็นแบบ Draft 7 ของเธรด POSIX มาตรฐาน แอ็พพลิเคชันเหล่านี้ จะทำงานโดยไม่ต้องทำการลิงก์ใหม่ไลบรารี **libpthreads_compat.a** ถูกจัดเตรียมไว้เฉพาะสำหรับความเข้ากันได้กับแอ็พพลิเคชันในเวอร์ชันก่อนหน้านี้ซึ่งเขียนไว้โดยใช้ Draft 7 ของ POSIX Thread Standard แอ็พพลิเคชันใหม่ทั้งหมด ต้องใช้ไลบรารี **libpthreads.a** ซึ่งสนับสนุน แอ็พพลิเคชัน 32

บิตและ 64 บิต ไลบรารี **libpthreads_compat.a** สนับสนุนเฉพาะแอ็พพลิเคชัน 32 บิตเท่านั้น การขึ้นต้นด้วย AIX 5.1 ไลบรารี **libpthreads.a** จะสนับสนุน Single UNIX Specification, Version 2 ซึ่งประกอบด้วย POSIX 1003.1c Pthread Standard ล่าสุด

หลักการที่เกี่ยวข้อง:

"ประโยชน์ของเธรด" ในหน้า 559
โปรแกรมแบบมัลติเธรดสามารถช่วยเพิ่มประสิทธิภาพการทำงานเมื่อเทียบ กับโปรแกรแบบขนาดแบบเดิมที่ใช้หลายกระบวนการ นอกจากนี้ ประสิทธิภาพยิ่งเพิ่มขึ้นเมื่อใช้ระบบมัลติโพรเซสเซอร์ที่ใช้เธรด

"อ็อพชันไลบรารีเธรด" ในหน้า 526
ส่วนนี้อธิบายถึงแอ็ตทริบิวต์ของเธรดชนิดพิเศษ mutex และตัวแปรเงื่อนไข

"การพัฒนาโปรแกรมแบบมัลติเธรด" ในหน้า 544
การพัฒนาโปรแกรมแบบมัลติเธรดคล้ายกับการพัฒนา โปรแกรมที่มีหลายกระบวนการ การพัฒนาโปรแกรมยังประกอบด้วยการคอมไพล์ และการดีบักโค้ด

การสร้างเธรด

การสร้างเธรดต่างจากการสร้างกระบวนการคือ ไม่มีความสัมพันธ์ parent-child ระหว่างเธรด

เธรดทั้งหมด ยกเว้น *initial thread* ถูกสร้างโดยอัตโนมัติเมื่อกระบวนการถูกสร้าง อยู่ในระดับลำดับชั้นเดียวกัน เธรดไม่คงรายการของ เธรดที่สร้าง ไม่รู้จักเธรดที่สร้างเธรดขึ้นมา

เมื่อสร้างเธรด รูทีน entry-point และอาร์กิวเมนต์ต้องถูก ระบุ ทุกเธรดมีรูทีน entry-point ที่มีหนึ่งอาร์กิวเมนต์ รูทีน entry-point เดียวกันอาจถูกใช้โดยหลายเธรด

เธรด มีแอ็ตทริบิวต์ ซึ่งระบุคุณลักษณะของ เธรด เพื่อควบความแอ็ตทริบิวต์เธรด อ็อบเจ็กต์แอ็ตทริบิวต์เธรดต้อง ถูกกำหนดก่อนการสร้างเธรด

อ็อบเจ็กต์แอ็ตทริบิวต์เธรด

แอ็ตทริบิวต์เธรด ถูกเก็บในอ็อบเจ็กต์ opaque *อ็อบเจ็กต์แอ็ตทริบิวต์ เธรด* ที่ใช้เมื่อสร้างเธรด มีหลายแอ็ตทริบิวต์ ขึ้นกับการการนำไปปฏิบัติ ของตัวเลือก POSIX อ็อบเจ็กต์ถูกเข้าถึงผ่านตัวแปรชนิด **pthread_attr_t** ใน AIX, ชนิดข้อมูล **pthread_attr_t** เป็นตัวชี้; บนระบบอื่นอาจเป็น structure หรือชนิดข้อมูลอื่น

การสร้สและการทำลายอ็อบเจ็กต์แอ็ตทริบิวต์เธรด

o

อ็อบเจ็กต์แอ็ตทริบิวต์เธรด ถูกกำหนดเป็นค่าเริ่มต้นโดยรูทีนย่อย **pthread_attr_init** แอ็ตทริบิวต์ถูกจัดการโดยรูทีนย่อย อ็อบเจ็กต์แอ็ตทริบิวต์เธรดถูกทำลายโดยรูทีนย่อย **pthread_attr_destroy** รูทีนย่อยนี้สามารถรีลีส พื้นที่จัดเก็บข้อมูลแบบไดนามิกที่จัดสรรโดยรูทีนย่อย **pthread_attr_init** ขึ้นกับการนำไปปฏิบัติของไลบรารีเธรด

ในตัวอย่างดังต่อไปนี้ อ็อบเจ็กต์แอ็ตทริบิวต์เธรดถูกสร้างและกำหนด ด้วยค่าเริ่มต้น แล้วใช้และถูกทำลายในตอนสุดท้าย:

```
pthread_attr_t attributes;
               /* the attributes object is created */
...
if (!pthread_attr_init(&attributes)) {
```

```
                /* the attributes object is initialized */
        ...
                /* using the attributes object */
        ...
        pthread_attr_destroy(&attributes);
                /* the attributes object is destroyed */
}
```

อ็อบเจ็กต์แอ็ตทริบิวต์เดียวกันสามารถถูกใช้เพื่อสร้างหลาย เธรด ซึ่งสามารถถูกแก้ไขระหว่างสองการสร้างเธรด เมื่อเธรดถูกสร้าง อ็อบเจ็กต์แอ็ตทริบิวต์สามารถถูกทำลายได้โดยไม่มีผลกับเธรดที่ สร้าง

แอ็ตทริบิวต์ Detachstate

แอ็ตทริบิวต์ดังต่อไปนี้ถูกกำหนดเสมอ:

Detachstate
: ระบุภาวะที่แยกออกของเธรด

ค่าของแอ็ตทริบิวต์ถูกส่งคืนโดยรูทีนย่อย **pthread_attr_getdetachstate** ซึ่งสามารถถูกตั้งค่า โดยรูทีนย่อย **pthread_attr_setdetachstate** ค่าที่เป็นไปได้สำหรับแอ็ตทริบิวต์นี้ คือค่าคงที่สัญลักษณ์ดังต่อไปนี้:

PTHREAD_CREATE_DETACHED
: ระบุว่าเธรดจะถูกสร้างในภาวะแยกออก

PTHREAD_CREATE_JOINABLE
: ระบุว่าเธรดจะถูกสร้างในภาวะรวมกันได้

ค่าดีฟอลต์คือ **PTHREAD_CREATE_JOINABLE**

ถ้าคุณสร้างเธรดในภาวะรวมกันได้ คุณต้องเรียกรูทีนย่อย **pthread_join** พร้อมกับเธรด มิฉะนั้น คุณอาจมีพื้นที่เก็บข้อมูล ไม่เพียงพอเมื่อสร้างเธรดใหม่ เนื่องจากเธรดแต่ละตัวใช้หน่วยความจำ จำนวนมาก สำหรับข้อมูลเพิ่มเติมเกี่ยวกับ รูทีนย่อย pthread_join โปรดดูที่ รูทีนย่อย การเรียกใช้ pthread_join

แอ็ตทริบิวต์เธรดอื่น

AIX ยังกำหนดแอ็ตทริบิวต์ดังต่อไปนี้ ซึ่งมีจุดประสงค์สำหรับโปรแกรมขั้นสูงและอาจต้องการสิทธิพิเศษในการกระทำการเพื่อให้มีผล โปรแกรมส่วนใหญ่ปฏิบัติการอย่างถูกต้องด้วย ค่ากำหนดดีฟอลต์ การใช้แอ็ตทริบิวต์ต่อไปนี้ อธิบายไว้ใน แอ็ตทริบิวต์การใช้ inheritsched

ขอบเขต Contention
: ระบุสโคป contention ของเธรด

Inheritsched
: ระบุการสืบทอดของคุณสมบัติการจัดตารางเวลาของเธรด

Schedparam
: ระบุพารามิเตอร์การจัดตารางเวลาของเธรด

Schedpolicy
: ระบุนโยบายการจัดตารางเวลาของเธรด

การใช้สแต็กแอ็ตทริบิวต์ต่อไปนี้อธิบายไว้ใน สแต็กแอ็ตทริบิวต์.

`Stacksize`
ระบุขนาดของสแตกของเธรด

`Stackaddr`
ระบุแอดเดรสของสแต็กของเธรด

`Guardsize`
ระบุขนาดของพื้นที่ guard ของสแต็กของเธรด

การสร้างเธรดโดยใช้รูทีนย่อย pthread_create

เธรดถูกสร้างโดยการเรียกรูทีนย่อย **pthread_create** รูทีนย่อยนี้สร้าง เธรดใหม่และทำให้สามารถรันได้

การใช้อ็อบเจ็กต์แอ็ตทริบิวต์เธรด

เมื่อเรียกรูทีนย่อย **pthread_create** คุณอาจระบุอ็อบเจ็กต์แอ็ตทริบิวต์เธรด ถ้าคุณระบุตัวชี้ **NULL** เธรดที่สร้างจะมีแอ็ตทริบิวต์ค่าเริ่มต้น ดังนั้นส่วนของโค้ดต่อไปนี้:

```
pthread_t thread;
pthread_attr_t attr;
...
pthread_attr_init(&attr);
pthread_create(&thread, &attr, init_routine, NULL);
pthread_attr_destroy(&attr);
```

เหมือนกับ:

```
pthread_t thread;
...
pthread_create(&thread, NULL, init_routine, NULL);
```

รูทีน Entry point

เมื่อ เรียกรูทีนย่อย **pthread_create** คุณต้องระบุรูทีน entry-point รูทีนนี้ จัดเตรียมโดยโปรแกรมของคุณ เหมือนกับ รูทีน **main** สำหรับกระบวนการ นี่เป็นรูทีนของผู้ใช้ แรกที่ถูกดำเนินการโดยเธรดใหม่ เมื่อเธรดส่งกลับจากรูทีนนี้ เธรดจะถูกยุติการทำงานโดยอัตโนมัติ

รูทีน entry-point มีหนึ่งพารามิเตอร์ void pointer ที่ระบุเมื่อเรียกรูทีนย่อย **pthread_create** คุณอาจระบุตัวชี้ไปที่ข้อมูลบางชนิด เช่นสตริงหรือ structure เธรดการสร้าง (เธรดที่เรียกรูทีนย่อย **pthread_create**) และเธรดที่สร้างต้อง ยอมรับชนิดจริงของตัวชี้นี้

รูทีน entry-point ส่งกลับ void pointer หลังจากการยุติการทำงานของเธรด ตัวชี้นี้ถูกเก็บโดยไลบรารีเธรด นอกจากเธรดถูกแยกออก สำหรับข้อมูลเพิ่มเติมเกี่ยวกับการใช้ตัวชี้นี้ โปรดดูที่ การส่งคืนข้อมูลจากเธรด

ข้อมูลที่ส่งกลับ

รูทีนย่อย **pthread_create** ส่งกลับ thread ID ของเธรดใหม่ ผู้เรียกสามารถใช้ thread ID นี้เพื่อการดำเนินการหลายแบบในเธรด

ขึ้นกับพารามิเตอร์การจัดตารางเวลาของทั้งสองเธรด เธรดใหม่อาจเริ่มรันก่อนการเรียกรูทีนย่อย **pthread_create** ส่งกลับ ซึ่งอาจเกิดขึ้น เมื่อรูทีนย่อย **pthread_create** ส่งกลับ เธรดใหม่ได้หยุดการทำงาน ไปแล้ว thread ID ที่ส่งกลับโดยรูทีนย่อย **pthread_create** ผ่านพารามิเตอร์ *thread* จึงไม่ถูกต้อง ไปแล้ว ดังนั้น เป็นสิ่งสำคัญที่จะตรวจสอบโค้ดระบุความผิดพลาด **ESRCH** ที่ส่งกลับโดยรูทีนย่อยไลบรารีเธรด โดยใช้ thread ID เป็นพารามิเตอร์

ถ้ารูทีนย่อย **pthread_create** ไม่สำเร็จ จะไม่มีเธรดใหม่ถูกสร้าง thread ID ในพารามิเตอร์ *thread* ไม่ถูกต้อง และโค้ดระบุความผิดพลาดที่เหมาะสมถูกส่งกลับ สำหรับข้อมูลเพิ่มเติม โปรดดูที่ ตัวอย่างของโปรแกรมแบบ Multi-Threaded

การจัดการ Thread IDs

thread ID ของเธรดที่สร้างใหม่ถูกส่งกลับไปที่เธรดการสร้าง ผ่านพารามิเตอร์ *thread* thread ID ปัจจุบันถูกส่งกลับโดยรูทีนย่อย **pthread_self**

thread ID เป็นอ็อบเจ็กต์ opaque ที่เป็นชนิด **pthread_t** ใน AIX ชนิดข้อมูล **pthread_t** เป็นจำนวนเต็ม บนระบบอื่น อาจเป็น structure ตัวชี้ หรือ ชนิดข้อมูลอื่น

เพื่อเพิ่มความสามารถในการทำงานข้ามระบบของโปรแกรม โดยใช้ไลบรารีเธรด thread ID ควรถูกจัดการเป็นอ็อบเจ็กต์ opaque เสมอ สำหรับเหตุผลนี้ thread ID ควรถูกเปรียบเทียบ โดยใช้รูทีนย่อย **pthread_equal** อย่าใช้ตัวดำเนินการเท่ากับของ C (==) เนื่องจากชนิดข้อมูล **pthread_t** อาจไม่ใช่ทั้งชนิด คณิตศาสตร์หรือตัวชี้

หลักการที่เกี่ยวข้อง:

"การพัฒนาโปรแกรมแบบมัลติเธรด" ในหน้า 544
การพัฒนาโปรแกรมแบบมัลติเธรดคล้ายกับการพัฒนา โปรแกรมที่มีหลายกระบวนการ การพัฒนาโปรแกรมยังประกอบด้วย การคอมไพล์ และการดีบักโค้ด

การยกเลิกเธรด

เธรดจะยกเลิกโดยอัตโนมัติ เมื่อส่งคืนค่าจากรูทีน entry-point

เธรดยังสามารถยกเลิกได้ด้วยตนเอง หรือยกเลิกเธรดอื่นใดในการประมวลผล การใช้กลไกเช่นนี้เรียกว่า *การยกเลิก* เนื่องจากเธรดทั้งหมดแบ่งใช้พื้นที่ข้อมูลเดียวกัน เธรดต้องดำเนินการกับการดำเนินการล้างข้อมูลในช่วงเวลาของการยกเลิก ไลบรารีเธรดจะเตรียม handler การล้างข้อมูลสำหรับวัตถุประสงค์นี้

การออกจากเธรด

การประมวลผล สามารถออกได้ทุกเวลา เมื่อเธรดเรียกรูทีน **exit** เช่นเดียวกัน เธรดสามารถออกได้ทุกเวลาโดยเรียกรูทีนย่อย **pthread_exit**

การเรียกรูทีนย่อย **exit** จะยกเลิกการประมวลผลทั้งหมด ซึ่งรวมถึงเธรดทั้งหมดด้วย ในโปรแกรมแบบมัลติเธรด รูทีนย่อย **exit** ควรใช้ต่อเมื่อทั้งกระบวนการต้องถูกยกเลิกการทำงาน ตัวอย่างเช่น ในกรณีที่เกิดข้อผิดพลาดที่ไม่สามารถแก้ได้ ซึ่งควรมีรูทีนย่อย **pthread_exit** สำหรับการออกจาก initial thread

การเรียกรูทีนย่อย **pthread_exit** จะยกเลิกการเรียกเธรด พารามิเตอร์ *status* จะถูกบันทึกไว้โดยไลบรารี และสามารถนำไปใช้ในอนาคตได้ เมื่อเชื่อมกับเธรดที่ยกเลิก การเรียกรูทีนย่อย **pthread_exit** จะดูคล้ายกัน แต่ไม่เฉพาะเจาะจง ซึ่งจะส่งคืนค่าจากรูทีนเริ่มต้นของเธรด ผลลัพธ์ของการส่งคืนค่าจากรูทีนเริ่มต้นของเธรด จะขึ้นอยู่กับเธรด:

- การส่งคืนจาก initial thread จะเรียกรูทีนย่อย **exit** ดังนั้น จึงยกเลิกเธรดทั้งหมดในการประมวลผล

- การส่งคืนจากเธรดอื่นจะเรียกรูทีนย่อย **pthread_exit** ค่าที่ส่งคืนอจะมีบทบาทเดียวกับพารามิเตอร์ *status* ของรูทีนย่อย **pthread_exit**

หากต้องการหลีกเลี่ยงการเรียกรูทีนย่อย **exit** ให้ใช้รูทีนย่อย **pthread_exit** เพื่อออกจากเธรด

การออกจาก initial thread (ตัวอย่างเช่น โดยการเรียกรูทีนย่อย **pthread_exit** จากรูทีน **main**) จะไม่ยกเลิกการประมวลผล ซึ่งจะยกเลิกเฉพาะ initial thread ถ้า initial thread ถูกยกเลิก การประมวลผลจะถูกยกเลิก เมื่อเธรดสุดท้ายที่มีอยู่ถูกยกเลิก ในกรณีนี้ โค้ดสำหรับส่งคืนของการประมวลผลคือ 0

โปรแกรมต่อไปนี้จะแสดงข้อความ 10 ข้อความ ในภาษาแต่ละภาษา ซึ่งจะบรรลุเป้าหมายโดยเรียกรูทีนย่อย **pthread_exit** ในรูทีน **main** หลังจากการสร้างเธรดสองตัว และสร้างลูปในรูทีน **Thread**

```
#include <pthread.h>    /* include file for pthreads - the 1st */
#include <stdio.h>      /* include file for printf()           */

void *Thread(void *string)

{
        int i;

        for (i=0; i<10; i++)
                printf("%s\n", (char *)string);
        pthread_exit(NULL);
}

int main()
{
        char *e_str = "Hello!";
        char *f_str = "Bonjour !";

        pthread_t e_th;
        pthread_t f_th;

        int rc;

        rc = pthread_create(&e_th, NULL, Thread, (void *)e_str);
        if (rc)
                exit(-1);
        rc = pthread_create(&f_th, NULL, Thread, (void *)f_str);
        if (rc)
                exit(-1);
        pthread_exit(NULL);
}
```

รูทีนย่อย **pthread_exit** จะปล่อยข้อมูลที่ระบุเฉพาะเธรด ซึ่งรวมถึงสแต็กของเธรด ข้อมูลใดๆ ที่จัดสรรไว้บนสแต็กจะไม่ถูกต้อง เนื่องจากสแต็กจะเป็นอิสระ และหน่วยความจำที่สอดคล้องกัน อาจถูกนำกลับมาใช้ใหม่โดยเธรดอื่น ดังนั้น อ็อบเจ็กต์การประสานเวลา (mutexes และตัวแปรเงื่อนไข) ที่จัดสรรไว้บนสแต็กของเธรด ต้องถูกทำลายก่อนที่เธรดจะเรียกรูทีนย่อย **pthread_exit**

ไม่เหมือนกับรูทีนย่อย **exit** รูทีนย่อย **pthread_exit** ไม่ได้ล้างข้อมูลรีซอร์สของระบบ ที่แบ่งใช้ระหว่างเธรด ตัวอย่างเช่น ไฟล์จะไม่ถูกปิดโดยรูทีนย่อย **pthread_exit** เนื่องจากอาจใช้งานอยู่โดยเธรดอื่น

การยกเลิกเธรด

กลไลการยกเลิกเธรด อนุญาตให้เธรดยกเลิกการประมวลผลของเธรดอื่นใด ในการประมวลผลที่ถูกควบคุมไว้ เธรดเป้าหมาย (นั่นคือ เธรดที่กำลังถูกยกเลิก) สามารถพักคำร้องขอการยกเลิกที่ค้างอยู่ด้วยวิธีการต่างๆ และดำเนินการกับการล้างข้อมูลการประมวลผล ที่ระบุเฉพาะแอ็พพลิเคชัน เมื่อข้อสังเกตของการยกเลิกถูกกระทำตาม เมื่อยกเลิกแล้ว เธรดจะเรียกรูทีนย่อย **pthread_exit((void *)-1)**

การยกเลิกเธรดจะถูกร้องขอโดยการเรียกรูทีนย่อย **pthread_cancel** เมื่อการเรียกส่งคืนคำร้องขอที่ได้ลงทะเบียนแล้ว แต่เธรดอาจยังคงรันอยู่ การเรียกรูทีนย่อย **pthread_cancel** จะไม่เป็นผลสำเร็จเมื่อ ID เธรดที่ระบุไม่ถูกต้อง

สถานะความสามารถในการยกเลิกและชนิด

สถานะและชนิดความสามารถในการยกเลิกของเธรด จะเป็นตัวกำหนดการดำเนินการที่ใช้เพื่อรับคำร้องขอการยกเลิก แต่ละเธรดจะควบคุมสถานะและชนิดความสามารถในการร้องขอที่เป็นของตน ด้วยรูทีนย่อย **pthread_setcancelstate** และ **pthread_setcanceltype**

สถานะของความสามารถในการยกเลิกและชนิดความสามารถในการยกเลิกต่อไปนี้ นำไปสู่กรณีที่อาจเป็นไปได้สามกรณี ดังที่แสดงอยู่ในตารางต่อไปนี้

สภาวะความสามารถในการยกเลิก	ชนิดความสามารถในการยกเลิก	กรณีผลลัพธ์
เปิดใช้งาน	ชนิดใดๆ (ชนิดไม่ถูกละเว้น)	ปิดใช้งานความสามารถในการยกเลิก
เปิดใช้งาน	ปฏิบัติตาม	ปฏิบัติตามความสามารถในการยกเลิก
เปิดใช้งาน	อะซิงโครนัส	ความสามารถในการใช้งานแบบอะซิงโครนัส

กรณที่อาจเป็นไปได้ตามที่กล่าวไว้มีดังนี้:

- *ปิดใช้งานความสามารถในการยกเลิก* คำร้องขอการยกเลิกใดๆ จะถูกตั้งค่าค้างอยู่ จนกระทั่งสถานะของความสามารถในการยกเลิกจะเปลี่ยนแปลงไป หรือเธรดจะถูกยกเลิกด้วยวิธีอื่น

  เธรดควรปิดใช้งานความสามารถในการยกเลิก เมื่อดำเนินการกับการดำเนินการที่ไม่สามารถอินเตอร์รัปต์ได้เท่านั้น ตัวอย่างเช่น ถ้าเธรดกำลังดำเนินการกับการดำเนินการบันทึกไฟล์ที่ซับซ้อนบางไฟล์ (เช่น ฐานข้อมูลที่ทำดัชนีแล้ว) และจะถูกยกเลิกในระหว่างการดำเนินการ ไฟล์อาจจะคงอยู่ในสถานะที่ไม่สอดคล้องกัน หากต้องการหลีกเลี่ยงกรณีนี้ เธรดความปิดใช้งานความสามารถในการยกเลิกในระหว่างการดำเนินการบันทึกไฟล์
- *ปฏิบัติตามความสามารถในการยกเลิก* คำร้องขอการยกเลิกใดๆ จะถูกตั้งค่าค้างอยู่ จนกระทั่งเธรดจะเข้าใกล้จุดของการยกเลิกถัดไป ซึ่งเป็นสถานะของ ความสามารถในการยกเลิกที่เป็นค่าดีฟอลต์

  สถานะของความสามารถในการยกเลิกนี้ มั่นใจว่า เธรดสามารถยกเลิกได้ แต่จำกัดการยกเลิกในช่วงเวลาที่ระบุไว้ในการประมวลผลของเธรดเท่านั้น เรียกว่า *จุดของการยกเลิก* เธรดที่ถูกยกเลิกบนจุดของการยกเลิกจะปล่อยให้ระบบอยู่ในสถานะที่มีความปลอดภัย อย่างไรก็ตาม ข้อมูลของผู้ใช้อาจไม่สอดคล้องกัน หรือการล็อกอาจถูกพักไว้โดยเธรดที่ถูกยกเลิก หากต้องการหลีกเลี่ยงสถานการณ์เหล่านี้ ให้ใช้ handler การล้างข้อมูล หรือปิดใช้งานความสามารถในการยกเลิกภายในส่วนที่สำคัญ สำหรับข้อมูลเพิ่มเติม โปรดดูที่ การใช้ Cleanup Handlers
- *ความสามารถในการยกเลิกแบบอะซิงโครนัส* คำร้องขอการยกเลิกใดๆ จะดำเนินการโดยทันที

เธรดจะถูกยกเลิกแบบอะซิงโครนัส ขณะที่การพักรีซอร์สอาจปล่อยการประมวลผลไว้ หรือแม้ว่า ระบบจะอยู่ในสถานะที่ยากหรือเป็นไปไม่ได้ที่จะกู้คืน สำหรับ ข้อมูลเพิ่มเติมเกี่ยวกับความปลอดภัยในการยกเลิกอะซิงโครนัส โปรดดูที่ Async-Cancel Safety

Async-cancel safety

ฟังก์ชันจะถูกพูดได้ว่าเป็น *ความปลอดภัยในการยกเลิกแบบอะซิงโครนัส* ถ้าฟังก์ชันนั้นถูกเขียน ดังนั้น การเรียกฟังก์ชันด้วยความสามารถในการยกเลิกแบบอะซิงโครนัสที่เปิดใช้งาน ไม่ได้เป็นสาเหตุที่ทำให้รีซอร์สใดๆ พัง แม้ว่า คำร้องขอการยกเลิกจะส่งคืนที่คำสั่งใดๆ

ฟังก์ชันใดๆ ที่ได้รับรีซอร์สซึ่งส่งผลทำให้ไม่สามารถสร้าง ความปลอดภัยในการยกเลิกแบบอะซิงโครนัสได้ ตัวอย่างเช่น ถ้ารูทีนย่อย **malloc** ถูกเรียกด้วยความสามารถในการยกเลิกแบบอะซิงโครนัสที่เปิดใช้งาน รูทีนย่อยอาจได้รับรีซอร์สเป็นผลสำเร็จ แต่อาจจะส่งคืนอให้กับตัวเรียก ซึ่งสามารถดำเนินการตามคำร้องขอการยกเลิก ในกรณีนี้ โปรแกรมจะไม่มีวิธีที่จะทำให้ทราบได้ว่า ได้รับรีซอร์สแล้วหรือไม่

สำหรับเหตุผลนี้ ไลบรารีรูทีนส่วนใหญ่ ไม่สามารถพิจารณาความปลอดภัยในการยกเลิกแบบอะซิงโครนัส ซึ่งแนะนำให้คุณใช้ความสามารถในการยกเลิกแบบอะซิงโครนัสเท่านั้น หากคุณแน่ใจว่า คุณได้ดำเนินการกับการดำเนินการที่ไม่ได้พักรีซอร์ส และเรียกไลบรารีรูทีนที่เป็นความปลอดภัยในการยกเลิกแบบอะซิงโครนัส

รูทีนย่อยต่อไปนี้คือความปลอดภัยในการยกเลิกแบบอะซิงโครนัส ซึ่งมั่นใจได้ว่า การยกเลิกจะถูกจัดการได้อย่างถูกต้อง แม้ว่าความความสามารถในการยกเลิกแบบอะซิงโครนัสจะถูกเปิดใช้งานก็ตาม :

- **pthread_cancel**
- **pthread_setcancelstate**
- **pthread_setcanceltype**

ตัวเลือกความสามารถในการยกเลิกแบบอะซิงโครนัส คือ การปฏิบัติตามความสามารถในการยกเลิก และการเพิ่มจุดของการยกเลิก โดยเรียกรูทีนย่อย **pthread_testcancel**

จุดการยกเลิก

จุดของการยกเลิกคือจุดที่อยู่ภายในรูทีนย่อยบางตัว ซึ่งเธรดต้องดำเนินการกับคำร้องขอการยกเลิกที่ค้างอยู่ หากการปฏิบัติตามความสามารถในการยกเลิกถูกเปิดใช้งาน รูทีนเหล่านี้ทั้งหมดอาจบล็อกการเรียก หรือคำนวณเธรด

จุดของการยกเลิกยังสามารถสร้างได้โดยเรียกรูทีนย่อย **pthread_testcancel** รูทีนย่อยนี้จะสร้างจุดของการยกเลิก ถ้าการปฏิบัติตามความสามารถในการยกเลิกถูกเปิดใช้งาน และถ้าคำร้องขอยกเลิกค้างอยู่ คำร้องขอจะถูกดำเนินการ และเธรดจะถูกยกเลิก มิฉะนั้น รูทีนย่อยจะส่งคืน

จุดของการยกเลิกอื่นๆ จะเกิดขึ้น ขณะที่เรียกรูทีนย่อยต่อไปนี้:

- **pthread_cond_wait**
- **pthread_cond_timedwait**
- **pthread_join**

รูทีนย่อย **pthread_mutex_lock** และ **pthread_mutex_trylock** ไม่ได้จัดเตรียมการยกเลิกไว้ ถ้ารูทีนย่อยได้จัดเตรียมไว้ ฟังก์ชันทั้งหมดที่เรียกรูทีนย่อยเหล่านี้ (และฟังก์ชันจำนวนมากทำสิ่งนี้) จะจัดเตรียมจุดของการยกเลิก การมีจุดของการยกเลิกมาก

เกินไปทำให้โปรแกรมมิงค่อนข้างยาก ซึ่งต้องการการปิดใช้งานและการเรียกคืนความสามารถในการยกเลิกจำนวนมาก หรือความพยายามเป็นพิเศษในการพยายามจัดเรียงความเชื่อถือของการล้างข้อมูลในทุกตำแหน่งที่อาจเป็นไปได้ สำหรับข้อมูลเพิ่มเติมเกี่ยวกับรูทีนย่อยเหล่านี้ โปรดดูที่ การใช้ Mutexes

จุดของการยกเลิกจะเกิดขึ้นเมื่อเธรดกำลังเรียกใช้งานฟังก์ชันต่อไปนี้:

ฟังก์ชัน	
aio_suspend	close
creat	fcntl
fsync	getmsg
getpmsg	lockf
mq_receive	mq_send
msgrcv	msgsnd
msync	nanosleep
open	pause
poll	pread
pthread_cond_timedwait	pthread_cond_wait
pthread_join	pthread_testcancel
putpmsg	pwrite
อ่าน	readv
select	sem_wait
sigpause	sigsuspend
sigtimedwait	sigwait
sigwaitinfo	sleep
ระบบ	tcdrain
usleep	wait
wait3	waitid
waitpid	write
writev	

จุดของการยกเลิกยังสามารถเกิดขึ้นได้เมื่อเธรดกำลังเรียกใช้งานฟังก์ชันต่อไปนี้ :

Function Function Function		
catclose	catgets	catopen
closedir	closelog	ctermid
dbm_close	dbm_delete	dbm_fetch
dbm_nextkey	dbm_open	dbm_store
dlclose	dlopen	endgrent
endpwent	fwprintf	fwrite
fwscanf	getc	getc_unlocked
getchar	getchar_unlocked	getcwd
getdate	getgrent	getgrgid
getgrgid_r	getgrnam	getgrnam_r
getlogin	getlogin_r	popen
printf	putc	putc_unlocked
putchar	putchar_unlocked	puts
pututxline	putw	putwc
putwchar	readdir	readdir_r
remove	rename	rewind
endutxent	fclose	fcntl
fflush	fgetc	fgetpos
fgets	fgetwc	fgetws
fopen	fprintf	fputc
fputs	getpwent	getpwnam
getpwnam_r	getpwuid	getpwuid_r
gets	getutxent	getutxid

Function	Function	Function
getutxline	**getw**	**getwc**
getwchar	**getwd**	**rewinddir**
scanf	**seekdir**	**semop**
setgrent	**setpwent**	**setutxent**
strerror	**syslog**	**tmpfile**
tmpnam	**ttyname**	**ttyname_r**
fputwc	**fputws**	**fread**
freopen	**fscanf**	**fseek**
fseeko	**fsetpos**	**ftell**
ftello	**ftw**	**glob**
iconv_close	**iconv_open**	**ioctl**
lseek	**mkstemp**	**nftw**
opendir	**openlog**	**pclose**
perror	**ungetc**	**ungetwc**
unlink	**vfprintf**	**vfwprintf**
vprintf	**vwprintf**	**wprintf**
wscanf		

ผลกระทบของการดำเนินการตามคำร้องขอให้ยกเลิกขณะที่หยุดทำงานชั่วคราว ในระหว่างที่เรียกฟังก์ชันจะเหมือนกับผลกระทบที่อาจมองเห็นได้ในโปรแกรมแบบเธรดเดี่ยว เมื่อเรียกฟังก์ชันที่ถูกอินเตอร์รัปต์โดยสัญญาณ และฟังก์ชันที่กำหนดไว้ส่งคืน [EINTR] ผลกระทบใดๆ จะเกิดขึ้นก่อนที่จะเรียก handler การล้างข้อมูลการยกเลิก

ไม่ว่าเธรดจะมีความสามารถในการยกเลิกที่เปิดใช้งาน และคำร้องขอให้ยกเลิก ได้ถูกทำด้วยเธรดที่เป็นเป้าหมาย และการเรียกเธรดสำหรับรูทีนย่อย **pthread_testcancel** คำร้องขอให้ยกเลิกจะถูกดำเนินการตามความต้องการ ก่อนที่รูทีนย่อย **pthread_testcancel** จะส่งคืนค่า ถ้าเธรดมีความสามารถในการยกเลิกที่เปิดใช้งาน และเธรดมีคำร้องขอให้ยกเลิกแบบอะซิงโครนัส และเธรดหยุดทำงานชั่วคราวที่จุดของการยกเลิกที่รอจนกว่าเหตุการณ์จะเกิดขึ้น คำร้องขอให้ยกเลิกจะถูกดำเนินการ อย่างไรก็ตาม ถ้าเธรดหยุดทำงานชั่วคราวที่จุดการยกเลิก และเหตุการณ์ที่รอจนกว่าจะเกิดขึ้น ก่อนที่คำร้องขอให้ยกเลิกจะดำเนินการ ลำดับของเหตุการณ์จะถูกพิจารณาคำร้องขอให้ยกเลิกว่าดำเนินการ หรือคำร้องจะคงค้างอยู่ และเธรดจะกลับสู่การทำงานด้วยการประมวลผลแบบปกติ

ตัวอย่างการยกเลิก

ตัวอย่างต่อไปนี้ เธรด "ตัวเขียน" ทั้งสองแบบ จะถูกยกเลิกหลัง 10 วินาที และหลังจากที่เธรดได้เขียนข้อความ อย่างน้อยห้าครั้ง

```
#include <pthread.h>    /* include file for pthreads - the 1st */
#include <stdio.h>      /* include file for printf()           */
#include <unistd.h>     /* include file for sleep()            */

void *Thread(void *string)
{
        int i;
        int o_state;

        /* disables cancelability */
        pthread_setcancelstate(PTHREAD_CANCEL_DISABLE, &o_state);

        /* writes five messages */
        for (i=0; i<5; i++)
                printf("%s\n", (char *)string);

        /* restores cancelability */
```

```
        pthread_setcancelstate(o_state, &o_state);

        /* writes further */
        while (1)
                printf("%s\n", (char *)string);
        pthread_exit(NULL);
}

int main()
{
        char *e_str = "Hello!";
        char *f_str = "Bonjour !";

        pthread_t e_th;
        pthread_t f_th;

        int rc;

        /* creates both threads */
        rc = pthread_create(&e_th, NULL, Thread, (void *)e_str);
        if (rc)
                return -1;
        rc = pthread_create(&f_th, NULL, Thread, (void *)f_str);
        if (rc)
                return -1;
        /* sleeps a while */
        sleep(10);

        /* requests cancelation */
        pthread_cancel(e_th);
        pthread_cancel(f_th);

        /* sleeps a bit more */
        sleep(10);
        pthread_exit(NULL);
}
```

รูทีนย่อย Timer และ sleep

รูทีน Timer จะเรียกใช้งานในบริบท ของการเรียกเธรด ดังนั้น ถ้าตัวแปรหมดอายุ ฟังก์ชัน watchdog timer จะถูกเรียกใน บริบทของเธรด เมื่อการประมวลผลหรือเธรด sleep การประมวลผลหรือเธรดนั้น จะละทิ้งตัวประมวลผล ในกระบวนการแบบมัลติเธรด เธรดที่เรียกใช้เท่านั้นที่ ถูกพักการทำงาน

การใช้ cleanup handlers

handler การล้างข้อมูลจะจัดเตรียมกลไลที่สามารถย้ายได้สำหรับการปล่อยรีซอร์ส และเรียกคืนค่าคงที่เมื่อเธรดยกเลิก

การเรียกใช้ cleanup handlers

handler การล้างข้อมูล จะระบุเธรดแต่ละเธรด เธรดสามารถมี handler การล้างข้อมูลหลายๆ แบบ ซึ่งจะเก็บอยู่ในสแต็กแบบ LIFO (เข้าหลังออกก่อน) ที่ระบุเฉพาะเธรด handler การล้างข้อมูล ยังถูกเรียกทั้งหมดในกรณีต่อไปนี้:

- เธรดส่งคืนจากรูทีน entry-point

- เธรดเรียกรูทีนย่อย **pthread_exit**
- เธรดทำหน้าที่ร้องขอการยกเลิก

handler การล้างข้อมูลจะถูกส่งไปยังสแต็กการล้างข้อมูลด้วยรูทีนย่อย **pthread_cleanup_push** รูทีนย่อย **pthread_cleanup_pop** จะแสดง handler การล้างข้อมูลสูงสุดจากสแต็กและเรียกใช้งาน ใช้รูทีนยอยนี้เมื่อ handler การล้างข้อมูลไม่มีความต้องการอีกต่อไป

handler การล้างข้อมูลคือรูทีนที่ผู้ใช้กำหนดเอง ซึ่งมีหนึ่งพารามิเตอร์ ตัวชี้ void ซึ่งระบุไว้เมื่อเรียกรูทีนย่อย **pthread_cleanup_push** คุณสามารถระบุตัวชี้ไปยังข้อมูลบางส่วน ซึ่ง handler การล้างข้อมูลจำเป็นต้องดำเนินการ

ในตัวอย่างต่อไปนี้ บัฟเฟอร์จะถูกจัดสรรไว้ สำหรับการดำเนินการบางอย่าง ด้วความสามารถในการยกเลิกที่เปิดใช้งาน การดำเนินการสามารถหยุดทำงานที่จุดของการยกเลิกใดๆ ในกรณีนั้น handler การล้างข้อมูลที่สร้างไว้เพื่รีลีสบัฟเฟอร์

```
/* the cleanup handler */

cleaner(void *buffer)

{
        free(buffer);
}

/* fragment of another routine */
...
myBuf = malloc(1000);
if (myBuf != NULL) {

        pthread_cleanup_push(cleaner, myBuf);

        /*
         *       perform any operation using the buffer,
         *       including calls to other functions
         *       and cancelation points
         */

        /* pops the handler and frees the buffer in one call */
        pthread_cleanup_pop(1);
}
```

การใช้ความสามารถในการยกเลิกที่เธรดจะไม่ดำเนินการเกี่ยวกับคำร้องขอให้ยกเลิก ระหว่างการจัดสรรบัฟเฟอร์และการลงทะเบียน handler การล้างข้อมูล เนื่องจากรูทีนย่อย **malloc** หรือรูทีนย่อย **pthread_cleanup_push** ไม่ได้จัดเตรียมจุดของการยกเลิกไว้ ขณะที่ pop handler การล้างข้อมูล handler จะถูกเรียกใช้งาน และปล่อยบัฟเฟอร์ โปรแกรมที่มีความซับซ้อนมากขึ้นอาจไม่เรียกใช้งาน handler ขณะที่ pop เนื่องจาก handler การล้างข้อมูลควรถูกพิจารณาเป็น "การออกแบบฉุกเฉิน" สำหรับส่วนของโค้ดที่ป้องกัน

การทำให้เกิดความสมดุลย์ระหว่างการดำเนินการ push และ pop

รูทีนย่อย **pthread_cleanup_push** และ **pthread_cleanup_pop** ควรปรากฏขึ้นเป็นคู่ภายในขอบเขต lexical นั่นคือ ภายในฟังก์ชันเดียวกัน และบล็อกคำสั่งเดียวกัน รูทีนย่อยเหล่านี้สามารถนำมาพิจารณาให้ปิดล้อมด้วยวงเล็บซ้ายและขวา ในส่วนของโค้ดที่ป้องกันไว้

เหตุผลสำหรับกฎนี้คือ ในระบบบางระบบ รูทีนย่อยเหล่านี้จะถูกนำไปใช้เป็นแมโคร รูทีนย่อย **pthread_cleanup_push** จะถูกนำมาใช้เป็นเครื่องหมายวงเล็บทางซ้าย แล้วตามด้วยคำสั่งอื่นๆ:

```
#define pthread_cleanup_push(rtm,arg) { \
        /* other statements */
```

รูทีนย่อย **pthread_cleanup_pop** จะถูกนำไปใช้เป็นเครื่องหมายวงเล็บ แล้วตามด้วยคำสั่งอื่นๆ ต่อไปนี้:

```
#define pthread_cleanup_pop(ex) \
        /* other statements */  \
}
```

ปฏิบัติตามกฎความสมดุลสำหรับรูทีนย่อย **pthread_cleanup_push** และ **pthread_cleanup_pop** แล้ว เพื่อหลีกเลี่ยงข้อผิดพลาดเกี่ยวกับคอมไพลเลอร์หรือลักษณะการทำงานที่คาดไม่ถึงของโปรแกรมของคุณ ขณะที่ย้ายไปยังระบบอื่น

ใน AIX รูทีนย่อย **pthread_cleanup_push** และ **pthread_cleanup_pop** คือไลบรารีรูทีน และอาจไม่เกิดความสมดุลภายในบล็อกคำสั่งเดียวกัน อย่างไรก็ตาม รูทีนย่อยเหล่านี้ต้องมีความสมดุลกันภายในโปรแกรม เนื่องจาก handler การล้างข้อมูลจะถูกทำเป็นสแต็กไว้

รูทีนย่อย	คำอธิบาย
pthread_attr_destroy	ลบอ็อบเจ็กต์แอ็ตทริบิวต์เธรด
pthread_attr_getdetachstate	ส่งคืนค่าของแอ็ตทริบิวต์ detachstate ของอ็อบเจ็กต์แอ็ตทริบิวต์เธรด
pthread_attr_init	สร้างอ็อบเจ็กต์แอ็ตทริบิวต์เธรด และกำหนดค่าเริ่มต้นด้วยค่าดีฟอลต์
pthread_cancel	ร้องขอการยกเลิกของเธรด
pthread_cleanup_pop	ลบออก และเรียกใช้งานตามการเลือก รูทีนที่อยู่ด้านบนสุดของสแต็กการล้างข้อมูลของเธรด
pthread_cleanup_push	ส่งรูทีนไปยังการเรียกสแต็กการล้างข้อมูลของเธรด
pthread_create	สร้างเธรดใหม่ กำหนดค่าเริ่มต้นให้กับแอ็ตทริบิวต์ และทำให้สามารถรันได้
pthread_equal	เปรียบเทียบสอง ID เธรด
pthread_exit	ยกเลิกการเรียกเธรด
pthread_self	ส่งคืนการเรียก ID ของเธรด
pthread_setcancelstate	ตั้งค่าการเรียกสถานะความสามารถในการยกเลิกของเธรด
pthread_setcanceltype	ตั้งค่าการเรียกชนิดความสามารถในการยกเลิกของเธรด
pthread_testcancel	สร้างจุดของการยกเลิกในการเรียกเธรด

หลักการที่เกี่ยวข้อง:

“การกำหนดค่าเริ่มต้นครั้งเดียว” ในหน้า 510

บางไลบรารี C ถูกออกแบบสำหรับการกำหนดค่าเริ่มต้น แบบไดนามิก ซึ่งการกำหนดค่าเริ่มต้นโกลบอลสำหรับไลบรารีถูกดำเนินการ เมื่อโพรซีเดอร์แรกในไลบรารีถูกเรียก

ภาพรวมการซิงโครไนซ์

ผลประโยชน์หลักส่วนหนึ่งที่ได้รับจากการใช้เธรดคือ ง่ายต่อการใช้สิ่งอำนวยความสะดวกในการประสานเวลา

เพื่อให้โต้ตอบกันอย่างมีประสิทธิผล เธรดต้องซิงโครไนซ์กับกิจกรรมเหล่านั้น ซึ่งประกอบด้วย:

- การสื่อสารทางอ้อมผ่านการแก้ไขข้อมูลที่แบ่งใช้
- การสื่อสารทางตรงโดยแจ้งให้ทราบถึงเหตุการณ์อื่นๆ แต่ละเหตุการณ์ที่เกิดขึ้น

อ็อบเจ็กต์การประสานเวลาที่ซับซ้อนมากขึ้นสามารถสร้างได้โดยใช้อ็อบเจ็กต์พื้นฐาน

ไลบรารีเธรดจัดเตรียมกลไกการซิงโครไนซ์ต่อไปนี้ แม้ว่า เมื่อก่อน กลไกที่มีประสิทธิภาพเหล่านี้สามารถใช้เพื่อสร้างกลไกที่มีความซับซ้อนมากขึ้น

ไลบรารีเธรดจัดเตรียมกลไกของการประสานเวลา ดังต่อไปนี้:

- Mutexes (โปรดดูที่ การใช้ Mutexes)
- ตัวแปรเงื่อนไข (โปรดดูที่ การใช้ตัวแปรเงื่อนไข)
- ล็อกการอ่าน–เขียน (โปรดดูที่ การใช้ล็อกการอ่าน–เขียน)
- การเชื่อม (โปรดดูที่ การเชื่อมเธรด)

ซึ่งแต่เดิมกลไกที่มีประสิทธิภาพเหล่านี้สามารถนำมาใช้ เพื่อสร้างกลไกที่มีความซับซ้อนมากขึ้น

หลักการที่เกี่ยวข้อง:

"การสร้างอ็อบเจ็กต์การซิงโครไนซ์ที่ซับซ้อน" ในหน้า 516

รูทีนย่อยที่จัดเตรียมในไลบรารีเธรดสามารถถูก ใช้เป็นหลักในการสร้างอ็อบเจ็กต์การซิงโครไนซ์ที่ซับซ้อนมากขึ้น

การใช้ mutexes

mutex คือล็อกการไม่เกิดร่วมกัน มีหนึ่งเธรดเท่านั้นที่สามารถ hold ล็อก

Mutexes ถูกใช้เพื่อป้องกันข้อมูลหรือทรัพยากรอื่นจาก การเข้าถึงพร้อมกัน mutex มีแอ็ตทริบิวต์ ซึ่งระบุคุณลักษณะของ mutex

อ็อบเจ็กต์แอ็ตทริบิวต์ Mutex

เหมือนกับ threads, mutexes ถูกสร้างด้วยความช่วยเหลือของอ็อบเจ็กต์แอ็ตทริบิวต์ อ็อบเจ็กต์แอ็ตทริบิวต์ mutex เป็นอ็อบเจ็กต์ abstract, มีหลายแอ็ตทริบิวต์ ขึ้นกับการการนำไปปฏิบัติ ของตัวเลือก POSIX โดยเข้าถึงผ่านตัวแปรชนิด **pthread_mutexattr_t** ใน AIX ชนิดข้อมูล **pthread_mutexattr_t** เป็นพอยเตอร์ บนระบบอื่นๆ อาจเป็นชนิดข้อมูลแบบโครงสร้าง หรือแบบอื่น

การสร้างและการทำลายอ็อบเจ็กต์แอ็ตทริบิวต์ mutex

อ็อบเจ็กต์แอ็ตทริบิวต์ mutex ถูกกำหนดเป็นค่าเริ่มต้น โดยรูทีนย่อย **pthread_mutexattr_init** แอ็ตทริบิวต์ถูกจัดการโดยรูทีนย่อย อ็อบเจ็กต์แอ็ตทริบิวต์ เธรดถูกทำลายโดยรูทีนย่อย **pthread_mutexattr_destroy** รูทีนย่อยนี้อาจรีลีส พื้นที่จัดเก็บข้อมูลแบบไดนามิกที่จัดสรรโดยรูทีนย่อย **pthread_mutexattr_init** ขึ้นกับการนำไปปฏิบัติของไลบรารีเธรด

ในตัวอย่างดังต่อไปนี้ อ็อบเจ็กต์แอ็ตทริบิวต์ mutex ถูกสร้างและกำหนด ด้วยค่าเริ่มต้น แล้วใช้และถูกทำลายในตอนสุดท้าย:

```
pthread_mutexattr_t attributes;
                /* the attributes object is created */
...
if (!pthread_mutexattr_init(&attributes)) {
                /* the attributes object is initialized */
        ...
                /* using the attributes object */
        ...
        pthread_mutexattr_destroy(&attributes);
                /* the attributes object is destroyed */
}
```

อ็อบเจ็กต์แอ็ตทริบิวต์เดียวกันสามารถถูกใช้เพื่อสร้างหลาย mutexes และสามารถ ถูกแก้ไขระหว่างการสร้าง mutex เมื่อ mutexes ถูกสร้าง อ็อบเจ็กต์แอ็ตทริบิวต์สามารถถูกทำลายได้โดยไม่มีผลกับ mutexes ที่ สร้าง

แอ็ตทริบิวต์ Mutex

แอ็ตทริบิวต์ mutex ดังต่อไปนี้ถูกกำหนด:

แอ็ตทริบิวต์	คำอธิบาย
Protocol	ระบุโปรโตคอลที่ใช้เพื่อป้องกันความผกผันลำดับความสำคัญสำหรับ mutex แอ็ตทริบิวต์นี้ขึ้นกับตัวเลือก priority inheritance หรือ priority protection POSIX
Process-shared	ระบุการแบ่งใช้การประมวลผลของ mutex แอ็ตทริบิวต์นี้ ขึ้นกับตัวเลือก process sharing POSIX

สำหรับข้อมูลเพิ่มเติมเกี่ยวกับแอ็ตทริบิวต์เหล่านี้ โปรดดูที่ อ็อพชันไลบรารีเธรด และ การกำหนดเวลาการซิงโครไนซ์

การสร้างและการทำลาย mutexes

mutex ถูกสร้างโดยการเรียกรูทีนย่อย **pthread_mutex_init** คุณอาจระบุอ็อบเจ็กต์ แอ็ตทริบิวต์ mutex ถ้าคุณระบุตัวชี้ **NULL**, mutex จะ มีแอ็ตทริบิวต์ค่าเริ่มต้น ดังนั้นส่วนของโค้ดต่อไปนี้:

```
pthread_mutex_t         mutex;
pthread_mutexattr_t    attr;
...
pthread_mutexattr_init(&attr);
pthread_mutex_init(&mutex, &attr);
pthread_mutexattr_destroy(&attr);
```

เหมือนกับ:

```
pthread_mutex_t         mutex;
...
pthread_mutex_init(&mutex, NULL);
```

ID ของ mutex ที่สร้างถูกส่งกลับไปที่เธรดที่เรียก ผ่านพารามิเตอร์ *mutex* mutex ID เป็นอ็อบเจ็กต์ opaque; ชนิดเป็น **pthread_mutex_t** ใน AIX ชนิดข้อมูล **pthread_mutex_t** เป็น structure บนระบบอื่น อาจเป็นตัวชี้หรือชนิดข้อมูลอื่น

mutex ต้องถูกสร้างครั้งเดียว อย่างไรก็ตาม หลีกเลี่ยงการเรียกรูทีนย่อย **pthread_mutex_init** มากกว่าหนึ่งครั้งด้วยพารามิเตอร์ *mutex* เดียวกัน (ตัวอย่างเช่น สองเธรดดำเนินการ กับโค้ดเหมือนกันพร้อมกัน) การตรวจสอบความไม่ซ้ำซ้อนของการสร้าง mutex กระทำได้ใน วิธีดังต่อไปนี้:

- เรียกรูทีนย่อย **pthread_mutex_init** ก่อน สร้างเธรดอื่นที่จะใช้ mutex นี้ ตัวอย่างเช่นใน initial thread
- เรียกรูทีนย่อย **pthread_mutex_init** ภายใน รูทีน one time initialization สำหรับข้อมูลเพิ่มเติม โปรดดูที่ การกำหนดคาเริ่มต้นแบบครั้งเดียว
- ใช้ static mutex ที่กำหนดค่าโดยแมโคร **PTHREAD_MUTEX_INITIALIZER** static initialization; mutex จะมีแอ็ตทริบิวต์เริ่มต้น

หลังจากไม่จำเป็นต้องใช้ mutex ให้ทำลายโดยเรียกรูทีนย่อย **pthread_mutex_destroy** รูทีนย่อยนี้อาจ เรียกคืนพื้นที่จัดเก็บข้อมูลที่จัดสรรโดยรูทีนย่อย **pthread_mutex_init** หลังจากทำลาย mutex ตัวแปร **pthread_mutex_t** เดิมสามารถถูกนำมาใช้ใหม่เพื่อสร้าง mutex อื่น ตัวอย่างเช่น ส่วนของโค้ด ต่อไปนี้ถูกต้อง แม้ว่าจะไม่นำมาใช้มากนัก:

```
pthread_mutex_t         mutex;
...
for (i = 0; i < 10; i++) {

        /* creates a mutex */
        pthread_mutex_init(&mutex, NULL);

        /* uses the mutex */

        /* destroys the mutex */
        pthread_mutex_destroy(&mutex);
}
```

เหมือนกับทรัพยากรระบบที่สามารถถูกแบ่งใช้ระหว่างเธรด mutex ที่จัดสรร บนสแต็กของเธรดต้องถูกทำลายก่อนที่จะหยุดการทำงานเธรด ไลบรารีเธรด ยังคงรายการที่ลิงก์ของ mutexes ไว้ ดังนั้น ถ้าสแต็กที่ mutex จัดสรรว่างลง list จะถูกทำให้เสียหาย

ชนิดของ mutexes

ชนิดของ mutex กำหนดวิธีที่ mutex ทำงานเมื่อ มีการดำเนินการ ชนิดของ mutexes ดังต่อไปนี้มีอยู่:

PTHREAD_MUTEX_DEFAULT หรือ PTHREAD_MUTEX_NORMAL
: มีผลให้เกิด deadlock ถ้า pthread เดียวกันพยายามล็อกในครั้งที่สอง โดยใช้รูทีนย่อย **pthread_mutex_lock** โดยไม่ได้ทำการปลดล็อกก่อน นี่เป็นชนิดเริ่มต้น

PTHREAD_MUTEX_ERRORCHECK
: หลีกเลี่ยง deadlocks โดยส่งคืนค่าที่ไม่เป็นศูนย์ถ้า thread เดียวกันพยายาม ล็อก mutex เดียวกันมากกว่าหนึ่งครั้งโดยไม่ได้ปลดล็อก mutex ก่อน

PTHREAD_MUTEX_RECURSIVE
: อนุญาตให้ pthread ทำการล็อก mutex แบบเรียกซ้ำโดยใช้รูทีนย่อย **pthread_mutex_lock** โดยไม่ส่งผลให้เกิด deadlock หรือได้รับค่าส่งกลับที่ไม่ใช่ศูนย์จาก **pthread_mutex_lock** pthread เดียวกันต้องเรียกรูทีนย่อย **pthread_mutex_unlock** จำนวนครั้งเท่ากับที่เรียกรูทีนย่อย **pthread_mutex_lock** เพื่อปลดล็อก mutex สำหรับให้ pthreads อื่นใช้

เมื่อแอ็ตทริบิวต์ mutex ถูกสร้างในครั้งแรก มีชนิดเริ่มต้นเป็น **PTHREAD_MUTEX_NORMAL** หลังการสร้าง mutex ชนิดสามารถถูกเปลี่ยน โดยใช้การเรียกไลบรารี API **pthread_mutexattr_settype**

ต่อไปนี้เป็นตัวอย่างของการสร้างและการใช้ชนิด recursive mutex:

```
pthread_mutexattr_t    attr;
pthread_mutex_t         mutex;

pthread_mutexattr_settype(&attr, PTHREAD_MUTEX_RECURSIVE);
pthread_mutex_init(&mutex, &attr);

struct {
        int a;
        int b;
        int c;
} A;
```

```
f()
{
        pthread_mutex_lock(&mutex);
        A.a++;
        g();
        A.c = 0;
        pthread_mutex_unlock(&mutex);
}

g()
{
        pthread_mutex_lock(&mutex);
        A.b += A.a;
        pthread_mutex_unlock(&mutex);
}
```

การล็อกและการปลดล็อก mutexes

mutex เป็นการล็อกปกติ มีสองภาวะ: locked และ unlocked เมื่อถูกสร้าง mutex มีสถานะ unlocked รูทีนย่อย **pthread_mutex_lock** ล็อก mutex ที่ระบุภายใต้ เงื่อนไขดังต่อไปนี้:

- ถ้า mutex ถูกปลดล็อก รูทีนย่อยจะล็อก mutex
- ถ้า mutex ถูกล็อกอยู่แล้วโดยเธรดอื่น รูทีนย่อยจะบล็อก เธรดที่เรียก จนกว่า mutex จะถูกปลดล็อก
- ถ้า mutex ถูกล็อกอยู่แล้วโดยเธรดที่เรียก รูทีนย่อยอาจ บล็อกไปตลอด หรือส่งกลับข้อผิดพลาด ขึ้นกับชนิดของ mutex

รูทีนย่อย **pthread_mutex_trylock** ทำงาน เหมือนรูทีนย่อย **pthread_mutex_lock** โดยไม่มี การบล็อกเธรดที่เรียกภายใต้เงื่อนไขดังต่อไปนี้:

- ถ้า mutex ถูกปลดล็อก รูทีนย่อยจะล็อก mutex
- ถ้า mutex ถูกล็อกอยู่แล้วโดยเธรดอื่น รูทีนย่อยส่งกลับ ข้อผิดพลาด

เธรดที่ล็อก mutex บ่อยครั้งจะเรียกว่า *owner* ของ mutex

รูทีนย่อย **pthread_mutex_unlock** รีเซ็ต mutex ที่ระบุเป็นภาวะ unlocked ถ้าเป็นของ mutex ที่เรียกภายใต้เงื่อนไขดังต่อไปนี้under the following conditions:

- ถ้า mutex ถูกปลดล็อกแล้ว รูทีนย่อยส่งกลับข้อผิดพลาด
- ถ้า mutex เป็นของเธรดที่เรียก รูทีนย่อยปลดล็อก mutex
- ถ้า mutex เป็นของเธรดอื่น รูทีนย่อยอาจส่งกลับ ข้อผิดพลาดหรือปลดล็อก mutex ขึ้นกับชนิดของ mutex ไม่แนะนำให้ปลดล็อก mutex เนื่องจาก mutexes โดยปกติถูกล็อกและเนื่องจาก โดย pthread เดียวกัน

เนื่องจากการล็อกไม่ให้ cancelation point, เธรดที่บล็อก ขณะรอ mutex ไม่สามารถถูกยกเลิกได้ ดังนั้น ขอแนะนำ ให้คุณใช้ mutexes เฉพาะช่วงเวลาสั้นๆ ตามตัวอย่าง ซึ่งคุณป้องกันข้อมูลจากการเข้าถึงพร้อมกัน สำหรับข้อมูลเพิ่มเติม โปรดดูที่ จุดการยกเลิก และ การยกเลิกเธรด

การป้องกันข้อมูลด้วย mutexes

Mutexes ถูกออกแบบให้รองรับเป็น low-level primitive ซึ่งฟังก์ชันการซิงโครไนซ์ ของเธรดอื่นสามารถถูกสร้างหรือเป็นล็อก การป้องกันข้อมูล สำหรับข้อมูลเพิ่มเติมเกี่ยวกับการประยุกต์ใช้ long locks และ writer-priority readers/writers locks โปรดดูที่ "การใช้ mutexes" ในหน้า 475

ตัวอย่างการใช้งาน Mutex

Mutexes สามารถถูกใช้เพื่อป้องกันข้อมูลจากการเข้าถึงพร้อมกัน ตัวอย่างเช่น แอ็พพลิเคชันฐานข้อมูลอาจสร้างหลายเธรดเพื่อจัดการหลายการร้องขอ พร้อมกัน ตัวฐานข้อมูลเองถูกป้องกันโดย mutex ชื่อว่า **db_mutex** ตัวอย่างเช่น:

```
/* the initial thread */
pthread_mutex_t         mutex;
int i;
...
pthread_mutex_init(&mutex, NULL);    /* creates the mutex      */
for (i = 0; i < num_req; i++)        /* loop to create threads */
        pthread_create(th + i, NULL, rtn, &mutex);
...                                  /* waits end of session   */
pthread_mutex_destroy(&mutex);       /* destroys the mutex     */
...

/* the request handling thread */
...                                  /* waits for a request  */
pthread_mutex_lock(&db_mutex);       /* locks the database   */
...                                  /* handles the request  */
pthread_mutex_unlock(&db_mutex);     /* unlocks the database */
...
```

initial thread สร้าง mutex และเธรด request-handling ทั้งหมด mutex ถูกส่งไปที่เธรดโดยใช้พารามิเตอร์ของรูทีน entry point ของเธรด ในโปรแกรมจริง แอดเดรสของ mutex อาจเป็นฟิลด์ ของ structure ข้อมูลที่ซับซ้อนที่ส่งไปที่เธรดที่สร้าง

การหลีกเลี่ยง Deadlocks

มีหลายวิธีที่ แอ็พพลิเคชันแบบมัลติเธรดอาจเกิด deadlock ต่อไปนี้เป็นตัวอย่างบางส่วน:

- mutex ที่สร้างด้วยชนิดเริ่มต้น **PTHREAD_MUTEX_NORMAL** ไม่สามารถถูกล็อกซ้ำโดย pthread เดียวกันโดยไม่ส่งผลให้เกิด deadlock
- แอ็พพลิเคชัน สามารถ deadlock ได้เมื่อล็อก mutexes ในลำดับย้อนกลับ ตัวอย่างเช่น ส่วนของโค้ดดังต่อไปนี้สามารถสร้าง deadlock ระหว่างเธรด A และ B

  ```
  /* Thread A */
  pthread_mutex_lock(&mutex1);
  pthread_mutex_lock(&mutex2);

  /* Thread B */
  pthread_mutex_lock(&mutex2);
  pthread_mutex_lock(&mutex1);
  ```

- แอ็พพลิเคชันสามารถ deadlock ที่เรียกว่า *resource* deadlock ตัวอย่างเช่น:

  ```
  struct {
                  pthread_mutex_t         mutex;
                  char *buf;
          } A;
  ```

```
struct {
                pthread_mutex_t         mutex;
                char *buf;
        } B;

struct {
                pthread_mutex_t         mutex;
                char *buf;
        } C;

use_all_buffers()
{
        pthread_mutex_lock(&A.mutex);
        /* use buffer A */

        pthread_mutex_lock(&B.mutex);
        /* use buffers B */

        pthread_mutex_lock(&C.mutex);
        /* use buffer C */

        /* All done */
        pthread_mutex_unlock(&C.mutex);
        pthread_mutex_unlock(&B.mutex);
        pthread_mutex_unlock(&A.mutex);
}

use_buffer_a()
{
        pthread_mutex_lock(&A.mutex);
        /* use buffer A */
        pthread_mutex_unlock(&A.mutex);
}

functionB()
{
        pthread_mutex_lock(&B.mutex);
        /* use buffer B */
        if (..some condition)
        {
          use_buffer_a();
        }
        pthread_mutex_unlock(&B.mutex);
}

/* Thread A */
use_all_buffers();

/* Thread B */
functionB();
```

แอ็พพลิเคชันนี้มีสองเธรด คือ thread A และ thread B Thread B จะเริ่มทำงานเพื่อรันก่อน จากนั้น thread A จะเริ่มทำงาน ไม่นานหลังจากนั้น ถ้า thread A รัน **use_all_buffers( )** และล็อก **A.mutex** สำเร็จ แล้วจะบล็อกเมื่อพยายามล็อก **B.**

mutex เนื่องจาก `thread B` ได้ทำการล็อกแล้ว ขณะที่ `thread B` เรียกทำงาน **functionB** และ `some_condition` เกิดขึ้น ขณะที่ `thread A` ถูกบล็อก `thread B` จะบล็อกความพยายามในการขอรับe **A.mutex** ซึ่งถูกล็อกไว้โดย `thread A` แล้ว สิ่งนี้จะส่งผลให้เกิด deadlock

วิธีแก้ปัญหาสำหรับ deadlock นี้เพื่อให้แต่ละเธรดได้รับล็อกทรัพยากรทั้งหมด ที่จำเป็น ก่อนการใช้ทรัพยากร ถ้าเธรดไม่ได้รับล็อก เธรดต้องรีลีสล็อก และเริ่มอีกครั้ง

เงื่อนไข Mutexes และ race

ล็อกที่ไม่เกิดพร้อมกัน (mutexes) สามารถป้องกันข้อมูลไม่สอดคล้องกัน เนื่องจากเงื่อนไข race เงื่อนไข race เกิดขึ้นบ่อยครั้งเมื่อเธรดตั้งแต่สองเธรดขึ้นไป ต้องดำเนินการกับพื้นที่หน่วยความจำเดียวกัน แต่ผลของการคำนวณขึ้นกับลำดับซึ่งการดำเนินการเหล่านี้ถูกสั่งการ

ตัวอย่างเช่น พิจารณาตัวนับเดี่ยว *X* ที่เพิ่มขึ้น โดยสองเธรด คือ A และ B ถ้า*X* มีค่าเดิมเป็น 1 ดังนั้น เมื่อเธรด A และ B เพิ่มตัวนับ *X* ควรเป็น 3 เธรดทั้งสองเป็นเอนทิตีที่ไม่ขึ้นต่อกัน และไม่มีการซิงโครไนซ์ กัน แม้ว่าคำสั่ง C X++ ดูธรรมดาพอที่จะเป็น atomic โค้ด assembly ที่สร้างขึ้น อาจไม่เป็นดังนั้น ดังที่แสดงในโค้ด pseudo-assembler ต่อไปนี้:

```
move    X, REG
inc     REG
move    REG, X
```

ถ้าทั้งสองเธรดในตัวอย่างก่อนหน้าถูกรัน พร้อมกันบนสอง CPU หรือถ้าการจัดตารางทำให้เธรดแยกรัน บนแต่ละคำสั่ง ขั้นตอนดังต่อไปนี้อาจเกิดขึ้น:

1. เธรด A รันคำสั่งแรกและใส่ *X* ซึ่งคือ 1 ลงในรีจิสเตอร์เธรด A จากนั้นเธรด B รันและใส่ *X* ซึ่งคือ 1 ลงในรีจิสเตอร์เธรด B ตัวอย่างต่อไปนี้แสดงผลของรีจิสเตอร์และเนื้อหาของ หน่วยความจำ *X*

   ```
   Thread A Register = 1
   Thread B Register = 1
   Memory X          = 1
   ```

2. เธรด A จะเรียกทำงานคำสั่งที่สองและเพิ่มเนื้อหาของรีจิสเตอร์ของตัวเอง เป็น 2 จากนั้นเธรด B จะเพิ่มรีจิสเตอร์ของตัวเองเป็น 2 ไม่มีข้อมูลใดที่ถูกย้ายไปยังหน่วยความจำ *X* ดังนั้นหน่วยความจำ *X* จะยัง เหมือนเดิม ตัวอย่างต่อไปนี้แสดงผลของรีจิสเตอร์และเนื้อหาของ หน่วยความจำ *X*

   ```
   Thread A Register = 2
   Thread B Register = 2
   Memory X          = 1
   ```

3. เธรด A ย้ายข้อมูลของรีจิสเตอร์ของตัวเอง ซึ่งขณะนี้คือ 2 ไปที่หน่วยความจำ *X* จากนั้นเธรด B ย้ายข้อมูลของรีจิสเตอร์ของตัวเอง ซึ่งขณะนี้คือ 2 ด้วยเช่นกันไปที่ *X* เขียนทับค่าของเธรด A ตัวอย่างต่อไปนี้แสดงผลของรีจิสเตอร์และเนื้อหาของ หน่วยความจำ *X*

   ```
   Thread A Register = 2
   Thread B Register = 2
   Memory X          = 2
   ```

ในกรณีส่วนใหญ่เธรด A และเธรด B รันสาม คำสั่งหนึ่งคำสั่งหลังจากอีกคำสั่ง และผลลัพธ์ควรจะเป็น 3 ตามที่คาดไว้ เงื่อนไข Race โดยปกติยากที่จะค้นพบ เนื่องจากเงื่อนไขนี้เกิดขึ้นน้อย

เมื่อต้องการหลีดเลี่ยงเงื่อนไข race นี้ แต่ละเธรดควรล็อกข้อมล ก่อนเข้าถึงตัวนับและการอัพเดตหน่วยความจำ *X* ตัวอย่างเช่น ถ้าเธรด A ใช้ล็อกและอัพเดตตัวนับ เธรด A จะปล่อยให้หน่วยความจำ *X* เป็นค่า 2 หลังจากที่เธรด A ปลดล็อก เธรด B จะใช้ล็อกและอัพเดตตัวนับ โดยใช้ค่า 2 เป็นค่าเริ่มต้น สำหรับ *X* โดยเพิ่มค่าเป็น 3 ซึ่งเป็นค่าที่ต้องการ

การใช้ตัวแปรเงื่อนไข

ตัวแปรเงื่อนไขยอมให้ threads รอจนกว่า จะมีเหตุการณ์หรือเงื่อนไขเกิดขึ้น

ตัวแปรเงื่อนไขมีแอ็ตทริบิวต์ ที่ระบุคุณลักษณะของเงื่อนไข โดยทั่วไป โปรแกรมใช้ อ็อบเจ็กต์ดังต่อไปนี้:

- ตัวแปร boolean บ่งชี้ว่าตรงตามเงื่อนไขหรือไม่
- mutex เพื่อ serialize การเข้าถึงกับตัวแปร boolean
- ตัวแปรเงื่อนไขเพื่อรอเงื่อนไข

การใช้ตัวแปรเงื่อนไขที่ต้องการการทำงานของ โปรแกรมเมอร์ อย่างไรก็ตาม ตัวแปรเงื่อนไขยอมให้มีการใช้กลไก การซิงโครไนซ์ความสามารสูงและมีประสิทธิภาพ สำหรับข้อมูลเพิ่มเติมเกี่ยวกับ การนำ long locks และ semaphores ไปใช้กับตัวแปรเงื่อนไข โปรดดูที่ การสร้าง Complex Synchronization Objects

เมื่อเธรดถูกหยุดการทำงาน พื้นที่จัดเก็บข้อมูลอาจไม่ได้ถูกเรียกคืน ขึ้นอยู่กับแอ็ตทริบิวต์ของเธรด เธรดดังกล่าวสามารถูกรวมโดยเธรดอื่น และส่งกลับข้อมูลไปที่เธรดนั้น เธรดที่ต้องการรวมกับเธรดอื่นถูกล็อก จนการเธรดเป้าหมายจะยุติการทำงาน กลไกการรวมนี้เป็นกรณี เฉพาะของการใช้ตัวแปรเงื่อนไข เงื่อนไขคือการสิ้นสุดการทำงานของเธรด

อ็อบเจ็กต์แอ็ตทริบิวต์เงื่อนไข

เหมือนกับเธรด และ mutexes ตัวแปรเงื่อนไขถูกสร้างด้วยความช่วยเหลือ ของอ็อบเจ็กต์แอ็ตทริบิวต์ *อ็อบเจ็กต์แอ็ตทริบิวต์เงื่อนไข* เป็นอ็อบเจ็กต์ abstract มีหนึ่งแอ็ตทริบิวต์เป็นอย่างมาก ขึ้นกับ การใช้ตัวเลือกของ POSIX ซึ่งถูกเข้าถึงผ่านตัวแปรชนิด **pthread_condattr_t** ใน AIX, ชนิดข้อมูล **pthread_condattr_t** เป็นตัวชี้; บนระบบอื่นอาจเป็น structure หรือชนิดข้อมูลอื่น

การสร้างและการทำลายอ็อบเจ็กต์แอ็ตทริบิวต์เงื่อนไข

อ็อบเจ็กต์แอ็ตทริบิวต์เงื่อนไข ถูกกำหนดเป็นค่าเริ่มต้น โดยรูทีนย่อย **pthread_condattr_init** แอ็ตทริบิวต์ถูกจัดการโดยรูทีนย่อย อ็อบเจ็กต์แอ็ตทริบิวต์ เธรดถูกทำลายโดยรูทีนย่อย **pthread_condattr_destroy** รูทีนย่อยนี้สามารถรีลีส พื้นที่จัดเก็บข้อมูลแบบไดนามิกที่จัดสรรโดยรูทีนย่อย **pthread_condattr_init** ขึ้นกับการนำไปปฏิบัติของไลบรารีเธรด

ในตัวอย่างดังต่อไปนี้ อ็อบเจ็กต์แอ็ตทริบิวต์ เงื่อนไขถูกสร้างและกำหนดด้วยค่าเริ่มต้น แล้วใช้และ ถูกทำลายในตอนสุดท้าย:

```
pthread_condattr_t attributes;
                /* the attributes object is created */
...
if (!pthread_condattr_init(&attributes)) {
                /* the attributes object is initialized */
        ...
                /* using the attributes object */
        ...
        pthread_condattr_destroy(&attributes);
                        /* the attributes object is destroyed */
}
```

อ็อบเจ็กต์แอ็ตทริบิวต์เดียวกันสามารถถูกใช้เพื่อสร้างหลาย ตัวแปรเงื่อนไข ซึ่งสามารถถูกแก้ไขระหว่างสองการสร้าง ตัวแปรเงื่อนไข เมื่อตัวแปรเงื่อนไขถูกสร้าง อ็อบเจ็กต์แอ็ตทริบิวต์สามารถถูกทำลายได้โดยไม่มีผลกับตัวแปรเงื่อนไขที่สร้าง

แอ็ตทริบิวต์เงื่อนไข

มีการสนับสนุนแอ็ตทริบิวต์เงื่อนไขดังต่อไปนี้:

`Process-shared`
ระบุการแบ่งใช้การประมวลผลของตัวแปรเงื่อนไข แอ็ตทริบิวต์นี้ขึ้นกับ ตัวเลือก POSIX การแบ่งใช้การประมวลผล

การสร้างและการทำลายตัวแปรเงื่อนไข

ตัวแปรเงื่อนไขถูกโดยการเรียกรูทีนย่อย **pthread_cond_init** คุณอาจระบุอ็อบเจ็กต์แอ็ตทริบิวต์ เงื่อนไข ถ้าคุณระบุตัวชี้ **NULL** ตัวแปรเงื่อนไขจะ มีแอ็ตทริบิวต์ค่าเริ่มต้น ดังนั้นส่วนของโค้ดต่อไปนี้:

```
pthread_cond_t cond;
pthread_condattr_t attr;
...
pthread_condattr_init(&attr);
pthread_cond_init(&cond, &attr);
pthread_condattr_destroy(&attr);
```

เหมือนกับ:

```
pthread_cond_t cond;
...
pthread_cond_init(&cond, NULL);
```

ID ของตัวแปรเงื่อนไขที่สร้างถูกส่งกลับไปที่เธรดที่เรียก ผ่านพารามิเตอร์ *เงื่อนไข* ID เงื่อนไขเป็นอ็อบเจ็กต์ opaque มีชนิดเป็น **pthread_cond_t** ใน AIX ชนิดข้อมูล **pthread_cond_t** เป็น structure; บนระบบอื่น อาจเป็นตัวชี้หรือชนิดข้อมูลอื่น

ตัวแปรเงื่อนไขต้องถูกสร้างครั้งเดียว หลีกเลี่ยงการเรียกรูทีนย่อย **pthread_cond_init** มากกว่าหนึ่งครั้งด้วยพารามิเตอร์ *condition* เดียวกัน (ตัวอย่างเช่น สองเธรดดำเนินการ กับโค้ดเหมือนกันพร้อมกัน) การตรวจสอบความไม่ซ้ำซ้อนของตัวแปรเงื่อนไขที่สร้างใหม่กระทำได้ใน วิธีดังต่อไปนี้:

- เรียกรูทีนย่อย **pthread_cond_init** ก่อน สร้างเธรดอื่นที่จะใช้ตัวแปรนี้ ตัวอย่างเช่นใน initial thread
- เรียกรูทีนย่อย **pthread_cond_init** ภายใน รูทีน one-time time initialization สำหรับข้อมูลเพิ่มเติม โปรดดูที่ One-Time Initializations
- ใช้ static condition ที่กำหนดค่าโดยแมโคร **PTHREAD_COND_INITIALIZER** static initialization; ตัวแปรเงื่อนไขจะมีแอ็ตทริบิวต์เริ่มต้น

หลังจากไม่จำเป็นต้องใช้ตัวแปรเงื่อนไขให้ ทำลายโดยเรียกรูทีนย่อย **pthread_cond_destroy** รูทีนย่อยนี้อาจเรียกคืนพื้นที่จัดเก็บข้อมูล ที่จัดสรรโดยรูทีนย่อย **pthread_cond_init** หลังจากทำลายตัวแปรเงื่อนไข **pthread_cond_t** เดิมสามารถถูกนำมาใช้ใหม่เพื่อสร้างเงื่อนไขอื่น ตัวอย่างเช่น ส่วนของโค้ด ต่อไปนี้ถูกต้อง แม้ว่าจะไม่นำมาใช้มากนัก:

```
pthread_cond_t cond;
...
for (i = 0; i < 10; i++) {

        /* creates a condition variable */
        pthread_cond_init(&cond, NULL);
```

```
        /* uses the condition variable */

        /* destroys the condition */
        pthread_cond_destroy(&cond);
}
```

เหมือนกับทรัพยากรระบบที่สามารถถูกแบ่งใช้ ระหว่างเธรดตัวแปรเงื่อนไขที่จัดสรรบนสแต็กของเธรดต้องถูกทำลาย ก่อนที่จะหยุดการทำงานเธรด ไลบรารีเธรดรักษา linked list ของตัวแปร เงื่อนไข ดังนั้น ถ้าสแต็กที่ mutex จัดสรรว่างลง list จะเสียหาย

การใช้ตัวแปรเงื่อนไข

ตัวแปรเงื่อนไขต้องถูกใช้ร่วมกันกับ mutex เสมอ ตัวแปรเงื่อนไขที่กำหนดมีหนึ่ง mutex เท่านั้นที่เชื่อมโยงกับ ตัวแปร แต่ mutex สามารถถูกใช้สำหรับตัวแปรเงื่อนไขได้มากกว่าหนึ่งตัวแปร เป็นไปได้ ที่จะบันเดิลลงในโครงสร้างของเงื่อนไข mutex และตัวแปรเงื่อนไข ตามที่แสดงในส่วนของโค้ดนี้:

```
struct condition_bundle_t {
        int             condition_predicate;
        pthread_mutex_t  condition_lock;
        pthread_cond_t   condition_variable;
};
```

การรอเงื่อนไข

การป้องกัน mutex เงื่อนไขต้องถูกล็อกก่อนการรอเงื่อนไข เธรดสามารถรอให้เงื่อนไขถูกส่งสัญญาณโดยเรียกรูทีนย่อย **pthread_cond_wait** หรือ **pthread_cond_timedwait** รูทีนย่อยปลดล็อก mutex อัตโนมัติและบล็อกเธรดที่เรียกจนกว่าเงื่อนไขจะส่งสัญญาณ เมื่อ ส่งกลับการเรียก mutex จะถูกล็อกอีกครั้ง

รูทีนย่อย **pthread_cond_wait** บล็อกเธรดอย่างไม่แน่นอน ถ้าไม่มีการส่งสัญญาณจากเงื่อนไข เธรด จะไม่พร้อมทำงาน เนื่องจากรูทีนย่อย **pthread_cond_wait** มีจุดการยกเลิก วิธีเดียวที่จะออกจาก deadlock นี้ได้คือยกเลิก เธรดที่บล็อก ถ้ามีการเปิดใช้งานความสามารถในการยกเลิก สำหรับข้อมูลเพิ่มเติม โปรดดูที่ การยกเลิกเธรด

รูทีนย่อย **pthread_cond_timedwait** บล็อก เธรดเฉพาะช่วงเวลาที่กำหนด รูทีนย่อยนี้มี พารามิเตอร์พิเศษ *timeout* ระบุวันที่สัมบูรณ์ซึ่งการ sleep ต้องสิ้นสุด พารามิเตอร์ *timeout* เป็น ตัวชี้ไปที่ **timespec** structure ชนิดข้อมูลนี้ถูกเรียก ว่า **timestruc_t** ซึ่งมีเขตข้อมูลดังต่อไปนี้:

`tv_sec`
: long unsigned integer, ระบุวินาที

`tv_nsec`
: long integer, ระบุนาโนวินาที

โดยทั่วไปรูทีนย่อย **pthread_cond_timedwait** ถูกใช้ในรูปแบบดังต่อไปนี้:

```
struct timespec timeout;
...
time(&timeout.tv_sec);
timeout.tv_sec += MAXIMUM_SLEEP_DURATION;
pthread_cond_timedwait(&cond, &mutex, &timeout);
```

พารามิเตอร์ *timeout* ระบุ วันที่สัมบูรณ์ บางส่วนของโค้ดก่อนหน้าแสดงวิธีระบุช่วงเวลา ที่ไม่ใช่วันที่สัมบูรณ์

เมื่อต้องการใช้รูทีนย่อย **pthread_cond_timedwait** กับวันที่สัมบูรณ์ คุณสามารถใช้รูทีนย่อย **mktime** เพื่อคำนวณค่าของเขตข้อมูล tv_sec ของ **timespec** structure ในตัวอย่างดังต่อไปนี้ เธรดรอเงื่อนไขจนถึง 08:00 January 1, 2001, เวลาท้องถิ่น:

```
struct tm       date;
time_t          seconds;
struct timespec timeout;
...

date.tm_sec = 0;
date.tm_min = 0;
date.tm_hour = 8;
date.tm_mday = 1;
date.tm_mon = 0;         /* the range is 0-11 */
date.tm_year = 101;      /* 0 is 1900 */
date.tm_wday = 1;        /* this field can be omitted -
                            but it will really be a Monday! */
date.tm_yday = 0;        /* first day of the year */
date.tm_isdst = daylight;
        /* daylight is an external variable - we are assuming
           that daylight savings time will still be used... */

seconds = mktime(&date);

timeout.tv_sec = (unsigned long)seconds;
timeout.tv_nsec = 0L;

pthread_cond_timedwait(&cond, &mutex, &timeout);
```

รูทีนย่อย **pthread_cond_timedwait** ยังมีจุดการยกเลิก แม้ว่า sleep จะกำหนดชัดเจน ดังนั้น เธรดที่มีการ sleep สามารถถูกยกเลิกได้ ไม่ว่า sleep จะมีการหมดเวลาใช้งานหรือไม่

การส่งสัญญาณเงื่อนไข

เงื่อนไขสามารถถูกส่งสัญญาณโดยเรียกรูทีนย่อย **pthread_cond_signal** หรือ **pthread_cond_broadcast**

รูทีนย่อย **pthread_cond_signal** พร้อมทำงานอย่างน้อย หนึ่งเธรดซึ่งถูกบล็อกอยู่ในปัจจุบันในเงื่อนไขที่ระบุ เธรดที่พร้อมใช้งาน ถูกเลือกตามนโยบายการกำหนดเวลา ซึ่งเป็นเธรด ที่มีลำดับความสำคัญของกำหนดเวลามากที่สุด (โปรดดูที่ การกำหนดเวลานโยบายและลำดับความสำคัญ) ซึ่งอาจเกิดขึ้นได้บนระบบมัลติโพรเซสเซอร์ หรือในบางระบบที่ไม่ใช่ AIX ที่มีเธรดที่พร้อมทำงานมากกว่าหนึ่งเธรด โปรดอย่าเข้าใจว่ารูทีนย่อยนี้ ทำให้เธรดพร้อมทำงานเพียงหนึ่งเธรด

รูทีนย่อย **pthread_cond_broadcast** ทำให้ซึ่ง ถูกบล็อกอยู่ในปัจจุบันทุกเธรดพร้อมทำงานในเงื่อนไขที่ระบุ อย่างไรก็ตาม เธรดสามารถเริ่มการรอบนเงื่อนไขเดียวกันหลังจาก การเรียกไปที่รูทีนย่อยมีการส่งกลับ

การเรียกไปที่รูทีนเหล่านี้สำเร็จเสมอ นอกจากว่า มีการระบุพารามิเตอร์ *cond* ที่ไม่ถูกต้อง ซึ่งไม่ได้หมายความว่า เธรดถูกทำให้พร้อมทำงานแล้ว นอกจากนี้ ไลบรารีไม่แนะนำให้มี การส่งสัญญาณไปที่เงื่อนไข ตัวอย่างเช่น พิจารณาเงื่อนไข C ไม่มีเธรดรอยู่ในเงื่อนไขนี้ ที่เวลา t, เธรด 1 ส่งสัญญานเงื่อนไข C การเรียกใช้เสร็จสิ้นแม้ว่าจะไม่มีเธรดถูกปลุกให้ตื่น ที่เวลา t+1, เธรด 2 เรียกรูทีนย่อย **pthread_cond_wait** โดยมี C เป็นพารามิเตอร์ *cond* เธรด 2 ถูกบล็อก ถ้าไม่มีเธรดอื่น ส่งสัญญาณ C, เธรด 2 อาจรอจนกระทั้งการประมวลผลหยุดการทำงาน

คุณสามารถหลีกเลี่ยง deadlock แบบนี้ได้โดยตรวจสอบโค้ดระบุความผิดพลาด **EBUSY** ที่ส่งกลับโดยรูทีนย่อย **pthread_cond_destroy** เมื่อมีการทำลายตัวแปรเงื่อนไข ตามบางส่วนของ โค้ดดังต่อไปนี้:

รูทีนย่อย **pthread_yield** ให้โอกาสแก่เธรดอื่นเพื่อให้ถูกจัดตารางเวลา ตัวอย่างเช่น หนึ่ง ในเธรดที่พร้อมใช้งานแล้ว สำหรับข้อมูลเพิ่มเติมเกี่ยวกับรูทีนย่อย **pthread_yield**

รูทีนย่อย **pthread_cond_wait** และ **pthread_cond_broadcast** ต้องไม่ถูกใช้ภายในตัวจัดการสัญญาณ เพื่อจัดเตรียมวิธีที่สะดวก สำหรับเธรดเพื่อรอสัญญาณ ไลบรารีเธรดได้จัดเตรียมรูทีนย่อย **sigwait** สำหรับข้อมูลเพิ่มเติม เกี่ยวกับรูทีนย่อย **sigwait** สำหรับข้อมูลเพิ่มเติมเกี่ยวกับรูทีนย่อย **sigwait** โปรดดูที่ การจัดการสัญญาณ

การซิงโครไนซ์เธรดกับตัวแปรเงื่อนไข

```
while (pthread_cond_destroy(&cond) == EBUSY) {
        pthread_cond_broadcast(&cond);
        pthread_yield();
}
```

ตัวแปรเงื่อนไขถูกใช้เพื่อรอจนกว่าเพรดิเคต เงื่อนไขเฉพาะจะมีค่าเป็น true เพรดิเคตเงื่อนไขถูกตั้งค่าโดยเธรดอื่น โดยทั่วไป คือเธรดที่ส่งสัญญาณเงื่อนไข

Condition wait semantics

เพรดิเคตเงื่อนไขต้องถูกป้องกันโดย mutex เมื่อรอเงื่อนไข รูทีนย่อยการรอ (รูทีนย่อย **pthread_cond_wait** หรือ **pthread_cond_timedwait**) จะปลดล็อก mutex และบล็อกเธรดแบบ atomic เมื่อเงื่อนไขถูกส่งสัญญาณ mutex จะถูกล็อกใหม่ และรอการส่งกลับของรูทีนย่อย สิ่งสำคัญคือ เมื่อการส่งกลับจากรูทีนย่อยไม่มีข้อผิดพลาด เพรดิเคต อาจยังคงมีค่าเป็น false

เหตุผลก็คืออาจมีเธรดมากกว่าหนึ่งเธรดถูกทำให้พร้อมใช้งาน: ไม่ว่าเธรดที่เรียกรูทีนย่อย **pthread_cond_broadcast** หรือเรซที่หลีกเลี่ยงไม่ได้ระหว่างสองโพรเซสเซอร์เรียกสองเธรดพร้อมกัน เธรดแรกล็อก mutex จะบล็อกเธรดที่ถูกทำให้พร้อมใช้งานอื่นทั้งหมดใน รูทีนย่อยการรอ จนกว่า mutex จะถูกปลดล็อกโดยโปรแกรม ดังนั้น เพรดิเคตอาจเปลี่ยนแปลง เมื่อเธรดที่สองได้รับ mutex และส่งกลับ จากรูทีนย่อยการรอ

โดยทั่วไป เมื่อใดก็ตามที่เงื่อนไขรอการส่งกลับ เธรดควรประเมินค่าเพรดิเคตซ้ำเพื่อกำหนดว่า สามารถดำเนินต่อได้อย่างปลอดภัย ควรรออีกครั้ง หรือควรแจ้งการหมดเวลา การส่งกลับจากรูทีนย่อย การรอ ไม่ได้แสดงว่าเพรดิเคตเป็น true หรือ false

ขอแนะนำว่าให้กำหนดการรอเงื่อนไข ใน "while loop" ที่ตรวจสอบเพรดิเคต การนำเงื่อนไขการรอ ไปใช้เบื้องต้นแสดงอยู่ในส่วนของโค้ดดังต่อไปนี้:

```
pthread_mutex_lock(&condition_lock);
while (condition_predicate == 0)
        pthread_cond_wait(&condition_variable, &condition_lock);
...
pthread_mutex_unlock(&condition_lock);
```

Timed wait semantics

เมื่อรูทีนย่อย **pthread_cond_timedwait** ส่งกลับข้อผิดพลาดหมดเวลาใช้งาน เพรดิเคตอาจมีค่าเป็น true เนื่องจาก เรซที่หลีกเลี่ยงไม่ได้อื่นระหว่างการหมดอายุของการหมดเวลาใช้งานและการเปลี่ยนแปลง สถานะของเพรดิเคต

สำหรับการรอที่ไม่ได้กำหนดเวลา เธรดควรถูกหาค่าเพรดิเคตซ้ำ เมื่อมีการหมดเวลาใช้งานเพื่อตรวจว่า ควรประกาศ การหมดเวลาใช้งานหรือควรดำเนินการต่อไป ขอแนะนำให้คุณตรวจสอบกรณีที่เป็นไปได้ ทั้งหมดอย่างระมัดระวัง เมื่อรูทีนย่อย **pthread_cond_timedwait** ส่งกลับ ส่วนของโค้ดดังต่อไปนี้แสดงวิธีการตรวจสอบควรดำเนินการอย่างไร ในโปรแกรมกันผิดพลาด:

```
int result = CONTINUE_LOOP;

pthread_mutex_lock(&condition_lock);
while (result == CONTINUE_LOOP) {
        switch (pthread_cond_timedwait(&condition_variable,
                &condition_lock, &timeout)) {

                case 0:
                if (condition_predicate)
                        result = PROCEED;
                break;

                case ETIMEDOUT:
                result = condition_predicate ? PROCEED : TIMEOUT;
                break;

                default:
                result = ERROR;
                break;
        }
}

...
pthread_mutex_unlock(&condition_lock);
```

ตัวแปร **result** สามารถถูกใช้เพื่อ เลือกการดำเนินการ คำสั่งก่อนการปลดล็อก mutex ควร ถูกดำเนินการเร็วที่สุดเท่าที่เป็นไปได้ เนื่องจาก mutex จะไม่ถูกเก็บไว้ เป็นเวลานาน

ระบุวันที่สัมบูรณ์ในพารามิเตอร์ *timeout* ทำให้การสร้างลักษณะการทำงาน real-time ง่าย การหมดเวลาใช้งาน สัมบูรณ์ไม่จำเป็นต้องถูกคำนวณซ้ำ ถ้าถูกใช้หลายครั้งในลูป เช่นอยู่ในเงื่อนไขรอ สำหรับกรณีที่นาฬิการะบบ เดินอย่างไม่ต่อเนื่องโดยตัวดำเนินการ การใช้การหมดเวลาใช้งานสัมบูรณ์ จะประกันว่าการกำหนดเวลารอจะจบทันทีที่เวลาระบบ ระบุวันที่หลังจากพารามิเตอร์ *timeout*

ตัวอย่างการใช้ตัวแปรเงื่อนไข

ตัวอย่างดังต่อไปนี้มีซอร์สโค้ดสำหรับ รูทีน synchronization point *synchronization point* คือจุดที่กำหนดในโปรแกรม ซึ่งเธรดที่ต่างกันต้องรอจนกว่า เธรดทั้งหมด (หรืออย่างน้อยเธรดจำนวนหนึ่ง) ได้มาถึงจุดนั้น

synchronization point สร้างได้โดยใช้ตัวนับ ซึ่งถูกป้องกันโดยล็อก และตัวแปรเงื่อนไข แต่ละเธรดใช้ล็อก เพิ่มค่าตัวนับ และรอจนเงื่อนไขถูกส่งสัญญาณ ถ้าตัวนับไม่ถึงค่าสูงสุด มิฉะนั้น เงื่อนไขจะถูกกระจายออกไป และเธรดทั้งหมดสามารถดำเนินการได้ เธรดสุดท้ายที่เรียกรูทีนกระจาย เงื่อนไข

```
#define SYNC_MAX_COUNT  10

void SynchronizationPoint()
{
        /* use static variables to ensure initialization */
```

```
        static mutex_t sync_lock = PTHREAD_MUTEX_INITIALIZER;
        static cond_t  sync_cond = PTHREAD_COND_INITIALIZER;
        static int sync_count = 0;

        /* lock the access to the count */
        pthread_mutex_lock(&sync_lock);

        /* increment the counter */
        sync_count++;

        /* check if we should wait or not */
        if (sync_count < SYNC_MAX_COUNT)

             /* wait for the others */
             pthread_cond_wait(&sync_cond, &sync_lock);

        else

            /* broadcast that everybody reached the point */
            pthread_cond_broadcast(&sync_cond);

        /* unlocks the mutex - otherwise only one thread
               will be able to return from the routine! */
        pthread_mutex_unlock(&sync_lock);
}
```

รูทีนนี้มีข้อจำกัดบางประการ: คือสามารถใช้ได้เพียงครั้งเดียว และจำนวนเธรด ที่จะเรียกรูทีนถูกโค้ดโดยค่าคงที่สัญลักษณ์ อย่างไรก็ตาม ตัวอย่างนี้แสดงการใช้พื้นฐานของตัวแปรเงื่อนไข สำหรับตัวอย่างการใช้งานที่ซับซ้อน สำหรับตัวอย่างการใช้งานที่ซับซ้อน โปรดดูที่ การสร้างอ็อบเจ็กต์การซิงโครไนซ์ แบบซับซ้อน

หลักการที่เกี่ยวข้อง:

"การรวมเธรด" ในหน้า 497

การรวมเธรดหมายถึงการรอให้เธรดจบการทำงาน ซึ่งอาจถูกมอบเป็นการใช้งานเฉพาะของตัวแปรเงื่อนไข

การใช้ล็อกอ่าน-เขียน

ในสถานการณ์ส่วนใหญ่ ข้อมูลจะถูกอ่านมากกว่าแก้ไขหรือเขียน

ในกรณีเหล่านี้ คุณสามารถอนุญาตให้เธรดอ่านข้อมูลแบบพร้อมกัน ขณะที่หยุดทำงานล็อก และอนุญาตให้มีเพียงเธรดเดียวที่พักล็อกได้ เมื่อแก้ไขข้อมูล ล็อกตัวอ่านจำนวนมาก ล็อกตัวเขียนเดี่ยว (หรือล็อกการอ่าน-เขียน) จะทำสิ่งนี้ ล็อกการอ่าน-เขียน ได้มาเพื่ออ่านหรือเขียน อย่างใดอย่างหนึ่ง จากนั้นจึงปล่อยออก เธรดที่ได้ล็อกการอ่าน-เขียน ต้องเป็นเธรดที่ปล่อยล็อกนั้น

อ็อบเจ็กต์แอ็ตทริบิวต์อ่าน-เขียน

รูทีนย่อย **pthread_rwlockattr_init** จะกำหนดค่าเริ่มต้นอ็อบเจ็กต์แอ็ตทริบิวต์สำหรับล็อกการอ่าน-เขียน (**attr**) ค่าดีฟอลต์สำหรับแอ็ตทริบิวต์ทั้งหมด จะถูกกำหนดด้วยการนำไปปฏิบัติ ผลลัพธ์ที่ไม่ได้คาดการณ์ไว้ สามารถเกิดขึ้นได้ ถ้ารูทีนย่อย **pthread_rwlockattr_init** ระบุอ็อบเจ็กต์แอ็ตทริบิวต์ล็อกสำหรับการอ่าน-เขียนที่ถูกกำหนดค่าเริ่มต้นแล้ว

ตัวอย่างต่อไปนี้แสดงให้เห็นภาพของวิธีการเรียกรูทีนย่อย **pthread_rwlockattr_init** ด้วยอ็อบเจ็กต์ **attr** :

```
pthread_rwlockattr_t    attr;
```

and:

```
pthread_rwlockattr_init(&attr);
```

หลังจากที่อ็อบเจ็กต์แอ็ตทริบิวต์สำหรับการอ่าน-เขียนได้ถูกนำมาใช้เพื่อกำหนดค่าเริ่มต้น สำหรับล็อกการอ่าน-เขียนตั้งแต่หนึ่งค่าขึ้นไป ฟังก์ชันใดๆ ที่มีผลกับอ็อบเจ็กต์แอ็ตทริบิวต์ (ซึ่งรวมถึงการทำลาย) ไม่ได้มีผลกับล็อกสำหรับการอ่าน-เขียนที่กำหนดค่าเริ่มต้นไว้ก่อนหน้า

รูทีนย่อย **pthread_rwlockattr_destroy** จะทำลายอ็อบเจ็กต์แอ็ตทริบิวต์สำหรับล็อกการอ่าน-เขียน ผลลัพธ์ที่ไม่ได้คาดการณ์ไว้อาจเกิดขึ้นได้ ถ้าอ็อบเจ็กต์ถูกนำมาใช้ก่อนที่จะกำหนดค่าเริ่มต้นใหม่ด้วยการเรียกรูทีนย่อย **pthread_rwlockattr_init** อื่น การนำไปปฏิบัติสามารถเป็นสาเหตุที่ทำให้รูทีนย่อย **pthread_rwlockattr_destroy** ตั้งค่าอ็อบเจ็กต์ ที่อ้างอิงโดยอ็อบเจ็กต์ **attr** ให้เป็นค่าที่ไม่ถูกต้อง

การสร้างและการทำลายล็อกการอ่าน-เขียน

รูทีนย่อย **pthread_rwlock_init** จะกำหนดค่าเริ่มต้นสำหรับ ล็อกการอ่าน-เขียนที่อ้างอิงโดยอ็อบเจ็กต์ **rwlock** ด้วยแอ็ตทริบิวต์ที่อ้างอิงโดยอ็อบเจ็กต์ **attr** ถ้าอ็อบเจ็กต์ **attr** มีค่า NULL แอ็ตทริบิวต์ล็อกการอ่าน-เขียน จะถูกนำมาใช้ ผลกระทบจะเหมือนกับการส่งแอดเดรสของอ็อบเจ็กต์แอ็ตทริบิวต์ล็อกการอ่าน-เขียน ที่เป็นค่าดีฟอลต์ สำหรับการกำหนดค่าเริ่มต้นที่เป็นผลสำเร็จ สถานะของล็อกการอ่าน-เขียน จะกลายเป็นการกำหนดค่าเริ่มต้นและปลดล็อก หลังจากที่กำหนดค่าเริ่มต้นแล้ว ล็อกสามารถใช้จำนวนของเวลาใดๆ ที่ไม่ได้ถูกกำหนดค่าเริ่มต้น ผลลัพธ์ที่ไม่ได้คาดการณ์ไว้ สามารถเกิดขึ้นได้ ถ้าการเรียกรูทีนย่อย **pthread_rwlock_init** จะถูกเรียกโดยระบุล็อกการอ่าน-เขียนที่ได้ถูกกำหนดค่าเริ่มต้นแล้ว หรือถ้าล็อกการอ่าน-เขียน ถูกใช้โดยไม่ได้ถูกกำหนดค่าเริ่มต้นไว้ในครั้งแรก

ถ้ารูทีนย่อย **pthread_rwlock_init** ถูกเรียก อ็อบเจ็กต์ **rwlock** จะไม่ถูกกำหนดค่าเริ่มต้น และเนื้อหาจะถูกถอนการกำหนดออก

รูทีนย่อย **pthread_rwlock_destroy** จะทำลายอ็อบเจ็กต์ล็อกการอ่าน-เขียน ที่อ้างถึงโดยอ็อบเจ็กต์ **rwlock** และรีลีสรีซอร์สใดๆ โดยใช้ล็อก ผลลัพธ์ที่ไม่ได้คาดการณ์ไว้สามารถเกิดขึ้นได้ในสถานการณ์ใดๆต่อไปนี้ :

- ถ้าล็อกไม่ได้ใช้ก่อนที่จะกำหนดค่าเริ่มต้นใหม่โดยการเรียกรูทีนย่อย **pthread_rwlock_init** อื่น
- การนำไปปฏิบัติสามารถเป็นสาเหตุทำให้รูทีนย่อย **pthread_rwlock_destroy** ตั้งค่าอ็อบเจ็กต์ที่อ้างอิงโดยอ็อบเจ็กต์ **rwlock** ให้เป็นค่าที่ไม่ถูกต้อง ผลลัพธ์ที่ไม่คาดการณ์ไว้สามารถเกิดขึ้นได้ถ้า **pthread_rwlock_destroy** ถูกเรียกเมื่อเธรดใดๆ พักอ็อบเจ็กต์ **rwlock** ไว้
- ความพยายามในการทำลายล็อกอ่าน-เขียนที่ไม่ได้กำหนดค่าเริ่มต้นจะส่งผลทำให้เกิดผลลัพธ์ที่ไม่คาดคิดไว้ อ็อบเจ็กต์ล็อกการอ่าน-เขียนจะถูกทำลายถ้าสามารถกำหนดค่าเริ่มต้นขึ้นใหม่ได้โดยใช้รูทีนย่อย **pthread_rwlock_init** ผลลัพธ์ที่ไม่ได้คาดการณ์ไว้สามารถเกิดขึ้นได้ ถ้าอ็อบเจ็กต์ล็อกการอ่าน-เขียนถูกอ้างอิงหลังจากที่ได้ถูกทำลายแล้ว

ในกรณีที่แอ็ตทริบิวต์ล็อกการอ่าน-เขียนตามค่าดีฟอลต์มีความเหมาะสม ให้ใช้แมโคร **PTHREAD_RWLOCK_INITIALIZER** เพื่อกำหนดค่าเริ่มต้นล็อกการอ่าน-เขียน ที่ถูกจัดสรรไว้ในเชิงสถิติ ตัวอย่างเช่น:

```
pthread_rwlock_t        rwlock1 = PTHREAD_RWLOCK_INITIALIZER;
```

ผลกระทบจะคล้ายคลึงกับการกำหนดค่าเริ่มต้นแบบไดนามิกโดยใช้การเรียกรูทีนย่อย **pthread_rwlock_init** ด้วยพารามิเตอร์ *attr* ที่ระบุเป็น NULL ยกเว้นว่า ไม่มีการตรวจสอบข้อผิดพลาดที่ถูกดำเนินการไว้ ตัวอย่างเช่น:

```
pthread_rwlock_init(&rwlock2, NULL);
```

ตัวอย่างต่อไปนี้จะแสดงให้เห็นภาพของวิธีการใช้รูทีนย่อย **pthread_rwlock_init** ด้วยพารามิเตอร์ attr ที่ถูกกำหนดค่าเริ่มต้นไว้ สำหรับตัวอย่างวิธีกำหนดค่าเริ่มต้น สำหรับพารามิเตอร์ *attr* โปรดดูที่ Read-Write Attributes Object

```
pthread_rwlock_init(&rwlock, &attr);
```

การล็อกอ็อบเจ็กต์ล็อกอ่าน เขียนสำหรับการอ่าน

รูทีนย่อย **pthread_rwlock_rdlock** ใช้ล็อกการอ่าน กับล็อกการอ่าน-เขียนที่อ้างถึงโดยอ็อบเจ็กต์ **rwlock** การเรียกเธรดจะได้รับล็อกการอ่าน ถ้าตัวเขียนไม่ได้พักล็อกไว้ และไม่มีตัวเขียนที่บล็อกการล็อก ซึ่งไม่ได้ระบุว่า การเรียกเธรดได้รับล็อก เมื่อตัวเขียนไม่ได้พักล็อกไว้ และมีตัวเขียนที่รอสำหรับล็อกอยู่ ถ้าตัวเขียนพักล็อกไว้ การเรียกเธรดจะไม่ได้รับล็อกการอ่าน ถ้าล็อกการอ่านไม่ได้รับ การเรียกเธรดไม่ได้ส่งคืนจากการเรียก **pthread_rwlock_rdlock** จนกว่าจะสามารถรับล็อกได้ ผลลัพธ์จะถูกยกเลิกถ้าการเรียกเธรดพักล็อกการเขียนบนอ็อบเจ็กต์ **rwlock** ในเวลาที่เรียก

เธรดอาจเก็บล็อกการอ่านที่พร้อมกันจำนวนมากบนอ็อบเจ็กต์ **rwlock** (นั้นคือ การเรียกใช้รูทีนย่อย **pthread_rwlock_rdlock** จำนวน *n* ครั้ง) ถ้าเป็นเช่นนั้น เธรดต้องดำเนินการจับคู่การปลดล็อก (นั้นคือ ต้องเรียกใช้รูทีนย่อย **pthread_rwlock_unlock** จำนวน *n* ครั้ง)

รูทีนย่อย **pthread_rwlock_tryrdlock** ใช้ล็อกการอ่าน ที่คล้ายคลึงกับรูทีนย่อย **pthread_rwlock_rdlock** ด้วยข้อยกเว้นที่รูทีนย่อยเกิดความล้มเหลว ถ้าเธรดใดๆ พักล็อกการเขียนบนอ็อบเจ็กต์ **rwlock** หรือมีตัวเขียนที่บล็อกอยู่บนอ็อบเจ็กต์ **rwlock** ผลลัพธ์จะถอดการกำหนดออก ถ้าฟังก์ชันเหล่านี้ใดๆ ถูกเรียกด้วยล็อกการอ่าน-เขียนที่ไม่ได้กำหนดค่าเริ่มต้นไว้

ถ้าสัญลักษณ์ถูกส่งไปยังเธรดที่รอสำหรับล็อกการอ่าน-เขียนสำหรับการอ่าน หากส่งคืนจาก handler สัญญาณ เธรดจะกลับสู่การรอสำหรับล็อกการอ่าน-เขียน สำหรับการเขียน หากไม่ได้ถูกอินเตอร์รัปต์

การล็อกอ็อบเจ็กต์ล็อกการอ่าน-เขียนสำหรับการเขียน

รูทีนย่อย **pthread_rwlock_wrlock** ใช้ล็อกการเขียน กับล็อกการอ่าน-เขียนที่อ้างอิงโดยอ็อบเจ็กต์ **rwlock** การเรียกเธรดได้รับล็อกการเขียนถ้าไม่มีเธรดอื่น (ตัวอ่านหรือตัวเขียน) พักล็อกการอ่าน-เขียนบนอ็อบเจ็กต์ **rwlock** มิฉะนั้น เธรดไม่ได้ส่งคืนจากการเรียก **pthread_rwlock_wrlock** จนกว่าจะสามารถได้รับล็อก ผลลัพธ์คือถอนการกำหนด ถ้าการเรียกเธรดพักล็อกการอ่าน-เขียน (ไม่ว่าจะล็อกการอ่านหรือเขียน) ในเวลาที่เรียก

รูทีนย่อย **pthread_rwlock_trywrlock** จะใช้ล็อกการเขียนที่คล้ายกับรูทีนย่อย **pthread_rwlock_wrlock** ด้วยข้อยกเว้นที่ฟังก์ชันเกิดความล้มเหลว ถ้าเธรดใดๆ พัก **rwlock** ไว้สำหรับการอ่านหรือการเขียนในปัจจุบัน ผลลัพธ์จะถอดการกำหนดออก ถ้าฟังก์ชันเหล่านี้ใดๆ ถูกเรียกด้วยล็อกการอ่าน-เขียนที่ไม่ได้กำหนดค่าเริ่มต้นไว้

ถ้าสัญญาณถูกส่งไปยังเธรดที่รอสำหรับล็อกการอ่าน-เขียนสำหรับการเขียน หากส่งคืนจาก handler สัญญาณ เธรดจะกลับสู่การรอสำหรับล็อกการอ่าน-เขียน สำหรับการเขียนหากไม่ได้ถูกอินเตอร์รัปต์

ตัวอย่างโปรแกรมล็อกการอ่าน-เขียน

ตัวอย่างโปรแกรมต่อไปนี้จะสาธิตวิธีการใช้รูทีนย่อยการล็อก หากต้องการรันโปรแกรมเหล่านี้ คุณต้องการไฟล์ **check.h** และ **makefile**

check.h file:

```
#include stdio.h
#include stdio.h
#include stdio.h
#include stdio.h

/* Simple function to check the return code and exit the program
   if the function call failed
   */
static void compResults(char *string, int rc) {
  if (rc) {
    printf("Error on : %s, rc=%d",
           string, rc);
    exit(EXIT_FAILURE);
  }
  return;
}
```

ไฟล์ Make:

```
CC_R = xlc_r

TARGETS = test01 test02 test03

OBJS = test01.o test02.o test03.o

SRCS = $(OBJS:.o=.c)

$(TARGETS): $(OBJS)
    $(CC_R) -o $@ $@.o

clean:
    rm $(OBJS) $(TARGETS)

run:
    test01
    test02
    test03
```

ตัวอย่างเทรดเดี่ยว

ตัวอย่างต่อไปนี้ใช้รูทีนย่อย pthread_rwlock_tryrdlock ด้วยเธรดเดี่ยว สำหรับตัวอย่างของการใช้รูทีนย่อย pthread_rwlock_tryrdlock ที่มีหลายเธรด โปรดดูที่ ตัวอย่าง หลายเธรด

Example: test01.c

```
#define _MULTI_THREADED
#include pthread.h
#include stdio.h
#include "check.h"

pthread_rwlock_t       rwlock = PTHREAD_RWLOCK_INITIALIZER;

void *rdlockThread(void *arg)
{
  int rc;
```

```
  int             count=0;

  printf("Entered thread, getting read lock with mp wait\n");
  Retry:
  rc = pthread_rwlock_tryrdlock(&rwlock);
  if (rc == EBUSY) {
    if (count >= 10) {
      printf("Retried too many times, failure!\n");

      exit(EXIT_FAILURE);
    }
    ++count;
    printf("Could not get lock, do other work, then RETRY...\n");
    sleep(1);
    goto Retry;
  }
  compResults("pthread_rwlock_tryrdlock() 1\n", rc);

  sleep(2);

  printf("unlock the read lock\n");
  rc = pthread_rwlock_unlock(&rwlock);
  compResults("pthread_rwlock_unlock()\n", rc);

  printf("Secondary thread complete\n");
  return NULL;
}

int main(int argc, char **argv)
{
  int                   rc=0;
  pthread_t thread;

  printf("Enter test case - %s\n", argv[0]);

  printf("Main, get the write lock\n");
  rc = pthread_rwlock_wrlock(&rwlock);
  compResults("pthread_rwlock_wrlock()\n", rc);

  printf("Main, create the try read lock thread\n");
  rc = pthread_create(&thread, NULL, rdlockThread, NULL);
  compResults("pthread_create\n", rc);

  printf("Main, wait a bit holding the write lock\n");
  sleep(5);

  printf("Main, Now unlock the write lock\n");
  rc = pthread_rwlock_unlock(&rwlock);
  compResults("pthread_rwlock_unlock()\n", rc);

  printf("Main, wait for the thread to end\n");
  rc = pthread_join(thread, NULL);
  compResults("pthread_join\n", rc);

  rc = pthread_rwlock_destroy(&rwlock);
```

```
  compResults("pthread_rwlock_destroy()\n", rc);
  printf("Main completed\n");
  return 0;
}
```

เอาต์พุตสำหรับตัวอย่างโปรแกรมนี้จะคล้ายคลึงกับที่แสดงต่อไปนี้:

```
Enter test case - ./test01
Main, get the write lock
Main, create the try read lock thread
Main, wait a bit holding the write lock

Entered thread, getting read lock with mp wait
Could not get lock, do other work, then RETRY...
Could not get lock, do other work, then RETRY...
Could not get lock, do other work, then RETRY...
Could not get lock, do other work, then RETRY...
Could not get lock, do other work, then RETRY...
Main, Now unlock the write lock
Main, wait for the thread to end
unlock the read lock
Secondary thread complete
Main completed
```

ตัวอย่าง Multiple-thread

ตัวอย่างต่อไปนี้จะใช้รูทีนย่อย **pthread_rwlock_tryrdlock** ที่มีเธรดจำนวนมาก สำหรับตัวอย่างของการใช้รูทีนย่อย **pthread_rwlock_tryrdlock** ที่มีเธรดเดียว โปรดดูที่ ตัวอย่าง เธรดเดี่ยว

```
Example: test02.c

#define _MULTI_THREADED
#include pthread.h
#include stdio.h
#include "check.h"

pthread_rwlock_t       rwlock = PTHREAD_RWLOCK_INITIALIZER;

void *wrlockThread(void *arg)
{
  int rc;
  int             count=0;

  printf("%.8x: Entered thread, getting write lock\n",
         pthread_self());
  Retry:
  rc = pthread_rwlock_trywrlock(&rwlock);
  if (rc == EBUSY) {
    if (count >= 10) {
      printf("%.8x: Retried too many times, failure!\n",
             pthread_self());
      exit(EXIT_FAILURE);
    }

    ++count;
```

```
    printf("%.8x: Go off an do other work, then RETRY...\n",
           pthread_self());
    sleep(1);
    goto Retry;
  }
  compResults("pthread_rwlock_trywrlock() 1\n", rc);
  printf("%.8x: Got the write lock\n", pthread_self());

  sleep(2);

  printf("%.8x: Unlock the write lock\n",
         pthread_self());
  rc = pthread_rwlock_unlock(&rwlock);
  compResults("pthread_rwlock_unlock()\n", rc);

  printf("%.8x: Secondary thread complete\n",
         pthread_self());
  return NULL;
}

int main(int argc, char **argv)
{
  int                   rc=0;
  pthread_t             thread, thread2;

  printf("Enter test case - %s\n", argv[0]);

  printf("Main, get the write lock\n");
  rc = pthread_rwlock_wrlock(&rwlock);
  compResults("pthread_rwlock_wrlock()\n", rc);

  printf("Main, create the timed write lock threads\n");
  rc = pthread_create(&thread, NULL, wrlockThread, NULL);
  compResults("pthread_create\n", rc);

  rc = pthread_create(&thread2, NULL, wrlockThread, NULL);
  compResults("pthread_create\n", rc);

  printf("Main, wait a bit holding this write lock\n");
  sleep(1);

  printf("Main, Now unlock the write lock\n");
  rc = pthread_rwlock_unlock(&rwlock);
  compResults("pthread_rwlock_unlock()\n", rc);

  printf("Main, wait for the threads to end\n");
  rc = pthread_join(thread, NULL);
  compResults("pthread_join\n", rc);

  rc = pthread_join(thread2, NULL);
  compResults("pthread_join\n", rc);

  rc = pthread_rwlock_destroy(&rwlock);
```

```
  compResults("pthread_rwlock_destroy()\n", rc);
  printf("Main completed\n");
  return 0;
}
```

เอาต์พุตสำหรับตัวอย่างโปรแกรมนี้จะคล้ายคลึงกับที่แสดงต่อไปนี้:

```
Enter test case - ./test02
Main, get the write lock
Main, create the timed write lock threads
Main, wait a bit holding this write lock
00000102: Entered thread, getting write lock
00000102: Go off an do other work, then RETRY...
00000203: Entered thread, getting write lock
00000203: Go off an do other work, then RETRY...
Main, Now unlock the write lock
Main, wait for the threads to end
00000102: Got the write lock
00000203: Go off an do other work, then RETRY...
00000203: Go off an do other work, then RETRY...
00000102: Unlock the write lock
00000102: Secondary thread complete
00000203: Got the write lock
00000203: Unlock the write lock
00000203: Secondary thread complete
Main completed
```

ตัวอย่าง Read-write read-lock

ตัวอย่างต่อไปนี้จะใช้รูทีนย่อย **pthread_rwlock_rdlock** เพื่อนำล็อกการอ่าน อ่าน-เขียนมาใช้:

ตัวอย่าง: test03.c

```
#define _MULTI_THREADED
#include pthread.h
#include stdio.h
#include "check.h"

pthread_rwlock_t       rwlock;

void *rdlockThread(void *arg)
{
  int rc;

  printf("Entered thread, getting read lock\n");
  rc = pthread_rwlock_rdlock(&rwlock);
  compResults("pthread_rwlock_rdlock()\n", rc);
  printf("got the rwlock read lock\n");

  sleep(5);

  printf("unlock the read lock\n");
  rc = pthread_rwlock_unlock(&rwlock);
  compResults("pthread_rwlock_unlock()\n", rc);
  printf("Secondary thread unlocked\n");
```

```
  return NULL;
}

void *wrlockThread(void *arg)
{
  int rc;

  printf("Entered thread, getting write lock\n");
  rc = pthread_rwlock_wrlock(&rwlock);
  compResults("pthread_rwlock_wrlock()\n", rc);

  printf("Got the rwlock write lock, now unlock\n");
  rc = pthread_rwlock_unlock(&rwlock);
  compResults("pthread_rwlock_unlock()\n", rc);
  printf("Secondary thread unlocked\n");
  return NULL;
}



int main(int argc, char **argv)
{
  int                   rc=0;
  pthread_t             thread, thread1;

  printf("Enter test case - %s\n", argv[0]);

  printf("Main, initialize the read write lock\n");
  rc = pthread_rwlock_init(&rwlock, NULL);
  compResults("pthread_rwlock_init()\n", rc);

  printf("Main, grab a read lock\n");
  rc = pthread_rwlock_rdlock(&rwlock);
  compResults("pthread_rwlock_rdlock()\n",rc);

  printf("Main, grab the same read lock again\n");
  rc = pthread_rwlock_rdlock(&rwlock);
  compResults("pthread_rwlock_rdlock() second\n", rc);

  printf("Main, create the read lock thread\n");
  rc = pthread_create(&thread, NULL, rdlockThread, NULL);
  compResults("pthread_create\n", rc);

  printf("Main - unlock the first read lock\n");
  rc = pthread_rwlock_unlock(&rwlock);
  compResults("pthread_rwlock_unlock()\n", rc);

  printf("Main, create the write lock thread\n");
  rc = pthread_create(&thread1, NULL, wrlockThread, NULL);
  compResults("pthread_create\n", rc);

  sleep(5);
  printf("Main - unlock the second read lock\n");
  rc = pthread_rwlock_unlock(&rwlock);
  compResults("pthread_rwlock_unlock()\n", rc);
```

```
  printf("Main, wait for the threads\n");
  rc = pthread_join(thread, NULL);
  compResults("pthread_join\n", rc);

  rc = pthread_join(thread1, NULL);
  compResults("pthread_join\n", rc);

  rc = pthread_rwlock_destroy(&rwlock);
  compResults("pthread_rwlock_destroy()\n", rc);

  printf("Main completed\n");
  return 0;
}
```

เอาต์พุตสำหรับตัวอย่างโปรแกรมนี้จะคล้ายคลึงกับที่แสดงต่อไปนี้:

```
$ ./test03
Enter test case - ./test03
Main, initialize the read write lock
Main, grab a read lock
Main, grab the same read lock again
Main, create the read lock thread
Main - unlock the first read lock
Main, create the write lock thread
Entered thread, getting read lock
got the rwlock read lock
Entered thread, getting write lock
Main - unlock the second read lock
Main, wait for the threads
unlock the read lock
Secondary thread unlocked
Got the rwlock write lock, now unlock
Secondary thread unlocked
Main completed
```

การรวมเธรด

การรวมเธรดหมายถึงการรอให้เธรดจบการทำงาน ซึ่งอาจถูกมอบเป็นการใช้งานเฉพาะของตัวแปรเงื่อนไข

การรอเธรด

การใช้รูทีนย่อย **pthread_join** อนุญาตให้เธรดรออีกเธรดหนึ่ง จบการทำงาน เงื่อนไขที่ซับซ้อนเพิ่มมากขึ้น เช่นการรอให้หลายๆ เธรด จบการทำงาน อาจถูกนำไปใช้ได้โดยการใช้ตัวแปรเงื่อนไข

การเรียกใช้รูทีนย่อย pthread_join

รูทีนย่อย **pthread_join** จะบล็อก การเรียกใช้เธรดไว้จนเธรดที่ระบุจะจบการทำงาน เธรดเป้าหมาย (เธรดที่ถูกรอให้จบการทำงาน) ต้องไม่ถูกแยกออก ถ้าเธรด เป้าหมายจบการทำงานแล้ว แต่ยังไม่ถูกแยกออก รูทีนย่อย **pthread_join** จะส่งกลับค่าในทันที หลังจากเธรดเป้าหมายถูกรวมเข้าแล้ว จะถูกแยกออกโดยอัตโนมัติ และสื่อบันทึกสามารถเรียกกลับคืนมาไว้ใช้ได้

ตารางต่อไปนี้บ่งชี้กรณีที่เป็นไปได้เมื่อ เธรดเรียกใช้รูทีนย่อย **pthread_join** ทั้งนี้ขึ้นอยู่กับ แอ็ตทริบิวต์ **state** และ **detachstate** ของเธรดเป้าหมาย

สภาวะปลายทาง	ปลายทางที่ไม่เชื่อมต่อ	ปลายทางที่เชื่อมต่อ
เป้าหมายที่ยังคงรันอยู่	ผู้เรียกจะถูกบล็อกไว้จนกระทั่งเป้าหมายจบการทำงาน	การเรียกจะส่งค่ากลับทันที โดยระบุข้อผิดพลาด
เป้าหมายจบการทำงาน	การเรียกจะส่งค่ากลับทันที โดยระบุการดำเนินงานสำเร็จเรียบร้อย	

การรวมหมาบเธรด

หลายๆ เธรดสามารถรวมกับเธรดเป้าหมายเดียวกันได้ ถ้า เป้าหมายไม่ถูกแยกออก ความสำเร็จของการดำเนินการนี้ขึ้นอยู่กับลำดับ การเรียกใช้รูทีนย่อย **pthread_join** และ ชั่วขณะที่เธรดเป้าหมายจบการทำงาน

- การเรียกใช้ใดๆ ไปยังรูทีนย่อย **pthread_join** เกิดขึ้นก่อนการจบการทงานของเธรดเป้าหมายจะบล็อกการเธรดการเรียกใช้
- เมื่อเธรดเป้าหมายจบการทำงาน เธรดที่ถูกบล็อกทั้งหมด จะถูกปลุกและเธรดเป้าหมายจะถูกแยกออกโดยอัตโนมัติ
- การเรียกใช้ใดๆ ไปยังรูทีนย่อย **pthread_join** ที่เกิดขึ้นก่อนการจบการทำงานของเธรดเป้าหมายจะล้มเหลว เนื่องจากเธรดถูกแยกออกโดยการรวมก่อนหน้า
- ถ้าไม่มีเธรดถูกเรียกใช้ไปยังรูทีนย่อย **pthread_join** ก่อนการจบการทำงานของเธรดเป้าหมาย การเรียกใช้แรก ไปยังรูทีนย่อย **pthread_join** จะส่งค่ากลับทันที ในระบุว่าการดำเนินการสำเร็จเรียบร้อย และการดำเนินการอื่นๆ จะล้มเหลว

ตัวอย่างการรวม

ในตัวอย่างต่อไปนี้ โปรแกรมสิ้นสุดหลัง ข้อความห้าข้อความแสดงในแต่ละภาษา นี้กระทำโดยการบล็อกเธรด เริ่มต้นจนกระทั่งเธรด ″writer″ ออกจากการทำงาน

```
#include <pthread.h>    /* include file for pthreads - the 1st */
#include <stdio.h>      /* include file for printf()           */

void *Thread(void *string)
{
        int i;

        /* writes five messages and exits */
        for (i=0; i<5; i++)
                printf("%s\n", (char *)string);
        pthread_exit(NULL);
}

int main()
{
        char *e_str = "Hello!";
        char *f_str = "Bonjour !";

        pthread_attr_t attr;
        pthread_t e_th;
        pthread_t f_th;

        int rc;
```

```
        /* creates the right attribute */
        pthread_attr_init(&attr);
        pthread_attr_setdetachstate(&attr,
                PTHREAD_CREATE_UNDETACHED);

        /* creates both threads */
        rc = pthread_create(&e_th, &attr, Thread, (void *)e_str);
        if (rc)
                exit(-1);
        rc = pthread_create(&f_th, &attr, Thread, (void *)f_str);
        if (rc)
                exit(-1);
        pthread_attr_destroy(&attr);

        /* joins the threads */
        pthread_join(e_th, NULL);
        pthread_join(f_th, NULL);

        pthread_exit(NULL);
}
```

เธรดไม่สามารถรวมกับตัวเองเนื่องจากจะเกิด deadlock และถูกตรวจพบโดยไลบรารี อย่างไรก็ตาม เธรดสองเธรดสามารถลองรวม กันได้ โดยจะเกิด deadlock แต่สถานการณ์นี้จะไม่ถูกตรวจพบโดย ไลบรารี

การส่งคืนข้อมูลจากเธรด

รูทีนย่อย **pthread_join** ยัง อนุญาตให้เธรดส่งกลับข้อมูลไปยังเธรดอื่น เมื่อเธรดเรียกใช้รูทีนย่อย **pthread_exit** หรือเมื่อส่งคืนจากรูทีน entry-point จะส่งคืนตัวชี้ (โปรดดูที่ การออกจากเธรด) ตัวชี้นี้ยังคงถูกเก็บไว้ตราบใดที่เธรดยังไม่ถูกแยกออก และรูทีนย่อย **pthread_join** สามารถส่งค่ากลับได้

ตัวอย่างเช่น คำสั่ง **grep** แบบมัลติเธรดอาจเลือก การประยุกต์ใช้ในตัวอย่างต่อไปนี้ ในตัวอย่างนี้ เธรดเริ่มต้นสร้างหนึ่งเธรดสำหรับหนึ่งไฟล์เพื่อสแกน แต่ละเธรด มีรูทีน entry point เหมือนกัน เธรดเริ่มต้นอาจรอให้ เธรดทั้งหมดจบการทำงาน แต่ละเธรด "scanning" จะเก็บบรรทัดที่พบใน บัฟเฟอร์ที่จัดสรรแบบไดนามิก และส่งกลับค่าตัวชี้ไปยังบัฟเฟอร์นี้ เธรด เริ่มต้นพิมพ์ค่าแต่ละบัฟเฟอร์ และรีลีสบัฟเฟอร์นั้น

```
/* "scanning" thread */
...
buffer = malloc(...);
        /* finds the search pattern in the file
           and stores the lines in the buffer   */
return (buffer);

/* initial thread */
...
for (/* each created thread */) {
        void *buf;
        pthread_join(thread, &buf);
        if (buf != NULL) {
                /* print all the lines in the buffer,
                        preceded by the filename of the thread */
```

```
                free(buf);
        }
}
...
```

ถ้าเธรดเป้าหมายถูกยกเลิก รูทีนย่อย **pthread_join** จะส่งคืนค่า -1 ซึ่งเป็นตัวชี้ (โปรดดูที่ การยกเลิกเธรด) เนื่องจาก -1 ไม่สามารถเป็นค่าตัวชี้ได้ การได้รับค่า -1 เป็นค่าตัวชี้ที่ส่งกลับจาก เธรดหมายความว่าเธรดถูกยกเลิก

ตัวชี้ที่ส่งกลับสามารถชี้ไปยังข้อมูลประเภทใดๆ ก็ได้ ตัวชี้ต้องยังคง ใช้ได้หลังจากเธรดจบการทำงาน และสื่อบันทึกถูกเรียกกลับคืน ดังนั้น หลีกเลี่ยงการส่งกลับค่า เนื่องจากรูทีน destructor ถูกเรียกใช้เมื่อสื่อบันทึกของเธรด ถูกเรียกกลับคืน

การส่งกลับตัวชี้ที่ชี้ไปยังหน่วยเก็บที่จัดสรรแบบไดนามิก ไปยังหลายๆ เธรดจำเป็นต้องใช้ข้อควรพิจารณาพิเศษ พิจารณาแฟรกเมนต์ของโค้ด ต่อไปนี้:

```
void *returned_data;
...
pthread_join(target_thread, &returned_data);
/* retrieves information from returned_data */
free(returned_data);
```

ตัวชี้ **returned_data** เป็น อิสระเมื่อถูกทำงานโดยเธรดเพียงเธรดเดียวเท่านั้น ถ้ามีหลายเธรดเรียกทำงาน แฟรกเมนต์ของโค้ดด้านบนพร้อมกัน ตัวชี้ **returned_data** จะ เป็นอิสระหลายครั้ง ซึ่งเป็นสถานการณ์ที่ควรหลีกเลี่ยง ในการป้องกันเหตุการณ์นี้ ใช้แฟล็ก mutex-protected เพื่อส่งสัญญาณว่าตัวชี้ **returned_data** เป็นอิสระ บรรทัดต่อไปนี้จากตัวอย่างก่อนหน้า:

```
free(returned_data);
```

จะแทนที่โดยบรรทัด ต่อไปนี้ โดยที่สามารถใช้ mutex เพื่อทำการล็อกการเข้าถึงส่วนที่สำคัญ (โดยถือว่าตัวแปร *flag* ค่าเริ่มต้นเป็น 0):

```
/* lock - entering a critical region, no other thread should
                run this portion of code concurrently */
if (!flag) {
        free(returned_data);
        flag = 1;
}
/* unlock - exiting the critical region */
```

การล็อกการเข้าถึงส่วนที่สำคัญช่วยให้แน่ใจว่าตัวชี้ **returned_data** เป็นอิสระเพียงครั้งเดียวเท่านั้น

เมื่อส่งกลับค่าตัวชี้ที่ชี้ไปยังหน่วยเก็บที่จัดสรรแบบไดนามิกไปยัง หลายๆ เธรด โดยทั้งหมดเรียกทำงานโค้ดที่ต่างกัน คุณต้องทำให้แน่ใจว่ามีเพียงหนึ่งเธรดจริงๆ เท่านั้น ที่ทำให้ตัวชี้เป็นอิสระ

หลักการที่เกี่ยวข้อง:

“การใช้ตัวแปรเงื่อนไข” ในหน้า 482
ตัวแปรเงื่อนไขยอมให้ threads รอจนกว่า จะมีเหตุการณ์หรือเงื่อนไขเกิดขึ้น
“ข้อมูลเฉพาะเธรด” ในหน้า 511
แอ็พพลิเคชันจำนวนมากต้องการข้อมูลบางส่วนสำหรับเธรดพื้นฐาน ระหว่างการเรียกฟังก์ชัน

เธรดการกำหนดเวลา

เธรดสามารถกำหนดตารางเวลาได้ และไลบรารีเธรดจะจัดเตรียมตัวช่วยต่างๆ เพื่อจัดการและควบคุมการกำหนดตารางเวลาของเธรด

และยังจัดเตรียมตัวช่วย เพื่อควบคุมการกำหนดตารางเวลาของเธรดในช่วงการดำเนินการประสานเวลา เช่น การล็อก mutex เธรดแต่ละเธรดมีชุดของพารามิเตอร์ของการกำหนดตารางเวลา ที่เป็นของตนเอง พารามิเตอร์เหล่านี้สามารถตั้งค่าโดยใช้อ็อบเจ็กต์แอ็ตทริบิวต์เธรด ก่อนที่เธรดจะถูกสร้างขึ้น พารามิเตอร์ยังถูกตั้งค่าแบบไดนามิก ในระหว่างการประมวลผลของเธรด

การควบคุมการกำหนดตารางเวลาของเธรด อาจเป็นภารกิจที่ซับซ้อน เนื่องจากตัวกำหนดเวลาจะจัดการกับเธรดในระบบแบบกว้างๆ ดังนั้นพารามิเตอร์การกำหนดเวลา ของเธรดจะโต้ตอบกับเธรดอื่นทั้งหมดใน กระบวนการและในกระบวนการอื่น ตัวอย่างต่อไปนี้จะถูกนำมาใช้เป็นอันดับแรก หากคุณต้องการควบคุมการกำหนดตารางเวลาของเธรด

ไลบรารีเธรดอนุญาตให้โปรแกรมเมอร์ควบคุมการประมวลผลการจัดตารางเวลาของเธรด ด้วยวิธีต่อไปนี้:

- โดยตั้งค่าแอ็ตทริบิวต์การกำหนดตารางเวลาขณะสร้างเธรด
- โดยการเปลี่ยนแอ็ตทริบิวต์การกำหนดตารางเวลาของเธรดที่สร้าง
- โดยการกำหนดผลกระทบของ mutex สำหรับการกำหนดตารางเวลาของเธรด ขณะที่สร้าง mutex (รู้จักกันในนามของ *การจัดตารางเวลาการประสานเวลา*)
- โดยการเปลี่ยนการกำหนดตารางเวลาของเธรดในช่วงการดำเนินการประสานเวลา (รู้จักกันในนามของ *การกำหนดตารางเวลาสำหรับการประสานเวลา*)

พารามิเตอร์การกำหนดเวลา

เธรดมีพารามิเตอร์การกำหนดตารางเวลาต่อไปนี้:

พารามิเตอร์	คำอธิบาย
ขอบเขต	ขอบเขต contention ของเธรดถูกกำหนดโดยแบบจำลองเธรด ที่ใช้ในไลบรารีเธรด
นโยบาย	นโยบายการกำหนดตารางเวลาของเธรดจะกำหนดวิธีที่ตัวกำหนดตารางเวลาใช้เธรด หลังจากที่ได้รับการควบคุม CPU
ลำดับความสำคัญ	ลำดับความสำคัญของการกำหนดตารางเวลาของเธรด จะกำหนดความสำคัญของงานที่ทำโดยเธรดแต่ละตัว

พารามิเตอร์การกำหนดตารางเวลาสามารถตั้งค่าได้ก่อนการสร้างเธรด หรือในระหว่างการประมวลผลของเธรด โดยทั่วไป การควบคุมพารามิเตอร์การกำหนดตารางเวลาของเธรด มีความสำคัญสำหรับเธรดที่มี CPU ที่ดีเยี่ยมเท่านั้น ดังนั้น ไลบรารีเธรดจะจัดเตรียมค่าดีฟอลต์ที่เพียงพอสำหรับกรณีส่วนใหญ่

การใช้แอ็ตทริบิวต์ inheritsched

แอ็ตทริบิวต์ **inheritsched** ของอ็อบเจ็กต์แอ็ตทริบิวต์เธรดที่ระบุวิธีที่แอ็ตทริบิวต์การกำหนดตารางเวลาของเธรด จะถูกกำหนด ค่าต่อไปนี้ คือค่าที่ถูกต้อง:

ค่า	คำอธิบาย
PTHREAD_INHERIT_SCHED	ระบุว่า เธรดใหม่จะได้รับแอ็ตทริบิวต์การกำหนดตารางเวลา (**schedpolicy** และแอ็ตทริบิวต์ **schedparam**) ของการสร้างเธรด แอ็ตทริบิวต์การกำหนดตารางเวลาที่นิยามในอ็อบเจ็กต์แอ็ตทริบิวต์จะถูกละเว้น
PTHREAD_EXPLICIT_SCHED	ระบุว่า เธรดใหม่จะได้รับแอ็ตทริบิวต์การกำหนดตารางเวลา ที่กำหนดอยู่ในอ็อบเจ็กต์แอ็ตทริบิวต์นี้

ค่าดีฟอลต์ของแอ็ตทริบิวต์ **inheritsched** คือ PTHREAD_INHERIT_SCHED แอ็ตทริบิวต์จะถูกตั้งค่าโดยการเรียกรูทีนย่อย **pthread_attr_setinheritsched** ค่าปัจจุบันของแอ็ตทริบิวต์จะถูกส่งคืน โดยการเรียกรูทีนย่อย **pthread_attr_getinheritsched**

หากต้องการตั้งค่าแอ็ตทริบิวต์การกำหนดตารางเวลาของเธรดในอ็อบเจ็กต์แอ็ตทริบิวต์ แอ็ตทริบิวต์ **inheritsched** ต้องถูกตั้งค่าเป็น PTHREAD_EXPLICIT_SCHED อันดับแรก มิฉะนั้น อ็อบเจ็กต์แอ็ตทริบิวต์สำหรับ แอ็ตทริบิวต์การกำหนดตารางเวลาจะถูกละเว้น

นโยบายและลำดับความสำคัญการกำหนดเวลา

ไลบรารีเธรดจัดเตรียมนโยบาย การกำหนดตารางเวลาต่อไปนี้:

ไลบรารี	คำอธิบาย
SCHED_FIFO	การกำหนดตารางเวลาแบบเข้าก่อนออกก่อน (FIFO) เธรดแต่ละตัวมีลำดับความสำคัญที่มีค่าคงที่ เมื่อเธรดจำนวนมากมีลำดับความสำคัญแบบเดียวกัน เธรดเหล่านั้นจะรันเพื่อให้เสร็จสิ้นด้วยลำดับ FIFO
SCHED_RR	การกำหนดตารางเวลาแบบวนรอบ (RR) เธรดแต่ละตัวมีลำดับความสำคัญที่มีค่าคงที่ เมื่อเธรดจำนวนมากมีลำดับความสำคัญแบบเดียวกัน เธรดเหล่านั้นจะรันด้วยการแบ่งเวลาที่คงที่ในลำดับ FIFO
SCHED_OTHER	ดีฟอลต์การกำหนดตารางเวลา AIX เธรดแต่ละตัวมีลำดับความสำคัญเริ่มต้น ซึ่งจะถูกแก้ไขโดยตัวกำหนดตารางเวลาแบบไดนามิกตามกิจกรรมของเธรด การประมวลผลเธรดคือการแบ่งเวลา สำหรับระบบอื่น นโยบายการกำหนดตารางเวลาอาจแตกต่างกัน

ในเวอร์ชันของ AIX ก่อนเวอร์ชัน 5.3 การเปลี่ยนลำดับความสำคัญของเธรด ขณะที่ตั้งค่านโยบายการกำหนดตารางเวลาให้เป็น **SCHED_OTHER** จะไม่อนุญาตให้ทำ ในกรณีนี้ เคอร์เนลจะจัดการกับลำดับความสำคัญโดยตรง และค่าที่ถูกต้องเท่านั้นที่สามารถส่งผ่านไปยังรูทีนย่อย **pthread_setschedparam** ซึ่งมีค่าเป็น **DEFAULT_PRIO** ค่า **DEFAULT_PRIO** ที่กำหนดในไฟล์ **pthread.h** ซึ่งมีค่า 1 และค่าที่ส่งผ่านอื่นๆ จะถูกละเว้น

การเริ่มต้นด้วย AIX 5.3 คุณสามารถเปลี่ยนลำดับความสำคัญของเธรดได้ เมื่อคุณตั้งค่านโยบายการกำหนดตารางเวลาไปเป็น **SCHED_OTHER** ค่าที่ถูกต้อง ซึ่งสามารถส่งไปยังรูทีนย่อย **pthread_setschedparam** มีค่าตั้งแต่ 40 ถึง 80 อย่างไรก็ตาม ผู้ใช้ที่มี privilege เท่านั้นที่สามารถตั้งค่าลำดับความสำคัญ ที่มากกว่า 60 ได้ ลำดับความสำคัญในช่วงตั้งแต่ 1 ถึง 39 จะมีลำดับความสำคัญเดียวกับ 40 และลำดับความสำคัญในช่วงของ 81 ถึง 127 จะมีลำดับความสำคัญเดียวกับ 80

หมายเหตุ: ใน AIX เคอร์เนลจะมีลำดับความสำคัญที่กลับกัน สำหรับเคอร์เนล AIX ลำดับความสำคัญที่อยู่ในช่วงตั้งแต่ 0 ถึง 127 โดยที่ 0 คือลำดับความสำคัญที่สูงสุดและ 127 คือลำดับความสำคัญที่ต่ำที่สุด คำสั่ง เช่น คำสั่ง **ps** จะรายงานลำดับความสำคัญเคอร์เนล

ไลบรารีเธรดจะจัดการกับลำดับความสำคัญผ่านโครงสร้าง **sched_param** ซึ่งกำหนดอยู่ในไฟล์ส่วนหัว **sys/sched.h** โครงสร้างนี้จะมีฟิลด์ต่อไปนี้:

ฟิลด์	คำอธิบาย
sched_priority	ระบุลำดับความสำคัญ
sched_policy	ฟิลด์นี้จะถูกละเว้นโดยไลบรารีเธรด ห้ามใช้

การตั้งค่านโนบายและลำดับความสำคัญการกำหนดเวลาขณะสร้าง

นโยบายการกำหนดตารางเวลาสามารถตั้งค่าขณะที่สร้างเธรด โดยตั้งค่าแอ็ตทริบิวต์ **schedpolicy** ของอ็อบเจ็กต์แอ็ตทริบิวต์เธรด รูทีนย่อย **pthread_attr_setschedpolicy** จะตั้งค่านโยบายการกำหนดตารางเวลาให้เป็นหนึ่งในนโยบายการกำหนดตารางเวลาที่ได้กำหนดไว้ก่อนหน้านี้ ค่าปัจจุบันของแอ็ตทริบิวต์ **schedpolicy** ของอ็อบเจ็กต์แอ็ตทริบิวต์สามารถขอรับได้โดยใช้รูทีนย่อย **pthread_attr_getschedpolicy**

นโยบายการกำหนดตารางเวลาสามารถตั้งค่าในเวลาที่สร้างเธรด โดยตั้งค่าแอ็ตทริบิวต์ **schedparam** ของอ็อบเจ็กต์แอ็ตทริบิวต์เธรด รูทีนย่อย **pthread_attr_setschedparam** จะตั้งค่าของแอ็ตทริบิวต์ **schedparam** คัดลอกค่าของโครงสร้างที่ระบุไว้ รูทีนย่อย **pthread_attr_getschedparam** จะได้รับแอ็ตทริบิวต์ **schedparam**

ในชิ้นส่วนของโค้ดต่อไปนี้ เธรดจะถูกสร้างด้วยนโยบายการกำหนดตารางเวลาแบบวนรอบ โดยใช้ลำดับความสำคัญ 3:

```
sched_param schedparam;

schedparam.sched_priority = 3;

pthread_attr_init(&attr);
pthread_attr_setinheritsched(&attr, PTHREAD_EXPLICIT_SCHED);
pthread_attr_setschedpolicy(&attr, SCHED_RR);
pthread_attr_setschedparam(&attr, &schedparam);

pthread_create(&thread, &attr, &start_routine, &args);
pthread_attr_destroy(&attr);
```

สำหรับข้อมูลเพิ่มเติมเกี่ยวกับ แอ็ตทริบิวต์ **inheritsched** โปรดดูที่ การใช้แอ็ตทริบิวต์ inheritsched

การตั้งค่าแอ็ตทริบิวต์การกำหนดเวลาเมื่อประมวลผล

รูทีนย่อย **pthread_getschedparam** จะส่งคืนแอ็ตทริบิวต์ **schedpolicy** และ **schedparam** ของเธรด แอ็ตทริบิวต์เหล่านี้สามารถตั้งค่าได้โดยเรียกรูทีนย่อย **pthread_setschedparam** ถ้าเธรดเป้าหมายกำลังทำงานอยู่บนตัวประมวลผล นโยบายการกำหนดตารางเวลาใหม่ และลำดับความสำคัญจะถูกนำไปใช้ในครั้งถัดไปที่เธรดถูกกำหนดตารางเวลา ถ้าเธรดเป้าหมายไม่ได้ทำงานอยู่ เธรดสามารถกำหนดตารางเวลาได้ในทันที ที่จุดสิ้นสุดของการเรียกรูทีนย่อย

ตัวอย่างเช่น ให้พิจารณาเธรด T ซึ่งกำลังรันด้วยนโยบายแบบวนรอบในปัจจุบัน ในขณะที่แอ็ตทริบิวต์ **schedpolicy** ของ T ถูกเปลี่ยนไปเป็น FIFO T จะรันจนกระทั่งถึงจุดสิ้นสุดของการแบ่งเวลา ซึ่งเป็นเวลาที่แอ็ตทริบิวต์การกำหนดตารางเวลาจะถูกประเมินผลใหม่ ถ้าไม่มีเธรดที่มีลำดับความสำคัญสูง T จะถูกกำหนดตารางเวลาใหม่ ก่อนที่เธรดอื่นจะมีลำดับความสำคัญในระดับเดียวกัน โปรดพิจารณาตัวอย่างที่สองซึ่งเธรดที่มีลำดับความสำคัญต่ำ ไม่ได้ทำงาน ถ้าลำดับความสำคัญของเธรดเกิดขึ้นโดยเธรดอื่นเรียกรูทีนย่อย **pthread_setschedparam** เธรดเป้าหมายจะถูกกำหนดตารางเวลาใหม่โดยทันที หากเธรดเป้าหมายคือเธรดที่สามารถรันได้ในลำดับความสำคัญสูงสุด

หมายเหตุ: รูทีนทั้งสองใช้พารามิเตอร์ *policy* และโครงสร้าง **sched_param** แม้ว่าโครงสร้างนี้จะมีฟิลด์ sched_policy โปรแกรมไม่สมควรใช้ รูทีนย่อยใช้พารามิเตอร์ *policy* เพื่อส่งค่านโยบายการกำหนดตารางเวลา และรูทีนย่อยจะละเว้นฟิลด์ sched_policy

ข้อควรพิจารณาเกี่ยวกับนโยบายการกำหนดเวลา

แอ็พพลิเคชันควรใช้นโยบายการกำหนดตารางเวลาที่เป็นค่าดีฟอลต์ เว้นเสียแต่แอ็พพลิเคชันเฉพาะต้องใช้นโยบายการกำหนดตารางเวลาในลำดับความสำคัญ ที่มีค่าที่กำหนดไว้ โปรดพิจารณาจุดต่อไปนี้เกี่ยวกับการใช้นโยบายที่ไม่ใช่ค่าดีฟอลต์:

- การใช้นโยบายแบบวนรอบจะตรวจสอบให้มั่นใจว่า เธรดทั้งหมดที่มีลำดับความสำคัญเดียวกัน จะถูกกำหนดตารางเวลาที่เหมือนกัน โดยไม่พิจารณาถึงกิจกรรมของเธรดเหล่านั้น ซึ่งจะมีประโยชน์ในโปรแกรม ที่เธรดต้องอ่านเซนเซอร์หรือเขียนตัวขับเคลื่อน
- การใช้นโยบายแบบเข้าก่อนออกก่อนควรถูกกระทำด้วยความระวังมากที่สุด เธรดที่รันด้วยนโยบาย FIFO จะรันเพื่อให้เสร็จสิ้น เว้นเสียแต่จะถูกบล็อกด้วยการเรียกบางขณะ เช่น ขณะดำเนินการกับการดำเนินการอินพุตและเอาต์พุต เธรดแบบ FIFO ที่มีลำดับความสำคัญสูงอาจไม่ได้จองไว้ และอาจกระทบกับผลการทำงานของระบบในระดับโกลบอล ตัวอย่างเช่น เธรดที่ทำการคำนวณ เช่น การกลับแมทริกซ์ขนาดใหญ่ ไม่ควรรันด้วยนโยบาย FIFO

การตั้งค่านโยบายการกำหนดตารางเวลาและลำดับความสำคัญ ยังมีอิทธิพลโยขอบเขต contention ของเธรด การใช้ FIFO หรือนโยบายแบบวนรอบ อาจไม่อนุญาตให้ใช้

รูทีนย่อย sched_yield

รูทีนย่อย **sched_yield** จะเทียบเท่ากับเธรดของรูทีนย่อย **yield** รูทีนย่อย **sched_yield** จะบังคับให้เรียกเธรดเพื่อยกเลิกการใช้ตัวประมวลผล และให้โอกาสแก่เธรดอื่นๆ เพื่อกำหนดตารางเวลา เธรดที่ถูกกำหนดตารางเวลาถัดไป อาจเป็นของการประมวลผลเดียวกันตาที่เรียกเธรดหรือการประมวลผลอื่นๆ ห้ามใช้รูทีนย่อย **yield** ในโปรแกรมแบบ มัลติเธรด

อินเตอร์เฟสรูทีนย่อย **pthread_yield** ไม่พร้อมใช้งานใน Single UNIX Specification, Version 2

หลักการที่เกี่ยวข้อง:

"การกำหนดเวลาการซิงโครไนซ์" ในหน้า 506
โปรแกรมเมอร์สามารถควบคุมการประมวลผลการกำหนดตารางเวลาของเธรด เมื่อมีข้อจำกัด โดยเฉพาะข้อจำกัดด้านเวลาที่ต้องการเธรดบางตัว ที่ต้องการเรียกใช้งานเร็วกว่าเธรดตัวอื่น

"รายการของรูทีนย่อยการกำหนดเวลา"
ส่วนนี้แสดงรายการรูทีนย่อยการกำหนดเวลา

"การพัฒนาโปรแกรมแบบมัลติเธรด" ในหน้า 544
การพัฒนาโปรแกรมแบบมัลติเธรดคล้ายกับการพัฒนา โปรแกรมที่มีหลายกระบวนการ การพัฒนาโปรแกรมยังประกอบด้วยการคอมไพล์ และการดีบักโค้ด

รายการของรูทีนย่อยการกำหนดเวลา

ส่วนนี้แสดงรายการรูทีนย่อยการกำหนดเวลา

รูทีนย่อย	คำอธิบาย
pthread_attr_getschedparam	ส่งกลับค่าของแอ็ตทริบิวต์ **schedparam** ของ อ็อบเจ็กต์แอ็ตทริบิวต์เธรด
pthread_attr_setschedparam	ตั้งค่าของแอ็ตทริบิวต์ **schedparam** ของ อ็อบเจ็กต์แอ็ตทริบิวต์เธรด
pthread_getschedparam	ส่งกลับค่าของแอ็ตทริบิวต์ **schedpolicy** และ **schedparam** ของเธรด
sched_yield	บังคับการเรียกเธรดเพื่อยกเลิกการใช้โพรเซสเซอร์

หลักการที่เกี่ยวข้อง:

"เธรดการกำหนดเวลา" ในหน้า 501

เธรดสามารถกำหนดตารางเวลาได้ และไลบรารีเธรดจะจัดเตรียมตัวช่วยต่างๆ เพื่อจัดการและควบคุมการกำหนดตารางเวลาของเธรด

ขอบเขต Contention และระดับสภาวะพร้อมกัน

ขอบเขต contention ของเธรดผู้ใช้จะกำหนดวิธีการ แม็พกับเคอร์เนลเธรด

. ไลบรารีเธรดจะกำหนดขอบเขต contention ต่อไปนี้:

PTHREAD_SCOPE_PROCESS
: ขอบเขตของการประมวลผล contention ซึ่งบางครั้งเรียกว่า *ขอบเขต contention แบบโลคัล* ระบุว่า เธรดจะถูกกำหนดตารางเวลากับเธรดในขอบเขต contention แบบโลคัลอื่นๆ ในการประมวลผล เธรดของผู้ใช้ในขอบเขตของการประมวลผล contention คือผู้ใช้เธรดที่แบ่งใช้เคอร์เนลเธรดด้วยเธรดของผู้ใช้ในขอบเขตของการประมวลผล contention ในการประมวลผล เธรดของผู้ใช้ทั้งหมดในแบบจำลองเธรดแบบ M:1 มีขอบเขตของการประมวลผล contention

PTHREAD_SCOPE_SYSTEM
: ขอบเขตของการประมวลผล contention ในบางครั้งเรียกว่า *ขอบเขต contention แบบโกลบอล* ระบุว่า เธรดจะถูกกำหนดตารางเวลากับเธรดอื่นๆ ทั้งหมดในระบบ และแม็พกับเคอร์เนลเธรดหนึ่งตัวโดยตรง เธรดของผู้ใช้ทั้งหมดในแบบจำลองเธรด 1:1 มีขอบเขต contention ของระบบ

ในแบบจำลองเธรดแบบ M:N เธรดของผู้ใช้สามารถมีระบบ หรือขอบเขตของการประมวลผล contention อย่างใดอย่างหนึ่ง ดังนั้น แบบจำลองเธรด M:N ถูกอ้างอิงเป็นแบบจำลอง *mixed-scope*

ระดับของสภาวะพร้อมกัน คือคุณสมบัติของไลบรารีเธรด M:N ซึ่งจะกำหนดจำนวนของตัวประมวลผลเสมือนที่ใช้เพื่อรันเธรดของผู้ใช้ใน ขอบเขตของการประมวลผล contention หมายเลขนี้ไม่สามารถมีค่าเกินจำนวนของเธรดของผู้ใช้ในขอบเขตของการประมวลผล contention และถูกตั้งค่าแบบไดนามิกโดยใช้ ไลบรารีเธรด ระบบยังตั้งค่าข้อจำกัดของจำนวนของเคอร์เนลเธรด ที่พร้อมใช้งาน

การตั้งค่าขอบเขต contention

ขอบเขต contention สามารถตั้งค่าก่อนที่เธรดจะถูกสร้างขึ้น โดยตั้งค่าแอ็ตทริบิวต์ contention-scope ของอ็อบเจ็กต์แอ็ตทริบิวต์เธรด รูทีนย่อย **pthread_attr_setscope** จะตั้งค่าของแอ็ตทริบิวต์ **pthread_attr_getscope** จะส่งคืน

ขอบเขต contention อาจมีความหมายในขอบเขตแบบผสมของการนำไลบรารีไปใช้งานแบบ M:N รูทีน **TestImplementation** สามารถเขียนได้ดังนี้:

```
int TestImplementation()
{
        pthread_attr_t a;
        int result;
```

```
        pthread_attr_init(&a);
        switch (pthread_attr_setscope(&a, PTHREAD_SCOPE_PROCESS))
        {
                case 0:          result = LIB_MN; break;
                case ENOTSUP:    result = LIB_1l; break;
                case ENOSYS:     result = NO_PRIO_OPTION; break;
                default:         result = ERROR; break;
        }

        pthread_attr_destroy(&a);
        return result;
}
```

ผลกระทบของขอบเขต contention กับการกำหนดเวลา

ขอบเขต contention ของเธรดจะมีอิทธิผลต่อการกำหนดตารางเวลา เธรดที่มีขอบเขต contention แต่ละตัวจะผูกกับเคอร์เนลเธรดหนึ่งตัว ดังนั้น การเปลี่ยนนโยบายการกำหนดตารางเวลาและลำดับความสำคัญของเธรดของผู้ใช้แบบโกลบอลจะส่งผลในการเปลี่ยนแปลงนโยบายการกำหนดตารางเวลาและลำดับความสำคัญของเคอร์เนลเธรด

ใน AIX เฉพาะเคอร์เนลเธรดที่มีสิทธิแบบ root เท่านั้น ที่สามารถใช้นโยบายการกำหนดตารางเวลาของลำดับความสำคัญที่กำหนดไว้ (FIFO หรือแบบวนรอบ) โค้ดต่อไปนี้ จะส่งคืนโค้ดระบุความผิดพลาด **EPERM** หากการเรียกเธรดมีขอบเขต contention ของระบบ แต่ไม่มีสิทธิแบบ root โค้ดนี้จะไม่ล้มเหลวหากการเรียกเธรดมีขอบเขตของการประมวลผล contention

```
schedparam.sched_priority = 3;
pthread_setschedparam(pthread_self(), SCHED_FIFO, schedparam);
```

หมายเหตุ: สิทธิแบบ root ไม่จำเป็นต้องใช้ในการควบคุมพารามิเตอร์การกำหนดตารางเวลา ของเธรดของผู้ใช้ที่มีขอบเขตของการประมวลผล contention

เธรดของผู้ใช้โลคัลสามารถตั้งค่านโยบายการกำหนดตารางเวลา และลำดับความสำคัญใดๆ ภายในช่วงของค่าที่ถูกต้อง อย่างไรก็ตาม เธรดสองตัวจะมีนโยบายการกำหนดตารางเวลา และลำดับความสำคัญที่เหมือนกับ แต่มีขอบเขต contention ที่ต่างกัน ซึ่งจะไม่ถูกกำหนดตารางเวลาในวิธีเดียวกัน เธรดที่มีขอบเขตของการประมวลผล contention จะถูกเรียกใช้งานโดยเคอร์เนลเธรดที่มีพารามิเตอร์การกำหนดตารางเวลาถูกตั้งค่าโดยไลบรารี

หลักการที่เกี่ยวข้อง:

"การทำความเข้าใจเธรดและกระบวนการ" ในหน้า 456

เธรด คือการควบคุมสายงานอย่างเป็นอิสระที่ทำงานภายในพื้นที่แอดเดรสเดียวกันกับ การควบคุมสายงานอย่างเป็นอิสระอื่นๆ ภายในกระบวนการ

การกำหนดเวลาการซิงโครไนซ์

โปรแกรมเมอร์สามารถควบคุมการประมวลผลการกำหนดตารางเวลาของเธรด เมื่อมีข้อจำกัด โดยเฉพาะข้อจำกัดด้านเวลาที่ต้องการเธรดบางตัว ที่ต้องการเรียกใช้งานเร็วกว่าเธรดตัวอื่น

อ็อบเจ็กต์การประสานเวลา เช่น mutexes อาจบล็อกเธรดที่มีลำดับความสำคัญสูง ในบางกรณี ลักษณะการทำงานที่ไม่สามารถกำหนดได้ ซึ่งรู้จักกันในนามของ *ความผกผันของลำดับความสำคัญ* อาจเกิดขึ้นได้ ไลบรารีเธรดจะมีตัวช่วย *โปรโตคอล mutex* เพื่อหลีกเลี่ยงความผกผันของลำดับความสำคัญ

การกำหนดตารางเวลาของการประสานเวลา จะนิยามวิธีการกำหนดตารางเวลาในการประมวลผลโดยเฉพาะอย่างยิ่งลำดับความสำคัญของเธรด ที่แก้ไขโดยการพัก mutex การกำหนดตารางเวลานี้อนุญาตให้ใช้ลักษณะการทำงานที่กำหนดไว้เอง และหลีกเลี่ยงความผกผันของลำดับความสำคัญ ซึ่งมีประโยชน์เมื่อใช้ scheme การล็อกที่ซับซ้อน การนำไปปฏิบัติของไลบรารีเธรดบางอย่าง ไม่ได้จัดเตรียมการกำหนดตารางเวลาการประสานเวลาไว้

ความผกผันของลำดับความสำคัญ

ความผกผันของลำดับความสำคัญ เกิดขึ้นเมื่อเธรดที่มีลำดับความสำคัญต่ำพัก mutex ไว้ แล้วบล็อกเธรดที่มีลำดับความสำคัญสูง เนื่องจากลำดับความสำคัญต่ำ เจ้าของ mutex อาจพัก mutex ไว้สำหรับช่วงเวลาที่ไม่มีข้อจำกัด ตามผลลัพธ์ที่ได้ ไม่มีความเป็นไปได้ที่จะรับประกันว่า เธรด deadline

ตัวอย่างต่อไปนี้ จะแสดงถึงความผกผันของลำดับความสำคัญโดยทั่วไป ในตัวอย่างนี้ กรณีของระบบแบบยูนิโพรเซสเซอร์จะถูกนำมาพิจารณา ความผกผันของลำดับความสำคัญจะเกิดขึ้นบนระบบแบบมัลติโพรเซสเซอร์ด้วยวิธีแบบเดียวกัน

ในตัวอย่างของเรา mutex *M* จะถูกนำมาใช้เพื่อป้องกันข้อมูลทั่วไปบางส่วน เธรด *A* มีลำดับความสำคัญเท่ากับ 100 และจะถูกกำหนดตารางเวลาให้ถี่ขึ้น เธรด *B* มีลำดับความสำคัญเท่ากับ 20 และมีเธรดแบบพื้นหลัง เธรดอื่นๆ ในกระบวนการ จะมีลำดับความสำคัญที่ใกล้กับ 60 ชิ้นส่วนของโค้ดจากเธรด *A* มีดังต่อไปนี้:

```
pthread_mutex_lock(&M);             /* 1 */
...
pthread_mutex_unlock(&M);
```

ชิ้นส่วนของโค้ดจากเธรด *B* มีดังต่อไปนี้:

```
pthread_mutex_lock(&M);          /* 2 */
...
fprintf(...);                    /* 3 */
...
pthread_mutex_unlock(&M);
```

โปรดพิจารณาการเรียงลำดับเหตุการณ์ของการประมวลผลต่อไปนี้ เธรด *B* ถูกกำหนดเวลา และเรียกใช้งานบรรทัดที่ 2 ขณะที่การเรียกใช้งานบรรทัดที่ 3 เธรด *B* ถูกจอง โดยเธรด *A* เธรด *A* เรียกใช้งานบรรทัดที่ 1 และ ถูกบล็อก เนื่องจาก mutex *M* ถูกยึดไว้โดยเธรด *B* ดังนั้น เธรดอื่นในกระบวนการจะถูกกำหนดเวลา เนื่องจากเธรด *B* มีลำดับความสำคัญที่ต่ำมาก ซึ่งอาจไม่ถูกกำหนดตารางเวลาขึ้นใหม่ในระยะเวลาที่ยาวนาน และบล็อกเธรด *A* แม้ว่าเธรด *A* จะมีลำดับความสำคัญสูงก็ตาม

โปรโตคอล Mutex

หากต้องการหลีกเลี่ยงความผกผัน โปรโตคอล mutex ต่อไปนี้ จะถูกจัดเตรียมไว้โดยไลบรารีเธรด:

โปรโตคอลการสืบทอดลำดับความสำคัญ

บางครั้งเรียกว่า *โปรโตคอลการสืบทอดลำดับความสำคัญพื้นฐาน* ในโปรโตคอลการสืบทอดลำดับความสำคัญ เจ้าของ mutex จะสืบทอดลำดับความสำคัญของเธรดที่ถูกบล็อกซึ่งมีลำดับความสำคัญสูงสุด เมื่อเธรดพยายามล็อก mutex โดยใช้โปรโตคอลนี้และถูกบล็อกไว้ เจ้าของ mutex จะได้รับลำดับความสำคัญของเธรดที่ถูกบล็อก ถ้าลำดับความสำคัญนั้นสูงกว่าลำดับความสำคัญของเจ้าของ และจะกู้คืนลำดับความสำคัญเดิม เมื่อปลดล็อก mutex แล้ว

โปรโตคอลการป้องกันลำดับความสำคัญ

บางครั้งเรียกว่า *อีมูเลชันโปรโตคอลลำดับความสำคัญสูงสุด* ในโปรโตคอลการป้องกันลำดับความสำคัญ mutex แต่ละตัวจะมี*ลำดับความสำคัญสูงสุด* ซึ่งเป็นลำดับความสำคัญภายในช่วงของลำดับความสำคัญที่ถูกต้อง เมื่อเธรดเป็นเจ้าของ mutex เธรดจะได้รับลำดับความสำคัญสูงสุดของ mutex แบบชั่วคราว หากระดับสูงสุดมีค่าสูงกว่าลำดับความ

สำคัญที่ตนเป็นเจ้าของ และจะกู้คืนลำดับความสำคัญเดิม เมื่อปลดล็อก mutex แล้ว ลำดับความสำคัญสูงสุดควรมีค่าของลำดับความสำคัญที่สูงที่สุดของเธรดทั้งหมด ซึ่งอาจล็อก mutex ไว้ หรือ ความผกผันของลำดับความสำคัญสูงสุด หรือ deadlock อาจเกิดขึ้นได้ และโปรโตคอลจะไม่มีประสิทธิภาพ

โปรโตคอลทั้งคู่จะเพิ่มลำดับความสำคัญของเธรดที่พัก mutex ที่ระบุเฉพาะ ดังนั้น deadline อาจสามารถรับประกันได้ นอกจากนี้ เมื่อใช้งานอย่างถูกต้อง โปรโตคอล mutex สามารถป้องกัน deadlock ได้ซึ่งกันและกัน โปรโตคอล mutex จะถูกกำหนดให้กับ mutex ที่ละตัว

การเลือกโปรโตคอล mutex

ตัวเลือกของโปรโตคอล mutex จะถูกทำขึ้นโดยตั้งค่าแอ็ตทริบิวต์ ขณะสร้าง mutex โปรโตคอล mutex จะควบคุมผ่านแอ็ตทริบิวต์ protocol แอ็ตทริบิวต์นี้สามารถตั้งค่าในอ็อบเจ็กต์แอ็ตทริบิวต์ mutex ได้โดยใช้รูทีนย่อย **pthread_mutexattr_getprotocol** และ **pthread_mutexattr_setprotocol** แอ็ตทริบิวต์ protocol สามารถมีหนึ่งในค่าต่อไปนี้:

ค่า	คำอธิบาย
PTHREAD_PRIO_DEFAULT	No value
PTHREAD_PRIO_NONE	หมายถึงไม่มีโปรโตคอล
PTHREAD_PRIO_INHERIT	หมายถึงโปรโตคอลการสืบทอดลำดับความสำคัญ
PTHREAD_PRIO_PROTECT	หมายถึงโปรโตคอลการป้องกันลำดับความสำคัญ

หมายเหตุ: The behavior of **PTHREAD_PRIO_DEFAULT** is the same as the **PTHREAD_PRIO_INHERIT** attribute. With reference to the mutex locking, the threads acting with the default attribute will temporarily boost the priority of a mutex holder when a user is locked and has a higher priority than the owner. Therefore, there are only three behaviors that are possible, although there are four values for the possible priority in the attribute structure.

โปรโตคอลการป้องกันลำดับความสำคัญจะใช้แอ็ตทริบิวต์เพิ่มเติมหนึ่งตัวคือ: แอ็ตทริบิวต์ prioceiling แอ็ตทริบิวต์นี้มีลำดับความสำคัญสูงสุดของ mutex แอ็ตทริบิวต์ prioceiling สามารถควบคุมได้ในอ็อบเจ็กต์แอ็ตทริบิวต์ mutex โดยใช้รูทีนย่อย **pthread_mutexattr_getprioceiling** และ **pthread_mutexattr_setprioceiling**

แอ็ตทริบิวต์ prioceiling ของ mutex ยังสามารถควบคุมแบบไดนามิกได้โดยใช้รูทีนย่อย **pthread_mutex_getprioceiling** และ **pthread_mutex_setprioceiling** เมื่อเปลี่ยนลำดับความสำคัญสูงสุดของ mutex แบบไดนามิกแล้ว mutex จะถูกล็อกโดยไลบรารี ซึ่งไม่ได้ถูกจัดการโดยเธรดที่เรียกรูทีนย่อย **pthread_mutex_setprioceiling** เพื่อหลีกเลี่ยง deadlock การตั้งค่าลำดับความสำคัญสูงสุดแบบไดนามิกของ mutex สามารถนำมาใช้ให้เกิดประโยชน์ขณะที่เพิ่มลำดับความสำคัญของเธรด

การนำไปปฏิบัติของโปรโตคอล mutex จะเป็นตัวเลือก แต่ละโปรโตคอลคืออ็อพชัน POSIX

การสืบทอดหรือการป้องกัน

โปรโตคอลทั้งสองแบบจะคล้ายคลึงกัน และส่งผลให้มีการพัฒนาลำดับความสำคัญของเธรดที่พัก mutex ไว้ ถ้าโปรโตคอลทั้งสองแบบพร้อมใช้งาน โปรแกรมเมอร์ต้องเลือกโปรโตคอล ตัวเลือกจะขึ้นอยู่กับว่า ลำดับความสำคัญของเธรด ที่จะล็อก mutex พร้อมให้โปรแกรมเมอร์ใช้งาน ซึ่งเป็นผู้ที่กำลังสร้าง mutex อยู่ โดยทั่วไป mutex ที่ถูกนิยามโดยไลบรารี และถูกใช้โดยเธรดของแอ็พพลิเคชันจะใช้โปรโตคอลการสืบทอด ขณะที่ mutex ที่สร้างอยู่ภายในแอ็พพลิเคชันโปรแกรม จะใช้โปรแกรมการป้องกัน

ในโปรแกรมผลการทำงานที่สำคัญ ข้อควรพิจารณาด้านผลการทำงาน อาจยังมีอิทธิพลต่อตัวเลือก ในการนำไปปฏิบัติส่วนใหญ่โดยเฉพาะใน AIX การเปลี่ยนแปลงลำดับความสำคัญของเธรดจะส่งผลทำให้เกิดการเรียกระบบ ดังนั้น โปรโตคอล mutex สองตัวจะแตกต่างกันในเรื่องของจำนวนของการเรียกของระบบที่โปรโตคอลสร้างไว้ ดังนี้:

- การใช้โปรโตคอลการสืบทอด การเรียกของระบบจะถูกทำขึ้นในแต่ละครั้งที่เธรดถูกบล็อก เมื่อพยายามล็อก mutex
- สำหรับการใช้โปรโตคอลการป้องกัน การเรียกของระบบหนึ่งครั้งจะถูกสร้างขึ้นในแต่ละครั้งที่ mutex ถูกล็อกโดยเธรด

สำหรับในโปรแกรมที่มีผลการทำงานเป็นส่วนสำคัญส่วนใหญ่ โปรโตคอลการสืบทอดจะถูกเลือก เนื่องจาก mutex คืออ็อบเจ็กต์ contention ทีร่มีระดับต่ำ Mutex จะไม่ถูกพักเป็นเวลานาน ดังนั้น จึงไม่เหมือนกับเธรดที่ถูกบล็อก ขณะที่พยายามล็อกเธรดนั้น

หลักการที่เกี่ยวข้อง:

"เธรดการกำหนดเวลา" ในหน้า 501
เธรดสามารถกำหนดตารางเวลาได้ และไลบรารีเธรดจะจัดเตรียมตัวช่วยต่างๆ เพื่อจัดการและควบคุมการกำหนดตารางเวลาของเธรด

รานการของรูทีนย่อยการซิงโครไนซ์

ส่วนนี้แสดงรายการรูทีนย่อยการซิงโครไนซ์

`pthread_mutex_destroy`
: ลบ mutex

`pthread_mutex_init`
: กำหนดค่าเริ่มต้น mutex และตั้งค่าแอ็ตทริบิวต์

`PTHREAD_MUTEX_INITIALIZER`
: กำหนดค่าเริ่มต้นสแตติก mutex ด้วยแอ็ตทริบิวต์เริ่มต้น

`pthread_mutex_lock` หรือ `pthread_mutex_trylock`
: ล็อก mutex

`pthread_mutex_unlock`
: ปลดล็อก mutex

`pthread_mutexattr_destroy`
: ลบอ็อบเจ็กต์แอ็ตทริบิวต์ mutex

`pthread_mutexattr_init`
: สร้างอ็อบเจ็กต์แอ็ตทริบิวต์ mutex และกำหนดค่าเริ่มต้นด้วย ค่าเริ่มต้น

`pthread_cond_destroy`
: ลบค่าเงื่อนไข

`pthread_cond_init`
: กำหนดค่าเริ่มต้นตัวแปรเงื่อนไขและตั้งค่าแอ็ตทริบิวต์

`PTHREAD_COND_INITIALIZER`
: กำหนดค่าเริ่มต้นตัวแปรเงื่อนไขสแตติกด้วยแอ็ตทริบิวต์เริ่มต้น

`pthread_cond_signal` หรือ `pthread_cond_broadcast`
: บล็อกเธรดหนึ่งเธรดหรือมากกว่านั้นที่บล็อกบนเงื่อนไข

pthread_cond_wait หรือ pthread_cond_timedwait
บล็อกเธรดการเรียกบนเงื่อนไข

pthread_condattr_destroy
ลบอ็อบเจ็กต์แอ็ตทริบิวต์เงื่อนไข

pthread_condattr_init
สร้างอ็อบเจ็กต์แอ็ตทริบิวต์เงื่อนไขและกำหนดค่าเริ่มต้นด้วย ค่าเริ่มต้น

การกำหนดค่าเริ่มต้นครั้งเดียว

บางไลบรารี C ถูกออกแบบสำหรับการกำหนดค่าเริ่มต้น แบบไดนามิก ซึ่งการกำหนดค่าเริ่มต้นโกลบอลสำหรับไลบรารีถูกดำเนินการ เมื่อโพรซีเดอร์แรกในไลบรารีถูกเรียก

ในโปรแกรมแบบ a single-thread ถูกทำขึ้นโดยใช้ตัวแปรสแตติกซึ่งค่าถูกตรวจสอบ เมื่อเข้าสู่แต่ละรูทีน ตามตัวอย่างของโค้ดดังต่อไปนี้:

```
static int isInitialized = 0;
extern void Initialize();

int function()
{
        if (isInitialized == 0) {
                Initialize();
                isInitialized = 1;
        }
        ...
}
```

สำหรับการกำหนดค่าเริ่มต้นไลบรารีแบบไดนามิกในโปรแกรมแบบมัลติเธรด แฟล็กการกำหนดค่าเริ่มต้นแบบง่ายไม่เพียงพอ แฟล็กนี้ต้องถูก ป้องกันการแก้ไขโดยหลายเธรดเรียก ฟังก์ชันไลบรารีพร้อมกัน การป้องกันแฟล็กต้องการการใช้ mutex; อย่างไรก็ตาม mutex ต้องถูกกำหนดค่าก่อนนำมาใช้ ตรวจสอบว่า mutex ถูกกำหนดค่าเริ่มต้นเมื่อต้องการวิธีแก้ปัญหาเรียกซ้ำกับปัญหานี้

เมื่อต้องการให้มีโครงสร้าง แบบเดียวกับในโปรแกรมแบบมัลติเธรด ให้ใช้รูทีนย่อย **pthread_once** หรือมิฉะนั้น การกำหนดค่าเริ่มต้นไลบรารีต้องกระทำ โดยการเรียกแบบ explicit ไปที่ไลบรารีที่เอ็กซ์พอร์ตฟังก์ชันการกำหนดค่าเริ่มต้น ก่อนการใช้ไลบรารี รูทีนย่อย **pthread_once** ยังมีทางเลือกสำหรับการกำหนดค่าเริ่มต้น mutexes และ ตัวแปรเงื่อนไข

อ็อบเจ็กต์การกำหนดค่าเริ่มต้นแบบครั้งเดียว

ความเป็นเอกลักษณ์ของการกำหนดค่าเริ่มต้น ถูกรับประกันโดยอ็อบเจ็กต์ one-time initialization ซึ่งเป็นตัวแปรที่มีชนิดข้อมูล **pthread_once_t** ใน AIX และการนำไปปฏิบัติอื่น ส่วนใหญ่ของไลบรารีเธรด ชนิดข้อมูล **pthread_once_t** เป็น structure

อ็อบเจ็กต์ one-time initialization โดยปกติแล้วคือ ตัวแปรโกลบอล ซึ่งต้องถูกกำหนดค่าเริ่มต้นด้วยแมโคร **PTHREAD_ONCE_INIT** ตามตัวอย่างดังต่อไปนี้:

```
static pthread_once_t once_block = PTHREAD_ONCE_INIT;
```

การกำหนดค่าเริ่มต้นยังสามารถทำได้ใน initial thread หรือในเธรดอื่น อ็อบเจ็กต์ one-time initialization หลายอ็อบเจ็กต์สามารถถูกใช้ใน โปรแกรมเดียวกัน ข้อกำหนดเดียวที่มีคือ อ็อบเจ็กต์ one-time initialization ต้องถูกกำหนดค่าเริ่มต้นด้วยแมโคร

รูทีนการกำหนดค่าเริ่มต้นแบบครั้งเดียว

รูทีนย่อย **pthread_once** เรียก รูทีนการกำหนดค่าเริ่มต้นที่ระบุที่สัมพันธ์กับอ็อบเจ็กต์ one-time initialization object ที่ระบุ ถ้าเป็นการเรียกครั้งแรก หรือมิฉะนั้น จะไม่มีการทำงานใด รูทีนการกำหนดค่าเริ่มต้นเดียวกันต้องถูกใช้เสมอกับ อ็อบเจ็กต์ one-time initialization เดิม รูทีนการกำหนดค่าเริ่มต้นต้องมี ต้นแบบดังต่อไปนี้:

```
void init_routine();
```

รูทีนย่อย **pthread_once** ไม่มี cancelation point อย่างไรก็ตาม รูทีนการกำหนดค่าเริ่มต้นอาจจัดเตรียม cancelation points และถ้า cancelability ถูกเปิดใช้งานเธรดแรกที่เรียกรูทีนย่อย **pthread_once** อาจถูกยกเลิกระหว่าง การดำเนินการรูทีนการกำหนดค่าเริ่มต้น ในกรณีนี้ รูทีนไม่ถูกพิจารณาว่ามีการดำเนินการ และการเรียกครั้งถัดไปที่รูทีนย่อย **pthread_once** จะมีผลให้มีการเรียกรูทีนการกำหนดค่าเริ่มต้นซ้ำ

ขอแนะนำให้ใช้ cleanup handlers ในรูทีน one-time initialization โดยเฉพาะเมื่อดำเนินการ non-idempotent เช่นเปิดไฟล์ ล็อก mutex หรือจัดสรรหน่วยความจำ

รูทีน One-time initialization สามารถถูกใช้เพื่อกำหนดค่าเริ่มต้น mutexes หรือตัวแปรเงื่อนไข หรือเพื่อทำการกำหนดค่าเริ่มต้นแบบไดนามิก ใน ไลบรารีแบบมัลติเธรด โค้ดแฟรกเมนต์ที่แสดงด้านบน (`void init_routine();`) จะถูกเขียนดังนี้:

```
static pthread_once_t once_block = PTHREAD_ONCE_INIT;
extern void Initialize();

int function()
{
        pthread_once(&once_block, Initialize);
        ...
}
```

หลักการที่เกี่ยวข้อง:

“การยกเลิกเธรด” ในหน้า 466
เธรดจะยกเลิกโดยอัตโนมัติ เมื่อส่งคืนค่าจากรูทีน entry-point

“ข้อมูลเฉพาะเธรด”
แอ็พพลิเคชันจำนวนมากต้องการข้อมูลบางส่วนสำหรับเธรดพื้นฐาน ระหว่างการเรียกฟังก์ชัน

“การเขียนโค้ด reentrant และ threadsafe” ในหน้า 538
ในกระบวนการแบบเธรดเดี่ยว มีได้เพียงหนึ่งโฟลว์ของการควบคุม โค้ดที่เรียกใช้งานโดยกระบวนการเหล่านี้ไม่จำเป็นต้องเป็น reentrant หรือ threadsafe ในโปรแกรมแบบมัลติเธรด ฟังก์ชันเดียวกัน และ รีซอร์สเดียวกันอาจถูกเข้าถึงพร้อมกันโดยโฟลว์ควบคุมหลายโฟลว์

ข้อมูลเฉพาะเธรด

แอ็พพลิเคชันจำนวนมากต้องการข้อมูลบางส่วนสำหรับเธรดพื้นฐาน ระหว่างการเรียกฟังก์ชัน

ตัวอย่างเช่น คำสั่ง **grep** มัลติเธรดใช้หนึ่งเธรด สำหรับแต่ละไฟล์ที่ต้องมี handlers ไฟล์ของเธรดเฉพาะ และรายการ สตริงที่พบ อินเตอร์เฟสของข้อมูลที่ระบุเฉพาะสำหรับเธรด จะถูกจัดเตรียมไว้โดยไลบรารีเธรดเพื่อให้ตรงกับความต้องการ

ข้อมูลที่ระบุเฉพาะสำหรับเธรด อาจมองเห็นเป็นค่าอาร์เรย์สองมิติด้วยที่คีย์ที่ใช้เป็นดัชนีแถว และ ID เธรดเป็นดัชนีคอลัมน์ ข้อมูลที่ระบุเฉพาะสำหรับเธรด *key* จะเป็นอ็อบเจ็กต์ opaque ของชนิดข้อมูล **pthread_key_t** คีย์ที่เหมือนกันนี้สามารถใช้โดยเธรดทั้งหมดในกระบวนการ แม้ว่า เธรดทั้งหมดใช้คีย์เดียวกัน ซึ่งตั้งค่าและเข้าถึงค่าข้อมูลที่ระบุเฉพาะสำหรับเธรดที่ต่างกัน ซึ่งเชื่อมโยงกับคีย์นั้น ข้อมูลที่ระบุเฉพาะสำหรับเธรดคือตัวชี้ void ซึ่งอนุญาตให้อ้างอิงชนิดข้อมูลใดๆ เช่น สตริงหรือโครงสร้างที่ถูกจัดสรรแบบไดนามิก

ในรูปภาพประกอบต่อไปนี้ เธรด T2 มีค่าข้อมูลที่ระบุเฉพาะสำหรับเธรดเท่ากับ 12 ซึ่งเชื่อมโยงกับคีย์ K3 เธรด T4 มีค่าเท่ากับ 2 ซึ่งเชื่อมโยงกับคีย์เดียวกัน

Keys	T1 Thread	T2 Thread	T3 Thread	T4 Thread
K1	6	56	4	1
K2	87	21	0	9
K3	23	**12**	61	**2**
K4	11	76	47	88

การสร้างและการทำลายคีย์

คีย์ข้อมูลที่ระบุเฉพาะสำหรับเธรด ต้องถูกสร้างขึ้นก่อนที่จะนำมาใช้ ค่าเหล่านี้สามารถถูกทำลายได้โดยอัตโนมัติ เมื่อเธรดที่สอดคล้องกันถูกยกเลิก คีย์ยังสามารถถูกทำลายได้ ตามคำร้องขอให้เรียกคืนหน่วยเก็บ

การสร้างคีย์

คีย์ข้อมูลที่ระบุเฉพาะสำหรับเธรด จะถูกสร้างขึ้นโดยเรียกรูทีนย่อย **pthread_key_create** รูทีนย่อยนี้จะส่งคืนคีย์ ข้อมูลที่ระบุเฉพาะสำหรับเธรด จะถูกตั้งค่า **NULL** สำหรับเธรดทั้งหมด ซึ่งรวมถึงเธรดที่ยังไม่ได้สร้างขึ้น

ตัวอย่างเช่น โปรดพิจารณาเธรดสองตัว นั่นคือ *A* และ *B* เธรด *A* จะดำเนินการกับการดำเนินการต่อไปนี้ตามลำดับ:

1. สร้างคีย์ข้อมูลที่ระบุเฉพาะสำหรับเธรด *K*

   เธรด *A* และ *B* สามารถใช้คีย์ *K* ได้ ค่าสำหรับเธรดทั้งสองค่าคือ **NULL**
2. สร้างเธรด *C*

   เธรด *C* ยังสามารถใช้คีย์ *K* ค่าสำหรับเธรด *C* คือ **NULL.**

จำนวนของคีย์ข้อมูลที่ระบุเฉพาะสำหรับเธรดจะถูกจำกัดให้มีค่า 450 ต่อกระบวนการ หมายเลขนี้สามารถดึงข้อมูลได้โดยใช้ค่าคงที่เชิงสัญลักษณ์ **PTHREAD_KEYS_MAX**

รูทีนย่อย **pthread_key_create** ต้องถูกเรียกเพียงหนึ่งครั้ง มิฉะนั้น คีย์อื่นๆ สองคีย์จะถูกสร้างขึ้น ตัวอย่างเช่น ให้พิจารณาชิ้นส่วนของโค้ดต่อไปนี้:

```
/* a global variable */
static pthread_key_t theKey;

/* thread A */
...
pthread_key_create(&theKey, NULL);   /* call 1 */
...
```

```
/* thread B */
...
pthread_key_create(&theKey, NULL);   /* call 2 */
...
```

ในตัวอย่างของเรา เธรด *A* และ *B* รันพร้อมกัน แต่การเรียกที่ 1 เกิดขึ้นก่อนการเรียกที่ 2 การเรียกที่ 1 จะสร้างคีย์ *K1* และ เรียงลำดับในตัวแปร **theKey** การเรียกที่ 2 จะสร้างคีย์*K2* ตัวอื่น และเก็บไว้ในตัวแปร **theKey** ดังนั้น จึงเป็นแทนที่ *K1* จากผลลัพธ์ เธรด *A* จะใช้ *K2* เนื่องเป็น *K1* สถานการณ์นี้ควรหลีกเลี่ยนด้วยเหตุผลต่อไปนี้:

- คีย์ K1 จะหายไป หน่วยเก็บจะไม่ถูกเรียกคืน จนกว่ากระบวนการจะยกเลิก เนื่องจากจำนวนคีย์ถูกจำกัดไว้ คุณอาจไม่มีคีย์เพียงพอ
- ถ้าเธรด *A* เก็บข้อมูลที่ระบุเฉพาะสำหรับเธรดโดยใช้ตัวแปร **theKey** ก่อนการเรียกที่ 2 ข้อมูลจะถูกผูกไว้กับคีย์ K1 หลังจากที่การเรียกที่ 2 ตัวแปร **theKey** จะมี K2 หากเธรด *A* พยายามดึงข้อมูลที่ระบุเฉพาะสำหรับเธรดออกมา ซึ่งจะได้รับค่า **NULL** เสมอ

ตรวจสอบให้มั่นใจว่า คีย์ได้ถูกสร้างขึ้นเฉพาะ และสามารถทำได้ด้วยวิธีต่อไปนี้:

- การใช้ตัวช่วยกำหนดค่าเริ่มต้นเพียงครั้งเดียว
- การสร้างคีย์ก่อนเธรดที่จะใช้ซึ่งมีความเป็นไปได้สูง ตัวอย่างเช่น ขณะที่ใช้พูลของเธรดด้วยข้อมูลที่ระบุเฉพาะสำหรับเธรดเพื่อดำเนินการกับการดำเนินการที่คล้ายกัน พูลของเธรดนี้จะถูกสร้างขึ้นโดยเธรดหนึ่งตัว นั่นคือ เธรดเริ่มต้น (หรือ "ไดรเวอร์" อื่นๆ)

ซึ่งเป็นความรับผิดชอบของโปรแกรมเมอร์เพื่อมั่นใจว่า การสร้างคีย์นั้นไม่ซ้ำกัน ไลบรารีเธรดไม่ได้จัดเตรียมวิธีที่ใช้ตรวจสอบ หากคีย์นั้นถูกสร้างขึ้นมากกว่าหนึ่งครั้ง

รูทีน Destructor

รูทีน destructor อาจเชื่อมโยงกับคีย์ข้อมูลที่ระบุเฉพาะสำหรับเธรดแต่ละตัว ไม่ว่าเธรดจะถูกยกเลิกหรือไม่ก็ตาม หากมีข้อมูลที่ระบุเฉพาะสำหรับเธรดที่ไม่ใช่ค่า **NULL** ข้อมูลสำหรับเธรดนี้จะผูกกับคีย์ใดๆ รูทีน destructor จะเชื่อมโยงกับคีย์ที่เรียก ซึ่งอนุญาตให้ข้อมูลที่ระบุเฉพาะสำหรับเธรดแบบไดนามิกถูกป้อนโดยอัตโนมัติ เมื่อเธรดถูกยกเลิก รูทีน destructor มีหนึ่งพารามิเตอร์ นั่นคือ ค่าของข้อมูลที่ระบุเฉพาะสำหรับเธรด

ตัวอย่างเช่น คีย์ข้อมูลที่ระบุเฉพาะสำหรับเธรด อาจถูกใช้สำหรับบัฟเฟอร์ที่ถูกจัดสรรแบบไดนามิก รูทีน destructor ควรจัดเตรียมไว้ เพื่อมั่นใจว่า เมื่อเธรดยกเลิกบัฟเฟอร์ที่ป้อนแล้ว รูทีน **free** สามารถนำมาใช้ได้ดังนี้:

```
pthread_key_create(&key, free);
```

destructor ที่มีความซับซ้อนมากขึ้นอาจถูกนำมาใช้ ถ้าคำสั่ง **grep** มัลติเธรด ใช้หนึ่งเธรดต่อไฟล์เพื่อสแกน มีข้อมูลเฉพาะเธรดเพื่อเก็บ โครงสร้างที่มีบัฟเฟอร์งาน และ file descriptor ของเธรด รูทีน destructor อาจเป็นดังนี้:

```
typedef struct {
        FILE *stream;
        char *buffer;
} data_t;
...

void destructor(void *data)
{
        fclose(((data_t *)data)->stream);
```

```
        free(((data_t *)data)->buffer);
        free(data);
        *data = NULL;
}
```

การเรียก Destructor สามารถทำซ้ำได้มากที่สุดสี่ครั้ง

การทำลายคีย์

คีย์ข้อมูลที่ระบุเฉพาะสำหรับเธรดสามารถถูกทำลายได้โดยเรียกรูทีนย่อย **pthread_key_delete** รูทีนย่อย **pthread_key_delete** ไม่ได้เรียกรูทีน destructor สำหรับเธรดแต่ละตัวที่มีข้อมูล หลังจากที่ทำลายคีย์ข้อมูลแล้ว คีย์นั้นสามารถนำกลับมาใช้ได้โดยการเรียกรูทีนย่อย **pthread_key_create** ตัวอื่น ดังนั้น รูทีนย่อย **pthread_key_delete** จะมีประโยชน์โดยเฉพาะ เมื่อใช้คีย์ข้อมูลจำนวนมาก ตัวอย่างเช่น ในชิ้นส่วนของโค้ดต่อไปนี้ ลูปจะไม่สิ้นสุด:

```
/* bad example - do not write such code! */
pthread_key_t key;

while (pthread_key_create(&key, NULL))
        pthread_key_delete(key);
```

การใช้ข้อมูลเฉพาะเธรด

ข้อมูลที่ระบุเฉพาะสำหรับเธรดจะเข้าถึงได้โดยใช้รูทีนย่อย **pthread_getspecific** และ **pthread_setspecific** รูทีนย่อย **pthread_getspecific** จะอ่านค่าที่ผูกกับคีย์ที่ระบุไว้ และเฉพาะกับเธรดการเรียก รูทีนย่อย **pthread_setspecific** จะถูกตั้งค่าไว้

การตั้งค่าค่าความสำเร็จ

ค่าที่ผูกกับคีย์เฉพาะควรเป็นตัวชี้ ซึ่งสามารถชี้ไปยังชนิดของข้อมูลใดๆ ข้อมูลที่ระบุเฉพาะสำหรับเธรดจะถูกใช้ สำหรับหน่วยเก็บที่จัดสรรแบบไดนามิก ดังที่แสดงอยู่ในชิ้นส่วนของโค้ดต่อไปนี้:

```
private_data = malloc(...);
pthread_setspecific(key, private_data);
```

ขณะที่ตั้งค่า ค่าก่อนหน้านี้จะหายไป ตัวอย่างเช่น ชิ้นส่วนของโค้ดต่อไปนี้ ค่าของตัวชี้ **เก่า** จะหายไป และหน่วยเก็บที่ถูกชี้อาจไม่ได้สามารถกู้คืนได้:

```
pthread_setspecific(key, old);
...
pthread_setspecific(key, new);
```

ซึ่งเป็นความรับผิดชอบของโปรแกรมเมอร์ที่จะดึงค่าข้อมูลที่ระบุเฉพาะสำหรับเธรด ที่เป็นค่าเก่าเพื่อเรียกคืนหน่วยเก็บก่อนที่จะตั้งค่าใหม่ ตัวอย่างเช่น มีความเป็นไปได้ที่จะนำรูทีน **swap_specific** ไปใช้งานด้วยวิธีต่อไปนี้:

```
int swap_specific(pthread_key_t key, void **old_pt, void *new)
{
        *old_pt = pthread_getspecific(key);
        if (*old_pt == NULL)
                return -1;
        else
                return pthread_setspecific(key, new);
}
```

รูทีนบางตัวไม่มีอยู่ในไลบรารีเธรด เนื่องจากรูทีนนั้นไม่จำเป็นต้องดึงค่าก่อนหน้านี้ของข้อมูลที่ระบุเฉพาะสำหรับเธรด ตัวอย่างเช่น หากกรณีนี้เกิดขึ้น เมื่อข้อมูลที่ระบุเฉพาะสำหรับเธรดคือตัวชี้ไปยังตำแหน่งเฉพาะ ในพูลหน่วยความจำที่จัดสรรโดยเธรดเริ่มต้น

การใช้รูทีน destructor

ขณะที่ใช้ข้อมูลที่ระบุเฉพาะสำหรับเธรดที่จัดสรรแล้ว โปรแกรมเมอร์ต้องจัดเตรียมรูทีน destructor ขณะเรียกรูทีนย่อย **pthread_key_create** โปรแกรมเมอร์ยังต้องมั่นใจว่า ขณะที่รีลีสหน่วยเก็บที่จัดสรรไว้สำหรับข้อมูลที่ระบุเฉพาะสำหรับเธรด ตัวชี้จะถูกตั้งค่าเป็น **NULL** มิฉะนั้น รูทีน destructor อาจถูกเรียกด้วยพารามิเตอร์ที่ผิดกฎเกณฑ์ได้ ตัวอย่างเช่น:

```
pthread_key_create(&key, free);
...

...
private_data = malloc(...);
pthread_setspecific(key, private_data);
...

/* bad example! */
...
pthread_getspecific(key, &data);
free(data);
...
```

เมื่อเธรดยกเลิกแล้ว รูทีน destructor จะถูกเรียกสำหรับข้อมูลที่ระบุเฉพาะสำหรับเธรด เนื่องจากค่าคือตัวชี้ไปยังหน่วยความจำที่รีลีสแล้ว ข้อผิดพลาดจะเกิดขึ้น หากต้องการแก้ไขปัญหานี้ ชิ้นส่วนของโค้ดต่อไปนี้ควรเข้าแทนที่:

```
/* better example! */
...
pthread_getspecific(key, &data);
free(data);
pthread_setspecific(key, NULL);
...
```

เมื่อเธรดยกเลิกแล้ว รูทีน destructor จะไม่ถูกเรียก เนื่องจากไม่มีข้อมูลที่ระบุเฉพาะสำหรับเธรด

การใช้ค่าที่ไม่ใช่ตัวชี้

แม้ว่ามีความเป็นไปได้ที่จะเก็บค่าที่ไม่ใช้ตัวชี้ไว้ แต่ไม่ขอแนะนำด้วยเหตุผลดังต่อไปนี้:

- การคัดเลือกตัวชี้ในชนิดสเกลาร์อาจไม่สามารถพอร์ตได้
- ค่าตัวชี้ **NULL** เป็นการพึ่งพาการนำไปปฏิบัติ ระบบหลายๆ ระบบจะกำหนดตัวชี้ **NULL** ให้มีค่าที่ไม่ใช่ศูนย์

ถ้าคุณมั่นใจว่า โปรแกรมของคุณไม่เคยถูกพอร์ตไปยังระบบอื่น คุณอาจใช้ค่าตัวเลขสำหรับข้อมูลที่ระบุเฉพาะสำหรับเธรด

หลักการที่เกี่ยวข้อง:

“การรวมเธรด” ในหน้า 497
การรวมเธรดหมายถึงการรอให้เธรดจบการทำงาน ซึ่งอาจถูกมอบเป็นการใช้งานเฉพาะของตัวแปรเงื่อนไข

“การกำหนดค่าเริ่มต้นครั้งเดียว” ในหน้า 510
บางไลบรารี C ถูกออกแบบสำหรับการกำหนดค่าเริ่มต้น แบบไดนามิก ซึ่งการกำหนดค่าเริ่มต้นโกลบอลสำหรับไลบรารีถูกดำเนินการ เมื่อโพรซีเดอร์แรกในไลบรารีถูกเรียก

“การสร้างอ็อบเจ็กต์การซิงโครไนซ์ที่ซับซ้อน”
รูทีนย่อยที่จัดเตรียมในไลบรารีเธรดสามารถถูก ใช้เป็นหลักในการสร้างอ็อบเจ็กต์การซิงโครไนซ์ที่ซับซ้อนมากขึ้น
“รายการของรูทีนย่อย threads-processes interactions” ในหน้า 524
ส่วนนี้แสดงรายการรูทีนย่อย threads-processes interactions

การสร้างอ็อบเจ็กต์การซิงโครไนซ์ที่ซับซ้อน

รูทีนย่อยที่จัดเตรียมในไลบรารีเธรดสามารถถูก ใช้เป็นหลักในการสร้างอ็อบเจ็กต์การซิงโครไนซ์ที่ซับซ้อนมากขึ้น

ล็อกเป็นเวลานาน

mutexes ที่จัดเตรียมโดยไลบรารีเธรดเป็นอ็อบเจ็กต์ low-contention และไม่ถูกคงไว้เป็นเวลานาน การล็อกเป็นเวลานานถูก ใช้ กับ mutexes และตัวแปรเงื่อนไข เพื่อที่การล็อกเป็นเวลานานสามารถถูก คงไว้ได้นานโดยไม่มีผลกับประสิทธิภาพของ โปรแกรม ไม่ควรใช้การล็อกเป็นเวลานาน ถ้ามีการเปิดใช้งานความสามารถในการยกเลิก

long lock มีชนิดข้อมูล **long_lock_t** ซึ่งต้องถูกกำหนดค่าโดยรูทีน **long_lock_init** รูทีนย่อย **long_lock, long_trylock** และ **long_unlock** ดำเนินการเหมือนกับรูทีนย่อย **pthread_mutex_lock, pthread_mutex_trylock** และ **pthread_mutex_unlock**

ตัวอย่างดังต่อไปนี้แสดงการใช้ตัวแปรเงื่อนไข ทั่วไป ในตัวอย่างนี้ เจ้าของล็อกไม่ได้ถูกตรวจสอบ ดังนั้น ทุกเธรด สามารถ ปลดล็อก ล็อกที่มีได้ การจัดการข้อผิดพลาดและการจัดการการยกเลิก ไม่ถูกดำเนินการ

```
typedef struct {
        pthread_mutex_t lock;
        pthread_cond_t cond;
        int free;
        int wanted;
} long_lock_t;

void long_lock_init(long_lock_t *ll)
{
        pthread_mutex_init(&ll->lock, NULL);
        pthread_cond_init(&ll->cond);
        ll->free = 1;
        ll->wanted = 0;
}

void long_lock_destroy(long_lock_t *ll)
{
        pthread_mutex_destroy(&ll->lock);
        pthread_cond_destroy(&ll->cond);
}

void long_lock(long_lock_t *ll)
{
        pthread_mutex_lock(&ll->lock);
        ll->wanted++;
        while(!ll->free)
                pthread_cond_wait(&ll->cond);
        ll->wanted--;
        ll->free = 0;
        pthread_mutex_unlock(&ll->lock);
}
```

```
int long_trylock(long_lock_t *ll)
{
        int got_the_lock;

        pthread_mutex_lock(&ll->lock);
        got_the_lock = ll->free;
        if (got_the_lock)
                ll->free = 0;
        pthread_mutex_unlock(&ll->lock);
        return got_the_lock;
}

void long_unlock(long_lock_t *ll)
{
        pthread_mutex_lock(&ll->lock);
        ll->free = 1;
        if (ll->wanted)
                pthread_cond_signal(&ll->cond);
        pthread_mutex_unlock(&ll->lock);
}
```

เซมาฟอร์

Semaphores ทั่วไปในระบบ UNIX คือหน่วยบริการการซิงโครไนซ์ระหว่างกระบวนการ สำหรับการใช้งานเจาะจง คุณสามารถสร้าง interthread semaphores

semaphore มีชนิดข้อมูลเป็น **sema_t** ซึ่งต้องถูกกำหนดค่าเริ่มต้นโดยรูทีน **sema_init** และทำลายด้วยรูทีน **sema_destroy** การดำเนินการ semaphore wait และ semaphore post ถูกดำเนินการตามลำดับโดยรูทีน **sema_p** และ **sema_v**

ในการนำไปปฏิบัติพื้นฐานดังต่อไปนี้ ไม่มีการจัดการข้อผิดพลาด แต่การยกเลิกถูกจัดการอย่างเหมาะสม ด้วยตัวจัดการ cleanup เมื่อจำเป็น:

```
typedef struct {
        pthread_mutex_t lock;
        pthread_cond_t cond;
        int count;
} sema_t;

void sema_init(sema_t *sem)
{
        pthread_mutex_init(&sem->lock, NULL);
        pthread_cond_init(&sem->cond, NULL);
        sem->count = 1;
}

void sema_destroy(sema_t *sem)
{
        pthread_mutex_destroy(&sem->lock);
        pthread_cond_destroy(&sem->cond);
}

void p_operation_cleanup(void *arg)
{
```

```
        sema_t *sem;

        sem = (sema_t *)arg;
        pthread_mutex_unlock(&sem->lock);
}

void sema_p(sema_t *sem)
{
        pthread_mutex_lock(&sem->lock);
        pthread_cleanup_push(p_operation_cleanup, sem);
        while (sem->count <= 0)
                pthread_cond_wait(&sem->cond, &sem->lock);
        sem->count--;
        /*
         *  Note that the pthread_cleanup_pop subroutine will
         *  execute the p_operation_cleanup routine
         */
        pthread_cleanup_pop(1);
}

void sema_v(sema_t *sem)
{
    pthread_mutex_lock(&sem->lock);
        if (sem->count <=0)
        pthread_cond_signal(&sem->cond);
    sem->count++;
    pthread_mutex_unlock(&sem->lock);
}
```

ตัวนับระบุจำนวนผู้ใช้ที่ได้รับอนุญาต ให้ใช้ semaphore ค่าจะไม่เป็นลบ ดังนั้นจึงไม่มีการ ระบุจำนวนของผู้ใช้ที่รออยู่ เหมือนกับ semaphores ตามปกติ การนำไปปฏิบัติ นี้จัดเตรียมวิธีแก้ปัญหาทั่วไปกับหลายๆ ปัญหาในการเรียกใช้งาน กับรูทีนย่อย **pthread_cond_wait** การรอของ semaphore สามารถยกเลิกได้ เนื่องจากรูทีนย่อย **pthread_cond_wait** ได้จัดเตรียมจุดการยกเลิก

Write-Priority Read/Write Locks

write-priority read/write lock จัดเตรียม การเข้าถึงแบบอ่านอย่างเดียวพร้อมกันหลายเธรด กับทรัพยากรที่ป้องกัน และสำหรับเธรดเดียวเป็นการเข้าถึงเพื่อเขียนทรัพยากรขณะทำการแยกการอ่าน เมื่อส่วนที่ เขียนปล่อยล็อก ส่วนที่รอการเขียนอื่นจะรับล็อกก่อน ส่วนการอ่าน ที่รออยู่ Write-priority read/write locks โดยปกติถูกใช้เพื่อป้องกัน ทรัพยากรที่ถูกอ่านมากกว่าเขียน

write-priority read/write lock มีชนิดข้อมูลเป็น **rwlock_t** ซึ่งต้องถูกกำหนดค่าเริ่มต้นโดยรูทีน **rwlock_init** รูทีน **rwlock_lock_read** ทำการล็อก สำหรับส่วนการอ่าน (สามารถมีหลายส่วนการอ่านได้) รูทีน **rwlock_unlock_read** ทำการปลดล็อก รูทีน **rwlock_lock_write** ทำการล็อกสำหรับส่วนการเขียน, รูทีน **rwlock_unlock_write** ทำการปลดล็อก รูทีนการปลดล็อกที่ถูกต้อง (สำหรับส่วนอ่านหรือเขียน) ต้องถูกเรียก

ในตัวอย่างดังต่อไปนี้ เจ้าของล็อกไม่ได้ถูกตรวจสอบ ดังนั้น ทุกเธรด สามารถปลดล็อก ล็อกที่มีได้ รูทีน เช่นรูทีนย่อย **pthread_mutex_trylock** หายไปและไม่มีการดำเนินการจัดการ ข้อผิดพลาด แต่การยกเลิกถูกจัดการอย่างถูกต้องด้วยตัวจัดการ cleanup เมื่อจำเป็น

```
typedef struct {
        pthread_mutex_t lock;
        pthread_cond_t rcond;
        pthread_cond_t wcond;
        int lock_count;  /* < 0 .. held by writer             */
                         /* > 0 .. held by lock_count readers */
                         /* = 0 .. held by nobody             */
        int waiting_writers;  /* count of wating writers      */
} rwlock_t;

void rwlock_init(rwlock_t *rwl)
{
        pthread_mutex_init(&rwl->lock, NULL);
        pthread_cond_init(&rwl->wcond, NULL);
        pthread_cond_init(&rwl->rcond, NULL);
        rwl->lock_count = 0;
        rwl->waiting_writers = 0;
}

void waiting_reader_cleanup(void *arg)
{
        rwlock_t *rwl;

        rwl = (rwlock_t *)arg;
        pthread_mutex_unlock(&rwl->lock);
}

void rwlock_lock_read(rwlock_t *rwl)
{
        pthread_mutex_lock(&rwl->lock);
        pthread_cleanup_push(waiting_reader_cleanup, rwl);
        while ((rwl->lock_count < 0) && (rwl->waiting_writers))
                pthread_cond_wait(&rwl->rcond, &rwl->lock);
        rwl->lock_count++;
        /*
         *  Note that the pthread_cleanup_pop subroutine will
         *  execute the waiting_reader_cleanup routine
         */
        pthread_cleanup_pop(1);
}

void rwlock_unlock_read(rwlock_t *rwl)
{
        pthread_mutex_lock(&rwl->lock);
        rwl->lock_count--;
        if (!rwl->lock_count)
                pthread_cond_signal(&rwl->wcond);
        pthread_mutex_unlock(&rwl->lock);
}

void waiting_writer_cleanup(void *arg)
{
        rwlock_t *rwl;

        rwl = (rwlock_t *)arg;
```

```
        rwl->waiting_writers--;
        if ((!rwl->waiting_writers) && (rwl->lock_count >= 0))
                /*
                         * This only happens if we have been canceled
                         */
                        pthread_cond_broadcast(&rwl->wcond);
                        pthread_mutex_unlock(&rwl->lock);
}

void rwlock_lock_write(rwlock_t *rwl)
{
        pthread_mutex_lock(&rwl->lock);
        rwl->waiting_writers++;
        pthread_cleanup_push(waiting_writer_cleanup, rwl);
        while (rwl->lock_count)
                pthread_cond_wait(&rwl->wcond, &rwl->lock);
        rwl->lock_count = -1;
        /*
         *  Note that the pthread_cleanup_pop subroutine will
         *  execute the waiting_writer_cleanup routine
         */
        pthread_cleanup_pop(1);
}

void rwlock_unlock_write(rwlock_t *rwl)
{
        pthread_mutex_lock(&rwl->lock);
        l->lock_count = 0;
        if (!rwl->wating_writers)
                pthread_cond_broadcast(&rwl->rcond);
        else
                pthread_cond_signal(&rwl->wcond);
        pthread_mutex_unlock(&rwl->lock);
}
```

ส่วนการอ่านเท่านั้นที่ถูกนับ เมื่อนับถึงศูนย์ ส่วนการเขียนที่รออยู่อาจใช้ล็อก มีเพียงหนึ่งส่วนการเขียนเท่านั้นถือมีล็อกไว้ได้ เมื่อ ล็อกถูกปล่อยโดยส่วนการเขียน ส่วนการเขียนอื่นจะถูกเรียก ถ้า มีอยู่ มิฉะนั้น ส่วนการอ่านที่รออยู่ทั้งหมดจะถูกเรียก

รูทีนการล็อกสามารถถูกยกเลิกได้ เนื่องจากรูทีน เรียกรูทีนย่อย **pthread_cond_wait** ดังนั้นตัวจัดการ Cleanup จึงถูกลงทะเบียนก่อนการเรียกรูทีนย่อย

หลักการที่เกี่ยวข้อง:

"ข้อมูลเฉพาะเธรด" ในหน้า 511

แอ็พพลิเคชันจำนวนมากต้องการข้อมูลบางส่วนสำหรับเธรดพื้นฐาน ระหว่างการเรียกฟังก์ชัน

"ภาพรวมการซิงโครไนซ์" ในหน้า 474

ผลประโยชน์หลักส่วนหนึ่งที่ได้รับจากการใช้เธรดคือ ง่ายต่อการใช้สิ่งอำนวยความสะดวกในการประสานเวลา

"รายการของรูทีนย่อย threads-processes interactions" ในหน้า 524

ส่วนนี้แสดงรายการรูทีนย่อย threads-processes interactions

การจัดการสัญญาณ

สัญญาณในกระบวนการแบบมัลติเธรดเป็นส่วนขยายของ สัญญาณในโปรแกรมแบบเธรดเดียวดั้งเดิม

การจัดการสัญญาณในกระบวนการแบบมัลติเธรดถูกแบ่งใช้โดย ระดับกระบวนการ และระดับเธรด และประกอบด้วยต่อไปนี้:

- handler สัญญาณต่อกระบวนการ
- สัญลักษณ์ตัวพรางต่อเธรด
- การส่งแต่ละสัญญาณแบบเดี่ยว

ตัวจัดการสัญญาณและมาสก์สัญญาณ

handler สัญญาณจะถูกคงไว้ที่ระดับของกระบวนการ ซึ่งขอแนะนำให้ใช้รูทีนย่อย **sigwait** ขณะรอสัญญาณ รูทีนย่อย **sigaction** จะไม่ถูกแนะนำให้ใช้ เนื่องจากรายการของ handler สัญญาณจะถูกคงไว้ที่ระดับของกระบวนการ และเธรดใดๆ ภายในกระบวนการอาจเปลี่ยนสัญญาณนั้น หากเธรดสองตัวตั้งค่า handler สัญญาณสำหรับสัญญาณเดียวกัน เธรดล่าสุดที่เรียกว่ารูทีนย่อย **sigaction** จะแทนทับค่าติดตั้งของการเรียกเธรดก่อนหน้านี้ และโดยส่วนใหญ่แล้ว ลำดับในเธรดที่กำหนดตารางเวลาไว้จะไม่สามารถคาดการณ์ได้

ตัวพรางสัญญาณจะถูกรักษาไว้ที่ระดับของเธรด เธรดแต่ละตัว สามารถมีชุดของสัญญาณของตนเองซึ่งจะถูกบล็อกไว้จากการส่ง รูทีนย่อย **sigthreadmask** ต้องนำมาใช้เพื่อขอรับและตั้งค่าตัวพรางสัญลักษณ์ของการเรียกเธรด รูทีนย่อย **sigprocmask** ต้องไม่ถูกใช้ในโปรแกรมแบบมัลติเธรด เนื่องจากลักษณะการทำงานที่ไม่คาดคิดอาจเกิดขึ้นได้

รูทีนย่อย **pthread_sigmask** จะคล้ายคลึงกับรูทีนย่อย **sigprocmask** พารามิเตอร์ และการใช้รูทีนย่อยทั้งสองตัวจะเป็นการใช้เฉพาะ ขณะที่พอร์ตโค้ดที่มีอยู่ไปยังไลบรารีเธรดที่สนับสนุน คุณสามารถแทนที่รูทีนย่อย **sigprocmask** ด้วยรูทีนย่อย **pthread_sigmask**

การสร้างสัญญาณ

สัญญาณที่ถูกสร้างโดยการดำเนินการบางอย่างที่สามารถเป็นแอ็ตทริบิวต์ให้กับเธรดเฉพาะได้ เช่น ข้อบกพร่องของฮาร์ดแวร์ จะถูกส่งไปยังเธรด ที่เป็นสาเหตุทำให้สัญญาณถูกสร้างขึ้น สัญญาณที่สร้างขึ้นในการเชื่อมโยงกับ ID ของกระบวนการ ID กลุ่มของกระบวนการ หรือเหตุการณ์แบบอะซิงโครนัส (เช่น กิจกรรมของเทอร์มินัล) จะถูกส่งไปยังกระบวนการ

- รูทีนย่อย **pthread_kill** จะส่งสัญญาณไปยังเธรด เนื่องจาก ID เธรดจะระบุเธรดภายในกระบวนการ รูทีนย่อยนี้สามารถส่งสัญญาณไปยังเธรดภายในกระบวนการเดียวกันได้เท่านั้น
- รูทีนย่อย **kill** (และคำสั่ง **kill**) จะส่งสัญญาณไปยังกระบวนการ เธรดสามารถส่งสัญญาณ **Signal** ไปยังกระบวนการโดยเรียกใช้งานการเรียกต่อไปนี้:

  ```
  kill(getpid(), Signal);
  ```

- รูทีนย่อย **raise** ไม่สามารถนำมาใช้เพื่อส่งสัญญาณไปยังกระบวนการของการเรียกเธรด รูทีนย่อย **raise** จะส่งสัญญาณไปยังการเรียกเธรด ตามที่อยู่ในการเรียกต่อไปนี้ :

  ```
  pthread_kill(pthread_self(), Signal);
  ```

  นี่จะช่วยทำให้มั่นใจว่า สัญญาณจะถูกส่งไปยังตัวเรียกของรูทีนย่อย **raise** ดังนั้น รูทีนไลบรารีที่เขียนไว้สำหรับโปรแกรมแบบมัลติเธรด สามารถพอร์ตไปยังระบบแบบมัลติเธรดได้ง่ายขึ้น เนื่องจากรูทีนย่อย **raise** จะทำตามคำสั่งเพื่อส่งสัญญาณไปยังตัวเรียก

- รูทีนย่อย **alarm** จะร้องขอสัญญาณที่ส่งไปยังกระบวนการในภายหลัง และสถานะของการเตือนที่คงไว้ที่ระดับของกระบวนการ ดังนั้น เธรดล่าสุดที่เรียกรูทีนย่อย **alarm** จะแทนทับค่าติดตั้งของเธรดอื่นในกระบวนการ ในโปรแกรมแบบมัลติเธรด สัญญาณ **SIGALRM** จะไม่ถูกส่งไปยังเธรดที่เรียกรูทีนย่อย **alarm** การเรียกเธรดอาจถูกยกเลิก และ ไม่สามารถรับสัญญาณ

การจัดการกับสัญญาณ

handler สัญญาณจะถูกเรียก ภายในเธรดที่สัญญาณนั้นถูกส่ง ข้อจำกัดต่อไปนี้ที่มีกับ handler สัญญาณ จะถูกแนะนำโดยไลบรารีเธรด:

- handler สัญญาณอาจเรียกรูทีนย่อย **longjmp** หรือรูทีนย่อย **siglongjmp** หากการเรียกที่สอดคล้องกันกับรูทีนย่อย **setjmp** หรือรูทีนย่อย **sigsetjmp** จะถูกดำเนินการในเธรดเดียวกัน

  โดยปกติแล้ว โปรแกรมที่ต้องการรอสัญญาณจะติดตั้ง handler สัญญาณที่เรียกรูทีนย่อย **longjmp** เพื่อดำเนินการกับการประมวลผลที่จุดที่รูทีนย่อย **setjmp** ที่สอดคล้องกันถูกเรียก ซึ่งไม่สามารถทำได้ในโปรแกรมแบบมัลติเธรด เนื่องจากสัญญาณอาจถูกส่งไปยังเธรดที่ไม่ใช่เธรดที่เรียกรูทีนย่อย **setjmp** ดังนั้น จึงเป็นสาเหตุที่ handler เรียกใช้งานด้วยเธรดที่ไม่ถูกต้อง

  หมายเหตุ: การใช้ **longjmp** จาก handler สัญญาณอาจส่งผลถึงลักษณะการทำงานที่ไม่ได้นิยามไว้
- ไม่มีรูทีนย่อย **pthread** ที่สามารถเรียกได้จาก handler สัญญาณ การเรียกรูทีน **pthread** จาก handler สัญญาณสามารถนำไปสู่ deadlock ของแอ็พพลิเคชันได้

หากต้องการอนุญาตให้เธรดรอสัญญาณที่ถูกสร้างแบบอะซิงโครนัส ไลบรารีเธรดจะจัดเตรียมรูทีนย่อย **sigwait** รูทีนย่อย **sigwait** จะบล็อกการเรียกเธรดจนกว่าสัญญาณที่รอจะถูกส่งไปยังกระบวนการ หรือเธรด ซึ่งต้องไม่ใช่ handler สัญญาณที่ติดตั้งไว้บนสัญญาณที่รอคอยการใช้รูทีนย่อย **sigwait**

โดยปกติแล้ว โปรแกรมอาจสร้างเธรดเฉพาะงาน เพื่อรอสัญญาณที่สร้างแบบอะซิงโครนัส เช่น ลูปของเธรดสำหรับการเรียกรูทีนย่อย **sigwait** และการจัดการกับสัญญาณ ซึ่งแนะนำว่า เธรดจะบล็อกสัญญาณทั้งหมด ชิ้นส่วนโค้ดต่อไปนี้ จะแสดงตัวอย่างของเธรดที่รอสัญญาณ:

```
#include <pthread.h>
#include <signal.h>

static pthread_mutex_t mutex;
sigset_t set;
static int sig_cond = 0;

void *run_me(void *id)
{
       int sig;
       int err;
       sigset_t sigs;
       sigset_t oldSigSet;
       sigfillset(&sigs);
       sigthreadmask(SIG_BLOCK, &sigs, &oldSigSet);

       err = sigwait(&set, &sig);

       if(err)
       {
              /* do error code */
       }
```

```
        else
        {
                printf("SIGINT caught\n");
                pthread_mutex_lock(&mutex);
                sig_cond = 1;
                pthread_mutex_unlock(&mutex);
        }

        return;
}

main()
{
        pthread_t tid;

        sigemptyset(&set);
        sigaddset(&set, SIGINT);
        pthread_sigmask(SIG_BLOCK, &set, 0);

        pthread_mutex_init(&mutex, NULL);

        pthread_create(&tid, NULL, run_me, (void *)1);

        while(1)
        {
                sleep(1);
                /* or so something here */

                pthread_mutex_lock(&mutex);
                if(sig_cond)
                {
                        /* do exit stuff */
                        return;
                }
                pthread_mutex_unlock(&mutex);

        }

}
```

ถ้าเธรดตั้งแต่หนึ่งตัวขึ้นไปเรียกรูทีนย่อย **sigwait** มีเพียงแค่หนึ่งการเรียกเท่านั้นที่จะส่งคืน เมื่อการจับคู่สัญญาณถูกส่งออก เธรดที่ถูกเรียกจะไม่สามารถคาดการณ์ได้ ถ้าเธรดต้องการทำ **sigwait** พร้อมกับจัดการสัญญาณอื่นๆ บางอย่างที่ไม่ได้ทำ **sigwait** ไว้ handler สัญญาณที่ผู้ใช้กำหนดเองจำเป็นต้องบล็อกสัญญาณ sigwaiter สำหรับการจัดการให้ถูกต้อง หมายเหตุ รูทีนย่อย **sigwait** จะจัดเตรียมจุดของการยกเลิก

เนื่องจากเธรดเฉพาะงานไม่ใช่ handler สัญญาณจริง ซึ่งอาจเป็นสัญญาณที่มีเงื่อนไขกับเธรดอื่นใด และมีความเป็นไปได้ที่จะนำรูทีน **sigwait_multiple** ที่ต้องใช้เรียกใช้เธรดทั้งหมดสำหรับสัญญาณเฉพาะ ตัวเรียกแต่ละตัวของรูทีน **sigwait_multiple** จะลงทะเบียนชุดของสัญญาณ ตัวเรียกจะรอตัวแปรเงื่อนไข เธรดเดี่ยวจะเรียกรูทีนย่อย **sigwait** บน union ของสัญญาณที่ลงทะเบียนไว้ทั้งหมด เมื่อการเรียกรูทีนย่อย **sigwait** ส่งคืนค่า สถานะที่เหมาะสมจะถูกตั้งค่าและตัวแปรเงื่อนไข ที่กระจายข้อความ ตัวเรียกใหม่ในรูทีนย่อย **sigwait_multiple** จะเป็นสาเหตุทำให้การเรียกรูทีนย่อย **sigwait** ที่คงค้างอยู่ถูกยกเลิก และออกคำสั่งให้อัพเดตชุดของสัญญาณที่รออยู่อีกครั้ง

ส่งส่งสัญญาณ

สัญญาณจะถูกส่งไปยังเธรด จนกว่าการดำเนินจะถูกตั้งค่าให้ละเว้น กฎต่อไปนี้ควบคุมการส่งสัญญาณในกระบวนการแบบมัลติเธรด:

- สัญญาณที่มีการดำเนินการถูกตั้งค่าให้ยกเลิก หยุดทำงาน หรือดำเนินการกับเธรดเป้าหมาย หรือกระบวนการจะยกเลิก หยุดทำงาน หรือดำเนินการกับกระบวนการทั้งหมด ตามลำดับ (ดังเช่น เธรดทั้งหมด) โปรแกรมแบบเธรด เดียวสามารถถูกเขียนใหม่เป็นโปรแกรมแบบมัลติเธรดโดยไม่ต้อง เปลี่ยนแปลงลักษณะการทำงานของสัญญาณที่ภายนอกเห็นได้ ตัวอย่างเช่น พิจารณาคำสั่งผู้ใช้แบบมัลติเธรด เช่นคำสั่ง **grep** ผู้ใช้สามารถเริ่มต้นคำสั่งใน shell รายการโปรดของตนเองได้ จากนั้น ตัดสินใจที่จะหยุดการทำงานโดยส่งสัญญาณด้วยคำสั่ง **kill** สัญญาณควรจะหยุดทำงานกระบวนการทั้งหมดที่รันคำสั่ง **grep**
- สัญญาณที่สร้างไว้สำหรับเธรดที่ระบุ การใช้รูทีนย่อย **pthread_kill** หรือรูทีนย่อย **raise** จะถูกส่งไปยังเธรดนั้น ถ้าเธรดถูกบล็อกสัญญาณจากการจัดส่ง สัญญาณจะตั้งค่าการคงค้างบนเธรดจนกว่าสัญญาณจะถูกปลดบล็อกจากการจัดส่ง ถ้าเธรดถูกยกเลิกก่อนที่จะส่งสัญญาณ สัญญาณจะถูกละเว้น
- สัญญาณที่สร้างไว้สำหรับกระบวนการ การใช้รูทีนย่อย **kill** จะส่งไปยังเธรดหนึ่งในกระบวนการ ถ้าเธรดตั้งแต่หนึ่งตัวขึ้นไปเรียกรูทีนย่อย **sigwait** สัญญาณจะถูกส่งไปยังหนึ่งในเธรดเหล่านี้ หรือ สัญญาณจะถูกส่งไปยังเธรดหนึ่งเธรด ที่ไม่ได้บล็อกสัญญาณจากการจัดส่ง ถ้าไม่มีเธรดที่ตรงกับเงื่อนไขเหล่านี้ สัญญาณจะตั้งค่าค้างอยู่สำหรับกระบวนการ จนกว่าเธรดจะเรียกรูทีนย่อย **sigwait** ที่ระบุสัญญาณนี้ หรือเธรดที่ปลดล็อกสัญญาณนี้ออกจากการส่ง

ถ้าการดำเนินการที่เชื่อมโยงกับสัญญาณที่ค้างอยู่ (เธรด หรือกระบวนการ) ที่ถูกตั้งค่าให้ละเว้น สัญญาณจะถูกละเว้น

หลักการที่เกี่ยวข้อง:

"การทำกระบวนการซ้ำและการยกเลิก" ในหน้า 525
เนื่องจากกระบวนการทั้งหมดมีอย่างน้อยหนึ่งเธรด การสร้าง (นั่นคือ การทำซ้ำ) และการยกเลิกกระบวนการหมายความถึงการสร้าง และการยกเลิกเธรด

รายการของรูทีนย่อย threads-processes interactions

ส่วนนี้แสดงรายการรูทีนย่อย threads–processes interactions

รูทีนย่อย	คำอธิบาย
alarm	สาเหตุที่ทำให้สัญญาณถูกส่งไปยังกระบวนการที่เรียก หลังจากหมดเวลาใช้งานตามที่ระบุไว้
kill หรือ **killpg**	ส่งสัญญาณไปยังกระบวนการหรือกลุ่มของกระบวนการต่างๆ
pthread_atfork	ลงทะเบียน handler การล้างข้อมูลส่วน
pthread_kill	ส่งสัญญาณไปยังเธรดที่ระบุไว้
pthread_sigmask	ตั้งค่าตัวพรางสัญญาณของเธรด
raise	ส่งสัญญาณไปยังเธรดที่กำลังเรียกใช้งานอยู่
sigaction, sigvec หรือ **signal**	ระบุการดำเนินการที่ใช้ตามการส่งสัญญาณ
sigsuspend หรือ **sigpause**	เปลี่ยนชุดสัญญาณที่บล็อกไว้โดยอัตโนมัติ และรอสัญญาณ
sigthreadmask	ตั้งค่าตัวพรางสัญญาณของเธรด
sigwait	บล็อกการเรียกเธรดจนกว่าจะได้รับสัญญาณที่ระบุไว้

หลักการที่เกี่ยวข้อง:

"ข้อมูลเฉพาะเธรด" ในหน้า 511
แอ็พพลิเคชันจำนวนมากต้องการข้อมูลบางส่วนสำหรับเธรดพื้นฐาน ระหว่างการเรียกฟังก์ชัน

"การสร้างอ็อบเจ็กต์การซิงโครไนซ์ที่ซับซ้อน" ในหน้า 516
รูทีนย่อยที่จัดเตรียมในไลบรารีเธรดสามารถถูก ใช้เป็นหลักในการสร้างอ็อบเจ็กต์การซิงโครไนซ์ที่ซับซ้อนมากขึ้น

“การทำกระบวนการซ้ำและการยกเลิก”
เนื่องจากกระบวนการทั้งหมดมีอย่างน้อยหนึ่งเธรด การสร้าง (นั่นคือ การทำซ้ำ) และการยกเลิกกระบวนการหมายความถึงการสร้าง และการยกเลิกเธรด

การทำกระบวนการซ้ำและการยกเลิก

เนื่องจากกระบวนการทั้งหมดมีอย่างน้อยหนึ่งเธรด การสร้าง (นั่นคือ การทำซ้ำ) และการยกเลิกกระบวนการหมายความถึงการสร้าง และการยกเลิกเธรด

ส่วนนี้อธิบายถึงการโต้ตอบระหว่างเธรด และกระบวนการ ขณะที่ทำซ้ำและยกเลิกกระบวนการ

การ Fork

โปรแกรมเมอร์เรียกรูทีนย่อย **fork** ในกรณีต่อไปนี้:

- เมื่อต้องการสร้างการไหลใหม่ของการควบคุมภายในโปรแกรมเดียวกัน AIX จะสร้างกระบวนการใหม่
- เมื่อต้องการสร้างกระบวนการใหม่ที่ทำงานในโปรแกรมอื่น ในกรณีนี้ การเรียกรูทีนย่อย **fork** ยังคงตามด้วยการเรียกหนึ่งในรูทีนย่อย **exec**

ในโปรแกรมแบบมัลติเธรด ครั้งแรกที่ใช้รูทีนย่อย **fork** การสร้างโฟลว์ควบคุม ถูกระบุโดยรูทีนย่อย **pthread_create** ดังนั้น รูทีนย่อย **fork** จึงควรถูกใช้เพื่อรันโปรแกรมใหม่เท่านั้น

รูทีนย่อย **fork** จะทำซ้ำกระบวนการหลัก แต่ทำซ้ำเฉพาะการเรียกเธรดเท่านั้น child process ยังคงเป็นกระบวนการเธรดเดี่ยว การเรียกเธรดของกระบวนการหลัก จะเป็น initial thread ของ child process ซึ่งอาจไม่เริ่มต้นเธรดของกระบวนการหลัก ดังนั้น หาก initial thread ของ child process ส่งคืนจากรูทีน entry-point แล้ว child process จะถูกยกเลิก

เมื่อทำซ้ำกระบวนการหลักแล้ว รูทีนย่อย **fork** ยังทำซ็ตัวแปรการประสานเวลาทั้งหมดด้วย ซึ่งรวมถึงสถานะของตัวแปรเหล่านั้นด้วยเช่นกัน ตัวอย่างเช่น mutexes อาจถูกพักโดยเธรด ซึ่งไม่มีอยู่ใน child process อีกต่อไป และรีซอร์สที่เชื่อมโยงอาจไม่สอดคล้องกัน

ขอแนะนำว่า รูทีนย่อย **fork** ควรนำมาใช้เพื่อรันโปรแกรมใหม่เท่านั้นและเพื่อเรียกหนึ่งในรูทีนย่อย **exec** ในทันทีหลังจากที่เรียกรูทีนย่อย **fork** ใน child process

Fork handlers

กฎการ fork ก่อนหน้าไม่ระบุความต้องการไลบรารี มัลติเธรด แอ็พพลิเคชันโปรแกรมอาจไม่ทราบว่ามีไลบรารีมัลติเธรด ถูกใช้งานอยู่ และจะเรียกใช้ไลบรารีรูทีนระหว่างรูทีนย่อย **fork** และ **exec** หลายๆ ครั้ง เหมือนที่เคย ทำเสมอ ตามความเป็นจริงแล้ว รูทีนย่อยเหล่านั้นอาจเป็นโปรแกรมเธรดเดี่ยวที่เก่า ดังนั้น จึงไม่สามารถคาดหวังเพื่อให้ทำตามข้อจำกัดใหม่ ที่กำหนดโดยไลบรารีเธรด

อีกนัยหนึ่ง ไลบรารีแบบมัลติเธรดต้องการวิธีการป้องกัน สภาวะภายในของตนระหว่าง fork ในกรณีที่รูทีนถูกเรียใช้ภายหลังในกระบวนการชายน์ ปัญหาเกิดขึ้นโดยเฉพาะในอินพุต/เอาต์พุต ไลบรารีแบบมัลติเธรด ซึ่งเกือบแน่ใจได้ว่าถูกเรียกใช้ระหว่าง รูทีนย่อย **fork** และ **exec** เพื่อให้มีผลต่อการเปลี่ยนทิศทาง อินพุต/เอาต์พุต

รูทีนย่อย **pthread_atfork** จะมี วิธีสำหรับไลบรารีแบบมัลติเธรดเพื่อป้องกันตนเองจากเรียกโปรแกรม ที่ไม่รู้เรื่องซึ่งเรียกใช้รูทีนย่อย **fork** รวมทั้งมีแอ็พพลิเคชันโปรแกรม แบบมัลติเธรดที่มีกลไกมาตรฐานสำหรับการป้องกันตนเอง จากการเรียกใช้รูทีนย่อย **fork** ในไลบรารีรูทีน หรือ แอ็พพลิเคชันเอง

รูทีนย่อย **pthread_atfork** จะลงทะเบียน fork handler ที่ต้องการเรียกก่อนและหลังการเรียกรูทีนย่อย **fork** fork handler จึงถูกคาดหวังในเธรดว่าจะเรียกรูทีนย่อย **fork** fork handler ต่อไปนี้คือ fork handler ที่มีอยู่:

รูทีนย่อย	คำอธิบาย
Prepare	prepare fork handler จะถูกเรียกก่อนที่การประมวลผลรูทีนย่อย **fork** จะเริ่มต้นขึ้น
Parent	parent fork handler จะถูกเรียกหลังจากการประมวลผลรูทีนย่อย **fork** เสร็จสิ้นแล้วในกระบวนการหลัก
Child	child fork handler จะถูกเรียกหลังจากการประมวลผลรูทีนย่อย **fork** เสร็จสิ้นแล้วใน child process

การยุติโปรเซส

prepare fork handler จะถูกเรียกตามลำดับแบบเข้าหลังออกก่อน (LIFO) ขณะที่ parent และ child fork handler จะถูกเรียกตามลำดับแบบเข้าก่อนออกก่อน (FIFO) ซึ่งอนุญาตให้โปรแกรมสงวนการล็อกลำดับที่ต้องการไว้

เมื่อกระบวนการยกเลิกโดยเรียกรูทีนย่อย **exit, atexit** หรือ **_exit** ไม่ว่าจะทางตรง หรือทางอ้อม เธรดทั้งหมดที่อยู่ภายในกระบวนจะถูกยกเลิก ไม่มี cleanup handler หรือตัวทำลายข้อมูลที่ระบุเฉพาะสำหรับเธรดถูกเรียก

หมายเหตุ: รูทีนย่อย **unatexit** จะถอนการลงทะเบียนฟังก์ชันที่ลงทะเบียนไว้ก่อนหน้านี้โดยรูทีนย่อย **atexit** ถ้าพบฟังก์ชันที่อ้างถึง ฟังก์ชันนั้นจะถูกลบออกจากรายการฟังก์ชันที่เรียกไว้ในช่วงการยกเลิก โปรแกรมปกติ

เหตุผลสำหรับลักษณะการทำงานแบบนี้คือ ไม่มีสถานะที่ต้องล้างข้อมูล และไม่มีหน่วยเก็บที่ระบุเฉพาะสำหรับเธรดที่ต้องเรียกคืน เนื่องจากกระบวนการทั้งหมดถูกยกเลิก ซึ่งรวมถึงเธรดทั้งหมด และหน่วยเก็บในกระบวนการทั้งหมดจะถูกเรียกคืน ซึ่งรวมถึงหน่วยเก็บที่ระบุเฉพาะเธรดทั้งหมด

หลักการที่เกี่ยวข้อง:

"การทำความเข้าใจเธรดและกระบวนการ" ในหน้า 456
เธรด คือการควบคุมสายงานอย่างเป็นอิสระที่ทำงานภายในพื้นที่แอดเดรสเดียวกันกับ การควบคุมสายงานอย่างเป็นอิสระอื่นๆ ภายในกระบวนการ

"การจัดการสัญญาณ" ในหน้า 521
สัญญาณในกระบวนการแบบมัลติเธรดเป็นส่วนขยายของ สัญญาณในโปรแกรมแบบเธรดเดียวดั้งเดิม

"รายการของรูทีนย่อย threads-processes interactions" ในหน้า 524
ส่วนนี้แสดงรายการรูทีนย่อย threads-processes interactions

อ็อพชันไลบรารีเธรด

ส่วนนี้อธิบายถึงแอ็ตทริบิวต์ของเธรดชนิดพิเศษ mutex และตัวแปรเงื่อนไข

POSIX มาตรฐานสำหรับไลบรารีเธรดจะระบุการนำไปปฏิบัติบางส่วน ตามที่เลือกไว้ รูทีนย่อยทั้งหมดจะกำหนดโดยไลบรารีเธรด API จะพร้อมใช้งานเสมอ ซึ่งขึ้นอยู่กับอ็อพชันที่ใช้งานได้ รูทีนย่อยบางตัวอาจไม่นำมาปฏิบัติ รูทีนย่อยที่ไม่ได้นำไปใช้สามารถเรียกได้โดยแอ็พพลิเคชัน แต่รูทีนย่อยเหล่านั้นจะส่งคืนโค้ดระบุความผิดพลาด **ENOSYS**

แอ็ตทริบิวต์สแต็ก

สแต็กจะถูกจัดสรรไว้สำหรับแต่ละเธรด การจัดการกับสแต็ก คือการนำไปปฏิบัติที่มีการพึ่งพา ดังนั้น ข้อมูลต่อไปนี้จะใช้กับ AIX เท่านั้น แม้ว่า คุณลักษณะที่คล้ายคลึงกันอาจมีอยู่บนระบบอื่น

สแต็กจะถูกจัดสรรแบบไดนามิก เมื่อเธรดถูกสร้างขึ้น สำหรับการใช้แอ็ตทริบิวต์เธรดระดับสูง มีความเป็นไปได้สำหรับผู้ใช้ที่ควบคุมขนาดของสแต็ก และที่อยู่ของสแต็ก ข้อมูลต่อไปนี้ไม่ได้ใช้กับ initial thread ซึ่งจะถูกสร้างโดยระบบ

ขนาดของสแต็ก

อ็อพชันขนาดของสแต็กเปิดใช้งานการควบคุมของแอ็ตทริบิวต์ **stacksize** ของอ็อบเจ็กต์แอ็ตทริบิวต์เธรด แอ็ตทริบิวต์เธรดระบุขนาดต่ำสุดของสแต็กที่ต้องการใช้สำหรับเธรดที่สร้าง

แอ็ตทริบิวต์ **stacksize** จะถูกกำหนดอยู่ใน AIX แอ็ตทริบิวต์และรูทีนย่อยต่อไปนี้พร้อมใช้งาน เมื่ออ็อพชันถูกนำไปใช้:

- แอ็ตทริบิวต์ **stacksize** ของอ็อบเจ็กต์แอ็ตทริบิวต์เธรด
- **pthread_attr_getstacksize** ส่งคืนค่าของแอ็ตทริบิวต์
- และรูทีนย่อย **pthread_attr_setstacksize** จะตั้งค่า

ค่าดีฟอลต์ของแอ็ตทริบิวต์ **stacksize** คือ 96 KB ค่าต่ำสุดของแอ็ตทริบิวต์ **stacksize** คือ 16 KB ถ้าค่าที่กำหนดไว้น้อยกว่าค่าต่ำสุด ค่าต่ำสุดจะถูกนำมาจัดสรร

ใน AIX การนำไปปฏิบัติของไลบรารีเธรด ชิ้นข้อมูลที่เรียกว่า *พื้นที่เธรดของผู้ใช้* จะถูกจัดสรรไว้สำหรับเธรดที่สร้างขึ้น แต่ละตัว พื้นที่จะถูกแบ่งออกตามส่วนต่อไปนี้:

- *โซนสีแดง* คือการป้องกันการบันทึกและการป้องกันการอ่าน สำหรับการตรวจหาโอเวอร์โฟลว์สแต็ก ไม่มีโซนแดงในโปรแกรมที่ใช้เพจขนาดใหญ่
- ดีฟอลต์สแต็ก
- โครงสร้าง pthread
- โครงสร้างเธรด
- โครงสร้างแอ็ตทริบิวต์เธรด

หมายเหตุ: พื้นที่เธรดของผู้ใช้ที่อธิบายไว้ที่นี่จะไม่มีความสัมพันธ์กับโครงสร้าง **uthread** ที่ใช้ในเคอร์เนล AIX พื้นที่เธรดของผู้ใช้เข้าถึงได้ในโหมดผู้ใช้เท่านั้น และจะถูกจัดการโดยไลบรารีเธรด ขณะที่โครงสร้าง **uthread** จะมีอยู่ภายในสภาวะแวดล้อมของเคอร์เนลเท่านั้น

อ็อพชันสแต็กแอดเดรส POSIX

อ็อพชันสแต็กแอดเดรสเปิดใช้งานการควบคุมของแอ็ตทริบิวต์ **stackaddr** ของอ็อบเจ็กต์แอ็ตทริบิวต์เธรด แอ็ตทริบิวต์นี้จะระบุตำแหน่งของหน่วยเก็บที่ต้องการใช้ สำหรับสแต็กของเธรดที่สร้างไว้

แอ็ตทริบิวต์และรูทีนย่อยต่อไปนี้พร้อมใช้งาน เมื่ออ็อพชันถูกนำไปใช้:

- แอ็ตทริบิวต์ **stackaddr** ของแอ็ตทริบิวต์เธรด จะระบุแอดเดรสของสแต็กที่จะจัดสรรไว้สำหรับเธรด
- รูทีนย่อย **pthread_attr_getstackaddr** จะส่งคืนค่าของแอ็ตทริบิวต์
- และรูทีนย่อย **pthread_attr_setstackaddr** จะตั้งค่า

ถ้าไม่ได้ระบุแอดเดรสสแต็กไว้ สแต็กจะถูกจัดสรรไว้โดยระบบที่สุ่มแอดเดรส ถ้าคุณต้องการสแต็กที่รู้ตำแหน่ง คุณสามารถใช้แอ็ตทริบิวต์ **stackaddr** ตัวอย่างเช่น ถ้าคุณต้องการสแต็กที่มีขนาดใหญ่มาก คุณสามารถตั้งค่าแอดเดรสให้เป็นเซ็กเมนต์ที่ไม่ได้ใช้ และการรับประกันว่าการจัดสรรจะดำเนินการเป็นผลสำเร็จ

ถ้าระบุสแต็กแอดเดรสไว้ ขณะเรียกรูทีนย่อย **pthread_create** ระบบจะพยายามจัดสรรสแต็กที่แอดเดรสที่กำหนดไว้ ถ้าเกิดความล้มเหลว รูทีนย่อย **pthread_create** จะส่งคืน **EINVAL** เนื่องจากรูทีนย่อย **pthread_attr_setstackaddr** ไม่ได้จัดสรรสแต็กไว้ รูทีนย่อยนั้นจะส่งคืนข้อผิดพลาด ถ้าสแต็กแอดเดรสที่ระบุไว้มีขนาดเกินกว่าการกำหนดแอดเดรส

อ็อพชันการกำหนดเวลาลำดับความสำคัญ POSIX

อ็อพชันลำดับความสำคัญในการกำหนดตารางเวลาจะเปิดใช้งานการควบคุมของการเรียกใช้งานการกำหนดตารางเวลา ที่ระดับเธรด เมื่อปิดใช้งานอ็อพชันนี้ เธรดทั้งหมดภายในการประมวลผลจะแบ่งใช้คุณสมบัติการกำหนดตารางเวลา ของการประมวลผล เมื่อเปิดใช้งานอ็อพชันนี้ เธรดแต่ละเธรดจะมีคุณสมบัติการกำหนดตารางเวลาของตนเอง สำหรับเธรดที่มีขอบเขตของ contention แบบโลคัล คุณสมบัติของการกำหนดตารางเวลาจะถูกจัดการที่ระดับของการประมวลผลด้วยตัวกำหนดตารางเวลาของไลบรารี ขณะที่เธรดที่มีขอบเขตของ contention แบบโกลบอล คุณสมบัติของการกำหนดตารางเวลาจะถูกจัดการที่ระดับของระบบด้วยตัวกำหนดตารางเวลาเคอร์เนล

แอ็ตทริบิวต์ต่อไปนี้และรูทีนย่อยพร้อมใช้งาน เมื่ออ็อพชันถูกนำไปใช้:

- แอ็ตทริบิวต์ **inheritsched** ของอ็อบเจ็กต์แอ็ตทริบิวต์เธรด
- แอ็ตทริบิวต์ **schedparam** ของอ็อบเจ็กต์แอ็ตทริบิวต์เธรด และเธรด
- แอ็ตทริบิวต์ **schedpolicy** ของอ็อบเจ็กต์แอ็ตทริบิวต์เธรด และเธรด
- แอ็ตทริบิวต์ **contention-scope** ของอ็อบเจ็กต์แอ็ตทริบิวต์เธรด และเธรด
- รูทีนย่อย **pthread_attr_getschedparam** และ **pthread_attr_setschedparam**
- รูทีนย่อย **pthread_getschedparam**

การตรวจสอบความพร้อมใช้งานของอ็อพชัน

อ็อพชันสามารถตรวจสอบได้ในเวลาการคอมไพล์ หรือในเวลารันไทม์โปรแกรมที่สามารถเคลื่อนย้ายได้ควรตรวจสอบสภาพพร้อมใช้งานของอ็อพชันก่อนที่จะใช้อ็อพชันเหล่านั้นได้ ดังนั้น จึงไม่ต้องการเขียนขึ้นใหม่ เมื่อย้ายไปยังระบบอื่น

การตรวจสอบ Compile-time

เมื่ออ็อพชันไม่พร้อมใช้งาน คุณสามารถหยุดการคอมไพล์ได้ ดังตัวอย่างต่อไปนี้:

```
#ifndef _POSIX_THREAD_ATTR_STACKSIZE
#error "The stack size POSIX option is required"
#endif
```

ไฟล์ส่วนหัว **pthread.h** ยังกำหนดสัญลักษณ์ต่อไปนี้ ซึ่งสามารถนำไปใช้ได้โดยไฟล์ส่วนหัวอื่นๆ หรือใช้โดยโปรแกรม:

_POSIX_REENTRANT_FUNCTIONS
หมายถึง การทำงานแบบ reentrant ที่ต้องการ

_POSIX_THREADS
หมายถึง การนำไปปฏิบัติของไลบรารีเธรด

การตรวจสอบรันไทม์

รูทีนย่อย **syscont** สามารถนำมาใช้เพื่อขอรับสภาพพร้อมใช้งานของอ็อพชันบนระบบ ที่โปรแกรมถูกเรียกใช้งาน สิ่งนี้มีประโยชน์ ขณะที่ย้ายโปรแกรมระหว่างระบบ ที่มีความเข้ากันได้แบบไบนารี เช่น สองเวอร์ชันของ AIX

รายการต่อไปนี้บ่งชี้ถึงสัญลักษณ์ที่เชื่อมโยงกับอ็อพชันแต่ละตัว และต้องใช้สำหรับพารามิเตอร์ *Name* ของรูทีนย่อย **sysconf** ค่าคงที่เชิงสัญลักษณ์จะถูกกำหนดไว้ในไฟล์ส่วนหัว **unistd.h**

สแต็กแอดเดรส
_SC_THREAD_ATTR_STACKADDR

ขนาดของสแต็ก
_SC_THREAD_ATTR_STACKSIZE

ลำดับความสำคัญในการกำหนดตารางเวลา
_SC_THREAD_PRIORITY_SCHEDULING

ลำดับความสำคัญในการสืบทอด
_SC_THREAD_PRIO_INHERIT

ลำดับความสำคัญในการป้องกัน
_SC_THREAD_PRIO_PROTECT

การประมวลผลที่แบ่งใช้
_SC_THREAD_PROCESS_SHARED

หากต้องการตรวจสอบอ็อพชันทั่วไป ให้ใช้รูทีนย่อย **sysconf** ค่าพารามิเตอร์ *Name* ต่อไปนี้:

_SC_REENTRANT_FUNCTIONS
หมายถึง การทำงานแบบ reentrant ที่ต้องการ

_SC_THREADS
หมายถึง การนำไปปฏิบัติของไลบรารีเธรด

การประมวลผลที่แบ่งใช้

AIX และระบบ UNIX ส่วนใหญ่อนุญาตให้หลายกระบวนการ แบ่งใช้พื้นที่ข้อมูลร่วมกัน ซึ่งเรียกว่า *หน่วยความจำแบบแบ่งใช้* แอ็ตทริบิวต์ process-sharing สำหรับตัวแปรเงื่อนไขและ mutexes หมายถึงอนุญาตให้จัดสรรอ็อบเจ็กต์เหล่านี้ในหน่วยความจำแบบแบ่งใช้เพื่อสนับสนุน การซิงโครไนซ์ระหว่างเธรดที่เป็นของกระบวนการอื่น อย่างไรก็ตาม เนื่องจากไม่มีอินเตอร์เฟสมาตรฐานอุตสาหกรรม สำหรับการจัดการหน่วยความจำที่แบ่งใช้ อ็อพชันการประมวลผลที่แบ่งใช้ POSIX ไม่ได้นำไปปฏิบัติในไลบรารีเธรด AIX

ชนิดข้อมูลเธรด

ชนิดข้อมูลต่อไปนี้ ถูกกำหนดไว้สำหรับไลบรารีเธรด นิยามของชนิดข้อมูลเหล่านี้ สามารถเปลี่ยนแปลงระหว่างระบบได้:

pthread_t
ระบุเธรด

pthread_attr_t
ระบุอ็อบเจ็กต์แอ็ตทริบิวต์เธรด

pthread_cond_t
ระบุตัวแปรเงื่อนไข

pthread_condattr_t
ระบุอ็อบเจ็กต์แอ็ตทริบิวต์เงื่อนไข

pthread_key_t
ระบุคีย์ข้อมูลที่ระบุเฉพาะสำหรับเธรด

pthread_mutex_t
ระบุ mutex

pthread_mutexattr_t
ระบุอ็อบเจ็กต์แอ็ตทริบิวต์ mutex

pthread_once_t
ระบุอ็อบเจ็กต์การกำหนดค่าเริ่มต้นแบบครั้งเดียว

ข้อจำกัดและค่าดีฟอลต์

ไลบรารีเธรดมีข้อจำกัดเกี่ยวกับการนำไปปฏิบัติที่ต้องพึ่งพา และค่าดีฟอลต์ ข้อจำกัดเหล่านี้และค่าดีฟอลต์จะสามารถเรียกข้อมูลได้ด้วยค่าคงที่เชิงสัญลักษณ์ เพื่อพัฒนาความสามารถในการย้ายโปรแกรม:

- จำนวนสูงสุดของเธรดต่อการประมวลผลคือ 512 จำนวนสูงสุดของเธรดสามารถเรียกข้อมูลได้ ที่เวลาการคอมไพล์โดยใช้ค่าคงที่เชิงสัญลักษณ์ **PTHREAD_THREADS_MAX** ที่กำหนดอยู่ในไฟล์ส่วนหัว **pthread.h** ถ้าแอ็พพลิเคชันถูกคอมไพล์ด้วยแฟล็ก **-D_LARGE_THREADS** จำนวนสูงสุดของเธรดต่อการประมวลผลคือ 32767
- ขนาดต่ำสุดของสแต็กสำหรับเธรดคือ 8 K ขนาดสแต็กดีฟอลต์คือ 96 KB จำนวนนี้สามารถเรียกข้อมูลที่เวลาการคอมไพล์โดยใช้ค่าคงที่เชิงสัญลักษณ์ **PTHREAD_STACK_MIN** ที่กำหนดอยู่ในไฟล์ส่วนหัว **pthread.h**

  หมายเหตุ: ขนาดต่ำสุดของสแต็กคือ 256 MB ขนาดของเซ็กเมนต์ ข้อจำกัดนี้จะบ่งชี้โดยค่าคงที่เชิงสัญลักษณ์ **PTHREAD_STACK_MAX** ในไฟล์ส่วนหัว **pthread.h**
- จำนวนสูงสุดของคีย์ข้อมูลที่ระบุเฉพาะสำหรับเธรดจะถูกจำกัดไว้ที่ 508 จำนวนนี้สามารถเรียกข้อมูลได้ในเวลาการคอมไพล์โดยใช้ค่าคงที่เชิงสัญลักษณ์ **PTHREAD_KEYS_MAX** ที่กำหนดอยู่ในไฟล์ส่วนหัว **pthread.h**

ค่าแอ็ตทริบิวต์ดีฟอลต์

ค่าดีฟอลต์สำหรับอ็อบเจ็กต์แอ็ตทริบิวต์เธรด จะถูกกำหนดอยู่ในไฟล์ส่วนหัว **pthread.h** ด้วยค่าคงที่เชิงสัญลักษณ์ต่อไปนี้:

- ค่าดีฟอลต์สำหรับค่าคงที่เชิงสัญลักษณ์ **DEFAULT_DETACHSTATE** คือ **PTHREAD_CREATE_DETACHED** ซึ่งจะระบุค่าดีฟอลต์สำหรับแอ็ตทริบิวต์ **detachstate**
- ค่าดีฟอลต์สำหรับค่าคงที่เชิงสัญลักษณ์ **DEFAULT_JOINABLE** คือ **PTHREAD_CREATE_JOINABLE** ซึ่งจะระบุค่าดีฟอลต์สำหรับ สถานะที่สามารถเชื่อมกันได้
- ค่าดีฟอลต์สำหรับค่าคงที่เชิงสัญลักษณ์ **DEFAULT_INHERIT** คือ **PTHREAD_INHERIT_SCHED** ซึ่งจะระบุค่าดีฟอลต์สำหรับแอ็ตทริบิวต์ **inheritsched**

- ค่าดีฟอลต์สำหรับค่าคงที่เชิงสัญลักษณ์ **DEFAULT_PRIO** คือ 1 ซึ่งจะระบุค่าดีฟอลต์สำหรับฟิลด์ sched_prio ของแอ็ตทริบิวต์ **schedparam**
- ค่าดีฟอลต์สำหรับค่าคงที่เชิงสัญลักษณ์ **DEFAULT_SCHED** คือ **SCHED_OTHER** ซึ่งจะระบุค่าดีฟอลต์สำหรับแอ็ตทริบิวต์ **schedpolicy** ของอ็อบเจ็กต์แอ็ตทริบิวต์เธรด
- ค่าดีฟอลต์สำหรับค่าคงที่เชิงสัญลักษณ์ **DEFAULT_SCOPE** คือ **PTHREAD_SCOPE_LOCAL** ซึ่งจะระบุค่าดีฟอลต์สำหรับแอ็ตทริบิวต์ **contention-scope**

หลักการที่เกี่ยวข้อง:

"ไลบรารี Threadsafe และที่ถูกเธรดใน AIX" ในหน้า 462
ส่วนนี้อธิบายเธรดไลบรารีใน AIX

รายการของรูทีนย่อยคุณลักษณะขั้นสูงของเธรด

ส่วนนี้แสดงรายการรูทีนย่อยคุณลักษณะขั้นสูงของเธรด

รูทีนย่อย	คำอธิบาย
pthread_attr_getstackaddr	ส่งกลับค่าของแอ็ตทริบิวต์ stackaddr ของอ็อบเจ็กต์แอ็ตทริบิวต์ เธรด
pthread_attr_getstacksize	ส่งกลับค่าของแอ็ตทริบิวต์ stacksize ของอ็อบเจ็กต์แอ็ตทริบิวต์ เธรด
pthread_attr_setstackaddr	ตั้งค่าของแอ็ตทริบิวต์ stackaddr ของอ็อบเจ็กต์แอ็ตทริบิวต์ เธรด
pthread_attr_setstacksize	ตั้งค่าของแอ็ตทริบิวต์ stacksize ของอ็อบเจ็กต์แอ็ตทริบิวต์ เธรด
pthread_condattr_getpshared	ส่งกลับค่าของแอ็ตทริบิวต์ process-shared ของอ็อบเจ็กต์แอ็ตทริบิวต์ เงื่อนไข
pthread_condattr_setpshared	ตั้งค่าของแอ็ตทริบิวต์ process-shared ของอ็อบเจ็กต์แอ็ตทริบิวต์ เงื่อนไข
pthread_getspecific	ส่งกลับข้อมูลจำเพาะเธรดที่สัมพันธ์กับคีย์ที่ระบุ
pthread_key_create	สร้างคีย์ข้อมูลจำเพาะเธรด
pthread_key_delete	ลบคีย์ข้อมูลจำเพาะเธรด
pthread_mutexattr_getpshared	ส่งกลับค่าของแอ็ตทริบิวต์ process-shared ของอ็อบเจ็กต์แอ็ตทริบิวต์ mutex
pthread_mutexattr_setpshared	ตั้งค่าของแอ็ตทริบิวต์ process-shared ของอ็อบเจ็กต์แอ็ตทริบิวต์ mutex
pthread_once	ดำเนินการรูทีนเพียงครั้งเดียวในกระบวนการ
PTHREAD_ONCE_INIT	กำหนดค่าเริ่มต้นโครงสร้างควบคุมการซิงโครไนซ์ one-time
pthread_setspecific	ตั้งค่าข้อมูลจำเพาะเธรดที่สัมพันธ์กับคีย์ที่ระบุ

อินเตอร์เฟสที่สนับสนุน

สำหรับระบบ AIX สัญลักษณ์ **_POSIX_THREADS**, **_POSIX_THREAD_ATTR_STACKADDR**, **_POSIX_THREAD_ATTR_STACKSIZE** และ **_POSIX_THREAD_PROCESS_SHARED** จะถูกกำหนดไว้

ดังนั้น อินเตอร์เฟสของเธรดต่อไปนี้จะได้รับการสนับสนุน

อินเตอร์เฟส POSIX

ต่อไปนี้คือรายการของอินเตอร์เฟส POSIX:

- pthread_atfork
- pthread_attr_destroy
- pthread_attr_getdetachstate
- pthread_attr_getschedparam
- pthread_attr_getstacksize
- pthread_attr_getstackaddr
- pthread_attr_init

- pthread_attr_setdetachstate
- pthread_attr_setschedparam
- pthread_attr_setstackaddr
- pthread_attr_setstacksize
- pthread_cancel
- pthread_cleanup_pop
- pthread_cleanup_push
- pthread_detach
- pthread_equal
- pthread_exit
- pthread_getspecific
- pthread_join
- pthread_key_create
- pthread_key_delete
- pthread_kill
- pthread_mutex_destroy
- pthread_mutex_init
- pthread_mutex_lock
- pthread_mutex_trylock
- pthread_mutex_unlock
- pthread_mutexattr_destroy
- pthread_mutexattr_getpshared
- pthread_mutexattr_init
- pthread_mutexattr_setpshared
- pthread_once
- pthread_self
- pthread_setcancelstate
- pthread_setcanceltype
- pthread_setspecific
- pthread_sigmask
- pthread_testcancel
- pthread_cond_broadcast
- pthread_cond_destroy
- pthread_cond_init
- pthread_cond_signal
- pthread_cond_timedwait

- pthread_cond_wait
- pthread_condattr_destroy
- pthread_condattr_getpshared
- pthread_condattr_init
- pthread_condattr_setpshared
- pthread_create
- sigwait

Single UNIX Specification, Version 2Interfaces

ต่อไปนี้คือรายการของอินเตอร์เฟส Single UNIX Specification, Version 2:

- pthread_attr_getguardsize
- pthread_attr_setguardsize
- pthread_getconcurrency
- pthread_mutexattr_gettype
- pthread_mutexattr_settype
- pthread_rwlock_destroy
- pthread_rwlock_init
- pthread_rwlock_rdlock
- pthread_rwlock_tryrdlock
- pthread_rwlock_trywrlock
- pthread_rwlock_unlock
- pthread_rwlock_wrlock
- pthread_rwlockattr_destroy
- pthread_rwlockattr_getpshared
- pthread_rwlockattr_init
- pthread_rwlockattr_setpshared
- pthread_setconcurrency

สำหรับระบบ AIX สัญลักษณ์ **_POSIX_THREAD_SAFE_FUNCTIONS** จะถูกกำหนดไว้เสมอ ดังนั้น อินเตอร์เฟสต่อไปนี้จะได้รับการสนับสนุน :

- asctime_r
- ctime_r
- flockfile
- ftrylockfile
- funlockfile
- getc_unlocked

- getchar_unlocked
- getgrgid_r
- getgrnam_r
- getpwnam_r
- getpwuid_r
- gmtime_r
- localtime_r
- putc_unlocked
- putchar_unlocked
- rand_r
- readdir_r
- strtok_r

AIX ไม่สนับสนุนอินเตอร์เฟสต่อไปนี้ สัญลักษณ์ถูกจัดเตรียมไว้ แต่ส่งคืนข้อผิดพลาดเสมอ และตั้งค่า errno ให้มีค่า ENOSYS:

- pthread_mutex_getprioceiling
- pthread_mutex_setprioceiling
- pthread_mutexattr_getprioceiling
- pthread_mutexattr_getprotocol
- pthread_mutexattr_setprioceiling
- pthread_mutexattr_setprotocol

Non-threadsafe interfaces

libc.a library (ฟังก์ชันมาตรฐาน):

- advance
- asctime
- brk
- catgets
- chroot
- compile
- ctime
- cuserid
- dbm_clearerr
- dbm_close
- dbm_delete
- dbm_error

- dbm_fetch
- dbm_firstkey
- dbm_nextkey
- dbm_open
- dbm_store
- dirname
- drand48
- ecvt
- encrypt
- endgrent
- endpwent
- endutxent
- fcvt
- gamma
- gcvt
- getc_unlocked
- getchar_unlocked
- getdate
- getdtablesize
- getgrent
- getgrgid
- getgrnam
- getlogin
- getopt
- getpagesize
- getpass
- getpwent
- getpwnam
- getpwuid
- getutxent
- getutxid
- getutxline
- getw
- getw
- gmtime
- l64a

- lgamma
- localtime
- lrand48
- mrand48
- nl_langinfo
- ptsname
- putc_unlocked
- putchar_unlocked
- pututxline
- putw
- rand
- random
- readdir
- re_comp
- re_exec
- regcmp
- regex
- sbrk
- setgrent
- setkey
- setpwent
- setutxent
- sigstack
- srand48
- srandom
- step
- strerror
- strtok
- ttyname
- ttyslot
- wait3

อินเตอร์เฟส AIX ต่อไปนี้ไม่เป็น threadsafe

ไลบรารี libc.a (ไลบรารี AIX-specific):

- endfsent
- endttyent

- endutent
- getfsent
- getfsfile
- getfsspec
- getfstype
- getttyent
- getttynam
- getutent
- getutid
- getutline
- pututline
- setfsent
- setttyent
- setutent
- utmpname

ไลบรารี **libbsd.a** :

- timezone

ไลบรารี **libm.a** และ **libmsaa.a** :

- gamma
- lgamma

ไม่มีฟังก์ชันใดในไลบรารีต่อไปนี้ที่เป็น threadsafe:

- libPW.a
- libblas.a
- libcur.a
- libcurses.a
- libplot.a
- libprint.a

อินเตอร์เฟส **ctermid** และ **tmpnam** ไม่เป็น threadsafe ถ้าถูกส่งอาร์กิวเมนต์ NULL

ในโปรแกรมแบบมัลติเธรด ไม่แนะนำให้เรียกทำงาน รูทีนย่อย **setlocale( )** พร้อมกันจากเธรดจำนวนมาก หากหนึ่งในเธรดเรียกใช้รูทีนย่อย **setlocale( )** จากภายในรูทีนการทำงานเริ่มต้นของโมดูล

หมายเหตุ: รูทีนย่อยบางตัวอาจนำมาใช้เป็นแมโครในบางระบบ ให้หลีกเลี่ยงการใช้แอดเดรสของรูทีนย่อยเธรด

การเขียนโค้ด reentrant และ threadsafe

ในกระบวนการแบบเธรดเดี่ยว มีได้เพียงหนึ่งโฟลว์ของการควบคุมโค้ดที่เรียกใช้งานโดยกระบวนการเหล่านี้ไม่จำเป็นต้องเป็น reentrant หรือ threadsafe ในโปรแกรมแบบมัลติเธรด ฟังก์ชันเดียวกัน และ รีซอร์สเดียวกันอาจถูกเข้าถึงพร้อมกันโดยโฟลว์ควบคุมหลายโฟลว์

เพื่อปกป้อง integrity ของรีซอร์ส โค้ดที่เขียนขึ้นสำหรับโปรแกรมแบบมัลติเธรด ต้องเป็น reentrant และ threadsafe

reentrance และ thread safety จะเกี่ยวกับวิธีการที่ฟังก์ชันจัดการกับรีซอร์ส การเข้าใหม่ และความปลอดภัยของเธรดเป็นแนวคิดแยกกัน: ฟังก์ชันสามารถเป็น reentrant, threadsafe, ทั้งสอง หรือไม่ใช่ทั้งหมด

ส่วนนี้ให้ข้อมูลเกี่ยวกับการเขียนโปรแกรม reentrant และ threadsafe ซึ่งจะไม่ครอบคลุมหัวข้อของการเขียนโปรแกรมแบบเธรดที่มีประสิทธิภาพ โปรแกรมแบบเธรดที่มีประสิทธิภาพจะป็นโปรแกรมที่ทำงานแบบขนาน คุณต้องพิจารณาถึงประสิทธิภาพของเธรดในระหว่างที่ออกแบบโปรแกรม โปรแกรมแบบเธรดเดี่ยวที่มีอยู่ สามารถทำเป็นโปรแกรมแบบเธรดที่มีประสิทธิภาพได้ แต่การทำเช่นนี้จำเป็นต้องออกแบบและเขียนโปรแกรมขึ้นใหม่

reentrance

การทำงานแบบ reentrant ไม่ได้พักข้อมูลสแตติกผ่านการเรียกแบบต่อเนื่อง หรือไม่ได้ส่งคืนตัวชี้ไปยังข้อมูลสถิติ ข้อมูลทั้งหมด จะถูกจัดเตรียมไว้โดยตัวเรียกของการทำงาน การทำงานแบบ reentrant ต้องไม่เรียกการทำงานแบบไม่ใช่ reentrant

บ่อยครั้งที่การทำงานที่ไม่ใช่ reentrant สามารถระบุได้โดยอินเตอร์เฟส และการใช้งานที่อยู่ภายนอก แต่ไม่เสมอไป ตัวอย่าง เช่น รูทีนย่อย **strtok** ไม่ใช่ reentrant เนื่องจากรูทีนย่อยจะพักสตริงที่แตกอยู่ในโทเค็น รูทีนย่อย **ctime** ยังคงไม่ใช่ reentrant แต่ส่งคืนตัวชี้ไปยังข้อมูลสถิติที่ถูกเขียนทับโดยการเรียกแต่ละครั้ง

Thread safety

ฟังก์ชัน threadsafe ปกป้องรีซอร์สที่แบ่งใช้มิให้เข้าถึง พร้อมกันโดยการล็อก Thread safety จะเกี่ยวข้องกับการนำฟังก์ชันไปใช้งาน และไม่ส่งผลถึงอินเตอร์เฟสภายนอก

ในภาษา C ตัวแปรโลคัลจะถูกจัดสรรไว้ บนสแต็กแบบไดนามิก ดังนั้น ฟังก์ชันใดๆ ที่ไม่ใช่ข้อมูลสแตติก หรือรีซอร์ส ที่แบ่งใช้อื่นๆ จะเป็น threadsafe ดังในตัวอย่างต่อไปนี้:

```
/* threadsafe function */
int diff(int x, int y)
{
        int delta;

        delta = y - x;
        if (delta < 0)
                delta = -delta;

        return delta;
}
}
```

การใช้ข้อมูลโกลบอลคือ thread-unsafe ข้อมูลโกลบอลควรถูกรักษาไว้ต่อเธรด หรือต่อการครอบคลุม ดังนั้น การเข้าถึงสามารถ serialize ได้ เธรดอาจอ่านโค้ดระบุความผิดพลาดที่สอดคล้องกับข้อผิดพลาด ที่มีต้นเหตุมาจากเธรดอื่น ใน AIX เธรดแต่ละตัวมีค่า **errno** เป็นของตนเอง

การสร้างฟังก์ชัน reentrant

ในกรณีส่วนใหญ่ การทำงานแบบไม่ใช่ reentrant ต้องแทนที่ด้วยการทำงานที่มีอินเตอร์เฟสที่ถูกแก้ไขให้เป็นแบบ reentrant การทำงานแบบไม่ใช่ reentrant ไม่สามารถนำมาใช้ได้โดยเธรดที่มีจำนวนมาก นอกนั้นจาก อาจเป็นไปไม่ได้ ที่จะทำให้ฟังกชันที่ไม่ใช่ reentrant เป็น threadsafe

การส่งคืนข้อมูล

การทำงานที่ไม่มี reentrant จำนวนมากจะส่งคืนตัวชี้ไปยังข้อมูลสถิติ ซึ่งสามารถหลีกเลี่ยงได้ด้วยวิธีต่อไปนี้:

- การส่งข้อมูลที่จัดสรรแบบไดนามิก ในกรณีนี้ เป็นความรับผิดชอบต่อตัวเรียก ที่ต้องจัดหาหน่วยเก็บที่ว่าง ประโยชน์คืออินเตอร์เฟสที่ทำไม่ต้องการการแก้ไข อย่างไรก็ตาม ไม่มีความแน่นอนสำหรับความเข้ากันได้แบบย้อนกลับ โปรแกรมสำหรับเธรดเดี่ยวที่มีอยู่โดยใช้ฟังก์ชันที่ถูกแก้ไขโดยไม่มีการแก้ไข จะไม่เพิ่มหน่วยเก็บที่ว่าง และนำไปสู่การขาดแคลนหน่วยความจำ
- การใช้หน่วยเก็บที่จัดเตรียมโดยตัวเรียก เมธอดนี้ขอแนะนำให้ใช้ แม้ว่าอินเตอร์เฟสต้องถูกแก้ไขก็ตาม

ตัวอย่างเช่น ฟังก์ชัน **strtoupper** คือการแปลงสตริงให้เป็นตัวพิมพ์ใหญ่สามารถนำมาใช้กับชิ้นส่วนของโค้ดต่อไปนี้ :

```
/* non-reentrant function */
char *strtoupper(char *string)
{
        static char buffer[MAX_STRING_SIZE];
        int index;

        for (index = 0; string[index]; index++)
                buffer[index] = toupper(string[index]);
        buffer[index] = 0

        return buffer;
}
```

ฟังก์ชันนี้ไม่ใช่ reentrant (หรือ threadsafe) หากต้องการสร้างฟังก์ชันแบบ reentrant โดยส่งคืนข้อมูลที่จัดสรรแล้วแบบไดนามิก ฟังก์ชันนั้นต้องคล้ายกับชิ้นส่วนของโค้ดที่แสดงดังต่อไปนี้:

```
/* reentrant function (a poor solution) */
char *strtoupper(char *string)
{
        char *buffer;
        int index;

        /* error-checking should be performed! */
        buffer = malloc(MAX_STRING_SIZE);

        for (index = 0; string[index]; index++)
                buffer[index] = toupper(string[index]);
```

```
        buffer[index] = 0

        return buffer;
}
```

ฟังก์ชันที่ดีกว่าประกอบด้วยการแก้ไขอินเตอร์เฟส ตัวเรียกต้องจัดเตรียมหน่วยเก็บสำหรับสตริงทั้งแบบอินพุต และเอาต์พุต ในชิ้นส่วนของโค้ดต่อไปนี้:

```
/* reentrant function (a better solution) */
char *strtoupper_r(char *in_str, char *out_str)
{
        int index;

        for (index = 0; in_str[index]; index++)
        out_str[index] = toupper(in_str[index]);
        out_str[index] = 0

        return out_str;
}
```

รูทีนย่อยไลบรารีภาษา C มาตรฐานที่ไม่ใช่ reentrant ถูกสร้างให้เป็นแบบ reentrant โดยใช้หน่วยเก็บข้อมูลที่ตัวเรียกจัดเตรียมไว้

การเก็บข้อมูลผ่านการเรียกแบบต่อเนื่อง

ไม่มีข้อมูลที่เก็บไว้ผ่านการเรียกแบบต่อเนื่อง เนื่องจากความแตกต่างของเธรดอาจเรียกฟังก์ชันในแบบต่อเนื่อง ถ้าฟังก์ชันต้องรักษาข้อมูลบางส่วน ผ่านการเรียกในแบบต่อเนื่อง เช่น บัฟเฟอร์การทำงานหรือตัวชี้ ตัวเรียกควรจัดเตรียมข้อมูลนี้

โปรดพิจารณาตัวอย่างต่อไปนี้ ฟังก์ชันจะส่งคืนอักขระตัวพิมพ์เล็กแบบต่อเนื่อง ของสตริง สตริงจะถูกจัดเตรียมเฉพาะกับการเรียกในครั้งแรก ด้วยรูทีนย่อย **strtok** ฟังก์ชันจะส่งคืนค่า 0 เมื่อเข้าถึงจุดสิ้นสุดของสตริง ฟังก์ชันสามารถนำไปใช้ได้ ดังที่แสดงอยู่ในชิ้นส่วนของโค้ดต่อไปนี้:

```
/* non-reentrant function */
char lowercase_c(char *string)
{
        static char *buffer;
        static int index;
        char c = 0;

        /* stores the string on first call */
        if (string != NULL) {
                buffer = string;
                index = 0;
        }

        /* searches a lowercase character */
        for (; c = buffer[index]; index++) {
                if (islower(c)) {
                        index++;
                        break;
```

```
                }
        }
        return c;
}
```

h

ฟังก์ชันนี้ไม่ใช่แบบ reentrant หากต้องการทำเป็นแบบ reentrant ข้อมูลสถิติ นั่นคือตัวแปร **index** ต้องถูกรักษาไว้ด้วยตัวเรียก reentrant ในเวอร์ชันของฟังก์ชันที่สามารถนำมาใช้ได้ ดังที่แสดงอยู่ในชิ้นส่วนของโค้ดต่อไปนี้:

```
/* reentrant function */
char reentrant_lowercase_c(char *string, int *p_index)
{
        char c = 0;

        /* no initialization - the caller should have done it */

        /* searches a lowercase character */
        for (; c = string[*p_index]; (*p_index)++) {
                if (islower(c)) {
                        (*p_index)++;
                        break;
                  }
        }
        return c;
}
```

อินเตอร์เฟสของฟังก์ชันจะเปลี่ยนไป และทำให้การใช้เปลี่ยนไปด้วย ตัวเรียกต้องจัดเตรียมสตริงสำหรับการเรียกแต่ละครั้ง และต้องกำหนดค่าเริ่มต้นของดัชนีให้มีค่า 0 ก่อนการเรียกในครั้งแรก ดังที่แสดงไว้ในชิ้นส่วนของโค้ดต่อไปนี้:

```
char *my_string;
char my_char;
int my_index;
...
my_index = 0;
while (my_char = reentrant_lowercase_c(my_string, &my_index)) {
        ...
}
```

การทำให้ฟังก์ชันเป็น threadsafe

ในโปรแกรมแบบมัลติเธรด ฟังก์ชันทั้งหมดที่เรียกใช้โดยหลายเธรด ต้องเป็น threadsafe อย่างไรก็ตาม วิธีแก้ปัญหาจะมีสำหรับการใช้รูทีนย่อยที่ thread-unsafe ในโปรแกรมแบบมัลติเธรด ฟังก์ชันที่ไม่ใช่ reentrant โดยปกติ เป็น thread-unsafe แต่การทำให้เป็น reentrant มักทำให้เป็น threadsafe เช่นกัน

การล็อกรีซอร์สแบบแบ่งใช้

ฟังก์ชันที่ใช้ข้อมูลสแตติก หรือรีซอร์สที่แบ่งใช้ใดๆ เช่น ไฟล์หรือเทอร์มินัล ต้อง serialize การเข้าถึงรีซอร์สเหล่านี้โดยการล็อกเพื่อให้เป็น threadsafe ตัวอย่างเช่น ฟังก์ชันต่อไปนี้เป็นแบบ thread-unsafe:

```
/* thread-unsafe function */
int increment_counter()
{
```

```
        static int counter = 0;

        counter++;
        return counter;
}
```

เมื่อต้องการให้เป็น threadsafe ตัวแปรสแตติก **counter** ต้องถูกป้องกัน โดยสแตติกล็อก ดังในตัวอย่างต่อไปนี้:

```
/* pseudo-code threadsafe function */
int increment_counter();
{
        static int counter = 0;
        static lock_type counter_lock = LOCK_INITIALIZER;

        pthread_mutex_lock(counter_lock);
        counter++;
        pthread_mutex_unlock(counter_lock);
        return counter;
}
```

ในแอ็พพลิเคชันโปรแกรมแบบมัลติเธรดที่ใช้ไลบรารีเธรด mutexes ควรถูกใช้สำหรับการ serialize รีซอร์สที่แบ่งใช้ ไลบรารีที่เป็นอิสระ อาจจำเป็นต้องทำงานอยู่ภายนอกบริบทของเธรด และใช้ล็อกชนิดอื่น

การแก้ไขปัญหาสำหรับฟังก์ชัน thread-unsafe

มีความเป็นไปได้ที่จะใช้วิธีแก้เพื่อใช้ฟังก์ชันแบบ thread-unsafe ที่ถูกเรียกโดยเธรดจำนวนมาก ซึ่งจะเป็นประโยชน์โดยเฉพาะเมื่อใช้ไลบรารี thread-unsafe ในโปรแกรมแบบมัลติเธรด สำหรับการทดสอบหรือ ขณะรอเวอร์ชัน threadsafe ของไลบรารีให้พร้อมใช้าน วิธีแก้ นำไปสู่การนำไปใช้ในบางส่วน เนื่องจากมีการ serialize ฟังก์ชันทั้งหมดหรือกลุ่มของฟังก์ชัน ต่อไปนี้คือวิธีแก้ที่เป็นไปได้:

- ใช้ล็อกแบบโกลบอลสำหรับไลบรารี และล็อกไว้ในแต่ละครั้งที่คุณใช้ไลบรารี (การเรียกไลบรารีรูทีน หรือการใช้ตัวแปรไลบรารีโกลบอล) โซลูชันนี้ สามารถสร้างผลการทำงานที่เป็นคอขวดได้ เนื่องจากมีเพียงเธรดเดียวเท่านั้นที่สามารถเข้าถึงส่วนใดๆ ของไลบรารี ณ เวลาที่กำหนดไว้ใดๆ โซลูชันในโค้ดจำลองต่อไปนี้ สามารถยอมรับได้ หากไลบรารีถูกเข้าถึงวิธีแก้ที่นำมาใช้ นานๆ ครั้ง หรือขณะที่เริ่มต้น

  ```
  /* this is pseudo code! */

  lock(library_lock);
  library_call();
  unlock(library_lock);

  lock(library_lock);
  x = library_var;
  unlock(library_lock);
  ```

- ใช้ล็อกสำหรับคอมโพเนนต์ไลบรารีแต่ละตัว (ตัวแปรรูทีนหรือตัวแปรโกลบอล) หรือกลุ่มของคอมโพเนนต์ โซลูชันนี้คือสิ่งที่ซับซ้อนในการนำไปปฏิบัติ มากกว่าตัวอย่างก่อนหน้านี้ แต่สามารถปรับปรุงผลการทำงานได้ เนื่องจากวิธีแก้นี้ ควรนำมาใช้ในแอ็พพลิเคชันโปรแกรมและไม่ได้ใช้ในไลบรารี mutexes สามารถนำมาใช้สำหรับการล็อกไลบรารีได้

  ```
  /* this is pseudo-code! */

  lock(library_moduleA_lock);
  library_moduleA_call();
  unlock(library_moduleA_lock);
  ```

```
lock(library_moduleB_lock);
x = library_moduleB_var;
unlock(library_moduleB_lock);
```

ไลบรารี Reentrant และ threadsafe

ไลบรารี Reentrant และ threadsafe เป็นประโยชน์ในขอบเขตที่กว้างของ สภาวะแวดล้อมการโปรแกรมแบบขนาน (และอะซิงโครนัส) ไม่เพียงภายใน เธรด โดยเป็นแนวทางโปรแกรมมิงที่ดีที่จะใช้และเขียนฟังก์ชัน reentrant และ threadsafe เสมอ

การใช้ไลบรารี

หลายๆ ไลบรารีที่มากับ AIX Base Operating System เป็น threadsafe ในเวอร์ชันปัจจุบันของ AIX ไลบรารีต่อไปนี้ เป็น threadsafe:

- ไลบรารี C มาตรฐาน (**libc.a**)
- ไลบรารีที่เข้ากันได้แบบ berkeley (**libbsd.a**)

รูทีนย่อยใน C มาตรฐานบางตัวไม่ใช่ reentrant เช่น รูทีนย่อย **ctime** and **strtok** reentrant ในเวอร์ชันของรูทีนย่อยมีชื่อของรูทีนย่อยเดิมที่มีคำต่อท้าย **_r** (ขีดเส้นใต้แล้วตามด้วยตัวอักษร *r*)

เมื่อเขียนโปรแกรมแบบมัลติเธรด ให้ใช้เวอร์ชัน reentrant ของรูทีนย่อยแทนเวอร์ชันต้นฉบับ ตัวอย่างเช่น ชิ้นส่วนของโค้ดต่อไปนี้:

```
token[0] = strtok(string, separators);
i = 0;
do {
        i++;
        token[i] = strtok(NULL, separators);
} while (token[i] != NULL);
```

ควรแทนด้วยโปรแกรมแบบมัลติเธรดโดยส่วนของ โค้ดต่อไปนี้:

```
char *pointer;
...
token[0] = strtok_r(string, separators, &pointer);
i = 0;
do {
        i++;
        token[i] = strtok_r(NULL, separators, &pointer);
} while (token[i] != NULL);
```

ไลบรารีแบบ thread-unsafe อาจไม่ถูกนำมาใช้โดยเธรดเดียวในโปรแกรม โปรดมั่นใจว่า ความไม่เป็นลักษณะเฉพาะของเธรดที่ใช้ไลบรารี มิฉะนั้นโปรแกรมจะมีลักษณะการทำงานที่ไม่คาดการณ์ไว้ หรืออาจหยุดทำงาน

การแปลงไลบรารี

พิจารณาสิ่งต่อไปนี้เมื่อแปลงไลบรารีที่มีอยู่ไปเป็น ไลบรารี reentrant และ threadsafe ข้อมูลนี้จะใช้กับไลบรารีในภาษา C เท่านั้น

- ระบุตัวแปรโกลบอลที่ถูกเอ็กซ์พอร์ต ตัวแปรเหล่านั้นจะถูกกำหนดในไฟล์ส่วนหัวด้วยคีย์เวิร์ด **export** ตัวแปรโกลบอลที่ถูกเอ็กซ์พอร์ต ควรถูกครอบคลุมไว้ ตัวแปรควรถูกทำให้เป็นส่วนตัว (กำหนดโดยคีย์เวิร์ด **static** ในซอร์สโค้ดไลบรารี) และเข้าถึงรูทีนย่อย (อ่านและเขียน) ควรถูกสร้างขึ้น
- ระบุตัวแปรสแตติก และรีซอร์สแบ่งใช้อื่นๆ ตัวแปรสแตติก จะถูกกำหนดด้วยคีย์เวิร์ด **static** ล็อกควรเชื่อมโยงกับ รีซอร์สแบ่งใช้ใดๆ เศษของการล็อก คือการเลือกจำนวนของล็อก จะได้รับผลกระทบกับผลการทำงานของไลบรารี หากต้องการกำหนดค่าเริ่มต้นให้กับล็อก ตัวช่วยกำหนดค่าเริ่มต้นเพียงครั้งเดียวอาจถูกนำมาใช้
- ระบุการทำงานที่ไม่ใช่ reentrant และทำเป็นแบบ reentrant สำหรับ ข้อมูลเพิ่มเติม โปรดดูที่ การสร้าง ฟังก์ชัน Reentrant
- ระบุฟังก์ชัน thread-unsafe และทำให้เป็น threadsafe สำหรับ ข้อมูลเพิ่มเติม โปรดดูที่ การทำให้ฟังก์ชันเป็น threadsafe

หลักการที่เกี่ยวข้อง:

“การกำหนดค่าเริ่มต้นครั้งเดียว” ในหน้า 510

บางไลบรารี C ถูกออกแบบสำหรับการกำหนดค่าเริ่มต้น แบบไดนามิก ซึ่งการกำหนดค่าเริ่มต้นโกลบอลสำหรับไลบรารีถูกดำเนินการ เมื่อโพรซีเดอร์แรกในไลบรารีถูกเรียก

ข้อมูลที่เกี่ยวข้อง:

admin

cdc

เดลต้า

get

prs

sccsdiff

sccsfile

การพัฒนาโปรแกรมแบบมัลติเธรด

การพัฒนาโปรแกรมแบบมัลติเธรดคล้ายกับการพัฒนา โปรแกรมที่มีหลายกระบวนการ การพัฒนาโปรแกรมยังประกอบด้วย การคอมไพล์ และการดีบักโค้ด

การคอมไพล์โปรแกรมแบบมัลติเธรด

ส่วนนี้อธิบายวิธีสร้างโปรแกรมแบบมัลติเธรด ซึ่งจะอธิบาย:

- ไฟล์ header ที่จำเป็น
- การเรียกใช้คอมไพลเลอร์ ซึ่งใช้เพื่อสร้างโปรแกรม แบบมัลติเธรด

ไฟล์ส่วนหัว

ต้นแบบรูทีนย่อย แมโคร และข้อกำหนดอื่นทั้งหมด สำหรับใช้ไลบรารีเธรดอยู่ในไฟล์ header **pthread.h** ซึ่งอยู่ในไดเร็กทอรี **/usr/include** ไฟล์ส่วนหัว **pthread.h** ต้องถูกรวมในซอร์สไฟล์แต่ละไฟล์โดยใช้ไลบรารีเธรด

ส่วนหัว **pthread.h** จะรวมส่วนหัว **unistd.h** ซึ่งให้คำนิยามโกลบอลต่อไปนี้:

`_POSIX_REENTRANT_FUNCTIONS`
ระบุว่าฟังก์ชันทั้งหมดควรเป็น reentrant หลายไฟล์ header หลายไฟล์ใช้สัญลักษณ์นี้เพื่อกำหนดรูทีนย่อย reentrant เสริม เช่น รูทีนย่อย **localtime_r**

_POSIX_THREADS
แสดงถึง POSIX threads API สัญลักษณ์นี้ถูกใช้เพื่อตรวจสอบว่า POSIX threads API พร้อมใช้งานหรือไม่ แมโครหรือรูทีนย่อยอาจถูกกำหนดในวิธีต่างกัน ขึ้นกับว่า POSIX หรือเธรด API ถูกใช้อยู่

ไฟล์ **pthread.h** ยังมี **errno.h** ที่ตัวแปรโกลบอล **errno** ถูกกำหนดใหม่สำหรับเธรดโดยเฉพาะ ดังนั้น ตัวบ่งชี้ **errno** จะไม่เป็น **l-value** ในโปรแกรม แบบมัลติเธรดอีกต่อไป

การเรียกใช้คอมไพลเลอร์

เมื่อคอมไพล์โปรแกรมแบบมัลติเธรด เรียกใช้คอมไพลเลอร์ C โดยใช้ หนึ่งในคำสั่งต่อไปนี้:

xlc_r
เรียกคอมไพลเลอร์ด้วยระดับภาษาเริ่มต้นของ **ansi**

cc_r
เรียกคอมไพลเลอร์ด้วยระดับภาษาเริ่มต้นของ **extended**

คำสั่งเหล่านี้ประกันว่าตัวเลือกและไลบรารี ที่เพียงพอถูกใช้เพื่อเข้ากันได้กับ Single UNIX Specification, Version 2 POSIX Threads Specification 1003.1c เป็นเซ็ตย่อยของ Single UNIX Specification, Version 2

ไลบรารีดังต่อไปนี้ถูกเชื่อมโยงโดยอัตโนมัติกับโปรแกรมของคุณ เมื่อใช้คำสั่ง **xlc_r** และ **cc_r**:

libpthreads.a
ไลบรารีเธรด

libc.a
ไลบรารี C มาตรฐาน

ตัวอย่างเช่น คำสั่งต่อไปนี้คอมไพล์ซอร์สไฟล์ C มัลติเธรด **foo.c** และสร้างไฟล์เรียกทำงาน **foo**:

```
cc_r -o foo foo.c
```

การเรียกใช้คอมไพลเลอร์สำหรับ draft 7 ของ POSIX 1003.1c

AIX จัดเตรียมความเข้ากันได้ขอซอร์สโค้ดสำหรับแอ็พพลิเคชัน Draft 7 ขอแนะนำให้ผู้พัฒนาพอร์ตแอ็พพลิเคชันเธรดไปที่มาตรฐานล่าสุด

เมื่อคอมไพล์โปรแกรมแบบมัลติเธรดสำหรับการสนับสนุน Draft 7 ของเธรด เรียกใช้คอมไพลเลอร์ C โดยใช้หนึ่งในคำสั่งต่อไปนี้:

xlc_r7
เรียกคอมไพลเลอร์ด้วยระดับภาษาเริ่มต้นของ **ansi**

cc_r7
เรียกคอมไพลเลอร์ด้วยระดับภาษาเริ่มต้นของ **extended**

ไลบรารีดังต่อไปนี้ถูกเชื่อมโยงโดยอัตโนมัติกับโปรแกรมของคุณ เมื่อใช้คำสั่ง **xlc_r7** และ **cc_r7**:

libpthreads_compat.a
ไลบรารีเธรดความเข้ากันได้ของ Draft 7

`libpthreads.a`
ไลบรารีเธรด

`libc.a`
ไลบรารี C มาตรฐาน

เพื่อให้มีความเข้ากันได้ของซอร์สโค้ด ใช้คอมไพลเลอร์ไดเรกทีฟ **_AIX_PTHREADS_D7** เป็นสิ่งจำเป็นที่จะเชื่อมโยงไลบรารีใน ลำดับดังต่อไปนี้: **libpthreads_compat.a, libpthreads.a** และ **libc.a** ผู้ใช้ส่วนใหญ่ไม่จำเป็นต้อง ทราบข้อมูลนี้ เนื่องจากคำสั่งจัดให้มีตัวเลือกที่จำเป็น ตัวเลือกเหล่านี้ถูกจัดเตรียมให้สำหรับผู้ที่ไม่มีคอมไพลเลอร์ AIX ล่าสุด

การพอร์ตแอ็พพลิเคชัน draft 7 ไปยัง &Symbol.unixspec;

ความแตกต่างมีอยู่ระหว่าง Draft 7 และมาตรฐานสุดท้ายรวมทั้ง:

- ความแตกต่าง **errno** ส่วนน้อย ที่พบได้บ่อยที่สุดคือ การใช้ **ESRCH** เพื่อแสดงว่าไม่พบ pthread ที่ระบุ บ่อยครั้งที่ Draft 7 ส่งกลับ **EINVAL** สำหรับ ความล้มเหลวนี้
- ภาวะเริ่มต้นเมื่อ pthread ถูกสร้างคือ *joinable* นี่เป็นการเปลี่ยนแปลงที่สำคัญเนื่องจาก อาจมีผลให้หน่วยความจำรั่วไหล ถ้าละเว้น
- พารามิเตอร์การจัดตารางเวลา pthread เริ่มต้นคือ *scope*
- รูทีนย่อย **pthread_yield** ได้ถูกแทนโดย รูทีนย่อย **sched_yield**
- นโยบายการจัดตารางเวลาต่างๆ ที่สัมพันธ์กับการล็อก mutex ต่างกัน นิดหน่อย

ข้อกำหนดหน่วยความจำของโปรแกรมแบบมัลติเธรด

AIX สนับสนุนถึง 32768 เธรดในกระบวนการเดี่ยว แต่ละ pthread ต้องการพื้นที่แอดเดรสกระบวนการจำนวนหนึ่ง ดังนั้น จำนวน pthreads จริงสูงสุดที่กระบวนการจะมีได้ขึ้นอยู่กับโมเดลหน่วยความจำและ การใช้พื้นที่แอดเดรสของกระบวนการสำหรับวัตถุประสงค์อื่น จำนวนของหน่วยความจำ ที่ pthread ต้องการรวมถึงขนาดสแต็กและขนาดขอบเขตป้องกัน บวก พื้นที่บางส่วนสำหรับการใช้ภายใน ผู้ใช้สามารถควบคุมขนาดของสแต็กด้วย รูทีนย่อย **pthread_attr_setstacksize** และขนาดของพื้นที่ป้องกันด้วยรูทีนย่อย **pthread_attr_setguardsize**

หมายเหตุ: ข้อจำกัดอย่างน้อย บนขนาดสแต็กกำหนดโดยคำสั่ง **ulimit** *–s* ใช้เฉพาะ กับสแต็กของเธรดหลักของแอ็พพลิเคชัน

ตารางดังต่อไปนี้บ่งชี้จำนวนสูงสุดของ pthreads ที่ควร ถูกสร้างในกระบวนการ 32–บิต โดยใช้โปรแกรมธรรมดา ซึ่งไม่ทำงานอย่างอื่น นอกจากสร้าง pthreads ในลูปโดยใช้แอ็ตทริบิวต์ NULL pthread ในโปรแกรม จริง ตัวเลขจริงขึ้นอยู่กับการใช้งานหน่วยความจำในโปรแกรม สำหรับ กระบวนการ 64–บิต รูทีนย่อย **ulimit** ควบคุมจำนวนเธรด ที่สามารถสร้างได้ ดังนั้นโมเดลข้อมูลขนาดใหญ่ไม่จำเป็น และโดยข้อเท็จจริง สามารถลดจำนวนของเธรดสูงสุดได้

โมเดลข้อมูล	-bmaxdata	Pthreads สูงสุด
ข้อมูลขนาดเล็ก	n/a	1084
ข้อมูลขนาดใหญ่	0x10000000	2169
ข้อมูลขนาดใหญ่	0x20000000	4340
ข้อมูลขนาดใหญ่	0x30000000	6510
ข้อมูลขนาดใหญ่	0x40000000	8681
ข้อมูลขนาดใหญ่	0x50000000	10852
ข้อมูลขนาดใหญ่	0x60000000	13022
ข้อมูลขนาดใหญ่	0x70000000	15193
ข้อมูลขนาดใหญ่	0x80000000	17364

ตัวแปรสภาวะแวดล้อม **NUM_SPAREVP** สามารถถูกตั้งค่าเพื่อควบคุมจำนวนของ โพรเซสเซอร์เสมือนสำรอง ที่ถูกรักษาโดยไลบรารี ไม่ จำเป็นที่จะต้องปรับเปลี่ยนตัวแปรนี้ ในบางสถานการณ์ แอ็พพลิเคชัน ที่ใช้เพียงหน่วยความจำไม่กี่เมกะไบต์ สามารถลดการสิ้นเปลืองหน่วยความจำโดย การตั้งค่าตัวแปรสภาวะแวดล้อม **NUM_SPAREVP** ให้มีค่าต่ำลง การตั้งค่าปกติรวมทั้งจำนวนของ CPUs บนระบบหรือหมายเลขสูงสุด ของเธรดกระบวนการ การตั้งค่าตัวแปรนี้ไม่มีผลกับประสิทธิภาพของกระบวนการ ค่ากำหนดดีฟอลต์คือ 256

หมายเหตุ: ตัวแปรสภาวะแวดล้อม **NUM_SPAREVP** พร้อมใช้งานเฉพาะใน AIX 5.1

ตัวอย่างของโปรแกรมแบบมัลติเธรด

โปรแกรมแบบมัลติเธรดสั้นๆ ต่อไปนี้ แสดง "Hello!" ทั้งในภาษาอังกฤษ และฝรั่งเศสเป็นเวลาห้าวินาที คอมไพล์ด้วย **cc_r** หรือ **xlc_r** F

```
#include <pthread.h>    /* include file for pthreads - the 1st */
#include <stdio.h>      /* include file for printf()           */
#include <unistd.h>     /* include file for sleep()            */

void *Thread(void *string)
{
        while (1)
                printf("%s\n", (char *)string);
        pthread_exit(NULL);
}

int main()
{
        char *e_str = "Hello!";
        char *f_str = "Bonjour !";

        pthread_t e_th;
        pthread_t f_th;

        int             rc;

        rc = pthread_create(&e_th, NULL, Thread, (void *)e_str);
```

```
        if (rc)
                exit(-1);
        rc = pthread_create(&f_th, NULL, Thread, (void *)f_str);
        if (rc)
                exit(-1);
        sleep(5);

        /* usually the exit subroutine should not be used
           see below to get more information */
        exit(0);
}
```

initial thread (ดำเนินรูทีน **main**) สร้างสองเธรด ทั้งสองเธรดมีรูทีน entry-point เหมือนกัน (รูทีน **Thread**) แต่พารามิเตอร์ต่างกัน พารามิเตอร์เป็นตัวชี้ไปที่สตริงซึ่งจะถูกแสดง

การดีบักโปรแกรมแบบมัลติเธรด

เครื่องมือต่อไปนี้มีให้ใช้สำหรับดีบักโปรแกรมแบบมัลติเธรด:

- โปรแกรมเมอร์แอ็พพลิเคชันสามารถ ใช้คำสั่ง **dbx** เพื่อ ทำการดีบัก หลายคำสั่งย่อยพร้อมใช้งานสำหรับการแสดงอ็อบเจ็กต์ ที่สัมพันธ์กับเธรด รวมถึง **attribute, condition, mutex** และ **thread**
- โปรแกรมเมอร์เคอร์เนลสามารถใช้โปรแกรมดีบักเคอร์เนลเพื่อทำการดีบักบนส่วนขยายเคอร์เนลและ ไดรเวอร์อุปกรณ์ โปรแกรมดีบักเคอร์เนลจัดเตรียมการเข้าถึงที่จำกัดแก่เธรด user และจัดการเธรดเคอร์เนลเป็นหลัก มีหลายคำสั่งย่อยสนับสนุนหลายเธรด เคอร์เนลและโพรเซสเซอร์ รวมทั้ง:
  - คำสั่งย่อย **cpu**, ซึ่งเปลี่ยนโพรเซสเซอร์ปัจจุบัน
  - คำสั่งย่อย **ppd**, ซึ่งแสดงโครงสร้างข้อมูลต่อโพรเซสเซอร์
  - คำสั่งย่อย **thread**, ซึ่งแสดงรายการตารางเธรด
  - คำสั่งย่อย **uthread** , ซึ่งแสดงโครงสร้าง **uthread** ของเธรด

  สำหรับข้อมูลเพิ่มเติมเกี่ยวกับโปรแกรมดีบักเคอร์เนล โปรดดูที่ *Kernel Extensions and Device Support Programming Concepts*

ข้อกำหนด Core File ของโปรแกรมแบบมัลติเธรด

ตามค่าเริ่มต้น กระบวนการไม่สร้างไฟล์คอร์แบบเต็มสมบูรณ์ ถ้าแอ็พพลิเคชัน ต้องดีบักข้อมูลในขอบเขตหน่วยความจำที่แบ่งใช้ สแต็กเธรดจำเพาะ แอ็พพลิเคชัน จำเป็นต้องสร้างดัมพ์คอร์แบบสมบูรณ์ เมื่อต้องการสร้างข้อมูลไฟล์คอร์ ให้รันคำสั่งดังต่อไปนี้เป็นผู้ใช้ root:

```
chdev -l sys0 -a fullcore=true
```

แต่ละ pthread เพิ่มให้กับขนาดของไฟล์คอร์ที่สร้าง จำนวน ของพื้นที่ไฟล์คอร์ที่ pthread ต้องการรวมถึงขนาดสแต็ก ซึ่งผู้ใช้สามารถควบคุมด้วยรูทีนย่อย **pthread_attr_setstacksize** สำหรับ pthreads ที่สร้างด้วยแอ็ตทริบิวต์ NULL pthread, แต่ละ pthread ในกระบวนการ 32-บิต เพิ่มขนาด 128 KB ให้กับขนาดของไฟล์คอร์ และแต่ละ pthread ในกระบวนการ 64-บิต เพิ่ม 256 KB ให้กับขนาดของไฟล์คอร์

หลักการที่เกี่ยวข้อง:

"ไลบรารี Threadsafe และที่ถูกเธรดใน AIX" ในหน้า 462
ส่วนนี้อธิบายเธรดไลบรารีใน AIX

“การสร้างเธรด” ในหน้า 463
การสร้างเธรดต่างจากการสร้างกระบวนการคือ ไม่มีความสัมพันธ์ parent-child ระหว่างเธรด

“เธรดการกำหนดเวลา” ในหน้า 501
เธรดสามารถกำหนดตารางเวลาได้ และไลบรารีเธรดจะจัดเตรียมตัวช่วยต่างๆ เพื่อจัดการและควบคุมการกำหนดตารางเวลาของเธรด

“การพัฒนาโปรแกรมแบบมัลติเธรด” ในหน้า 544
การพัฒนาโปรแกรมแบบมัลติเธรดคล้ายกับการพัฒนา โปรแกรมที่มีหลายกระบวนการ การพัฒนาโปรแกรมยังประกอบด้วยการคอมไพล์ และการดีบักโค้ด

การพัฒนาโปรแกรมแบบมัลติเธรดเพื่อตรวจสอบและแก้ไขไลบรารีอ็อบเจ็กต์ pthread

ไลบรารีดีบัก pthread (**libpthdebug.a**) จัดเตรียม ชุดของฟังก์ชันที่ช่วยให้ผู้พัฒนาแอ็พพลิเคชันตรวจสอบและปรับเปลี่ยนอ็อบเจ็กต์ไลบรารี pthread

ไลบรารีนี้สามารถถูกใช้สำหรับทั้งแอ็พพลิเคชัน 32-บิต และ 64-บิต ไลบรารีนี้เป็น threadsafe ไลบรารีดีบัก pthread มีอ็อบเจ็กต์ที่แบ่งใช้ 32-บิต และ 64-บิต

ไลบรารีดีบัก pthread จัดเตรียมแอ็พพลิเคชันที่มีการเข้าถึงข้อมูลไลบรารี pthread ซึ่งรวมถึงข้อมูลเกี่ยวกับ pthreads แอ็ตทริบิวต์ pthread, mutexes, แอ็ตทริบิวต์ mutex, condition variables แอ็ตทริบิวต์ condition variable, read/write locks แอ็ตทริบิวต์ read/write lock และข้อมูลเกี่ยวกับภาวะของ ไลบรารี pthread

หมายเหตุ: ข้อมูลทั้งหมด (แอดเดรส รีจิสเตอร์) ที่ส่งกลับโดยไลบรารีอยู่ในรูปแบบ 64-บิต ทั้งสำหรับแอ็พพลิเคชัน 64-บิต และ 32-บิต เป็นความรับผิดชอบ ของแอ็พพลิเคชันในการแปลงค่าเหล่านี้เป็นรูปแบบ 32-บิตสำหรับแอ็พพลิเคชัน 32-บิต เมื่อดีบักแอ็พพลิเคชัน 32-บิต แอดเดรสและรีจิสเตอร์ ครึ่งบน จะถูกละเว้น

ไลบรารีดีบักไม่รายงานข้อมูลบน mutexes แอ็ตทริบิวต์ mutex, condition variables แอ็ตทริบิวต์ condition variable, read/write locks และแอ็ตทริบิวต์ read/write lock ซึ่งมีค่า pshared เป็น **PTHREAD_PROCESS_SHARED**

การกำหนดค่าเริ่มต้น

แอ็พพลิเคชันต้องกำหนดค่าเริ่มต้นเซสชันไลบรารีดีบัก pthread สำหรับแต่ละ กระบวนการ pthreaded ฟังก์ชัน **pthdb_sessison_init** ต้องถูกเรียกจากแต่ละกระบวนการ pthreaded หลังจากกระบวนการ ได้ถูกโหลด ไลบรารีดีบัก pthread สนับสนุนหนึ่งเซสชันสำหรับ กระบวนการเดี่ยว แอ็พพลิเคชันต้องกำหนดตัวระบุผู้ใช้เฉพาะและส่งไปที่ ฟังก์ชัน **pthdb_session_init** ซึ่งตามลำดับจะกำหนด ตัวระบุเซสชันเฉพาะ ที่ต้องถูกผ่านเป็นพารามิเตอร์แรกให้กับ ฟังก์ชัน **pthread debug library** อื่น นอกจากฟังก์ชัน **pthdb_session_pthreaded** ตามลำดับ เมื่อใดก็ตามที่ไลบรารีดีบัก pthread ร้องขอฟังก์ชัน call back ไลบรารีจะส่งแอ็พพลิเคชันเฉพาะที่กำหนดรหัสผู้ใช้ กลับไปที่แอ็พพลิเคชัน ฟังก์ชัน **pthdb_session_init** ตรวจสอบรายการของฟังก์ชัน call back ที่จัดเตรียมโดยแอ็พพลิเคชัน และกำหนดค่าเริ่มต้น โครงสร้างข้อมูลของเซสชัน นอกจากนี้ ฟังก์ชันนี้ตั้งค่าแฟล็กเซสชัน แอ็พพลิเคชันต้องผ่านแฟล็ก **PTHDB_FLAG_SUSPEND** ไปที่ฟังก์ชัน **pthdb_session_init** โปรดดูที่ฟังก์ชัน **pthdb_session_setflags** สำหรับรายการที่สมบูรณ์ของแฟล็ก

ฟังก์ชัน Call back

ไลบรารีดีบัก pthread ใช้ฟังก์ชัน call back เพื่อรับและเขียนข้อมูล เช่นเดียวกับให้การจัดการพื้นที่จัดเก็บข้อมูลกับแอ็พพลิเคชัน ฟังก์ชัน call back ที่จำเป็นสำหรับแอ็พพลิเคชันมีดังนี้:

read_data
เรียกข้อมูลอ็อบเจ็กต์ **ไลบรารี pthread**

alloc จัดสรรหน่วยความจำในไลบรารีดีบัก pthread

realloc จัดสรรหน่วยความจำใหม่ในไลบรารีดีบัก pthread

dealloc ทำหน่วยความจำที่จัดสรรในไลบรารีดีบัก pthread ให้ว่าง

ฟังก์ชัน call back ที่เป็นทางเลือกสำหรับแอ็พพลิเคชันมีดังนี้:

read_regs
จำเป็นเฉพาะสำหรับรูทีนย่อย **pthdb_pthread_context** และ **pthdb_pthread_setcontext**

write_data
จำเป็นเฉพาะสำหรับรูทีนย่อย **pthdb_pthread_setcontext**

write_regs
จำเป็นเฉพาะสำหรับรูทีนย่อย **pthdb_pthread_setcontext**

ฟังก์ชัน Update

แต่ละครั้งที่แอ็พพลิเคชันหยุดทำงาน หลังจากเซสชันได้ถูกกำหนดค่าเริ่มต้น จำเป็นต้องมีการเรียกฟังก์ชัน **pthdb_session_update** ฟังก์ชันนี้ตั้งค่าหรือรีเซ็ตรายการของ pthreads, แอ็ตทริบิวต์ pthread, mutexes, แอ็ตทริบิวต์ mutex, condition variables, แอ็ตทริบิวต์condition variable, read/write locks, แอ็ตทริบิวต์ read/write lock, คีย์จำเพาะ pthread และคีย์แอคทีฟ โดยใช้ฟังก์ชัน call back เพื่อจัดการหน่วยความจำสำหรับ รายการ

ฟังก์ชัน Context

ฟังก์ชัน **pthdb_pthread_context** รับข้อมูล context และฟังก์ชัน **pthdb_pthread_setcontext** ตั้งค่า context ฟังก์ชัน **pthdb_pthread_context** รับข้อมูล context ของ pthread จากเคอร์เนลหรือโครงสร้างข้อมูล pthread ในพื้นที่แอดเดรส ของแอ็พพลิเคชัน ถ้า pthread ไม่ถูกเชื่อมโยงกับเธรดเคอร์เนล ข้อมูล context จะถูกบันทึกโดยไลบรารีที่ได้รับ ถ้า pthread เชื่อมโยงกับเธรดเคอร์เนล ข้อมูลได้รับจากแอ็พพลิเคชัน โดยใช้ฟังก์ชัน call back แอ็พพลิเคชันต้องระบุ ว่าเธรดเคอร์เนลอยู่ในโหมด kernel หรือ user และจากนั้นจัดเตรียม ข้อมูลที่ถูกต้องสำหรับโหมดนั้น

เมื่อ pthread ที่มีเธรดโหมดอยู่ในโหมด kernel, คุณไม่สามารถกรับบริบทโหมด full user เนื่องจากเคอร์เนลไม่ได้บันทึกบริทบที่จุดเดียว ฟังก์ชัน **getthrds** สามารถถูกใช้เพื่อรับส่วนของข้อมูลนี้ เนื่องจากฟังก์ชันจะบันทึกสแตกโหมด user เสมอ แอ็พพลิเคชัน สามารถพบข้อมูลนี้ได้โดยตรวจสอบโครงสร้าง **thrdsinfo64.ti_scount** ถ้าค่าไม่เป็นศูนย์ สแตกโหมด user พร้อมใช้งาน ในโครงสร้าง **thrdsinfo64.ti_ustk** จากสแต็กโหมด user เป็นไปได้ที่จะ ระบุ instruction address register (IAR) และเฟรม call back แต่ทำไม่ได้กับค่ารีจิสเตอร์อื่น โครงสร้าง **thrdsinfo64** ถูกกำหนดในไฟล์ **procinfo.h**

ฟังก์ชันลิสต์

ไลบรารีดีบัก pthread รักษารายการสำหรับ pthreads, แอ็ตทริบิวต์ pthread, mutexes, แอ็ตทริบิวต์ mutex, condition variables, แอ็ตทริบิวต์ condition variables, read/write locks, แอ็ตทริบิวต์ read/write lock, คีย์จำเพาะ pthread และคีย์แอคทีฟ, แต่ละค่าแสดงโดยแฮนเดิลจำเพาะชนิด ฟังก์ชัน **pthdb_*object*** ส่งกลับ แฮนเดิลถัดไปในรายการที่เหมาะสม โดยที่ *object* คือ หนึ่งในข้อมูลต่อไปนี้: **pthread, attr, mutex, mutexattr, cond, condattr, rwlock, rwlockattr** หรือ **key** ถ้ารายการ ว่างหรือถึงจุดสิ้น

สุดของรายการ **PTHDB_INVALID_*OBJECT*** จะถูกรายงานโดยที่ *OBJECT* คือหนึ่งในข้อมูลต่อไปนี้: **PTHREAD, ATTR, MUTEX, MUTEXATTR, COND, CONDATTR, RWLOCK, RWLOCKATTR** หรือ **KEY**

ฟังก์ชัน Field

ข้อมูลละเอียดเกี่ยวกับอ็อบเจ็กต์สามารถรับได้โดยการใช้ ฟังก์ชันสมาชิกอ็อบเจ็กต์ **pthdb_*object_field*** โดยที่ *object* คือหนึ่งในข้อมูลต่อไปนี้: **pthread, attr, mutex, mutexattr, cond, condattr, rwlock, rwlockattr** หรือ **key** และโดยที่ *field* คือชื่อของฟิลด์ของข้อมูลโดยละเอียด สำหรับอ็อบเจ็กต์.

การกำหนดเซสชันเอง

ฟังก์ชัน **pthdb_session_setflags** อนุญาตให้แอ็พพลิเคชันเปลี่ยนแปลงแฟล็กที่กำหนดค่าเซสชันเอง แฟล็กเหล่านี้ควบคุมจำนวนของรีจิสเตอร์ที่ถูกอ่านหรือเขียนระหว่างการดำเนินการ context

ฟังก์ชัน **pthdb_session_flags** ได้รับแฟล็กปัจจุบันสำหรับเซสชัน

การยกเลิกเซสชัน

ที่จุดสิ้นสุดของเซสชัน โครงสร้างข้อมูลเซสชันต้องถูกคืนค่าที่จัดสรรมา และข้อมูลเซสชันต้องถูกลบ ซึ่งทำได้โดยการเรียกฟังก์ชัน **pthdb_session_destroy** ซึ่งใช้ฟังก์ชัน call back เพื่อคืนหน่วยความจำ หน่วยความจำทั้งหมด ที่จัดสรรโดยฟังก์ชัน **pthdb_session_init** และ **pthdb_session_update** จะถูกคืนข้อมูลที่จัดสรร

ตัวอย่างของการเชื่อมต่อกับไลบรารีดีบัก debug

ตัวอย่างดังต่อไปนี้แสดงวิธีที่แอ็พพลิเคชันสามารถเชื่อมต่อกับไลบรารีดีบัก pthread:

```
/* includes */

#include &lt;thread.h>
#include &lt;ys/pthdebug.h>

...

int my_read_data(pthdb_user_t user, pthdb_symbol_t symbols[],int count)
{
  int rc;

  rc=memcpy(buf,(void *)addr,len);
  if (rc==NULL) {
    fprintf(stderr,&odq;Error message\n&cdq;);
    return(1);
  }
  return(0);
}
int my_alloc(pthdb_user_t user, size_t len, void **bufp)
{
  *bufp=malloc(len);
  if(!*bufp) {
    fprintf(stderr,&odq;Error message\n&cdq;);
    return(1);
  }
```

```
  return(0);
}
int my_realloc(pthdb_user_t user, void *buf, size_t len, void **bufp)
{
  *bufp=realloc(buf,len);
  if(!*bufp) {
    fprintf(stderr,"Error message\n");
    return(1);
  }
  return(0);
}
int my_dealloc(pthdb_user_t user,void *buf)
{
  free(buf);
  return(0);
}

status()
{
  pthdb_callbacks_t callbacks =
                   {  NULL,
                      my_read_data,
                      NULL,
                      NULL,
                      NULL,
                      my_alloc,
                      my_realloc,
                      my_dealloc,
                      NULL
                   };

  ...

  rc=pthread_suspend_others_np();
  if (rc!=0)
    deal with error

  if (not initialized)
    rc=pthdb_session_init(user,exec_mode,PTHDB_SUSPEND|PTHDB_REGS,callbacks,
                          &session);
    if (rc!=PTHDB_SUCCESS)
       deal with error

  rc=pthdb_session_update(session);
  if (rc!=PTHDB_SUCCESS)
        deal with error

   retrieve pthread object information using the object list functions and
   the object field functions

  ...

  rc=pthread_continue_others_np();
  if (rc!=0)
    deal with error
```

```
}

...

main()
{
  ...
}
```

การพัฒนาดีบักเกอร์โปรแกรมแบบมัลติเธรด

ไลบรารีดีบัก pthread (**libpthdebug.a**) จัดเตรียม ชุดของฟังก์ชันที่อนุญาตให้ผู้พัฒนาจัดเตรียมความสามารถในการดีบักสำหรับ แอ็พพลิเคชันที่ใช้ไลบรารี pthread

ไลบรารีดีบัก pthread ถูกใช้เพื่อดีบักแอ็พพลิเคชัน pthread ทั้ง 32-บิต และ 64-บิต ไลบรารีนี้ถูกใช้เพื่อดีบักกระบวนการดีบักที่เป็นเป้าหมายเท่านั้น และยังสามารถถูกใช้เพื่อตรวจสอบข้อมูล pthread ของตัวแอ็พพลิเคชันของไลบรารีเอง ไลบรารีนี้สามารถใช้โดยดีบักเกอร์ แบบมัลติเธรดเพื่อดีบักแอ็พพลิเคชันแบบมัลติเธรด ดีบักเกอร์แบบมัลติเธรด ได้รับการสนับสนุนในไลบรารี **libpthreads.a** ซึ่งเป็น threadsafe ไลบรารีดีบัก pthread มีอ็อบเจ็กต์ที่แบ่งใช้ 32-บิต และ 64-บิต

ดีบักเกอร์ใช้ความสามารถ ptrace ต้องเชื่อมโยงไปที่เวอร์ชัน 32-บิต ของ ไลบรารี เนื่องจากความสามารถ ptrace ไม่ถูกสนับสนุนในโหมด 64-บิต ดีบักเกอร์ที่ใช้ความสามารถ /proc สามารถเชื่อมโยงไปที่ไลบรารีนี้ทั้งเวอร์ชัน 32-บิต หรือ 64-บิต

ไลบรารีดีบัก pthread จัดเตรียมดีบักเกอร์ที่มีการเข้าถึงข้อมูลไลบรารี pthread ซึ่งรวมถึงข้อมูลเกี่ยวกับ pthreads, แอ็ตทริบิวต์ pthread, mutexes, แอ็ตทริบิวต์ mutex, condition variables, แอ็ตทริบิวต์ condition variable, read/write locks, แอ็ตทริบิวต์ read/write lock และข้อมูลเกี่ยวกับภาวะของ ไลบรารี pthread ไลบรารีนี้ยังจัดเตรียมวิธีใช้พร้อมกับการควบคุมการกระทำการของ pthreads

หมายเหตุ: ข้อมูลทั้งหมด (แอดเดรส รีจิสเตอร์) ที่ส่งกลับโดยไลบรารีอยู่ในรูปแบบ 64-บิต ทั้งสำหรับแอ็พพลิเคชัน 64-บิต และ 32-บิต เป็นความรับผิดชอบของดีบักเกอร์ในการแปลงค่าเหล่านี้เป็นรูปแบบ 32-บิตสำหรับแอ็พพลิเคชัน 32-บิต เมื่อดีบักแอ็พพลิเคชัน 32-บิต แอดเดรสและรีจิสเตอร์ ครึ่งบน จะถูกละเว้น

ไลบรารีดีบัก pthread ไม่รายงาน mutexes, แอ็ตทริบิวต์ mutex, condition variables, แอ็ตทริบิวต์ condition variable, read/write locks และแอ็ตทริบิวต์ read/write lock ที่มีค่า pshared เป็น PTHREAD_PROCESS_SHARED

การกำหนดค่าเริ่มต้น

ดีบักเกอร์ต้องกำหนดค่าเริ่มต้นเซสชันไลบรารีดีบัก pthread สำหรับแต่ละ กระบวนการดีบัก ซึ่งทำไม่ได้จนกว่าไลบรารี pthread ได้ถูกกำหนดค่าเริ่มต้น ในกระบวนการดีบัก ฟังก์ชัน **pthdb_session_pthreaded** ได้ถูกจัดเตรียมเพื่อแจ้งดีบักเกอร์เมื่อไลบรารี pthread ได้ถูก กำหนดค่าเริ่มต้นในกระบวนการดีบัก แต่ละครั้งที่ฟังก์ชัน **pthdb_session_pthreaded** ถูกเรียกฟังก์ชันจะ ตรวจสอบเพื่อดูว่าไลบรารี pthread ได้ถูกกำหนดค่าเริ่มต้นหรือไม่ ถ้ามีการกำหนดค่าเริ่มต้น จะส่งกลับ PTHDB_SUCCESS มิฉะนั้นจะส่งกลับ PTHDB_NOT_PTHREADED ในทั้งสองกรณี ฟังก์ชันส่งกลับชื่อฟังก์ชันซึ่ง สามารถถูกใช้เพื่อตั้งค่าจุดหยุดสำหรับการแจ้งเตือนทันทีซึ่งไลบรารี pthread ได้ถูกกำหนดค่าเริ่มต้น ดังนั้นฟังก์ชัน **pthdb_session_pthreaded** จัดเตรียมเมธอดดังต่อไปนี้สำหรับกำหนดเวลาที่ไลบรารี pthread ได้ถูกกำหนดค่าเริ่มต้น:

- ดีบักเกอร์เรียกฟังก์ชันแต่ละครั้งที่กระบวนการดีบักหยุด เพื่อดู ว่าโปรแกรมที่กำลังถูกดีบักถูก pthreaded หรือไม่

- ดีบักเกอร์เรียกฟังก์ชันหนึ่งครั้ง และถ้าโปรแกรมที่กำลังถูก ดีบักไม่ถูก pthreaded, ตั้งค่าจุดหยุดเพื่อแจ้งดีบักเกอร์เมื่อกระบวนการดีบักถูก pthreaded

หลังจากกระบวนการดีบักถูก pthreaded, ดีบักเกอร์ต้องเรียกฟังก์ชัน **pthdb_session_init**, เพื่อกำหนดค่าเริ่มต้นเซสชันสำหรับกระบวนการดีบัก ไลบรารีดีบัก pthread สนับสนุนหนึ่งเซสชันสำหรับ กระบวนการดีบักเดี่ยว ดีบักเกอร์ต้องกำหนดตัวระบุผู้ใช้เฉพาะ และส่งไปที่ **pthdb_session_init** ซึ่งตามลำดับจะกำหนด ตัวระบุเซสชันเฉพาะ ซึ่งต้องถูกผ่านเป็นพารามิเตอร์แรก ไปที่ฟังก์ชันไลบรารีดีบัก pthread อื่น ยกเว้น **pthdb_session_pthreaded**, ตามลำดับ เมื่อใดก็ตามที่ไลบรารีดีบัก pthread ร้องขอฟังก์ชัน call back ไลบรารีจะส่งดีบักเกอร์เฉพาะที่กำหนดรหัสผู้ใช้กลับไปที่ ดีบักเกอร์ ฟังก์ชัน **pthdb_session_init** ตรวจสอบ รายการของฟังก์ชัน call back ที่จัดเตรียมโดยดีบักเกอร์ และกำหนดค่าเริ่มต้น โครงสร้างข้อมูลของเซสชัน นอกจากนี้ฟังก์ชันนี้ตั้งค่าแฟล็กเซสชัน โปรดดูที่ฟังก์ชัน **pthdb_session_setflags** ใน *Technical Reference: Base Operating System and Extensions, Volume 1*

ฟังก์ชัน Call back

ไลบรารีดีบัก pthread ใช้ฟังก์ชัน call back เพื่อทำงานดังต่อไปนี้:
- รับแอดเดรสและข้อมูล
- เขียนข้อมูล
- ให้การจัดการพื้นที่จัดเก็บข้อมูลกับดีบักเกอร์
- ช่วยการดีบักของไลบรารีดีบัก pthread

ฟังก์ชัน Update

แต่ละครั้งที่ดีบักเกอร์หยุดทำงาน หลังจากเซสชันได้ถูกกำหนดค่าเริ่มต้น จำเป็นต้องมีการเรียกฟังก์ชัน **pthdb_session_update** ฟังก์ชันนี้ตั้งค่าหรือรีเซ็ตรายการของ pthreads, แอ็ตทริบิวต์ pthread, mutexes, แอ็ตทริบิวต์ mutex, condition variables, แอ็ตทริบิวต์condition variable, read/write locks, แอ็ตทริบิวต์ read/write lock, คีย์จำเพาะ pthread และคีย์แอ็คทีฟ โดยใช้ฟังก์ชัน call back เพื่อจัดการหน่วยความจำสำหรับ รายการ

ฟังก์ชัน Hold และ unhold

ดีบักเกอร์ต้องสนับสนุน hold และ unhold ของเธรดด้วยเหตุผลดังต่อไปนี้:
- เพื่ออนุญาตให้ผู้ใช้ทำการดีบักแบบขั้นตอนเดียวในเธรดเดี่ยว ซึ่งต้องเป็นไปได้ที่จะ เก็บค่าเธรดอื่นหนึ่งเธรดหรือมากกว่านั้น
- สำหรับผู้ใช้ที่ดำเนินการต่อผ่านชุดย่อยของเธรดที่พร้อมใช้งาน ต้องเป็นไปได้ ที่จะเก็บค่าเธรดที่ไม่ได้อยู่ในชุด

รายการของฟังก์ชันดังต่อไปนี้ทำงาน hold และ unhold:
- ฟังก์ชัน **pthdb_pthread_hold** ตั้งค่า *hold state* ของ pthread เป็น `hold`
- ฟังก์ชัน **pthdb_pthread_unhold** ตั้งค่า *hold state* ของ pthread เป็น `unhold`

  หมายเหตุ: ฟังก์ชัน **pthdb_pthread_hold** และ **pthdb_pthread_unhold** ต้องถูกใช้เสมอ ไม่ว่า pthread จะมีเคอร์เนลเธรดหรือไม่
- ฟังก์ชัน **pthdb_pthread_holdstate** ส่งกลับ *hold state* ของ pthread.

- ฟังก์ชัน **pthdb_session_committed** รายงานชื่อฟังก์ชันของฟังก์ชันที่ถูกเรียก หลังจากการเปลี่ยนแปลงของ hold และ unhold ทั้งหมดถูกส่งข้อมูล จุดหยุดสามารถ ถูกวางไว้ที่ฟังก์ชันนี้เพื่อแจ้งดีบักเกอร์เมื่อการเปลี่ยนแปลงของ hold และ unhold ได้ถูกส่งข้อมูล
- ฟังก์ชัน **pthdb_session_stop_tid** แจ้งแก่ **ไลบรารี pthread debug** ซึ่งแจ้ง **ไลบรารี pthread** ถึงข้อมูล thread ID (TID) ของเธรดซึ่ง หยุดการทำงานของดีบักเกอร์
- ฟังก์ชัน **pthdb_session_commit_tid** ส่งกลับรายการของเธรดเคอร์เนล ครั้งละหนึ่งเธรดเคอร์เนล ซึ่งต้องดำเนินต่อเพื่อส่งข้อมูลการเปลี่ยนแปลง hold และ unhold ฟังก์ชันนี้ต้องถูกเรียกซ้ำจนกว่า PTHDB_INVALID_TID ถูกรายงาน ถ้ารายการเธรดเคอร์เนลว่างเปล่า ไม่มีความจำเป็น ต้องดำเนินเธรดใดๆ ต่อสำหรับการส่งข้อมูล

ดีบักเกอร์สามารถระบุเวลาที่การเปลี่ยนแปลง hold และ unhold ทั้งหมด ถูกส่งข้อมูลในวิธีดังต่อไปนี้:

- ก่อนการดำเนินการส่งข้อมูล (ดำเนินการกับ tids ที่หมดที่ส่งกลับโดยฟังก์ชัน **pthdb_session_commit_tid**) ถูกเริ่มต้นขึ้น ดีบักเกอร์ สามารถเรียกฟังก์ชัน **pthdb_session_committed** เพื่อรับ ชื่อฟังก์ชันและกำหนดจุดหยุด (เมธอดนี้สามารถกระทำได้หนึ่งครั้ง ในช่วงอายุของกระบวนการ)
- ก่อนการดำเนินการส่งข้อมูลถูกเริ่มต้น ดีบักเกอร์จะเรียกฟังก์ชัน **pthdb_session_stop_tid** ที่มี TID ของเธรดที่หยุดการทำงาน ของดีบักเกอร์ เมื่อการดำเนินการส่งข้อมูลสมบูรณ์ ไลบรารี pthread ยืนยัน ว่า TID เดียวกันถูกหยุดก่อนการดำเนินการส่งข้อมูล

hold หรือ unhold pthreads ใช้โพรซีเดอร์ดังต่อไปนี้ ก่อน ดำเนินการต่อกับกลุ่มของ pthreads หรือทำการ single-stepping เธรดเดี่ยว:

1. ใช้ฟังก์ชัน **pthdb_pthread_hold** และ **pthdb_pthread_unhold** เพื่อตั้งค่า pthreads ใดที่จะถูกคงค่า และ pthreads ใดจะถูกปลดค่า
2. เลือกวิธีซึ่งจะระบุเวลาที่การเปลี่ยนแปลงของ hold และ unhold ทั้งหมดจะถูกส่งข้อมูล
3. ใช้ฟังก์ชัน **pthdb_session_commit_tid** เพื่อกำหนด รายการของ TIDs ซึ่งต้องถูกดำเนินต่อเพื่อส่งข้อมูลการเปลี่ยนแปลง hold และ unhold
4. ดำเนิน TIDs ต่อในขั้นตอนก่อนหน้า รวมทั้งเธรดที่หยุดการทำงานของ ดีบักเกอร์

ฟังก์ชัน **pthdb_session_continue_tid** อนุญาตให้ดีบักเกอร์รับรายการของเธรดเคอร์เนล ซึ่งต้องถูกดำเนินการต่อก่อนที่ฟังก์ชันจำเริ่มการทำงานกับ single-stepping pthread เกี่ยวหรือดำเนินการกับกลุ่มของ pthreads ต่อ ฟังก์ชันนี้ต้องถูกเรียกซ้ำจนกว่า PTHDB_INVALID_TID ถูกรายงาน ถ้ารายการ ของเธรดเคอร์เนลไม่ว่าง ดีบักเกอร์ต้องดำเนินเธรดเคอร์เนลเหล่านี้ต่อตามด้วยเธรดอื่นที่เกี่ยวข้องชัดเจน ดีบักเกอร์รับผิดชอบ ในการพักเธรดที่หยุดทำงานและดำเนินเธรดที่หยุดทำงานต่อ เธรดที่หยุดการทำงานคือเธรดที่ทำให้เข้าสู่ดีบักเกอร์

ฟังก์ชัน Context

ฟังก์ชัน **pthdb_pthread_context** รับข้อมูล context และฟังก์ชัน **pthdb_pthread_setcontext** ตั้งค่า context ฟังก์ชัน **pthdb_pthread_context** รับข้อมูล context ของ pthread จากเคอร์เนลหรือโครงสร้างข้อมูล pthread ในพื้นที่แอดเดรส ของกระบวนการดีบัก ถ้า pthread ไม่ถูกเชื่อมโยงกับเธรดเคอร์เนล ข้อมูล context จะถูกบันทึกโดยไลบรารีที่ได้รับ ถ้า pthread เชื่อมโยงกับเธรดเคอร์เนล ข้อมูลได้รับจากดีบักเกอร์โดยใช้ call backs เป็นหน้าที่ของดีบักเกอร์ในการระบุ ว่าเธรดเคอร์เนลอยู่ในโหมด kernel หรือ user และจากนั้นจัดเตรียม ข้อมูลที่ถูกต้องสำหรับโหมดนั้น

เมื่อ pthread ที่มีเธรดโหมดอยู่ในโหมด kernel, คุณไม่สามารถกรับบริบท โหมด full user เนื่องจากเคอร์เนลไม่ได้บันทึกบริทบที่จุดเดียว ฟังก์ชัน **getthrds** สามารถถูกใช้เพื่อรับส่วนของข้อมูลนี้ เนื่องจากฟังก์ชันจะบันทึกสแต็กโหมด user เสมอ ดีบักเกอร์

สามารถพบข้อมูลนี้ได้โดยตรวจสอบโครงสร้าง **thrdsinfo64.ti_scount** ถ้าค่าไม่เป็นศูนย์ สแต็กโหมด user พร้อมใช้งานในโครงสร้าง **thrdsinfo64.ti_ustk** จากสแต็กโหมด user เป็นไปได้ที่จะ ระบุ instruction address register (IAR) และเฟรม call back แต่ทำไม่ได้กับค่ารีจิสเตอร์อื่น โครงสร้าง **thrdsinfo64** ถูกกำหนดในไฟล์ **procinfo.h**

ฟังก์ชันลิสต์

ไลบรารีดีบัก pthread รักษารายการสำหรับ pthreads แอ็ตทริบิวต์ pthread, mutexes แอ็ตทริบิวต์ mutex, condition variables แอ็ตทริบิวต์ condition variables, read/write locks, แอ็ตทริบิวต์ read/write lock คีย์จำเพาะ pthread และคีย์แอ็คทีฟ แต่ละค่าแสดงโดยแฮนเดิลจำเพาะชนิด ฟังก์ชัน **pthdb_*object*** ส่งกลับ แฮนเดิลถัดไปในรายการที่เหมาะสม โดยที่ *object* คือหนึ่งในข้อมูลต่อไปนี้: **pthread, attr, mutex, mutexattr, cond, condattr, rwlock, rwlockattr** หรือ **key** ถ้ารายการว่างเปล่าหรือถึงจุดสิ้นสุดของรายการ **PTHDB_INVALID_*object*** จะถูกรายงาน โดยที่ *object* คือหนึ่งในข้อมูลต่อไปนี้: **PTHREAD, ATTR, MUTEX, MUTEXATTR, COND, CONDATTR, RWLOCK, RWLOCKATTR** หรือ **KEY**

ฟังก์ชัน Field

ข้อมูลละเอียดเกี่ยวกับอ็อบเจ็กต์สามารถรับได้โดยการใช้ ฟังก์ชันสมาชิกอ็อบเจ็กต์ **pthdb_*object_field***, โดยที่ ***object*** คือหนึ่งในข้อมูลต่อไปนี้: **pthread, attr, mutex, mutexattr, cond, condattr, rwlock, rwlockattr** หรือ **key** โดยที่ ***field*** คือชื่อ ของฟิลด์ของข้อมูลโดยละเอียดสำหรับอ็อบเจ็กต์

การกำหนดเซสชันเอง

ฟังก์ชัน **pthdb_session_setflags** อนุญาตให้ดีบักเกอร์เปลี่ยนแปลงแฟล็กที่กำหนดค่าเซสชันเอง แฟล็กเหล่านี้ ควบคุมจำนวนของรีจิสเตอร์ที่ถูกอ่านหรือเขียนระหว่างการดำเนินการ context และในการควบคุมการพิมพ์ของข้อมูลดีบัก

ฟังก์ชัน **pthdb_session_flags** ได้รับแฟล็กปัจจุบันสำหรับเซสชัน

การยกเลิกเซสชัน

ที่จุดสิ้นสุดของเซสชันดีบัก โครงสร้างข้อมูลเซสชันต้องถูกคืนค่าที่จัดสรรมา และข้อมูลเซสชันต้องถูกลบ ซึ่งทำได้โดยการเรียกฟังก์ชัน **pthdb_session_destroy** ซึ่งใช้ฟังก์ชัน call back เพื่อคืนหน่วยความจำ หน่วยความจำทั้งหมด ที่จัดสรรโดยฟังก์ชัน **pthdb_session_init** และ **pthdb_session_update** จะถูกคืนข้อมูลที่จัดสรร

ตัวอย่างของฟังก์ชัน hold/unhold

ตัวอย่างโค้ดจำลองดังต่อไปนี้แสดงวิธีที่ดีบักเกอร์ใช้โค้ด hold/unhold:

```
/* includes */

#include <sys/pthdebug.h>

main()
{
    tid_t stop_tid; /* thread which stopped the process */
    pthdb_user_t user = <unique debugger value>;
    pthdb_session_t session; /* <unique library value> */
    pthdb_callbacks_t callbacks = <callback functions>;
    char *pthreaded_symbol=NULL;
    char *committed_symbol;
    int pthreaded = 0;
```

```
   int pthdb_init = 0;
   char *committed_symbol;

   /* fork/exec or attach to the program that is being debugged */

   /* the program that is being debugged uses ptrace()/ptracex() with PT_TRACE_ME */

   while (/* waiting on an event */)
   {
     /* debugger waits on the program that is being debugged */

     if (pthreaded_symbol==NULL) {
       rc = pthdb_session_pthreaded(user, &callbacks, pthreaded_symbol);
       if (rc == PTHDB_NOT_PTHREADED)
       {
           /* set breakpoint at pthreaded_symbol */
       }
       else
         pthreaded=1;
     }
     if (pthreaded == 1 && pthdb_init == 0) {
         rc = pthdb_session_init(user, &session, PEM_32BIT, flags, &callbacks);
         if (rc)
             /* handle error and exit */
         pthdb_init=1;
     }

     rc = pthdb_session_update(session)
     if ( rc != PTHDB_SUCCESS)
/* handle error and exit */

     while (/* accepting debugger commands */)
     {
         switch (/* debugger command */)
         {
             ...
             case DB_HOLD:
                 /* regardless of pthread with or without kernel thread */
                 rc = pthdb_pthread_hold(session, pthread);
                 if (rc)
                     /* handle error and exit */
             case DB_UNHOLD:
                 /* regardless of pthread with or without kernel thread */
                 rc = pthdb_pthread_unhold(session, pthread);
                 if (rc)
                     /* handle error and exit */
             case DB_CONTINUE:
                 /* unless we have never held threads for the life */
                 /* of the process */
                 if (pthreaded)
                 {
                     /* debugger must handle list of any size */
                     struct pthread commit_tids;
                     int commit_count = 0;
                     /* debugger must handle list of any size */
                     struct pthread continue_tids;
                     int continue_count = 0;

               rc = pthdb_session_committed(session, committed_symbol);
```

```
        if (rc != PTHDB_SUCCESS)
          /* handle error */
                              /* set break point  at committed_symbol */

                        /* gather any tids necessary to commit hold/unhold */
                        /* operations */
                        do
                        {
                            rc = pthdb_session_commit_tid(session,
                                                  &commit_tids.th[commit_count++]);
                            if (rc != PTHDB_SUCCESS)
                                /* handle error and exit */
                        } while (commit_tids.th[commit_count - 1] != PTHDB_INVALID_TID);

                        /* set up thread which stopped the process to be */
                        /* parked using the stop_park function*/

                    if (commit_count > 0) {
                            rc = ptrace(PTT_CONTINUE, stop_tid, stop_park, 0,
                                                                 &commit_tids);
                            if (rc)
                                /* handle error and exit */

                            /* wait on process to stop */
        }

                        /* gather any tids necessary to continue */
                        /* interesting threads */
                        do
                        {
                            rc = pthdb_session_continue_tid(session,
                                          &continue_tids.th[continue_count++]);
                             if (rc != PTHDB_SUCCESS)
                                /* handle error and exit */
                        } while (continue_tids.th[continue_count - 1] != PTHDB_INVALID_TID);

                        /* add interesting threads to continue_tids */

                        /* set up thread which stopped the process to be parked */
                        /* unless it is an interesting thread */

                        rc = ptrace(PTT_CONTINUE, stop_tid, stop_park, 0,
                                                               &continue_tids);
                        if (rc)
                            /* handle error and exit */
                    }
                case DB_EXIT:
      rc = pthdb_session_destroy(session);
      /* other clean up code */
    exit(0);
                ...
            }
        }

      }
      exit(0);
}
```

ประโยชน์ของเธรด

โปรแกรมแบบมัลติเธรดสามารถช่วยเพิ่มประสิทธิภาพการทำงานเมื่อเทียบ กับโปรแกรแบบขนาดแบบเดิมที่ใช้หลาย กระบวนการ นอกจากนี้ ประสิทธิภาพยิ่งเพิ่มขึ้นเมื่อใช้ระบบมัลติโพรเซสเซอร์ที่ใช้เธรด

การจัดการเธรด

การจัดการเธรด คือการสร้างเธรด และการควบคุม การดำเนินการ ต้องการทรัพยากรระบบน้อยกว่าการจัดการกระบวนการ ตัวอย่างเช่น การสร้างเธรด ต้องการเพียงการจัดสรรพื้นที่ข้อมูลส่วนตัว ของเธรด ปกติคือ 64 KB และสองการเรียกการทำงาน ของระบบ การสร้างกระบวนการ ใช้ทรัพยากรมากกว่ามาก เนื่องจากพื้นที่การกำหนดแอดเดรสของกระบวนการ parent ทั้งหมดถูกทำซ้ำ

และเธรดไลบรารี API ยังใช้งานง่ายกว่า ไลบรารีสำหรับการจัดการกระบวนการ การสร้างเธรดต้องการเพียงรูทีนย่อย **pthread_create**

การสื่อสารระหว่างเธรด

การสื่อสารระหว่างเธรดมีประสิทธิภาพมากกว่าและ ง่ายในการใช้มากกว่าการสื่อสารระหว่างกระบวนการ เนื่องจากเธรดทั้งหมดอยู่ภายใน กระบวนการใช้พื้นที่แอดเดรสเดียวกัน เธรดไม่จำเป็นต้องใช้หน่วยความจำที่แบ่งใช้ ป้องกัน ข้อมูลที่แบ่งใช้จากการเข้าถึงพร้อมกันโดยการใช้ mutexes หรือเครื่องมือ การซิงโครไนซ์อื่น

ความสามารถในการซิงโครไนซ์จัดเตรียมโดยเธรด ไลบรารีเป็นเครื่องมือการซิงโครไนซ์ที่ยืดหยุ่นและความสามารถสูงที่ง่าย ต่อการนำไปใช้ เครื่องมือเหล่านี้สามารถแทนที่ความสามารถในการสื่อสารระหว่างกระบวนการแบบเดิม เช่นคิวข้อความ ไพพ์ สามารถถูกใช้เป็นพาธการสื่อสาร ระหว่างเธรด

ระบบมัลติโพรเซสเซอร์

บนระบบมัลติโพรเซสเซอร์ เธรดหลายเธรดสามารถรัน พร้อมกันบน CPU หลายตัว ดังนั้น โปรแกรมแบบมัลติเธรดสามารถรันได้เร็ว มากยิ้นบนระบบตัวประมวลผลเดียว และยังสามารถรันได้เร็วกว่าโปรแกรมที่ใช้ หลายกระบวนการ เนื่องจาก ต้องการทรัพยากรน้อยกว่าและมี ความสิ้นเปลืองน้อยกว่า ตัวอย่าง การสลับเธรดในกระบวนการเดียวกันจะเร็วกว่าได้โดยเฉพาะในโมเดลไลบรารี M:N ซึ่งการสลับบริบทบ่อยครั้งที่สามารถหลีกเลี่ยงได้ สุดท้าย ประโยชน์เด่น ของการใช้เธรดก็คือ โปรแกรมแบบมัลติเธรดโปรแกรมเดียวจะทำงาน บนระบบตัวประมวลผลเดียว แต่โดยธรรมชาติแล้วสามารถใช้ประโยชน์ของระบบ หลายตัวประมวลผล โดยไม่ต้องคอมไพล์ใหม่

ข้อจำกัด

โปรแกรมมิงแบบมัลติเธรดเป็นประโยชน์สำหรับการประยุกต์ใช้อัลกอริทึมแบบ ขนาดโดยใช้เอนทิตีอิสระหลายเอนทิตี อย่างไรก็ตาม มี บางกรณีที่การทำงานแบบหลายกระบวนการควรถูกใช้แทน การทำงานแบบหลายเธรด

ตัวระบุ ทรัพยากร ภาวะ หรือข้อจำกัด ของระบบปฏิบัติการ จำนวนมากถูกกำหนดที่ระดับกระบวนการ ดังนั้นจึงถูกแบ่งใช้โดยเธรด ทั้งหมดในกระบวนการ ตัวอย่างเช่น user และ group IDs และสิทธิที่เกี่ยวข้อง ถูกจัดการที่ระดับกระบวนการ โปรแกรมที่ต้องกำหนด ID ผู้ใช้ที่ต่างกันให้แก่โปรแกรมมิงเอนทิตีต้องใช้หลาย กระบวนการ แทนการใช้กระบวนการแบบมัลติเธรดเดียว ตัวอย่างอื่น รวมถึงแอ็ตทริบิวต์ระบบไฟล์ เช่น ไดเร็กทอรีทำงานปัจจุบัน และภาวะ และจำนวนสูงสุด ของไฟล์ที่เปิด โปรแกรมแบบมัลติเธรด อาจไม่เหมาะสมถ้าแอ็ตทริบิวต์เหล่านี้ถูกจัดการแบบอิสระได้ดีกว่า ตัวอย่างเช่น โปรแกรม multi-processed สามารถ ปล่อยให้แต่ละกระบวนการเปิดไฟล์จำนวนมากโดยไม่มีการรบกวนจาก กระบวนการอื่น

หลักการที่เกี่ยวข้อง:

"ไลบรารี Threadsafe และที่ถูกเธรดใน AIX" ในหน้า 462
ส่วนนี้อธิบายเธรดไลบรารีใน AIX

ข้อมูลโปรแกรม lex และ yacc

สำหรับโปรแกรมที่จะรับอินพุต ในสภาวะแวดล้อมแบบโต้ตอบ หรือแบบแบ็ตช์ คุณต้องจัดให้มีอีกโปรแกรมหรือรูทีนหนึ่ง เพื่อรับอินพุต อินพุต ที่ซับซ้อนต้องใช้โค้ดเพิ่มเติมเพื่อแบ่งย่อยอินพุตออกเป็นส่วนเล็กๆ ที่มีความหมายบางอย่าง ต่อโปรแกรม

คุณสามารถใช้คำสั่ง **lex** และ **yacc** เพื่อพัฒนาโปรแกรมอินพุตประเภทนี้

คำสั่ง **lex** สร้าง โปรแกรมตัววิเคราะห์คำที่จะวิเคราะห์อินพุตและแบ่งย่อยออกเป็นโทเค็น เช่น ตัวเลข ตัวอักษร หรือตัวดำเนินการ โทเค็นถูกกำหนดโดยกฎไวยากรณ์ ที่ตั้งค่าในไฟล์ข้อมูลจำเพาะ **lex** คำสั่ง **yacc** สร้างโปรแกรมวิเคราะห์คำที่วิเคราะห์อินพุตโดยใช้โทเค็นที่ระบุโดยระบุคำ (สร้างขึ้นโดยคำสั่ง **lex** และเก็บในไฟล์ข้อมูลจำเพาะ **lex**) และทำหน้าทีในการดำเนินการตามที่ระบุ เช่นการแฟล็กไวยากรณ์ที่ไม่เหมาะสม รวมทั้ง คำสั่งเหล่านี้จะสร้างตัววิเคราะห์คำและโปรแกรมวิเคราะห์คำเพื่อใช้ในการแปลความหมาย การจัดการอินพุตและเอาต์พุต

ข้อมูลที่เกี่ยวข้อง:

printf

ed

ex

sed

yacc

การสร้างตัววิเคราะห์คำด้วยคำสั่ง lex

คำสั่ง **lex** ช่วยในการเขียน โปรแกรมภาษา C ที่สามารถรับและแปลอินพุตสตรีมอักขระ เป็นการดำเนินการของโปรแกรม

เมื่อต้องการใช้คำสั่ง **lex** คุณต้อง ระบุหรือเขียนไฟล์ข้อกำหนดที่มี:

ส่วนขยายนิพจน์ทั่วไป
: รูปแบบอักขระที่ lexical analyzer ที่สร้างรู้จัก

คำสั่งดำเนินการ
: ส่วนของโปรแกรมภาษา C ที่กำหนดวิธีที่ lexical analyzer ที่สร้างทำงานกับส่วนขยายนิพจน์ทั่วไปที่รู้จัก

For information about the format and logic allowed in this file, see the **lex** command in *Commands Reference, Volume 3*.

คำสั่ง **lex** สร้างโปรแกรมภาษา C ที่สามารถวิเคราะห์สตรีมอินพุตโดยใช้ข้อมูลใน ไฟล์ข้อกำหนด จากนั้นคำสั่ง **lex** เก็บโปรแกรมเอาต์พุตไว้ในไฟล์ **lex.yy.c** ถ้าโปรแกรมเอาต์พุต ทำงานได้กับข้อมูลง่ายๆ โครงสร้างอินพุตที่มีหนึ่งคำ คุณสามารถคอมไพล์ไฟล์เอาต์พุต **lex.yy.c** ด้วยคำสั่งดังต่อไปนี้เพื่อสร้าง lexical analyzer ที่รันได้:

```
cc lex.yy.c -ll
```

อย่างไรก็ตาม ถ้า lexical analyzer ต้องรู้จัก ไวยากรณ์ซับซ้อนเพิ่มเติม คุณสามารถสร้างโปรแกรมวิเคราะห์คำเพื่อ ใช้กับไฟล์เอาต์พุตเพื่อประกันความถูกต้องในการจัดการอินพุต

คุณสามารถย้ายไฟล์เอาต์พุต **lex.yy.c** ไปที่ระบบอื่น ถ้าระบบนั้นมีคอมไพลเลอร์ C ที่สนับสนุนฟังก์ชันไลบรารี **lex**

lexical analyzer ที่คอมไพล์ดำเนินฟังก์ชัน ดังต่อไปนี้:

- อ่านสตรีมอินพุตของอักขระ
- คัดลอกสตรีมอินพุตไปที่สตรีมเอาต์พุต
- แยกสตรีมอินพุตออกเป็นสตริงที่เล็กลงที่ตรงกับส่วนขยาย นิพจน์ทั่วไปในไฟล์ข้อกำหนด **lex**
- รันการดำเนินการสำหรับแต่ละส่วนขยายนิพจน์ทั่วไปที่รู้จัก การดำเนินการเหล่านี้เป็นส่วนของโปรแกรมภาษา C ใน ไฟล์ข้อกำหนด **lex** แต่ละส่วนของการดำเนินการ สามารถเรียกการทำงานหรือรูทีนย่อยภายนอกได้

lexical analyzer ที่สร้างโดยคำสั่ง **lex** ใช้เมธอดการวิเคราะห์ที่เรียกว่า *deterministic finite-state automaton* เมธอดนี้จัดเตรียมไว้สำหรับจำนวนจำกัด ของเงื่อนไขซึ่ง lexical analyzer มีอยู่ได้ ตามด้วยกฎที่ กำหนดภาวะของ lexical analyzer

การทำงานอัตโนมัติอนุญาตให้ lexical analyzer ที่สร้างค้นหาเพิ่มเติมจากหนึ่งหรือสองอักขระในสตรีมอินพุต ตัวอย่างเช่น คุณได้กำหนดกฎสองกฎในไฟล์ข้อกำหนด **lex** : ข้อแรกค้นหาสตริง `ab` อีกข้อหนึ่ง ค้นหาสตริง `abcdefg` ถ้า lexical analyzer ได้รับสตริงอินพุต `abcdefh` ซึ่งอ่าน อักขระไปจนจบสตริงอินพุต ก่อนกำหนดว่าไม่ ตรงกับสตริง `abcdefg` จากนั้น lexical analyzer ส่งกลับกฎที่ค้นหาสตริง `ab` ตัดสินว่าตรงกับส่วนของอินพุต และเริ่มการค้นหา ข้อมูลอื่นที่ตรงกันโดยใช้อินพุต `cdefh` ที่เหลือ

การคอมไพล์ตัววิเคราะห์คำ

เมื่อคอมไพล์โปรแกรม **lex** ให้ทำดังต่อไปนี้:

1. ใช้โปรแกรม **lex** เพื่อเปลี่ยนไฟล์ข้อกำหนด ลงในโปรแกรมภาษา C โปรแกรมผลลัพธ์อยู่ในไฟล์ **lex.yy.c**
2. ใช้คำสั่ง **cc** กับแฟล็ก **–ll** เพื่อคอมไพล์และเชื่อมโยงโปรแกรมกับโปรแกรมของรูทีนย่อย **lex** โปรแกรมเรียกทำงานผลลัพธ์อยู่ในไฟล์ **a.out**

ตัวอย่างเช่น ถ้าไฟล์ข้อกำหนด **lex** ชื่อว่า **lextest** ให้ป้อนคำสั่งดังต่อไปนี้:

```
lex lextest
cc lex.yy.c -ll
```

หลักการที่เกี่ยวข้อง:

“เครื่องมือและยูทิลีตี” ในหน้า 2
ส่วนนี้ให้ภาพรวมของเครื่องมือและยูทิลิตีที่คุณสามารถใช้เพื่อพัฒนาโปรแกรมภาษาที่คอมไพล์ด้วย C
“การสร้างโปรแกรมวิเคราะห์คำด้วยโปรแกรม yacc” ในหน้า 573
โปรแกรม **yacc** สร้างโปรแกรมวิเคราะห์คำ ที่กำหนดโครงสร้างการบังคับใช้สำหรับอินพุตอักขระไปยังโปรแกรมคอมพิวเตอร์

การใช้โปรแกรม lex กับโปรแกรม yacc

คุณยังสามารถใช้โปรแกรม **lex** ด้วยตัวสร้าง parser เช่น คำสั่ง **yacc** คำสั่ง **yacc** จะสร้างโปรแกรมที่เรียกว่า *parser* ซึ่งวิเคราะห์โครงสร้างของอินพุตแบบหนึ่งคำ

โปรแกรม parser นี้จะดำเนินการได้ดีด้วยตัววิเคราะห์ lexical ที่คำสั่ง **lex** สร้าง parser จะจดจำชนิดของไวยากรณ์ต่างๆ ที่ไม่คำนึงถึงบริบท parser เหล่านี้ต้องการตัวประมวลผลก่อนเพื่อจดจำโทเค็นอินพุต เช่น ตัวประมวลผลก่อนที่คำสั่ง **lex** สร้างขึ้น

โปรแกรม **lex** จะจดจำเฉพาะนิพจน์ปกติที่ขยายแล้ว และจัดรูปแบบนิพจน์ลงในแพ็กเกจอักขระที่เรียกว่า *tokens* ตามที่ระบุโดยไฟล์อินพุต ขณะที่ใช้โปรแกรม **lex** เพื่อสร้างตัววิเคราะห์สำหรับ parser ตัววิเคราะห์ lexical (สร้างจากคำสั่ง **lex**) จะแบ่งพาร์ติชันอินพุตสตรีม parser (จากคำสั่ง **yacc**) จะกำหนดโครงสร้างให้กับส่วนของผลลัพธ์ คุณยังสามารถใช้โปรแกรมอื่นๆ พร้อมกับโปรแกรมที่สร้างด้วยคำสั่ง **lex** หรือ **yacc** อย่างใดอย่างหนึ่ง

โทเค็นคือยูนิตขนาดเล็กที่สุดที่เป็นอิสระในความหมายที่ได้นิยามไว้โดย parser หรือตัววิเคราะห์ lexical อย่างใดอย่างหนึ่ง โทเค็นสามารถมีข้อมูล คีย์เวิร์ดภาษา ตัวระบุ หรือส่วนอื่นๆ ของไวยากรณ์ภาษา

โปรแกรม **yacc** จะมองหารูทีนย่อยตัววิเคราะห์ lexical ที่ชื่อ **yylex** ซึ่งถูกสร้างด้วยคำสั่ง **lex** โดยปกติแล้ว โปรแกรมหลักที่เป็นค่าดีฟอลต์ในไลบรารี **lex** จะเรียกรูทีนย่อย **yylex** อย่างไรก็ตาม ถ้าคำสั่ง **yacc** ถูกติดตั้งไว้และโปรแกรมหลักถูกใช้โปรแกรม **yacc** จะเรียกรูทีนย่อย **yylex** ในกรณีนี้ ค่า `token` ที่เหมาะสมที่ถูกส่งคืน กฎของโปรแกรม **lex** แต่ละตัวควรสิ้นสุดด้วยคำสั่งต่อไปนี้:

```
return(token);
```

คำสั่ง **yacc** จะกำหนดค่าจำนวนเต็มให้กับโทเค็นแต่ละตัว ที่ได้นิยามไว้ในไฟล์ไวยากรณ์ **yacc** ผ่านข้อความตัวประมวลผลก่อน **#define** ตัววิเคราะห์ lexical ต้องมีสิทธิเข้าถึงแมโครเหล่านี้เพื่อส่งคืนโทเค็นให้กับ parser ใช้อ็อพชัน **yacc -d** เพื่อสร้างไฟล์ **y.tab.h** และสอดแทรกไฟล์ **y.tab.h** ลงในไฟล์ข้อกำหนดคุณลักษณะ **lex** โดยเพิ่มบรรทัดต่อไปนี้ในส่วนของการนิยามของไฟล์ข้อกำหนดคุณลักษณะ **lex** :

```
%{
#include "y.tab.h"
%}
```

หรือ คุณสามารถสอดแทรกไฟล์ **lex.yy.c** ลงในเอาต์พุต **yacc** โดยการเพิ่มบรรทัดต่อไปนี้หลังตัวคั่น `%%` ตัวที่สอง (เครื่องหมายเปอร์เซ็นต์ เครื่องหมายเปอร์เซ็นต์) ในไฟล์ไวยากรณ์ **yacc** :

```
#include "lex.yy.c"
```

ไลบรารี **yacc** ควรถูกโหลด ก่อนไลบรารี **lex** เพื่อขอรับโปรแกรมหลักที่เรียกใช้งาน **yacc** คุณสามารถสร้างโปรแกรม **lex** และ **yacc** ตามลำดับอย่างใดอย่างหนึ่ง

นิพจน์ทั่วไปส่วนขยายในคำสั่ง lex

การระบุนิพจน์ทั่วไปที่ขยายเพิ่มในไฟล์ข้อมูลจำเพาะ **lex** คล้ายกับวิธีที่ใช้ในคำสั่ง **sed** หรือ **ed**

นิพจน์ทั่วไปที่ขยายเพิ่ม ระบุชุดของสตริงที่จะจับคู่ นิพจน์มีทั้งอักขระข้อความ และอักขระตัวดำเนินการ อักขระข้อความจับคู่กับอักขระที่สอดคล้องกัน ในสตริงที่กำลังเปรียบเทียบ อักขระตัวดำเนินการระบุการซ้ำ ตัวเลือก และคุณลักษณะอื่นๆ

ตัวเลขและตัวอักษรถูกพิจารณา ว่าเป็นอักขระข้อความ ตัวอย่าง นิพจน์ทั่วไปที่ขยายเพิ่ม `integer` จับคู่สตริง `integer` และนิพจน์ `a57D` ค้นหาสตริง `a57D`

โอเปอเรเตอร์

รายการต่อไปนี้อธิบายวิธีใช้ตัวดำเนินการ เพื่อระบุนิพจน์ทั่วไปที่ขยายเพิ่ม:

อักขระ จับคู่อักขระ *Character*

ตัวอย่าง: a จับคู่อักขระที่เป็นตัวอักษร a โดย b จับคู่อักขระที่เป็นตัวอักษร b และ c จับคู่อักขระที่เป็นตัวอักษร c

"String"

จับคู่สตริงที่อยู่ภายในเครื่องหมายคำพูด แม้ว่าสตริง นั้นจะประกอบด้วยตัวดำเนินการ

ตัวอย่าง: เพื่อป้องกันมิให้คำสั่ง **lex** แปลความหมาย $ (เครื่องหมายดอลล่าร์) ว่าเป็นตัวดำเนินการ ให้ใส่สัญลักษณ์นั้นภายในเครื่องหมายคำพูด

อักขระ **หรือ** *ตัวเลข*

Escape character เมื่อนำหน้าตัวดำเนินการคลาสอักขระที่ใช้ใน สตริง อักขระ \ แสดงว่าสัญลักษณ์ ตัวดำเนินการนั้นแทนตัวอักขระนั้นๆ มากกว่าเป็นตัวดำเนินการ ลำดับ escape ที่ถูกต้องได้แก่:

\a แจ้งเตือน

\b ถอยกลับ (Backspace)

\f ป้อนกระดาษ

\n อักขระขึ้นบรรทัดใหม่ (อย่าใช้อักขระขึ้นบรรทัดใหม่จริงในนิพจน์)

\r Return

\t แท็บ (Tab)

\v แท็บแนวดิ่ง

\\ Backslash

Digits อักขระที่มีการเข้ารหัสถูกแทนด้วยจำนวนเต็มฐานแปดหนึ่งหลัก, สองหลัก หรือสามหลักที่ระบุโดยสตริง *Digits*

\x*Digits*

อักขระที่มีการเข้ารหัสถูกแทนด้วยลำดับของอักขระฐานสิบหก ที่ระบุโดยสตริง *Digits*

โดยที่อักขระ \ อยู่หน้าอักขระที่ ไม่อยู่ในรายการนำหน้าของ escape sequences คำสั่ง **lex** จะแปลความหมายอักขระตามตัวอักษร

ตัวอย่าง: \c ถูกแปลเป็น อักขระ c ไม่เปลี่ยนแปลง และ [\^abc] แทนคลาสของอักขระที่รวมอักขระ ^abc

หมายเหตุ: อย่าใช้ \0 หรือ \x0 ใน คำสั่ง **lex**

[*List*] จับคู่หนึ่งอักขระใดๆ ในช่วงที่อยู่ในเครื่องหมาย ([*x*-*y*]) หรือรายการที่อยู่ในเครื่องหมาย ([*xyz*]) ยึดตาม locale ที่คำสั่ง **lex** ถูกเรียกใช้ สัญลักษณ์ตัวดำเนินการทั้งหมด โดยมีข้อยกเว้น ต่อไปนี้ สูญเสียความหมายพิเศษภายในนิพจน์วงเล็บเหลี่ยม: - (เส้นประ), ^ (caret) และ \ (backslash)

ตัวอย่าง: [abc-f] จับคู่ a, b, c, d, e หรือ f ใน en_US locale

[:*Class*:]

จับคู่อักขระใดๆ ที่อยู่ในคลาสอักขระที่ระบุ ระหว่างตัวคั่น [::] ดังกำหนดใน หมวดหมู่ LC_TYPE ใน locale ปัจจุบัน ชื่อคลาสอักขระต่อไปนี้ ได้รับการสนับสนุนให้ใช้ได้ในทุก locales:

```
alnum   cntrl  lower   space

alpha   digit   print  upper

blank  graph   punct   xdigit
```

คำสั่ง **lex** ยังรู้จักชื่อคลาสที่ ผู้ใช้กำหนดเอง ตัวดำเนินการ `[::]` ใช้ได้เฉพาะใน นิพจน์ `[]` เท่านั้น

ตัวอย่าง: `[[:alpha:]]` จับคู่อักขระใดๆ ในคลาสอักขระ **alpha** ใน locale ปัจจุบัน แต่ `[:alpha:]` จับคู่เฉพาะอักขระ `:,a,l,p` และ `h`

`[.`*CollatingSymbol*`.]`

จับคู่สัญลักษณ์การเรียงที่ระบุภายในตัวคั่น `[..]` เป็นอักขระเดี่ยว ตัวดำเนินการ `[..]` ใช้ได้เฉพาะในนิพจน์ `[ ]` สัญลักษณ์การเรียงต้องเป็นสัญลักษณ์การเรียงที่ถูกต้อง สำหรับ locale ปัจจุบัน

ตัวอย่าง: `[[.ch.]]` จับคู่ `c` และ `h` ด้วยกัน ขณะที่ `[ch]` จับคู่ `c` หรือ `h`

`[=`*CollatingElement*`=]`

จับคู่อิลิเมนต์การเรียงที่ระบุภายในตัวคั่น `[==]` และอิลิเมนต์การเรียงทั้งหมดที่เป็นของคลาส ที่เทียบเท่ากัน ตัวดำเนินการ `[==]` ใช้ได้เฉพาะในนิพจน์ `[]` เท่านั้น

ตัวอย่าง: ถ้า `w` และ `v` เป็นของ คลาสที่เทียบเท่าเหมือนกัน `[[=w=]]` จะเหมือน กับ `[wv]` และจับคู่ `w` หรือ `v` ถ้า `w` ไม่ได้เป็นของคลาสที่เทียบเท่า ดังนั้น `[[=w=]]` จับคู่เฉพาะ `w` เท่านั้น

`[^`*Character*`]`

จับคู่อักขระใดๆ ยกเว้นอักขระที่ตามหลังสัญลักษณ์ `^` (caret) คลาสอักขระผลลัพธ์มีเพียงอักขระไบต์เดียว อย่างเดียวเท่านั้น อักขระที่ตามหลังสัญลักษณ์ `^` สามารถเป็นอักขระหลายไบต์ อย่างไรก็ตาม สำรหับตัวดำเนินการนี้ที่จะจับคู่อักขระ หลายไบต์ คุณต้องตั้งค่า **%h** และ **%m** ให้มีค่ามากกว่าศูนย์ในส่วนนิยาม

ตัวอย่าง: `[^c]` จับคู่อักขระใดๆ ยกเว้น `c`

CollatingElement`-`*CollatingElement*

ในคลาสอักขระ จะระบุช่วงของอักขระภายในลำดับการเรียง ที่กำหนดสำหรับ locale ปัจจุบัน ช่วงต้องเรียงลำดับจากน้อยไปหามาก จุดสิ้นสุดช่วงต้องเรียงลำดับค่าเท่ากับหรือมากกว่าจุดเริ่มต้น ช่วง เนื่องจากช่วงกำหนดตามลำดับการเรียงของ locale ปัจจุบัน ช่วงที่กำหนดอาจจับคู่อักขระที่ต่างต่าง ขึ้นอยู่กับ locale ที่คำสั่ง **lex** ถูกเรียกใช้

Expression`?`

จับคู่จำนวนการเกิดนิพจน์ที่มีค่าศูนย์หรือมากกว่าศูนย์ที่นำหน้า ติดกับตัวดำเนินการ ?

ตัวอย่าง: `ab?c` จับคู่ ac หรือ abc อย่างใดอย่างหนึ่ง

อักขระจุด (.)

จับคู่อักขระใดๆ ยกเว้นอักขระขึ้นบรรทัดใหม่ สำหรับอักขระ จุด (.) เพื่อให้ตรงกับอักขระแบบหลายไบต์ **%z** ต้องถูกตั้งค่าให้มากกว่า 0 ในส่วนของนิยามของไฟล์ข้อมูลจำเพาะ **lex** ถ้าไม่ได้ตั้งค่า **%z** ไว้ อักขระจุด (.) จะจับคู่กับอักขระไบต์เดี่ยว เท่านั้น

Expression`*`

จับคู่จำนวนการเกิดนิพจน์ที่มีค่าศูนย์ หรือมากกว่าศูนย์ที่นำหน้า ติดกับตัวดำเนินการ * ตัวอย่าง `a*` คือจำนวนอักขระ `a` ที่ต่อเนื่องตามลำดับ โดยรวมศูนย์ ข้อดีของการจับคู่การการเกิดศูนย์เห็นได้อย่างชัดเจน ในนิพจน์ที่มีความซับซ้อน

ตัวอย่าง: นิพจน์ `[A-Za-z][A-Za-z0-9]*` บ่งชี้สตริงตัวอักษรผสมตัวเลขทั้งหมด ที่มีอักขระตัวอักษรผสมตัวเลขนำหน้า รวมถึงสตริงที่มีอักขระแบบตัวอักษร หนึ่งอักขระเท่านั้น คุณสามารถใช้นิพจน์นี้เพื่อให้สามารถรู้จัก identifiers ใน ภาษาของคอมพิวเตอร์

Expression+

จับคู่การเกิดของรูปแบบหนึ่งหรือมากกว่าหนึ่งที่นำหน้า ติดกับตัวดำเนินการ +

ตัวอย่าง: `a+` จับคู่การมีอย่างน้อยหนึ่งตัวของ `a` รวมทั้ง `[a-z]+` จับคู่ สตริงทั้งหมดที่มีตัวอักษรเป็นตัวพิมพ์เล็ก

Expression | *Expression*

บ่งชี้การจับคู่สำหรับนิพจน์ที่นำหน้า หรือตามหลังตัวดำเนินการ `|` (ไพพ์)

ตัวอย่าง: `ab|cd` จับคู่ `ab` หรือ `cd`

(*Expression*)

จับคู่นิพจน์ในเครื่องหมายวงเล็บ ตัวดำเนินการ `()` (วงเล็บ) ถูกใช้เพื่อจัดกลุ่มและทำให้นิพจน์ที่อยู่ ภายในวงเล็บถูกอ่านเข้าไปในอาร์เรย์ **yytext** กลุ่มในวงเล็บสามารถถูกใช้แทนอักขระเดียวใน รูปแบบอื่นๆ

ตัวอย่าง: `(ab|cd+)?(ef)*` จับคู่สตริงเช่น `abefef`, `efefef`, `cdef` หรือ `cddd` แต่ไม่จับคู่ `abc`, `abcd` หรือ `abcdef`

^*Expression*

บ่งชี้การจับคู่ต่อเมื่อ *นิพจน์* อยู่ที่ ตำแหน่งเริ่มต้นของบรรทัดและตัวดำเนินการ ^ (caret) เป็นอักขระแรกในนิพจน์เท่านั้น

ตัวอย่าง: `^h` จับคู่ `h` ที่ตำแหน่งเริ่มต้นของบรรทัด

Expression$

บ่งชี้การจับคู่ต่อเมื่อ *นิพจน์* อยู่ที่ ตำแหน่งสิ้นสุดของบรรทัดและตัวดำเนินการ $ (ดอลล่าร์) เป็นอักขระสุดท้ายในนิพจน์เท่านั้น

ตัวอย่าง: `h$` จับคู่ `h` ที่ตำแหน่งสิ้นสุดบรรทัด

Expression1/*Expression2*

บ่งชี้การจับคู่เฉพาะเมื่อ *Expression2* ต่อท้าย *Expression1* เท่านั้น ตัวดำเนินการ / (slash) อ่านเฉพาะนิพจน์แรกเข้ามาในอาร์เรย์ **yytext**

ตัวอย่าง: `ab/cd` จับคู่สตริง `ab` แต่เฉพาะเมื่อตามด้วย `cd` เท่านั้น และจากนั้นอ่าน `ab` ไว้ในอาร์เรย์ **yytext**

หมายเหตุ: เพียงหนึ่งตัวดำเนินการบริบทต่อท้าย / เท่านั้นที่ สามารถนำไปใช้ในนิพจน์ทั่วไปที่ขยายเพิ่มนิพจน์เดียว ตัวดำเนินการ ^ (caret) และ $ (เครื่องหมายดอลล่าร์) ไม่สามารถ ใช้ในนิพจน์เดียวกันที่มีตัวดำเนินการ / เนื่องจากตัวดำเนินการทั้งสองระบุกรณีพิเศษของบริบทต่อท้าย

{*DefinedName*}

จับคู่ชื่อที่คุณกำหนดในส่วนนิยาม

ตัวอย่าง: ถ้าคุณได้กำหนด `D` เป็น หลักที่เป็นตัวเลข `{D}` จะจับคู่หลัก ตัวเลขทั้งหมด

{*Number1* , *Number2*}

จับคู่การเกิดเหตุการณ์ *Number1* ถึง *Number2* ของรูปแบบที่นำหน้า นิพจน์ {*Number*} และ {*Number*} ยังได้รับอนุญาตและจับคู่การเกิดเหตุการณ์ *Number* จำนวนแน่นอนของรูปแบบที่นำหน้านิพจน์

ตัวอย่าง: xyz{2,4} จับคู่ xyzxyz, xyzxyzxyz หรือ xyzxyzxyzxyz นี้ต่างจากตัวดำเนินการ +, * และ ? ตรงที่ตัวดำเนินการเหล่านี้จับคู่เฉพาะอักขระที่นำหน้าตัวดำเนินการเท่านั้น ในการจับคู่เฉพาะอักขระที่นำหน้านิพจน์ช่วง ให้ใช้ตัวดำเนินการ จัดกลุ่ม ตัวอย่าง xy(z{2,4}) จับคู่ xyzz, xyzzz หรือ xyzzzz

<*StartCondition*>

เรียกใช้งานการดำเนินการที่สัมพันธ์กันเฉพาะถ้าตัววิเคราะห์คำ อยู่ในเงื่อนไขเริ่มทำงานที่ระบุเท่านั้น

ตัวอย่าง: ถ้าตำแหน่งเริ่มต้น ของบรรทัดเป็นเงื่อนไขเริ่มทำงาน ONE ดังนั้นโอเปอเรเตอร์ ^ (caret) จะเท่ากับนิพจน์ <ONE>

ในการใช้อักขระตัวดำเนินการเป็นอักขระธรรมดา ให้ใช้ escape sequences ค่าใดค่าหนึ่ง: " " (เครื่องหมายอัญประกาศคู่) หรือ \ (backslash) ตัวดำเนินการ " " ระบุว่าสิ่งที่อยู่ภายในเครื่องหมาย เป็นข้อความ ดังนั้น ตัวอย่างต่อไปนี้จะจับคู่สตริง xyz++:

```
xyz"++"
```

ส่วนของสตริงสามารถอยู่ภายในเครื่องหมาย การใส่เครื่องหมายคำพูด อักขระข้อความธรรมดาจะไม่มีผลใดๆ ตัวอย่าง นิพจน์ต่อไปนี้เท่ากับ ตัวอย่างก่อนหน้า:

```
"xyz++"
```

ในการทำให้แน่ใจว่าข้อมูลถูกแปลความหมายเป็นข้อความ ให้ใส่เครื่องหมาย คำพูดอักขระทั้งหมดที่ไม่ใช่ตัวอักษรหรือตัวเลข

อีกวิธีหนึ่งคือแปลงอักขระตัวดำเนินการ เป็นอักขระข้อความโดยใส่อักขระ \ (backslash) ก่อนหน้าอักขระตัวดำเนินการ ตัวอย่าง นิพจน์ต่อไปนี้เท่ากับ ตัวอย่างก่อนหน้า:

```
xyz\+\+
```

หลักการที่เกี่ยวข้อง:

"เงื่อนไขการเริ่มทำงานโปรแกรม lex" ในหน้า 572
กฎอาจถูกเชื่อมโยงกับเงื่อนไขการเริ่มทำงานใดๆ

การผ่านโค้ดไปยังโปรแกรม lex ที่สร้างขึ้น

คำสั่ง **lex** ส่งโค้ดภาษา C ที่ไม่เปลี่ยนแปลง ไปที่ตัววิเคราะห์คำในกรณีเหตุการณ์ต่อไปนี้:

- บรรทัดขึ้นต้นด้วยช่องว่างหรือแท็บในส่วนนิยาม หรือที่ ตอนต้นของส่วนกฎก่อนกฎข้อแรก จะถูกทำสำเนาไปไว้ใน ตัววิเคราะห์คำ ถ้ารายการอยู่ในส่วนนิยาม จะถูกทำสำเนา ไปยังพื้นที่การประกาศภายนอกของไฟล์ **lex.yy.c** ถ้ารายการอยู่ที่ตอนเริ่มต้นของส่วนกฎ รายการจะถูกทำสำเนาไปยัง พื้นที่การประกาศโลคัลของรูทีนย่อย **yylex** ใน ไฟล์ **lex.yy.c**
- บรรทัดที่อยู่ระหว่างบรรทัดตัวคั่นที่มีเพียง %{ (เครื่องหมายเปอร์เซ็นต์ วงเล็บปีกกาซ้าย) และ %} (เครื่องหมาย เปอร์เซ็นต์ วงเล็บปีกกาขวา) เท่านั้นในส่วนนิยาม หรือที่ตอนเริ่มของ ส่วนกฎจะถูกทำสำเนาไปไว้ในตัววิเคราะห์คำวิธีเดียวกับบรรทัดที่ขึ้นต้นด้วยช่องว่างหรือแท็บ
- บรรทัดใดก็ตามที่เกิดขึ้นหลังตัวคั่น %% (เครื่องหมาย เปอร์เซ็นต์ เครื่องหมายเปอร์เซ็นต์) ที่สองถูกทำสำเนาไปที่ตัววิเคราะห์คำโดยไม่มีข้อจำกัด ด้านรูปแบบ

การกำหนดสตริงแทน lex

คุณสามารถกำหนดแมโครสตริงที่ โปรแกรม **lex** ขยายเมื่อสร้างตัววิเคราะห์ คำ

กำหนดสตริงก่อนตัวคั่น %% แรก ในไฟล์ข้อมูลจำเพาะ **lex** บรรทัดใดก็ตามในส่วนนี้ที่เริ่มต้นในคอลัมน์ 1 และที่ไม่ได้อยู่ระหว่าง %{ และ %} จะกำหนด สตริงแทน **lex** นิยามสตริงแทน มีรูปแบบทั่วไปต่อไปนี้:

```
name                    translation
```

โดยที่ name และ translation ถูกคั่นด้วยช่องว่างหรือแท็บอย่างน้อยหนึ่งอักขระ และ ชื่อที่ระบุที่ขึ้นต้นด้วยตัวอักษร เมื่อโปรแกรม **lex** พบ สตริงที่กำหนดโดย name อยู่ภายในเครื่องหมาย {} (วงเล็บปีกกา) ในส่วนของกฎในไฟล์ข้อมูลจำเพาะ โปรแกรมจะเปลี่ยน ชื่อนั้นเป็นสตริงที่กำหนดใน translation และ ลบวงเล็บปีกกา

ตัวอย่าง ในการกำหนดชื่อ D และ E ให้ใส่นิยามต่อปนี้ก่อน ตัวคั่น %% แรกในไฟล์ข้อมูลจำเพาะ:

```
D   [0-9]
E           [DEde][-+]{D}+
```

จากนั้นใช้ชื่อเหล่านี้ในส่วนกฎของ ไฟล์ข้อมูลจำเพาะเพื่อให้กฎสั้นลง:

```
{D}+                            printf("integer");
{D}+"."{D}*({E})?               |
{D}*"."{D}+({E})?               |
{D}+{E}                         printf("real");
```

คุณยังสามารถรวมรายการต่อไปนี้ในส่วน นิยาม:

- ตารางชุดอักขระ
- รายการเงื่อนไขการเริ่มทำงาน
- เปลี่ยนเป็นขนาดของอาร์เรย์เพื่อรองรับซอร์สโปรแกรมที่มีขนาดใหญ่ขึ้น

ไลบรารี lex

ไลบรารี **lex** มี รูทีนย่อยต่อไปนี้:

รูทีนย่อย	คำอธิบาย
main()	เรียกใช้ตัววิเคราะห์ค่า โดยการเรียกใช้รูทีนย่อย **yylex**
yywrap()	ส่งกลับค่า 1 เมื่อ สิ้นสุดอินพุต
yymore()	ต่อท้ายสตริงที่ตรง ถัดไปที่ค่าปัจจุบันของอาร์เรย์ **yytext** แทน การแทนที่เนื้อหาของอาร์เรย์ **yytext**
yyless(int *n*)	เก็บอักขระเริ่มต้น *n* ในอาร์เรย์ **yytext** และส่งกลับ อักขระที่เหลือไปยังสตรีมอินพุต
yyreject()	อนุญาตให้ตัววิเคราะห์ค่า จับคู่กฎหลายกฎสำหรับสตริงอินพุตที่เหมือนกัน (รูทีนย่อย **yyreject** ถูกเรียกใช้เมื่อมีการใช้การดำเนินการพิเศษ **REJECT**)

บางรูทีนย่อย **lex** สามารถถูกแทนที่ด้วยรูทีนย่อยที่ผู้ใช้กำหนด ตัวอย่าง คำสั่ง **lex** สนับสนุนเวอร์ชันที่ผู้ใช้ระบุของรูทีนย่อย **main** และ **yywrap** เวอร์ชันไลบรารี ของรูทีนเหล่านี้ ที่ถูกจัดให้มีเบื้องต้น มีดังนี้:

main subroutine

```
#include <stdio.h>
#include <locale.h>
main() {
     setlocale(LC_ALL, "");
     yylex();
     exit(0);
}
```

yywrap subroutine

```
yywrap() {
       return(1);
}
```

รูทีนย่อย **yymore, yyless** และ **yyreject** พร้อมใช้งาน ได้โดยผ่านไลบรารี **lex** เท่านั้น อย่างไรก็ตาม รูทีนย่อยเหล่านี้ จำเป็นต้องใช้เมื่อใช้ในการดำเนินการ **lex** เท่านั้น

การดำเนินการที่ทำโดยตัววิเคราะห์คำ

เมื่อตัววิเคราะห์คำจับคู่นิพจน์ทั่วไปที่ขยายเพิ่ม นิพจน์หนึ่งในส่วนกฎของไฟล์ข้อมูลจำเพาะ ตัววิเคราะห์จะทำ *การดำเนินการ* ที่สอดคล้องกับนิพจน์ทั่วไป ที่ขยายเพิ่มนั้น โดยไม่ต้องมีกฎพอที่จะตรงกับสตริงทั้งหมดในอินพุตสตรีม ตัววิเคราะห์คำจะทำสำเนาไปยังเอาต์พุตมาตรฐาน ดังนั้น อย่า สร้างกฎที่ทำสำเนาเอาต์พุตไปเอาต์พุตเท่านั้น เอาต์พุตดีฟอลต์ สามารถช่วยค้นหาช่องว่างในกฎ

เมื่อใช้คำสั่ง **lex** เพื่อ ประมวลผลอินพุตให้แก่โปรแกรมวิเคราะห์คำที่คำสั่ง **yacc** สร้างขึ้น ให้มีกฎเพื่อจับคู่สตริงอินพุตทั้งหมด กฎเหล่านั้นต้องสร้างเอาต์พุต ที่คำสั่ง **yacc** สามารถแปลความหมายได้

การดำเนินการ Null

ในการข้ามอินพุตที่เชื่อมโยงกับนิพจน์ทั่วไปที่ขยายเพิ่ม ให้ใช้ ; (ประโยคคำสั่ง null ของภาษา C) เป็นการดำเนินการ ตัวอย่างต่อไปนี้จะข้ามอักขระระยะห่าง (ช่องว่าง, แท็บ และขึ้นบรรทัดใหม่):

```
[ \t\n] ;
```

เหมือนกับการดำเนินการถัดไป

ในการเลี่ยงการเขียนการดำเนินการที่เหมือนกันซ้ำๆ ให้ใช้ | (สัญลักษณ์ไพพ์) อักขระนี้บ่งชี้ว่าการดำเนินการสำหรับกฎนี้เหมือนกับการดำเนินการสำหรับกฎข้อถัดไป ตัวอย่างเช่น ในตัวอย่างก่อนหน้า ที่ไม่สนใจอักขระช่องว่าง แท็บและขึ้นบรรทัดใหม่ ยังสามารถเขียนได้ดังนี้:

```
" "                     |
"DefaultKeyBindings"
"\n"                    ;
```

เครื่องหมายคำพูดที่ล้อมรอบ \n และ \t ไม่จำเป็นต้องมี

การพิมพ์สตริงที่ตรงกัน

ในการพิจารณาข้อความที่ตรงกับนิพจน์ใน ส่วนของกฎของไฟล์ข้อมูลจำเพาะ คุณสามารถรวมการเรียกใช้รูทีนย่อยภาษา C **printf** เป็นหนึ่งในการดำเนินการสำหรับ นิพจน์นั้น เมื่อตัววิเคราะห์คำพบค่าที่ตรงในสตรีมอินพุต โปรแกรม จะใส่สตริงที่ตรงนั้นลงในอาร์เรย์อักขระภายนอก (**char**) และอักขระตัวแทน (**wchar_t**) ที่เรียกใช้ **yytext** and **yywtext,** ตามลำดับ ตัวอย่าง คุณสามารถใช้กฎต่อไปนี้เพื่อพิมพ์สตริงที่ตรง:

```
[a-z]+          printf("%s",yytext);
```

รูทีนย่อย **printf** ในภาษา C ยอมรับอาร์กิวเมนต์การจัดรูปแบบ และข้อมูลที่จะถูกพิมพ์ ในตัวอย่างนี้ อาร์กิวเมนต์ สำหรับรูทีนย่อย **printf** มีความหมายต่อไปนี้:

`%s` สัญลักษณ์ที่แปลงข้อมูลเป็นประเภทสตริงก่อนทำการพิมพ์

`%S` สัญลักษณ์ที่แปลงข้อมูลเป็นสตริงอักขระตัวแทน (**wchar_t**) ก่อนทำการพิมพ์

`yytext`
ชื่อของอาร์เรย์ที่มีข้อมูลที่จะพิมพ์

`yywtext`
ชื่อของอาร์เรย์ที่มีข้อมูลประเภทหลายไบต์ (**wchar_t**) ที่จะพิมพ์

คำสั่ง **lex** กำหนด **ECHO;** เป็นการดำเนินการ พิเศษเพื่อพิมพ์เนื้อหาของ **yytext** ตัวอย่าง กฎสองข้อต่อไปนี้เทียบเท่ากัน:

```
[a-z]+      ECHO;
[a-z]+      printf("%s",yytext);
```

คุณสามารถเปลี่ยนการแสดงของ **yytext** ได้โดยใช้ **%array** หรือ **%pointer** ในส่วนข้อกำหนดของไฟล์ข้อมูลจำเพาะ **lex** ดังนี้:

`%array`
กำหนด **yytext** เป็นอาร์เรย์อักขระที่สิ้นสุด ด้วยค่า null นี้เป็นการดำเนินการดีฟอลต์

`%pointer`
กำหนด **yytext** เป็นตัวชี้ไปยังสตริงอักขระ ที่สิ้นสุดด้วยค่า null

การหาความยาวของสตริงที่ตรงกัน

ในการหาจำนวนอักขระ ที่ตัววิเคราะห์คำจับคู่สำหรับนิพจน์ทั่วไปที่ขยายเพิ่มเฉพาะ ให้ใช้ตัวแปรภายนอก **yyleng** หรือ **yywleng**

`yyleng`
ติดตามจำนวนไบต์ที่ตรง

`yywleng`
ติดตามจำนวนอักขระตัวแทนในสตริงที่ตรง อักขระ หลายไบต์มีความยาวมากกว่า 1

ในการนับทั้งจำนวนคำและจำนวน อักขระในคำในอินพุต ให้ใช้การดำเนินการต่อไปนี้:

```
[a-zA-Z]+       {words++;chars += yyleng;}
```

การดำเนินการนี้หาผลรวมจำนวนอักขระใน คำที่ตรงและใส่จำนวนนั้นใน `chars`

นิพจน์ต่อไปนี้ค้นหาอักขระสุดท้าย ในสตริงที่ตรง:

```
yytext[yyleng-1]
```

การจับคู่สตริงภายในสตริง

คำสั่ง **lex** แบ่งส่วน สตรีมอินพุตและไม่ค้นหาค่าที่เป็นไปได้ทั้งหมดที่ตรงกันของแต่ละนิพจน์ แต่ละอักขระจะถูกนำไปคำนวณเพียงครั้งเดียวเท่านั้น ในการแทนที่ตัวเลือกนี้ และค้นหา รายการที่อาจซ้อน หรือรวมอยู่ ให้ใช้การดำเนินการ **REJECT** ตัวอย่าง ในการนับจำนวนที่มีทั้งหมดของ `she` และ `he` รวมถึงจำนวนที่มีของ `he` ที่รวมอยู่ในคำว่า `she` ใช้การดำเนินการต่อไปนี้:

```
she             {s++; REJECT;}
he              {h++}
\n              |
.               ;
```

หลังการนับจำนวนการเกิดขึ้นของ she แล้ว คำสั่ง **lex** จะปฏิเสธสตริงอินพุตและ จากนั้นนับจำนวนที่เกิดขึ้นของ he เนื่องจาก he ไม่รวมใน she การดำเนินการ **REJECT** จึงไม่จำเป็นสำหรับ he

การเพิ่มผลลัพธ์ในอาร์เรย์ yytext

โดยทั่วไป สตริงถัดไปจากสตรีมอินพุตจะบันทึกทับ รายการปัจจุบันที่มีอยู่ในอาร์เรย์ **yytext** ถ้าคุณใช้รูทีนย่อย **yymore** สตรีมถัดไปจากสตรีมอินพุต จะถูกเพิ่มที่ท้ายรายการปัจจุบันในอาร์เรย์ **yytext**

ตัวอย่าง ตัววิเคราะห์คำต่อไปนี้ค้นหา สตริง:

```
%s instring
%%
<INITIAL>\"     {  /* start of string */
         BEGIN instring;
         yymore();
        }
<instring>\"    {  /* end of string */
         printf("matched %s\n", yytext);
         BEGIN INITIAL;
        }
<instring>. {
         yymore();
        }
<instring>\n    {
         printf("Error, new line in string\n");
         BEGIN INITIAL;
        }
```

แม้ว่าสตริงอ่านถูกรู้จักโดยการจับคู่ กฎหลายๆ ข้อ การเรียกใช้ไปยังรูทีนย่อย **yymore** ที่ถูกทำซ้ำ ตรวจสอบว่าอาร์เรย์ **yytext** จะมีครบทั้ง สตริง

การส่งกลับอักขระไปยังอินพุตสตรีม

ในการส่งกลับอักขระไปยังสตรีมอินพุต ใช้ การเรียกใช้ต่อไปนี้:

```
yyless(n)
```

โดยที่ n คือจำนวน อักขระของสตริงปัจจุบันที่จะเก็บไว้ อักขระในสตริงที่เกินจำนวนนี้ จะถูกส่งกลับไปยังสตรีมอินพุต รูทีนย่อย **yyless** จะมีฟังก์ชันที่เป็นประเภทค้นหาล่วงหน้าที่ตัวดำเนินการ / (เครื่องหมายทับ) ใช้ แต่มีการควบคุมการ ใช้งานมากกว่า

ใช้รูทีนย่อย **yyless** เพื่อ ประมวลผลข้อความมากกว่าหนึ่งครั้ง ตัวอย่าง เมื่อวิเคราะห์คำโปรแกรมภาษา C นิพจน์เช่น x=-a จะเข้าใจได้ยาก นิพจนี้นี้หมายความว่า x *เท่ากับ ลบ* a หรือเป็นการแทนการแบบเก่า ของ x -= a ซึ่งหมายถึง *ลดค่า* x *ด้วยค่าของ* a? ในการปฏิบัติต่อนิพจน์นี้เป็น x *เท่ากับลบ* a แต่พิมพ์ข้อความเตือน ให้ใช้กฎเช่นตัวอย่างต่อไปนี้:

```
=-[a-zA-Z]      {
                printf("Operator (=-) ambiguous\n");
                yyless(yyleng-1);
                ... action for = ...
                }
```

รูทีนย่อยอินพุต/เอาต์พุต

โปรแกรม **lex** อนุญาตให้โปรแกรมใช้รูทีนย่อยอินพุต/เอาต์พุต (I/O) ต่อไปนี้:

`input()`
ส่งกลับอักขระอินพุตถัดไป

`output(c)`
เขียนอักขระ c ไปที่เอาต์พุต

`unput(c)`
วางอักขระ c กลับไปที่สตรีมอินพุตเพื่ออ่านภายหลัง โดยรูทีนย่อย **input**

`winput()`
ส่งกลับอักขระอินพุตหลายไบต์ถัดไป

`woutput(C)`
เขียนอักขระหลายไบต์ C กลับไปที่สตรีมเอาต์พุต

`wunput(C)`
วางอักขระหลายไบต์ C กลับไปที่สตรีมอินพุตเพื่อ อ่านโดยรูทีนย่อย **winput**

โปรแกรม **lex** จัดให้มี รูทีนย่อยเหล่านี้แบบเดียวกับ macro definitions รูทีนย่อยมีโค้ดอยู่ในไฟล์ **lex.yy.c** คุณสามารถแทนที่โค้ดเหล่านั้น และให้มีเวอร์ชันใหม่ได้

แมโคร **winput, wunput** และ **woutput** ถูกกำหนดให้ใช้รูทีนย่อย **yywinput, yywunput** และ **yywoutput** เพื่อความเข้ากันได้ รูทีนย่อย **yy** ใช้รูทีนย่อย **input, unput,** และ **output** ตามลำดับเพื่ออ่าน, แทนที่ และเขียน จำนวนไบต์ที่ต้องการในอักขระหลายไบต์ที่สมบูรณ์

รูทีนย่อยเหล่านี้กำหนดความสัมพันธ์ระหว่าง ไฟล์ภายนอกและอักขระภายใน ถ้าคุณเปลี่ยนรูทีนย่อย ให้เปลี่ยนทั้งหมด ด้วยวิธีเดียวกัน รูทีนย่อยเหล่านี้ควรปฏิบัติตามกฎเหล่านี้:

- รูทีนย่อยทั้งหมดต้องใช้ชุดอักขระเดียวกัน
- รูทีนย่อย **input** ต้องส่งกลับค่า 0 เพื่อ ระบุว่าสิ้นสุดไฟล์
- ห้ามเปลี่ยนความสัมพันธ์ของรูทีนย่อย **unput** ไปยังรูทีนย่อย **input** หรือมิฉะนั้นฟังก์ชันการค้นหาล่วงหน้า จะไม่ทำงาน

ไฟล์ **lex.yy.c** อนุญาตให้ ตัววิเคราะห์คำสำรองข้อมูลได้สูงสุด 200 อักขระ

ในการอ่านไฟล์ที่มีค่า nulls ให้สร้างเวอร์ชันใหม่ ของรูทีนย่อย **input** ในเวอร์ชันปกติ ของรูทีนย่อย **input** ค่าที่ส่งกลับ 0 (จากอักขระ null) ระบุว่าสิ้นสุดไฟล์ และสิ้นสุดอินพุต

ชุดอักขระ

ตัววิเคราะห์คำที่คำสั่ง **lex** สร้างขึ้นจะประมวลผล I/O อักขระผ่านทางรูทีนย่อย **input, output** และ **unput** ดังนั้น ในการส่งกลับค่าในรูทีนย่อย **yytext** คำสั่ง**lex** จะใช้การแทนค่าอักขระที่รูทีนย่อยเหล่านี้ใช้ อย่างไรก็ตาม สำหรับภายในคำสั่ง **lex** แทนค่าแต่ละอักขระด้วยค่าจำนวนเต็มที่เล็กลง เมื่อใช้ไลบรารีมาตรฐาน ค่าจำนวนเต็มนี้ คือค่าของรูปแบบของบิตที่คอมพิวเตอร์ใช้แทน อักขระ โดยปกติ ตัวอักษร *a* ถูกแทนใน รูปแบบเดียวกับค่าคงที่อักขระ *a* ถ้าคุณ เปลี่ยนการแปลความหมายนี้ด้วยรูทีนย่อย I/O อื่น ให้ใส่ตาราง คำแปลในส่วนนิยามของไฟล์ข้อมูลจำเพาะ ตาราง คำแปลเริ่มต้นและสิ้นสุดด้วยบรรทัดที่มีรายการต่อไปนี้เท่านั้น:

```
%T
```

ตารางคำแปลมีบรรทัดเพิ่มเติมที่ บ่งชี้ค่าที่เชื่อมโยงกับแต่ละอักขระ ตัวอย่างเช่น:

```
%T
{integer}       {character string}
{integer}       {character string}
{integer}       {character string}
%T
```

การประมวลผล End-of-file

เมื่อตัววิเคราะห์คำถึงจุดสิ้นสุดไฟล์ จะเรียกใช้รูทีนย่อยไลบรารี **yywrap** ซึ่งส่งกลับ ค่า 1 เพื่อบ่งชี้ไปที่ตัววิเคราะห์คำว่าควรดำเนินต่อ ด้วยการเริ่มต้นใหม่ตามปกติเมื่อสิ้นสุดอินพุต

อย่างไรก็ตาม ถ้าตัววิเคราะห์คำได้รับอินพุตจากแหล่งที่มามากกว่าหนึ่งแห่ง ให้เปลี่ยนรูทีนย่อย **yywrap** ฟังก์ชันใหม่ต้องรับค่าอินพุตใหม่และส่งกลับค่า 0 ไปยังตัววิเคราะห์คำ ค่าที่ส่งกลับ 0 บ่งชี้ว่าโปรแกรมควรดำเนินการประมวลผลต่อ

คุณยังสามารถรวมโค้ดเพื่อพิมพ์รายงานสรุป และตารางเมื่อตัววิเคราะห์คำสิ้นสุดในเวอร์ชันใหม่ของรูทีนย่อย **yywrap** รูทีนย่อย **yywrap** เป็นทางเดียว ที่จะบังคับให้รูทีนย่อย **yylex** ทราบถึง การสิ้นสุดอินพุต

เงื่อนไขการเริ่มทำงานโปรแกรม lex

กฎอาจถูกเชื่อมโยงกับเงื่อนไขการเริ่มทำงานใดๆ

อย่างไรก็ตาม โปรแกรม **lex** ทราบถึงกฎก็ต่อเมื่อ อยู่ในเงื่อนไขการเริ่มทำงานที่เชื่อมโยงนั้น คุณสามารถเปลี่ยนเงื่อนไขการเริ่มทำงานปัจจุบัน ได้ตลอดเวลา

กำหนดเงื่อนไขการเริ่มทำงานในส่วน *นิยาม* ของไฟล์ข้อมูลจำเพาะโดยใช้บรรทัดในฟอร์ม ต่อไปนี้:

```
%Start  name1 name2
```

โดยที `name1` และ `name2` กำหนดชื่อที่แทนเงื่อนไข โดยไม่มีข้อจำกัดสำหรับ จำนวนเงื่อนไข และสามารถปรากฏในลำดับใดๆ ก็ได้ คุณยังสามารถตัด คำ `Start` ให้สั้นลงเหลือเพียง `s` หรือ `S`

เมื่อใช้เงื่อนไขการเริ่มทำงานในส่วนของกฎของไฟล์ข้อมูลจำเพาะ ให้ชื่อของเงื่อนไขเริ่มทำงานในสัญลักษณ์ <> (น้อยกว่ามากกว่า) ที่ตอนเริ่มต้นของกฎ ตัวอย่างต่อไปนี้กำหนดกฎ นิพจน์ ที่โปรแกรม **lex** จะรู้จักต่อเมื่อโปรแกรม **lex** อยู่ในเงื่อนไขการเริ่มทำงาน `name1`:

```
<name1> expression
```

ในการวางโปรแกรม **lex** ให้มี เงื่อนไขการเริ่มทำงานเฉพาะ ให้ทำงานคำสั่งการดำเนินการในส่วนการดำเนินการ ของกฎ ตัวอย่างเช่น `BEGIN` ในบรรทัด ต่อไปนี้:

```
BEGIN name1;
```

คำสั่งนี้เปลี่ยนเงื่อนไขการเริ่มทำงานเป็น name1

ในการทำงานสถานะปกติต่อไปให้ป้อน:

```
BEGIN 0;
```

หรือ

```
BEGIN INITIAL;
```

โดยที่ INITIAL ถูกกำหนดให้เป็น 0 โดยโปรแกรม **lex** BEGIN 0; ตั้งค่าโปรแกรม **lex** ให้ให้เป็นเงื่อนไขเริ่มต้น

โปรแกรม **lex** ยังสนับสนุน เงื่อนไขการเริ่มทำงานเฉพาะที่ระบุด้วยตัวดำเนินการ **%x** (เครื่องหมาย เปอร์เซ็นต์ ตัวพิมพ์เล็ก x) หรือ **%X** (เครื่องหมายเปอร์เซ็นต์ ตัวพิมพ์ใหญ่ X) ตามด้วยรายการชื่อการเริ่มทำงานเฉพาะในรูปแบบเดียวกับ เงื่อนไขการเริ่มทำงานปกติ เงื่อนไขการเริ่มทำงานเฉพาะต่างจากเงื่อนไขการเริ่มทำงาน ปกติตรงที่กฎที่ไม่ขึ้นต้นด้วยเงื่อนไขการเริ่มทำงานจะไม่แอ็คทีฟ เมื่อตัววิเคราะห์คำอยู่ในสถานะเริ่มทำงานเฉพาะ ตัวอย่างเช่น:

```
%s      one
%x      two
%%
abc     {printf("matched ");ECHO;BEGIN one;}
<one>def        printf("matched ");ECHO;BEGIN two;}
<two>ghi        {printf("matched ");ECHO;BEGIN INITIAL;}
```

ในสถานะเริ่มทำงานในตัวอย่างก่อนหน้านี้ ทั้ง abc และ def สามารถแก้ไขได้ ในสถานะเริ่มทำงาน มีเพียง ghi เท่านั้นที่สามารถแก้ไขได้

หลักการที่เกี่ยวข้อง:

"นิพจน์ทั่วไปส่วนขยายในคำสั่ง lex" ในหน้า 562
การระบุนิพจน์ทั่วไปที่ขยายเพิ่มในไฟล์ข้อมูลจำเพาะ **lex** คล้ายกับวิธีที่ใช้ในคำสั่ง **sed** หรือ **ed**

การสร้างโปรแกรมวิเคราะห์คำด้วยโปรแกรม yacc

โปรแกรม **yacc** สร้างโปรแกรมวิเคราะห์คำ ที่กำหนดโครงสร้างการบังคับใช้สำหรับอินพุตอักขระไปยังโปรแกรมคอมพิวเตอร์

ในการใช้โปรแกรมนี้ คุณต้องกำหนดอินพุตต่อไปนี้:

ไฟล์ ไวยากรณ์
ไฟล์ต้นฉบับที่มีข้อมูลจำเพาะเพื่อให้ภาษา รู้จัก ไฟล์นี้ยังมีรูทีนย่อย **main, yyerror** และ **yylex** คุณต้องกำหนด รูทีนย่อยเหล่านี้

main
รูทีนย่อยภาษา C ที่อย่างน้อยที่สุดต้องมีการเรียกใช้ไปยังรูทีนย่อย **yyparse** ที่สร้างขึ้นโดยโปรแกรม **yacc** ฟอร์มที่จำกัดของรูทีนย่อยนี้มีอยู่ในไลบรารี **yacc**

yyerror
รูทีนย่อยภาษา C ที่จะจัดการข้อผิดพลาดที่อาจเกิดขึ้นระหว่างการดำเนินการ วิเคราะห์คำ ฟอร์มที่จำกัดของรูทีนย่อยนี้มีอยู่ในไลบรารี **yacc**

yylex
: รูทีนย่อยภาษา C ที่ทำหน้าที่ในการการวิเคราะห์ศัพท์บนสตรีม อินพุตและส่งโทเค็นไปยังโปรแกรมวิเคราะห์คำ คุณสามารถใช้คำสั่ง **lex** เพื่อสร้างรูทีนย่อยตัววิเคราะห์คำนี้

เมื่อคำสั่ง **yacc** ได้รับ ข้อมูลจำเพาะ จะสร้างไฟล์ที่ประกอบด้วยฟังก์ชันภาษา C ชื่อ **y.tab.c** เมื่อคอมไพล์โดยใช้คำสั่ง **cc** ฟังก์ชันเหล่านี้จะฟอร์มรูทีนย่อย **yyparse** และส่งกลับค่า เลขจำนวนเต็ม เมื่อถูกเรียกใช้ รูทีนย่อย **yyparse** จะเรียกใช้รูทีนย่อย **yylex** เพื่อรับโทเค็นอินพุต รูทีนย่อย **yylex** ดำเนินกาเตรียมอินพุตให้ต่อไปจนกระทั่งโปรแกรมวิเคราะห์คำตรวจพบ ข้อผิดพลาด หรือรูทีนย่อย **yylex** ส่งกลับโทเค็น end-marker เพื่อแจ้งว่าสิ้นสุดการดำเนินการ ถ้ามีข้อผิดพลาดเกิดขึ้นและรูทีนย่อย **yyparse** ไม่สามารถกู้คืนได้ จะส่งกลับค่า 1 ไปยังรูทีนย่อย **main** ถ้าพบโทเค็น end-marker รูทีนย่อย **yyparse** จะส่งกลับค่า 0 ไปยังรูทีนย่อย **main**

หลักการที่เกี่ยวข้อง:

"การสร้างตัววิเคราะห์คำด้วยคำสั่ง lex" ในหน้า 560
คำสั่ง **lex** ช่วยในการเขียน โปรแกรมภาษา C ที่สามารถรับและแปลอินพุตสตรีมอักขระ เป็นการดำเนินการของโปรแกรม

ไฟล์ไวยากรณ์ yacc

ในการใช้คำสั่ง **yacc** เพื่อสร้าง โปรแกรมวิเคราะห์คำ ควรมีไฟล์ไวยากรณ์ที่อธิบายสตรีมข้อมูลอินพุต และสิ่งที่โปรแกรมวิเคราะห์คำควรทำกับข้อมูล

ไฟล์ไวยากรณ์ประกอบด้วยกฎ ที่อธิบายโครงสร้างอินพุต โค้ดที่จะถูกเรียกใช้เมื่อกฎเหล่านี้ถูกพบ และรูทีนย่อยที่จะทำการอินพุตเบื้องต้น

คำสั่ง **yacc** ใช้ข้อมูล ในไฟล์ไวยากรณ์เพื่อสร้างโปรแกรมวิเคราะห์คำที่ใช้ควบคุมกระบวนการอินพุต โปรแกรมวิเคราะห์คำนี้เรียกใช้รูทีนย่อยอินพุต (ตัววิเคราะห์คำ) เพื่อเลือก รายการเบื้องต้น (เรียกว่า *โทเค็น*) จากสตรีมอินพุต โทเค็นคือสัญลักษณ์หรือชื่อที่บอกให้โปรแกรมวิเคราะห์คำทราบถึงรูปแบบที่จะถูกส่ง ไปโดยรูทีนย่อยอินพุต สัญลักษณ์ nonterminal คือโครงสร้างที่ โปรแกรมวิเคราะห์คำรู้จัก โปรแกรมวิเคราะห์คำจัดระเบียบโทเค็นเหล่านี้ตาม กฎโครงสร้างในไฟล์ไวยากรณ์ กฎโครงสร้างถูกเรียกเป็น *กฎไวยากรณ์* เมื่อโปรแกรมวิเคราะห์คำรู้จักหนึ่งในกฎเหล่านี้ จะเรียกใช้งาน โค้ดของผู้ใช้ที่กำหนดไว้สำหรับกฎนั้น โค้ดของผู้ใช้ถูกเรียกเป็น *การดำเนินการ* การดำเนินการส่งกลับค่าและใช้ค่าที่ส่งกลับโดยการดำเนินการอื่น

ใช้ภาษาโปรแกรมภาษา C เพื่อเขียนโค้ดการดำเนินการ และรูทีนย่อยอื่นๆ คำสั่ง **yacc** ใช้ระบบ ไวยากรณ์หลายๆ ข้อของภาษา C สำหรับไฟล์ไวยากรณ์

รูทีนย่อย main และ yyerror

คุณต้องจัดเตรียมรูทีนย่อย **main** และ **yyerror** สำหรับโปรแกรมวิเคราะห์คำ เพื่อช่วยให้ง่ายต่อการเริ่ม การใช้คำสั่ง **yacc** ไลบรารี **yacc** จะมีเวอร์ชันแบบง่ายของรูทีนย่อย **main** และ **yyerror** รวมรูทีนย่อยเหล่านี้โดยใช้ อาร์กิวเมนต์ *-ly* กับคำสั่ง **ld** (หรือกับคำสั่ง **cc**) ซอร์สโค้ดสำหรับโปรแกรมไลบรารี **main** เป็นดังนี้:

```
#include <locale.h>
main()
{
     setlocale(LC_ALL, "");
     yyparse();
}
```

ซอร์สโค้ดสำหรับโปรแกรมไลบรารี **yyerror** เป็นดังนี้:

```
#include <stdio.h>
yyerror(s)
        char *s;
{
        fprintf( stderr, "%s\n" ,s);
}
```

อาร์กิวเมนต์ไปยังรูทีนย่อย **yyerror** เป็นสตริงที่มี ข้อความแสดงข้อผิดพลาด โดยทั่วไปเป็นสตริง `syntax error`

เนื่องจากโปรแกรมเหล่านี้ถูกจำกัด ให้มีฟังก์ชันเพิ่มเติม ในรูทีนย่อยเหล่านี้ ตัวอย่าง การติดตามหมายเลขบรรทัดอินพุตและพิมพ์ หมายเลขควบคู่กับข้อความเมื่อตรวจพบข้อผิดพลาดทางไวยากรณ์ คุณอาจ ต้องการใช้ค่าในตัวแปรจำนวนเต็มภายนอก *yychar* ด้วย ตัวแปรนี้มีจำนวนโทเค็นที่คาดในอนาคต ณ เวลาที่ ตรวจพบข้อผิดพลาด

รูทีนย่อย yylex

รูทีนย่อยอินพุตที่คุณกำหนดให้แก่ไฟล์ไวยากรณ์ต้องสามารถทำต่อไปนี้:

- อ่านสตรีมอินพุต
- รู้จักรูปแบบเบื้องต้นในสตรีมอินพุต
- ส่งต่อรูปแบบไปยังโปรแกรมวิเคราะห์คำ พร้อมกับโทเค็นที่กำหนดรูปแบบ ไปยังโปรแกรมวิเคราะห์คำ

ตัวอย่าง รูทีนย่อยอินพุตแบ่งสตรีม อินพุตออกเป็นโทเค็น `WORD`, `NUMBER` และ `PUNCTUATION` และรับอินพุตต่อไปนี้:

```
I have 9 turkeys.
```

โปรแกรมสามารถเลือกส่งสตริงและ โทเค็นต่อไปนี้ไปยังโปรแกรมวิเคราะห์คำ:

สตริง	โทเค็น
`I`	WORD
`have`	WORD
`9`	NUMBER
`turkeys`	WORD
`.`	PUNCTUATION

โปรแกรมวิเคราะห์คำต้องมีข้อกำหนดสำหรับโทเค็น ที่ส่งมาให้โดยรูทีนย่อยอินพุต การใช้อ็อพชัน **-d** สำหรับคำสั่ง **yacc** จะสร้างรายการของโทเค็น ในไฟล์ชื่อ **y.tab.h** รายการนี้คือชุดของคำสั่ง **#define** ที่อนุญาตให้ตัววิเคราะห์คำ (**yylex**) ใช้โทเค็นเดียวกับโปรแกรมวิเคราะห์คำ

หมายเหตุ: ในการหลีกเลี่ยงความขัดแย้งกับโปรแกรมวิเคราะห์คำ ห้ามใช้ชื่อรูทีนย่อย ที่ขึ้นต้นด้วยตัวอักษร `yy`

คุณสามารถใช้คำสั่ง **lex** เพื่อสร้างรูทีนย่อยอินพุต หรือคุณสามารถเขียนรูทีนเป็นภาษา C ได้

การใช้ไฟล์ไวยากรณ์ yacc

ไฟล์ไวยากรณ์ **yacc** ประกอบด้วย ส่วนต่อไปนี้:

- การประกาศ
- กฎ
- โปรแกรม

เครื่องหมาย %% (เครื่องหมาย เปอร์เซ็นต์ เครื่องหมายเปอร์เซ็นต์) สองตัวติดกันคั่นแต่ละส่วนของไฟล์ไวยากรณ์ ในการทำให้ไฟล์อ่านได้ง่ายขึ้น ให้ใส่เครื่องหมาย %% บนบรรทัดของตนเอง ไฟล์ไวยากรณ์โดยสมบูรณ์จะเป็นดังต่อไปนี้:

```
declarations
%%
rules
%%
โปรแกรม
```

ส่วนการประกาศอาจว่าง ถ้าคุณเว้น ส่วนโปรแกรม ให้เว้นในชุดที่สองของ %% ดังนั้น ไฟล์ไวยากรณ์ **yacc** ที่มีขนาดเล็กที่สุดมีดังนี้:

```
%%
กฎ
```

คำสั่ง **yacc** ข้ามอักขระช่องว่าง แท็บ และขึ้นบรรทัดใหม่ในไฟล์ไวยากรณ์ ดังนั้น ใช้อักขระเหล่านี้ เพื่อทำให้ไฟล์ไวยากรณ์อ่านง่ายขึ้น อย่างไรก็ตาม อย่าใช้อักขระช่องว่าง แท็บ หรือขึ้นบรรทัดใหม่ในชื่อหรือสัญลักษณ์ที่สงวนไว้

การใช้หมายเหตุ

ในการอธิบายสิ่งที่โปรแกรมทำกลัง ให้ใส่หมายเหตุ ในไฟล์ไวยากรณ์ คุณสามารถใส่หมายเหตุได้ทุกที่ในไฟล์ไวยากรณ์ที่ คุณสามารถตั้งชื่อ อย่างไรก็ตาม ในการทำให้ไฟล์อ่านง่ายขึ้น ให้ใส่หมายเหตุ บนบรรทัดของตนเองที่เริ่มต้นบล็อกการทำงานของกฎ หมายเหตุ ในไฟล์ไวยากรณ์ **yacc** จะเหมือนกับหมายเหตุใน โปรแกรมภาษา C หมายเหตุถูกล้อมรอบระหว่าง /* (เครื่องหมายทับ เครื่องหมายดอกจัน) และ */ (เครื่องหมายดอกจัน เครื่องหมายทับ) ตัวอย่างเช่น:

```
/* This is a comment on a line by itself. */
```

การใช้สตริงตัวอักษร

สตริงค่าคงที่คืออักขระอย่างน้อยหนึ่งอักขระที่อยู่ภายใน '' (เครื่องหมายคำพูดเดี่ยว) อย่างในภาษา C เครื่องหมาย \ (backslash) เป็น escape character ภายในค่าคงที่ และรู้จัก escape codes ภาษา C ทั้งหมด ดังนั้น คำสั่ง **yacc** ยอมรับสัญลักษณ์ในตารางต่อไปนี้:

สัญลักษณ์	นิยาม
'**a**'	เตือน
'**b**'	อักขระถอยกลับ
'**f**'	ฟอร์มฟีด
'**n**'	บรรทัดใหม่
'**r**'	Return
'**t**'	แท็บ (Tab)
'**v**'	แท็บในแนวตั้ง
'\''	เครื่องหมายคำพูดเดี่ยว (')
'\"'	เครื่องหมายคำพูดคู่ (")
'\?'	เครื่องหมายคำถาม (?)
'\\'	Backslash (\)
'*Digits*'	อักขระที่มีการเข้ารหัสถูกแทนด้วยจำนวนเต็มฐานแปดหนึ่ง, สอง หรือสามหลักที่ระบุโดยสตริง *Digits*
'**x***Digits*'	อักขระที่มีการเข้ารหัสถูกแทนด้วยลำดับของอักขระฐานสิบหก ที่ระบุโดยสตริง *Digits*

เนื่องจากรหัส ASCII เป็นศูนย์ อักขระ null (\0 หรือ 0) ต้องไม่ถูกนำมาใช้ในกฎไวยากรณ์ รูทีนย่อย **yylex** ส่งกลับค่า 0 ถ้าใช้อักขระ null เป็นการแสดงว่าสิ้นสุดอินพุต

การจัดรูปแบบไฟล์ไวยากรณ์

เพื่อช่วยให้ไฟล์ไวยากรณ์ **yacc** อ่านง่ายขึ้น ให้ใช้แนวทางต่อไปนี้:

- ใช้ตัวอักษรตัวพิมพ์ใหญ่สำหรับชื่อโทเค็น และใช้ตัวอักษรตัวพิมพ์เล็กสำหรับชื่อสัญลักษณ์ nonterminal
- ใส่กฎไวยากรณ์และการดำเนินการบนบรรทัดแยกเพื่อให้การเปลี่ยนแปลง อย่างใดอย่างหนึ่งได้โดยไม่เปลี่ยนอีกอย่างหนึ่ง
- ใส่กฎทั้งหมดที่มีด้านซ้ายเหมือนกันไว้ด้วยกัน ป้อนด้านซ้ายเพียงครั้งเดียว และใช้แถบแนวดิ่งเพื่อเริ่มส่วนที่เหลือของกฎสำหรับด้านซ้ายนั้น
- สำหรับแต่ละชุดของกฎที่มีด้านซ้ายเหมือนกัน ให้ป้อนเซมิโคลอนครั้งเดียว บนบรรทัดของตนตามด้วยกฎสุดท้ายสำหรับด้านซ้ายนั้น จากนั้นคุณสามารถ เพิ่มกฎใหม่ได้โดยง่าย
- เยื้องเนื้อความของกฎเข้าไปสองแท็บ และเนื้อความส่วนการดำเนินการเข้าไปสามแท็บ

ข้อผิดพลาดในไฟล์ไวยากรณ์

คำสั่ง **yacc** ไม่สามารถสร้าง โปรแกรมวิเคราะห์คำสำหรับชุดของข้อมูลจำเพาะไวยากรณ์ทั้งหมดได้ ถ้ากฎไวยากรณ์ขัดแย้งกันเอง หรือจำเป็นต้องใช้เทคนิคการจับคู่ค่าที่แตกต่างจากสิ่งที่คำสั่ง **yacc** จัดให้มี คำสั่ง **yacc** จะไม่สร้างโปรแกรมวิเคราะห์คำ ในกรณีส่วนใหญ่ คำสั่ง **yacc** จะจัดให้มีข้อความที่แสดงข้อผิดพลาด ในการแก้ไขข้อผิดพลาดเหล่านี้ให้ออกแบบ กฎในไฟล์ไวยากรณ์ใหม่ หรือจัดให้มีตัววิเคราะห์คำ (อินพุตโปรแกรม เข้าในโปรแกรมวิเคราะห์คำ) เพื่อให้รู้จักรูปแบบที่คำสั่ง **yacc** ไม่รู้จัก

yacc grammar file declarations

ส่วนการประกาศของไฟล์ไวยากรณ์ **yacc** มีดังตอไปนี้:

- การประกาศตัวแปรหรือค่าคงที่ใดๆ ที่ใช้ในส่วนอื่นๆ ของ ไฟล์ไวยากรณ์
- คำสั่ง **#include** เพื่อใช้ไฟล์อื่นๆ เป็นส่วนหนึ่ง ในไฟล์นี้ (ใช้สำหรับไฟล์ส่วนหัวไลบรารี)
- คำสั่งที่กำหนดเงื่อนไขการประมวลผลสำหรับโปรแกรมวิเคราะห์คำที่สร้างขึ้น

คุณสามารถคงรักษาข้อมูลเกี่ยวกับความหมายที่สัมพันธ์กับ โทเค็นที่ขณะนี้อยู่บนสแต็กการวิเคราะห์คำในภาษา C *union* ที่ผู้ใช้กำหนดเอง ถ้าสมาชิกของ union ถูกเชื่อมโยงกับ ชื่อต่างๆ ในไฟล์ไวยากรณ์

การประกาศค่าตัวแปรหรือค่าคงที่ใช้ ไวยากรณ์ต่อไปนี้ของภาษาโปรแกรม C:

```
TypeSpecifier Declarator ;
```

TypeSpecifier คือคีย์เวิร์ด ชนิดข้อมูลและ *Declarator* คือชื่อของตัวแปร หรือค่าคงที่ ชื่อสามารถมีความยาวเท่าใดก็ได้ และประกอบด้วยตัวอักษร จุด เครื่องหมายขีดเส้นใต้ และตัวเลข ชื่อไม่สามารถขึ้นต้นด้วยตัวเลข ตัวอักษรตัวพิมพ์ใหญ่และตัวพิมพ์เล็ก จะถือว่าต่างกัน

ชื่อเทอร์มินัล (หรือโทเค็น) สามารถประกาศโดยใช้การประกาศ **%token** และชื่อ nonterminal names สามารถประกาศโดยใช้การประกาศ **%type** การประกาศ **%type** ไม่จำเป็นสำหรับชื่อ nonterminal ชื่อ Nonterminal ถูกกำหนดโดยอัตโนมัติถ้าปรากฏทางด้านซ้ายของกฎอย่างน้อย หนึ่งข้อ โดยไม่มีการประาศชื่อใช้ส่วนการประกาศ คุณสามารถใช้ชื่อเป็นสัญลักษณ์ nonterminal ได้เท่านั้น คำสั่ง **#include** เหมือนกับไวยากรณ์ภาษา C และดำเนินงาน ฟังก์ชันเหมือนกัน

โปรแกรม **yacc** มีชุดของ คีย์เวิร์ดที่กำหนดเงื่อนไขการประมวลผลสำหรับโปรแกรมวิเคราะห์คำที่สร้างขึ้น แต่ละ คีย์เวิร์ดจะขึ้นต้นด้วย % (เครื่องหมายเปอร์เซ็นต์) ซึ่งตามด้วยโทเค็นหรือชื่อ nonterminal คีย์เวิร์ดเหล่านี้มีดังนี้:

คีย์เวิร์ด	คำอธิบาย
%left	ระบุโทเค็นซึ่งเกี่ยวข้องทางด้านซ้ายกับโทเค็นอื่นๆ
%nonassoc	ระบุโทเค็นซึ่งไม่เกี่ยวข้องกับโทเค็นอื่นๆ
%right	ระบุโทเค็นซึ่งเกี่ยวข้องทางด้านขวากับโทเค็นอื่นๆ
%start	ระบุชื่อ nonterminal สำหรับสัญลักษณ์เริ่มทำงาน
%token	ระบุชื่อโทเค็นที่คำสั่ง **yacc** ยอมรับ ประกาศชื่อโทเค็นทั้งหมดในส่วนการประกาศ
%type	ระบุประเภทของ nonterminals การตรวจสอบประเภทถูกดำเนินการเมื่อ construct นี้ถูกแสดง
%union	ระบุสแต็กค่า yacc เป็นที่รวมของค่าประเภทต่างๆ ที่ต้องการ โดยดีฟอลต์ ค่าที่ส่งกลับจะเป็นจำนวนเต็ม ผลของ construct นี้คือเพื่อให้มีการประกาศของ **YYSTYPE** โดยตรงจากอินพุต
%{ *โค้ด* %}	ทำสำเนา *โค้ด* ที่ระบุ ลงในไฟล์โค้ด construct นี้สามารถใช้เพื่อเพิ่มการประกาศภาษา C และ นิยามให้แก่ส่วนการประกาศ **หมายเหตุ:** สัญลักษณ์ %{ (เครื่องหมายเปอร์เซ็นต์ วงเล็บปีกกาซ้าย) และ %} (เครื่องหมาย เปอร์เซ็นต์ วงเล็บปีกกาขวา) ต้องแสดงบนบรรทัดเอง

คีย์เวิร์ด **%token, %left, %right** และ **%nonassoc** เป็นทางเลือกเพื่อสนับสนุนชื่อของ สมาชิก C union (ดังอธิบายโดย **%union**) ที่ชื่อ <*Tag*> (เครื่องหมายวงเล็บที่ล้อมรอบ ชื่อสมาชิก union) คีย์เวิร์ด **%type** ต้องการ <*Tag*> การใช้ <*Tag*> จะระบุว่า โทเค็นที่กำหนดชื่อบนบรรทัดจะเป็นชนิดภาษา C เดียวกับสมาชิก union ที่ <*Tag*> อ้างถึง ตัวอย่าง การประกาศ ต่อไปนี้ประกาศพารามิเตอร์ *Name* ที่จะเป็นโทเค็น:

```
%token [<Tag>] Name [Number] [Name [Number]]...
```

ถ้ามี <*Tag*> ชนิดภาษา C สำหรับโทเค็นทั้งหมดบนบรรทัดนี้จะถูกประกาศเป็นชนิดที่อ้างถึง โดย <*Tag*> ถ้าเป็นจำนวนเต็มบวก *Number* ตามด้วยพารามิเตอร์ *Name* ค่านั้น ถูกกำหนดให้เป็นโทเค็น

โทเค็นทั้งหมดบนบรรทัดเดียวกันมีระดับการนำหน้า และความสัมพันธ์เท่ากัน บรรทัดที่แสดงในไฟล์ตามลำดับการเพิ่ม การนำหน้า หรือการเชื่อม strength ตัวอย่าง ต่อไปนี้อธิบายการนำหน้า และความสัมพันธ์ของตัวดำเนินการคำนวณสี่ตัว:

```
%left '+' '-'
%left '*' '/'
```

+ (เครื่องหมายบวก) และ - (เครื่องหมายลบ) เป็นความสัมพันธ์จากด้านซ้าย และมีระดับการมาก่อน ต่ำกว่า * (เครื่องหมายดอกจัน) และ / (เครื่องหมายทับ) ซึ่งเป็นความสัมพันธ์จากด้านซ้ายเช่นกัน

การกำหนดตัวแปรโกลบอล

ในการกำหนดตัวแปรเพื่อใช้โดยการดำเนินการบางอย่างหรือทั้งหมด รวมถึงใช้โดยตัววิเคราะห์ค่า ให้รวมไว้ในการประกาศสำหรับตัวแปรเหล่านั้น ระหว่างสัญลักษณ์ %{ (เครื่องหมายเปอร์เซ็นต์ วงเล็บปีกกาซ้าย) และ %} (เครื่องหมายเปอร์เซ็นต์ วงเล็บปีกกาขวา) การประกาศ ที่รวมอยู่ภายในสัญลักษณ์เหล่านี้เรียกว่า *ตัวแปรโกลบอล* ตัวอย่าง ในการทำให้ตัวแปร **var** มีอยู่ในทุกส่วน ของทั้งโปรแกรม ให้ใช้รายการต่อไปนี้ในส่วนการประกาศ ของไฟล์ไวยากรณ์:

```
%{
int var = 0;
%}
```

เงื่อนไขการเริ่มทำงาน

โปรแกรมวิเคราะห์คำรู้จักสัญลักษณ์พิเศษที่เรียกสัญลักษณ์ *เริ่มต้น* สัญลักษณ์เริ่มต้นคือชื่อของกฎ ในส่วนกฎของไฟล์ไวยากรณ์ที่อธิบายโครงสร้างทั่วไปส่วนใหญ่ ของภาษาที่จะถูกวิเคราะห์คำ เนื่องจากเป็นโครงสร้างทั่วไปส่วนใหญ่ โปรแกรมวิเคราะห์คำเริ่มการวิเคราะห์จากบนลงไปของสตรีมอินพุตที่จุดนี้ ประกาศสัญลักษณ์เริ่มต้นในส่วนการประกาศโดยใช้คีย์เวิร์ด **%start** ถ้าคุณไม่ประกาศชื่อของ สัญลักษณ์เริ่มต้น โปรแกรมวิเคราะห์คำจะใช้ชื่อของกฎไวยากรณ์ใน ไฟล์ไวยากรณ์

ตัวอย่าง เมื่อมีการวิเคราะห์คำฟังก์ชันภาษา C โครงสร้างทั่วไปส่วนใหญ่สำหรับโปรแกรมวิเคราะห์คำที่จะทราบเป็นดังนี้:

```
main()
{
        code_segment
}
```

สัญลักษณ์เริ่มต้นชี้ไปที่กฎที่อธิบาย โครงสร้างนี้ กฎที่เหลือทั้งหมดในไฟล์อธิบายวิธีระบุ โครงสร้างระดับที่ต่ำลงในภายในฟังก์ชัน

หมายเลขโทเค็น

หมายเลขโทเค็นคือเลขจำนวนเต็ม nonnegative ที่แทน ชื่อของโทเค็น ถ้าตัววิเคราะห์ค่าส่งค่าหมายเลขโทเค็นไปยัง โปรแกรมวิเคราะห์คำ แทนการส่งชื่อโทเค็นจริง ทั้งสองโปรแกรมต้องตกลง เกี่ยวกับหมายเลขที่กำหนดให้แก่โทเค็น

คุณสามารถกำหนดหมายเลขให้แก่โทเค็นที่ใช้ในไฟล์ไวยากรณ์ **yacc** ถ้าคุณไม่กำหนดหมายเลขให้แก่โทเค็น ไฟล์ไวยากรณ์ **yacc** จะกำหนดหมายเลขโดยใช้กฎ ต่อไปนี้:

- อักขระตามตัวอักษรเป็นค่า numerical ของอักขระในชุดอักขระ ASCII
- ชื่ออื่นๆ เป็นหมายเลขโทเค็นที่กำหนดเริ่มต้นที่ 257

  หมายเหตุ: อย่ากำหนดหมายเลขโทเค็นเป็น 0 หมายเลขนี้ถูกกำหนดให้แก่โทเค็น endmarker คุณไม่สามารถเปลี่ยนการกำหนดค่าได้

ในการกำหนดหมายเลขให้แก่โทเค็น (รวมถึงตัวอักษร) ในส่วนการประกาศของไฟล์ไวยากรณ์ให้ใส่เลขจำนวนเต็มบวก (ไม่ใช่ 0) ต่อท้ายชื่อโทเค็นในบรรทัด `%token` ทันที เลขจำนวนเต็มนี้คือหมายเลขโทเค็นของชื่อหรือค่าคงที่ใดๆ หมายเลข โทเค็นแต่ละค่าต้องเป็นค่าเฉพาะ ตัววิเคราะห์ค่าทั้งหมดที่ใช้กับคำสั่ง **yacc** ต้องส่งกลับค่า 0 หรือค่าลบสำหรับโทเค็นเมื่อ ถึงตำแหน่งสิ้นสุดของอินพุต

กฎ yacc

ส่วนกฎของไฟล์ไวยากรณ์มีกฎไวยากรณ์ อย่างน้อยหนึ่งข้อ กฎแต่ละข้ออธิบายโครงสร้างและตั้งชื่อ

กฎไวยากรณ์มีฟอร์มต่อไปนี้:

```
A : BODY;
```

โดยที่ A คือชื่อ nonterminal และ BODY คือลำดับของชื่ออย่างน้อย 0 ชื่อ ตัวอักษร และการดำเนินการเกี่ยวกับความหมายที่สามารถเลือกให้เป็นไปตามกฎ การมาก่อนได้ เฉพาะชื่อและตัวอักษรที่จำเป็นต้องฟอร์มไวยากรณ์ การดำเนินการ เกี่ยวกับความหมายและกฎการมาก่อนเป็นทางเลือก โคลอนและเซมิโคลอน ต้องการใช้เครื่องหมายวรรคตอน **yacc**

การดำเนินการเกี่ยวกับความหมายอนุญาตให้คุณเชื่อมโยงการดำเนินการ ที่จะดำเนินการในแต่ละครั้งที่กฎถูกรับรู้ในการประมวลผลอินพุต การดำเนินการ สามารถเป็นคำสั่ง C กำหนดเอง และอื่นๆ เช่น ดำเนินการอินพุตหรือเอาต์พุต การเรียกใช้โปรแกรมย่อย หรือปรับเปลี่ยนตัวแปรภายนอก การดำเนินการยังสามารถอ้างอิงไปยัง การดำเนินการของโปรแกรมวิเคราะห์คำ ตัวอย่างเช่น การเลื่อนและการลด

กฎการมาก่อนถูกกำหนดโดยคีย์เวิร์ด **%prec** และเปลี่ยนระดับการมาก่อนที่เชื่อมโยงกับ กฎไวยากรณ์เฉพาะ สัญลักษณ์ที่ถูกสงวนไว้ **%prec** สามารถแสดงทันทีหลังจากเนื้อหาของ กฎไวยากรณ์ และสามารถตามด้วยชื่อโทเค็นหรือค่าคงที่ใดๆ construct เป็นเหตุในการมาก่อนของกฎไวยากรณ์กลายเป็นของชื่อโทเค็น หรือค่าคงที่

การทวนชื่อที่ไม่ใช้เทอร์มินัล

ถ้าหลายกฎไวยากรณ์มีชื่อ nonterminal เดียวกัน ให้ใช้ | (สัญลักษณ์ไพพ์) เพื่อเลี่ยงการเขียน ด้านซ้ายซ้ำ นอกจากนั้น ใช้ ; (เซมิโคลอน) เฉพาะที่ท้ายของกฎทั้งหมดโดยรวมด้วยสัญลักษณ์ไพพ์ ตัวอย่าง กฎไวยากรณ์ ต่อไปนี้:

```
A  :  B  C  D  ;
A  :  E  F  ;
A  :  G  ;
```

สามารถกำหนดให้แก่คำสั่ง **yacc** โดยใช้สัญลักษณ์ ไพพ์ดังนี้:

```
A  :  B  C  D
   |  E  F
   |  G
   ;
```

การใช้การเรียกซ้ำในไฟล์ไวยากรณ์

การเรียกซ้ำ คือกระบวนการของ การใช้ฟังก์ชันเพื่อกำหนดตัวเอง ในนิยามของภาษา กฎเหล่านี้โดยทั่วไป มีฟอร์มต่อไปนี้:

```
rule    :       EndCase
        |       rule EndCase
```

ดังนั้น กรณีตัวอย่างง่ายที่สุดของ กฎ คือ *EndCase* แต่ กฎ ยังสามารถประกอบด้วยการเกิดเหตุการณ์ *EndCase* มากกว่าหนึ่งรายการในบรรทัดที่สองที่ใช้ กฎ ในนิยามของ กฎ คือการเรียกซ้ำ โปรแกรมวิเคราะห์คำหมุนรอบผ่านอินพุตจนกระทั่งสตรีมถูกลดจำนวนเหลือ *EndCase* สุดท้าย

เมื่อใช้การเรียกซ้ำในกฎ ให้ใส่การเรียก ไปยังชื่อของกฎเป็นรายการซ้ายสุดในกฎเสมอ (เหมือนที่อยู่ใน ตัวอย่างก่อนหน้า) ถ้าการเรียกใช้ไปยังชื่อของกฎเกิดขึ้นภายหลังใน บรรทัด เช่นในตัวอย่างต่อไปนี้ โปรแกรมวิเคราะห์อาจไม่เหลือพื้นที่สแต็ก ภายในและหยุดทำงาน

```
rule    :       EndCase
        |       EndCase rule
```

ตัวอย่างต่อไปนี้กำหนดกฎ บรรทัด เป็นการรวมกันอย่างน้อยหนึ่งค่าของสตริงตามด้วยอักขระ ขึ้นบรรทัดใหม่ (\n):

```
lines   :       line
        |       lines line
        ;

line    :       string '\n'
        ;
```

สตริงว่าง

ในการระบุสัญลักษณ์ nonterminal ที่ตรงกับ สตริงว่าง ให้ใช้ ; (เซมิโคลอน) ต่อท้าย เนื้อหาของกฎ ในการกำหนดสัญลักษณ์ว่าง ที่ ตรงกับสตริง ว่าง ให้ใช้กฎที่คล้ายกับ กฎต่อไปนี้:

```
empty   :  ;
        | x;
```

หรือ

```
empty   :
        | x
        ;
```

ตัวทำเครื่องหมายสิ้นสุดอินพุต

เมื่อตัววิเคราะห์คำถึงจุดสิ้นสุดสตรีมอินพุต จะส่งเครื่องหมายสิ้นสุดอินพุตไปยังโปรแกรมวิเคราะห์คำ เครื่องหมายนี้เป็นโทเค็น พิเศษที่ชื่อ *endmarker* ซึ่งมีค่าโทเค็นเป็น 0 เมื่อโปรแกรมวิเคราะห์คำได้รับเครื่อหมายสิ้นสุดอินพุต จะตรวจสอบเพื่อดูว่า ได้กำหนดอินพุตทั้งหมดให้แก่กฎไวยากรณ์หรือไม่ และได้ประมวลผล ฟอร์มอินพุตทั้งหน่วย (ตามที่กำหนดในไฟล์ไวยากรณ์ **yacc**) ถ้าอินพุตเป็นทั้งหน่วย โปรแกรมวิเคราะห์คำจะหยุดทำงาน ถ้าอินพุต ไม่ใช่ทั้งหน่วย โปรแกรมวิเคราะห์คำจะส่งสัญญาณแจ้งข้อผิดพลาดและหยุดทำงาน

ตัววิเคราะห์คำต้องส่งเครื่องหมายสิ้นสุดอินพุต ในเวลาที่เหมาะสม เช่นเมื่อสิ้นสุดไฟล์ หรือสิ้นสุดเร็กคอร์ด

แอ็คชัน yacc

ด้วยกฎไวยากรณ์แต่ละกฎ คุณสามารถระบุการดำเนินการที่ต้องการดำเนินการ ในแต่ละครั้งที่ parser จดจำกฎในอินพุตสตรีม การดำเนินการ คือคำสั่งในภาษา C ที่ทำอินพุตและเอาต์พุต เรียกโปรแกรมย่อย และเปลี่ยนเวกเตอร์และตัวแปรภายนอก

การดำเนินการส่งคืนค่า และขอรับค่าที่ส่งคืนโดยการดำเนินการก่อนหน้านี้ ตัววิเคราะห์ lexical ยังสามารถส่งคืนค่าสำหรับโทเค็นได้

ระบุการดำเนินการในไฟล์ไวยากรณ์ด้วยคำสั่งตั้งแต่หนึ่งคำสั่งขึ้นไปโดยห้อมล้อมด้วย {} (เครื่องหมายวงเล็บปีกกา) ตัวอย่างโปรแกรมต่อไปนี้ คือกฎไวยากรณ์พร้อมกับการดำเนินการ:

```
A  :  '('B')'
   {
     hello(1, "abc" );
   }
```

AND

```
XXX  :  YYY  ZZZ
     {
     printf("a message\n");
     flag = 25;
     }
```

การส่งค่าระหว่างการดำเนินการ

หากต้องการขอรับค่าที่สร้างโดยการดำเนินการอื่น การดำเนินการสามารถใช้คีย์เวิร์ดพารามิเตอร์ **yacc** ที่ขึ้นต้นด้วยเครื่องหมายดอลล่าร์ ($1, $2, ...) คีย์เวิร์ดเหล่านี้จะอ้างถึงค่าที่ส่งคืนโดยคอมโพเนนต์ที่อยู่ด้านขวาของกฎ ซึ่งอ่านจากซ้ายไปขวา ตัวอย่างเช่น ถ้ากฎคือ:

```
A  :  B  C  D  ;
```

ดังนั้น $1 จะมีค่าที่ส่งคืนโดยกฎที่ B จดจำไว้ $2 จะมีค่าที่ส่งคืนโดยกฎที่ C จดจำไว้ และ $3 จะมีค่าที่ส่งคืนโดยกฎที่ D จดจำไว้

หากต้องการส่งคืนค่า การดำเนินการจะตั้งค่าตัวแปรจำลอง $$ ให้มีค่าบางค่า ตัวอย่างเช่น การดำเนินการต่อไปนี้ จะส่งคืนค่าของ 1:

```
{ $$ = 1;}
```

ตามค่าดีฟอลต์แล้ว ค่าของกฎจะมีค่าขององค์ประกอบแรกที่อยู่ในกฎนั้น ($1) ดังนั้น คุณไม่จำเป็นต้องจัดเตรียมการดำเนินการสำหรับกฎที่มีรูปแบบต่อไปนี้:

```
A : B ;
```

คีย์เวิร์ดพารามิเตอร์ **yacc** ที่ขึ้นต้นด้วย $ (เครื่องหมายดอลล่าร์) จะอนุญาตให้ตรวจสอบชนิดต่อไปนี้:

- $<*Tag*>$
- $<*Tag*>*Number*

$<*Tag*>*Number* กำหนดบน การอ้างอิงชนิดของสมาชิกแบบ union ที่อ้างถึงโดย <*Tag*> ซึ่งจะเพิ่ม ***.tag*** ให้กับการอ้างอิง ดังนั้น สมาชิกแบบ union จะถูกระบุโดย *Tag* ซึ่งถูกเข้าถึง การสร้างนี้จะเทียบเท่ากับการระบุ $$. *Tag* หรือ $1 . *Tag* คุณสามารถใช้การสร้างนี้ เมื่อคุณใช้การดำเนินการที่จุดกึ่งกลางของกฎ ซึ่งชนิดการส่งคืนไม่สามารถระบุผ่าน **%type** declaration ได้ ถ้า **%type** ถูกประกาศสำหรับชื่อที่ไม่ใช่เทอร์มินัล ห้ามใช้การสร้าง <*Tag*> การอ้างอิงแบบ union จะถูกทำโดยอัตโนมัติ

การวางการดำเนินการในจุดกึ่งกลางของกฎ

หากต้องการขอรับการควบคุมกระบวนการวิเคราะห์คำ ก่อนที่กฎจะดำเนินการจนเสร็จสิ้น ให้บันทึกการดำเนินการลงในจุดกึ่งกลางของกฎ ถ้ากฎนี้ส่งคืนค่าผ่านคีย์เวิร์ด $ การดำเนินการที่ปฏิบัติตามกฎนี้ สามารถใช้ค่านั้นได้ กฎนี้ยังสามารถใช้ค่าที่ส่งคืนโดยการดำเนินการ ที่อยู่หน้ากฎนั้น ดังนั้น กฎต่อไปนี้จะตั้งค่า x ให้มีค่า 1 และตั้งค่า y ให้มีค่าที่ส่งคืนโดย C ค่าของกฎ A คือค่าที่ส่งคืนโดย B ตามค่าดีฟอลต์กฎต่อไปนี้

```
A  :  B
        {
           $$ =1;
        }
        C
   {
```

```
        x = $2;
        y = $3;
    }
    ;
```

ภายในโปรแกรม คำสั่ง **yacc** จะสร้างชื่อสัญลักษณ์ที่ไม่ใช่เทอร์มินัลใหม่สำหรับการดำเนินการที่เกิดขึ้นที่จุดกึ่งกลาง และยังสร้างกฎใหม่ที่ตรงกับชื่อนี้ให้กับสตริงว่าง ดังนั้น คำสั่ง **yacc** จะจัดการโปรแกรมที่อยู่ก่อนหน้านี้ หากโปรแกรมนี้ถูกเขียนไว้ในรูปแบบต่อไปนี้:

```
$ACT  :  /* empty */
      {
         $$ = 1;
      }
      ;
A     :  B  $ACT  C
      {
         x = $2;
         y = $3;
      }
      ;
```

โดยที่ $ACT การดำเนินการที่ว่าง

การจัดการกับข้อผิดพลาดของโปรแกรม yacc

เมื่อ parser อ่านอินพุตสตรีม อินพุตสตรีมนั้นอาจไม่ตรงกับกับกฎในไฟล์ไวยากรณ์

parser ตรวจพบปัญหาได้ก่อนเท่าที่จะเป็นไปได้ ถ้ามีรูทีนย่อยสำหรับการจัดการข้อผิดพลาดในไฟล์ไวยากรณ์แล้ว parser จะสามารถอนุญาตให้ป้อนข้อมูลได้อีกครั้ง โดยข้ามข้อมูลที่เสียหาย หรือเริ่มต้นการดำเนินการล้างข้อมูลและกู้คืน ตัวอย่างเช่น เมื่อ parser พบข้อผิดพลาด parser อาจต้องการเรียกคืนหน่วยเก็บแผนผังการวิเคราะห์คำ หรือเปลี่ยนแปลงรายการตารางสัญลักษณ์ และตั้งค่าให้สับเปลี่ยน เพื่อหลีกเลี่ยงการสร้างเอาต์พุตในอนาคต

เมื่อเกิดข้อผิดพลาดขึ้น parser จะหยุดทำงาน ยกเว้นเสียแต่คุณจะจัดเตรียมรูทีนย่อยการจัดการข้อผิดพลาดไว้ หากต้องการดำเนินการประมวลผลอินพุต เพื่อค้นหาข้อผิดพลาดเพิ่มเติม ให้รีสตาร์ท parser ที่จุดในสตรีมอินพุตที่ parser สามารถลองจัดการกับอินพุตเพิ่มเติมได้ใหม่ วิธีการหนึ่งในการรีสตาร์ท parser เมื่อเกิดข้อผิดพลาดคือ การละทิ้งโทเค็นบางตัวที่ตามหลังข้อผิดพลาด จากนั้น ลองรีสตาร์ท parser ที่จุดในสตรีมอินพุต

คำสั่ง **yacc** จะใช้ชื่อโทเค็นพิเศษ นั่นคือ **error** สำหรับการจัดการข้อผิดพลาด วางโทเค็นนี้ลงในไฟล์กฎ ที่ตำแหน่งที่ข้อผิดพลาดอินพุตอาจเกิดขึ้น ดังนั้น คุณจึงสามารถจัดเตรียมรูทีนย่อยสำหรับกู้คืนได้ ถ้าข้อผิดพลาดอินพุตนี้เกิดขึ้นในตำแหน่งนี้ parser จะเรียกใช้งานการดำเนินการสำหรับโทเค็น **error** แทนการเรียกใช้งานการดำเนินการปกติ

แมโครต่อไปนี้สามารถวางไว้ได้ในการดำเนินการ **yacc** เพื่อช่วยจัดการข้อผิดพลาด:

แมโคร	คำอธิบาย
YYERROR	สาเหตุที่ parser เริ่มต้นการจัดการข้อผิดพลาด
YYABORT	สาเหตุที่ parser ส่งคืนค่า 1
YYACCEPT	สาเหตุที่ parser ส่งคืนค่า 0
YYRECOVERING()	ส่งคืนค่า 1 ถ้าตรวจพบข้อผิดพลาดทางไวยากรณ์ และ parser ไม่ได้กู้คืนอย่างเต็มรูปแบบ

หากต้องการป้องกันข้อผิดพลาดเดี่ยวนี้จากการสร้างข้อความแสดงความผิดพลาดจำนวนมาก parser ยังคงอยู่ในสถานะข้อผิดพลาด จนกว่าจะประมวลผลโทเค็นสามตัวตามหลังข้อผิดพลาด ถ้าเกิดข้อผิดพลาดอื่นขึ้น ขณะที่ parser อยู่ในสถานะข้อผิดพลาด parser จะละทิ้งโทเค็นอินพุต และไม่ได้สร้างข้อความใดๆ

ตัวอย่างเช่น กฎของคำสั่งในรูปแบบต่อไปนี้:

```
stat  :  error ';'
```

บอกให้ parser ที่มีข้อผิดพลาดทราบ ซึ่งควรละเว้นโทเค็น และโทเค็นที่ตามหลังทั้งหมดจนกว่าจะพบเซมิโคลอนถัดไป โทเค็นทั้งหมดที่อยู่หลังข้อผิดพลาดและอยู่ก่อนเซมิโคลอนถัดไปจะถูกละทิ้ง หลังจากที่ค้นหาเซมิโคลอนแล้ว parser จะลดกฎนี้ลง และดำเนินการล้างข้อมูลใดๆ ที่เชื่อมโยงกับ parser

การจัดเตรียมสำหรับการแก้ไขข้อผิดพลาด

คุณยังสามารถอนุญาตให้บุคคลที่ป้อนสตรีมอินพุตในสภาวะแวดล้อมแบบโต้ตอบ แก้ไขข้อผิดพลาดของอินพุตใดๆ โดยป้อนบรรทัดใน data stream อีกครั้ง ต่อไปนี้คือตัวอย่างที่แสดงหนึ่งในวิธีที่สามารถกระทำได้

```
input : error '\n'
        {
          printf(" Reenter last line: " );
         }
         input
       {
         $$ = $4;
       }
       ;
```

อย่างไรก็ตาม ในตัวอย่างนี้ parser จะพักอยู่ในสถานะข้อผิดพลาดสำหรับโทเค็นอินพุตสามตัวที่ตามหลังข้อผิดพลาด ถ้าบรรทัดที่แก้ไข มีข้อผิดพลาดอยู่ในโทเค็นสามตัวแรก parser จะลบโทเค็น และไม่สร้างข้อความใดๆ หากต้องการอนุญาตให้ใช้เงื่อนไขนี้ ให้ใช้คำสั่ง **yacc** ดังต่อไปนี้:

```
yyerrok;
```

เมื่อ parser ค้นพบคำสั่งนี้ parser จะปล่อยให้อยู่ในสถานะข้อผิดพลาด และเริ่มต้นการประมวลผลตามปกติ ตัวอย่างการกู้คืนข้อผิดพลาดจึงเป็น :

```
input : error '\n'
        {
          yyerrok;
          printf(" Reenter last line: " );
         }
         input
       {
         $$ = $4;
       }
         ;
```

การล้างโทเค็น look-ahead

โทเค็น look-ahead คือโทเค็นถัดไปที่ parser ตรวจสอบ เมื่อข้อผิดพลาดเกิดขึ้น โทเค็น look-ahead จะกลายเป็นโทเค็นที่ตรวจพบข้อผิดพลาด อย่างไรก็ตาม ถ้าการดำเนินการกู้คืนข้อผิดพลาดสอดแทรกโค้ดเพื่อค้นหาตำแหน่งที่ถูกต้องในการเริ่มต้นการประมวลผลอีกครั้ง ซึ่งโค้ดต้องเปลี่ยนโทเค็น look-ahead ด้วยเช่นกัน หากต้องการล้างข้อมูลโทเค็น look-ahead ให้สอดแทรกคำสั่งต่อไปนี้ในการดำเนินการกู้คืนข้อผิดพลาด:

```
yyclearin ;
```

การดำเนินการตัววิเคราะห์คำที่สร้างด้วยคำสั่ง yacc

คำสั่ง **yacc** จะแปลงไฟล์ไวยากรณ์ไปเป็นโปรแกรมภาษา C

โปรแกรมนั้น เมื่อคอมไพล์หรือเรียกใช้งานแล้ว จะวิเคราะห์คำในอินพุตตามข้อกำหนดคุณสมบัติที่จัดเตรียมไว้

parser คือ finite state machine พร้อมกับสแต็ก parser สามารถอ่านและจดจำโทเค็น look-ahead สถานะปัจจุบันคือ สถานะที่อยู่ด้านบนสุดของสแต็กเสมอ สถานะของ finite state machine จะถูกแสดงโดยเลขจำนวนเต็มขนาดเล็ก ในครั้งแรก เครื่องจะอยู่ในสถานะ 0 สแต็กจะมี 0 เท่านั้น และไม่มีโทเค็น look-ahead ที่ถูกอ่าน

เครื่องสามารถดำเนินการกับหนึ่งในการดำเนินการต่อไปนี้:

แอ็คชัน	คำอธิบาย
shift *State*	parser จะส่งสถานะปัจจุบันไปยังสแต็ก เปลี่ยนค่า *State* ให้เป็นสถานะปัจจุบัน และล้างข้อมูลโทเค็น look-ahead
reduce *Rule*	เมื่อ parser พบสตริงที่กำหนดไว้ด้วย *Rule* (หมายเลขกฎ) ในอินพุตสตรีมแล้ว parser จะแทนที่สตริงนั้นด้วย *Rule* ในเอาต์พุตสตรีม
accept	parser จะมองที่อินพุตทั้งหมดที่ตรงกับข้อกำหนดคุณสมบัติไวยากรณ์ และจดจำอินพุตเพื่อให้ตอบสนองความต้องการโครงสร้างในระดับสูงสุด (กำหนดด้วยสัญลักษณ์สตาร์ท) การดำเนินการนี้จะปรากฏขึ้นก็ต่อเมื่อโทเค็น look-ahead คือตัวทำเครื่องหมายสิ้นสุด และบ่งชี้ว่า parser ทำงานสำเร็จแล้ว
error	parser ไม่สามารถดำเนินการประมวลผลอินพุตสตรีมต่อไปได้ และยังคงจับคู่การดำเนินการนั้นกับกฎใดๆ ที่ได้กำหนดไว้ในข้อกำหนดคุณสมบัติไวยากรณ์ได้สำเร็จ อินพุตโทเค็นที่ parser มองหาพร้อมกับโทเค็น look-ahead ไม่สามารถติดตามด้วยสิ่งใดๆ ที่จะส่งผลทำให้อินพุตไม่ถูกต้อง parser จะรายงานข้อผิดพลาด และพยายามกู้คืนสถานการณ์และกลับสู่การวิเคราะห์คำต่อไป

parser จะดำเนินการกับการดำเนินการต่อไปนี้ในระหว่างขั้นตอนของกระบวนการ:

1. อ้างอิงจากสถานะปัจจุบัน parser จะตัดสินว่า ต้องการโทเค็น look-ahead เพื่อกำหนดการดำเนินการที่ต้องทำ ถ้า parser ต้องการโทเค็น look-ahead และโทเค็นนั้นไม่มีอยู่ parser จะเรียกรูทีนย่อย **yylex** เพื่อขอรับโทเค็นถัดไป
2. การใช้สถานะปัจจุบันและโทเค็น look-ahead หากต้องการ parser จะตัดสินการดำเนินการถัดไปและจะเริ่มดำเนินการด้วยเช่นกัน จากผลลัพธ์ที่ได้ สถานะอาจส่งไปยังสแต็กอย่างรวดเร็ว และโทเค็น look-ahead อาจดำเนินการหรือปล่อยไว้ตามลำพัง

การดำเนินการ Shift

การดำเนินการ **shift** เป็นการดำเนินการทั่วไปที่ parser ใช้เป็นส่วนใหญ่ เมื่อไรก็ตามที่ parser ทำการเลื่อน โทเค็น look-ahead จะยังคงมีอยู่เสมอ ตัวอย่างเช่น ให้พิจารณากฎของข้อกำหนดคุณสมบัติของไวยากรณ์ต่อไปนี้ :

```
IF shift 34
```

ถ้า parser อยู่ในสถานะที่มีกฎนี้ และโทเค็น look-ahead คือ `IF` parser:

1. ส่งสถานะปัจจุบันไปยังสแต็ก

2. เปลี่ยนสถานะ 34 ให้เป็นสถานะปัจจุบัน (วางไว้ที่ด้านบนสุดของสแต็ก
3. ล้างข้อมูลโทเค็น look-ahead

การดำเนินการ Reduce

การดำเนินการ **reduce** จะคงสแต็กไว้จากการโตที่มีขนาดใหญ่เกินไป parser จะลดการดำเนินการหลังจากจับคู่ด้านซ้ายของกฎกับอินพุตสตรีม จากนั้น parser พร้อมที่จะแทนที่อักขระในอินพุตสตรีม ด้วยด้านซ้ายของกฎ parser อาจต้องใช้โทเค็น look-ahead เพื่อตัดสินว่า รูปแบบตรงกันทั้งหมด

การดำเนินการ reduce จะถูกเชื่อมโยงกับกฎไวยากรณ์แต่ละกฎ เนื่องจากกฎไวยากรณ์ยังคงมีเลขจำนวนเต็มขนาดเล็ก ซึ่งง่ายต่อความสับสนในความหมายของเลขจำนวนเต็มในสองการดำเนินการ นั่นคือ **shift** และ **reduce** ตัวอย่างเช่น การดำเนินการต่อไปนี้จะอ้างถึงกฎไวยากรณ์ 18:

```
. reduce 18
```

การดำเนินการต่อไปนี้จะอ้างถึงสถานะ 34:

```
IF shift 34
```

ตัวอย่างเช่น หากต้องการลดกฎต่อไปนี้ลง parser จะแสดงสามสถานะจากสแต็กในทันที:

```
A  :  x y z ;
```

หมายเลขของสถานะที่แสดงจะเท่ากับหมายเลขของสัญลักษณ์ ทางด้านขวาของกฎ สถานะเหล่านี้คือสถานะที่วางอยู่บนสแต็กขณะที่จดจำ x, y และ z หลังจากที่แสดงสถานะเหล่านี้แล้ว สถานะจะไม่ถูกครอบคลุม ซึ่งเป็นสถานะที่ parser เป็นอยู่ก่อนเริ่มต้นการประมวลผลกฎ นั่นคือ สถานะที่จำเป็นต่อการจดจำกฎ A เพื่อให้เป็นไปตามความต้องการของกฎ การใช้สถานะที่ไม่ครอบคลุมถึงนี้ และสัญลักษณ์ทางด้านซ้ายของกฎ parser จะดำเนินการกับการดำเนินการที่เรียก **goto** ซึ่งคล้ายกับ shift ของ A สถานะใหม่จะถูกรับ ส่งไปยังสแต็ก และวิเคราะห์คำต่อไป

การดำเนินการ **goto** จะแตกต่างจาก shift ของโทเค็น โทเค็น look-ahead จะถูกล้างข้อมูลด้วย shift แต่จะไม่ได้รับผลกระทบจากการดำเนินการ **goto** เมื่อสถานะทั้งสามถูกแสดง สถานะที่ไม่ครอบคลุมจะมีรายการดังต่อไปนี้:

```
A  goto 20
```

รายการนี้เป็สาเหตุทำให้สถานะ 20 ถูกส่งไปยังสแต็ก และกลายเป็นสถานะปัจจุบัน

การดำเนินการ reduce ยังคงเป็นการดำเนินการที่สำคัญ ในการดูแลการดำเนินการและค่าที่ผู้ใช้ระบุ เมื่อกฎถูกลดลง parser จะเรียกใช้งานโค้ดที่คุณสอดแทรกอยู่ในกฎ ก่อนที่จะปรับสแต็ก สแต็กอื่น ที่รันขนานกับสแต็กที่พักไว้ในสถานะพักค่า จะถูกส่งคืนจากตัววิเคราะห์ lexical และการดำเนินการ เมื่อ shift เข้าแทนที่ ตัวแปรภายนอก *yylval* จะถูกคัดลอกไปยังสแต็กที่พักค่าอยู่ หลังจากที่เรียกใช้งานโค้ดที่คุณจัดเตรียมไว้แล้ว parser จะดำเนินการลดลง เมื่อ parser ยืนยันการดำเนินการ **goto** แล้ว parser จะคัดลอกตัวแปรภายนอก *yylval* ไปยังค่าสแต็ก คีย์เวิร์ด **yacc** ที่ขึ้นต้นด้วย $ จะอ้างถึงค่าสแต็กนี้

การใช้กฎที่คลุมเครือในโปรแกรม yacc

ชุดของกฎไวยากรณ์จะ *กำกวม* ถ้าสามารถสร้างอินพุตสตริง ได้ตั้งแต่สองวิธีขึ้นไป

ตัวอย่างเช่น กฎไวยากรณ์ต่อไปนี้จะกล่าวถึงกฎที่มีรูปแบบนิพจน์เชิงคำนวณ โดยการใส่ค่านิพจน์อื่นๆ สองนิพจน์พร้อมกับเครื่องหมายลบ ระหว่างนิพจน์เหล่านั้น

```
expr : expr '-' expr
```

แต่น่าเสียเดียวที่กฎไวยากรณ์นี้ ไม่ได้ระบุวิธีการจัดโครงสร้างอินพุตที่ซับซ้อนทั้งหมด ตัวอย่างเช่น ถ้าอินพุตนี้คือ:

```
expr - expr - expr
```

โปรแกรมสามารถจัดโครงสร้างอินพุตนี้ตามด้านซ้ายที่มีการเชื่อมโยงกัน :

```
( expr - expr ) - expr
```

หรือด้านขวาที่มีการเชื่อมโยงกัน:

```
expr - ( expr - expr )
```

และสร้างผลลัพธ์ที่ต่างกัน

ข้อขัดแย้งของตัววิเคราะห์คำ

เมื่อ parser พยายามจัดการกับกฎที่กำกวม ความสับสนสามารถเกิดขึ้นได้เกินกว่าที่การดำเนินการทั้งสี่แบบสามารถดำเนินการได้ เมื่อประมวลผลอินพุต ชนิดของข้อขัดแย้งหลักต่อไปนี้จะได้รับการพัฒนา:

ความขัดแย้ง	คำอธิบาย
ข้อขัดแย้งแบบ shift/reduce	กฎสามารถนำมาประมวลผลได้อย่างถูกต้องโดยใช้การดำเนินการ **shift** หรือ **reduce** อย่างใดอย่างหนึ่ง แต่อาจมีผลลัพธ์ที่ต่างกัน
ข้อขัดแย้งแบบ reduce/reduce	กฎสามารถนำมาประมวลผลได้อย่างถูกต้องโดยใช้หนึ่งในสองการดำเนินการ reduce ที่ต่างกัน ซึ่งจะสร้างสองการดำเนินการที่ต่างกัน

ข้อขัดแย้งแบบ **shift/shift** ไม่สามารถเป็นไปได้ ข้อขัดแย้งแบบ **shift/reduce** และ **reduce/reduce** มีผลมาจากกฎที่ไม่ได้อยู่ในสถานะที่เสร็จสิ้นแล้ว ตัวอย่างเช่น การใช้กฎที่กำกวมจะถูกกล่าวถึงในส่วนก่อนหน้านี้ หาก parser ได้รับอินพุต:

```
expr - expr - expr
```

หลังจากที่อ่านส่วนสามส่วนแรกแล้ว parser จะมี:

```
expr - expr
```

ซึ่งตรงกับด้านขวาของกฎไวยากรณ์ที่อยู่ก่อนหน้า parser สามารถลดอินพุตได้โดยใช้กฎนี้ หลังจากที่ใช้กฎแล้ว อินพุตจะเป็น:

```
expr
```

ซึ่งจะอยู่ด้านซ้ายของกฎ จากนั้น parser จะอ่านส่วนท้ายสุดของอินพุต:

```
- expr
```

และลดอินพุตลง ซึ่งจะสร้างการตีความทางด้านซ้ายที่มีการเชื่อมโยงกัน

อย่างไรก็ตาม parser ยังมองไปถึงอินพุตสตรีม เมื่อ parser ได้รับส่วนสามส่วนแรก:

```
expr - expr
```

parser จะอ่านอินพุตสตรีมจนกว่าจะมีส่วนสองส่วนถัดไป จากนั้น จะมีอินพุตต่อไปนี้:

```
expr - expr - expr
```

การใช้กฎกับส่วนสามส่วนที่อยู่ด้านขวาสุดจะลดส่วนใน `expr` จากนั้น parser จะมีนิพจน์:

```
expr - expr
```

การลดนิพจน์ลงหนึ่งนิพจน์จะสร้างการตีความทางด้านขวา ที่มีการเชื่อมโยงกัน

ดังนั้น ณ จุดที่ parser อ่านเฉพาะสามส่วนแรก parser จะใช้การดำเนินการหนึ่งในสองการดำเนินการที่ถูกต้อง: **shift** หรือ **reduce** ถ้า parser ไม่มีกฎที่ต้องตัดสินระหว่างสองการดำเนินการ ข้อขัดแย้ง **shift/reduce** จะเกิดขึ้น

สถานการณ์ที่คล้ายกันนี้จะเกิดขึ้น หาก parser สามารถเลือกได้ระหว่างการดำเนินการ reduce ที่ถูกต้องสองการดำเนินการ ซึ่งเรียกว่า *ข้อขัดแย้งแบบ reduce/reduce*

วิธีที่ตัววิเคราะห์คำตอบสนองต่อข้อขัดแย้ง

เมื่อข้อขัดแย้ง **shift/reduce** หรือ **reduce/reduce** เกิดขึ้น คำสั่ง **yacc** จะสร้าง parser โดยเลือกหนึ่งในขั้นตอนที่ถูกต้องในทุกที่ที่มีตัวเลือก ถ้าคุณไม่ได้จัดเตรียมกฎที่สร้างตัวเลือกไว้ โปรแกรม **yacc** จะใช้กฎต่อไปนี้:

- ในข้อขัดแย้ง **shift/reduce** ให้เลือก shift
- ในข้อขัดแย้ง **reduce/reduce** ให้ลดกฎไวยากรณ์ลง ซึ่งสามารถใช้ได้ที่จุดเริ่มแรกในอินพุตสตรีม

การใช้การดำเนินการภายในกฎสามารถเป็นสาเหตุทำให้เกิดข้อขัดแย้งได้ ถ้าการดำเนินการต้องถูกดำเนินการก่อนที่ parser มั่นใจว่ากฎใดที่ถูกจดจำ ในกรณีเช่นนี้ กฎที่อยู่ก่อนหน้าส่งผลให้มี parser ที่ไม่ถูกต้อง สำหรับเหตุผลนี้ โปรแกรม **yacc** จะรายงานจำนวนข้อขัดแย้งแบบ **shift/reduce** และ **reduce/reduce** ที่แก้ปัญหาโดยการใช้กฎที่อยู่ก่อนหน้า

การเปิดโหมดดีบักสำหรับตัววิเคราะห์คำที่สร้างด้วยคำสั่ง yacc

คุณสามารถเข้าถึงโค้ดการดีบักได้โดยเรียกใช้งานคำสั่ง **yacc** พร้อมด้วยอ็อพชัน **-t** หรือคอมไพล์ไฟล์ **y.tab.c** ด้วย **-DYYDEBUG** อย่างใดอย่างหนึ่ง

สำหรับการดำเนินการปกติ ตัวแปรภายนอกสำหรับเลขจำนวนเต็ม *yydebug* ตั้งค่าเป็น 0 อย่างไรก็ตาม ถ้าคุณตั้งค่าตัวแปรนี้ให้มีค่าที่ไม่ใช่ศูนย์แล้ว parser จะสร้างคำอธิบายของโทเค็นอินพุตที่ได้รับ และการดำเนินการที่ใช้สำหรับแต่ละโทเค็น ขณะที่วิเคราะห์หาอินพุตสตรีม

ตั้งค่าตัวแปรนี้ด้วยหนึ่งในวิธีต่อไปนี้:

- วางข้อความภาษา C ต่อไปนี้ในส่วนที่มีการประกาศของไฟล์ไวยากรณ์ **yacc** :
  ```
  int yydebug = 1;
  ```
- ใช้โปรแกรม **dbx** เพื่อเรียกใช้งาน parser ล่าสุด และตั้งค่าตัวแปรให้มีค่า เปิด หรือ ปิด โดยใช้คำสั่ง **dbx**

ตัวอย่างโปรแกรมสำหรับโปรแกรม lex และ yacc

ส่วนนี้มีโปรแกรมตัวอย่างสำหรับคำสั่ง **lex** และ **yacc**

อีกทั้ง ตัวอย่าง โปรแกรมเหล่านี้สร้างโปรแกรมเครื่องคิดเลขตั้งโต๊ะแบบง่ายที่ดำเนินการบวก การลบ การคูณ และการหาร โปรแกรมเครื่องคิดเลขนี้ ยังอนุญาตให้คุณกำหนดค่าให้กับตัวแปร (แต่ละค่าถูกกำหนดด้วยตัวอักษร ตัวพิมพ์เล็กตัวเดียว) จากนั้นใช้ตัวแปรในการคำนวณ ไฟล์ที่ มีโปรแกรมตัวอย่าง **lex** และ **yacc** มีดังนี้:

ไฟล์	เนื้อหา
calc.lex	ระบุไฟล์ข้อมูลจำเพาะคำสั่ง **lex** ที่กำหนดกฎการวิเคราะห์ศัพท์
calc.yacc	ระบุไฟล์ไวยากรณ์คำสั่ง **yacc** ที่ กำหนดกฎการวิเคราะห์คำ และเรียกใช้รูทีนย่อย **yylex** ที่สร้างโดยคำสั่ง **lex** เพื่อจัดเตรียมอินพุต

คำอธิบายต่อไปนี้สมมุติว่าโปรแกรมตัวอย่าง **calc.lex** และ **calc.yacc** อยู่ใน ไดเร็กทอรีปัจจุบันของคุณ

การคอมไพล์โปรแกรมตัวอย่าง

ในการสร้างโปรแกรมตัวอย่างเครื่องคิดเลขตั้งโต๊ะ ทำดังนี้:

1. ประมวลผลไฟล์ไวยากรณ์ **yacc** โดยใช้แฟล็กทางเลือก **-d** (ซึ่งแจ้งให้คำสั่ง **yacc** สร้างไฟล์ที่กำหนดโทเค็นที่ใช้นอกเหนือจากซอร์สโค้ด ภาษา C):

   ```
   yacc -d calc.yacc
   ```

2. ใช้คำสั่ง **ls** เพื่อตรวจสอบว่าไฟล์ต่อไปนี้ ถูกสร้าง:

   y.tab.c ไฟล์ต้นฉบับภาษา C ที่คำสั่ง **yacc** สร้างขึ้นสำหรับเป็นโปรแกรมวิเคราะห์คำ

   y.tab.h ไฟล์ส่วนหัวที่มีคำสั่ง define สำหรับโทเค็นที่ใช้โดย โปรแกรมวิเคราะห์คำ

3. ประมวลผลไฟล์ข้อมูลจำเพาะ **lex** :

   ```
   lex calc.lex
   ```

4. ใช้คำสั่ง **ls** เพื่อตรวจสอบว่าไฟล์ต่อไปนี้ ถูกสร้าง:

   lex.yy.c ไฟล์ต้นฉบับภาษา C ที่คำสั่ง **lex** สร้างขึ้นสำหรับเป็นตัววิเคราะห์คำ

5. คอมไพล์และลิงก์ไฟล์ต้นฉบับภาษา C สองไฟล์:

   ```
   cc y.tab.c lex.yy.c
   ```

6. ใช้คำสั่ง **ls** เพื่อตรวจสอบว่าไฟล์ต่อไปนี้ ถูกสร้าง:

   y.tab.o อ็อบเจ็กต์ไฟล์สำหรับไฟล์ต้นฉบับ **y.tab.c**

   lex.yy.o
   อ็อบเจ็กต์ไฟล์สำหรับไฟล์ต้นฉบับ **lex.yy.c**

   a.out ไฟล์ของโปรแกรมเรียกทำงาน

ในการทำงานโปรแกรมโดยตรงจากไฟล์ **a.out** พิมพ์:

```
$ a.out
```

OR

ในการย้ายโปรแกรมไปยังไฟล์ที่มีชื่ออธิบายมากขึ้น ดังในตัวอย่าง ต่อไปนี้ ให้รันโปรแกรม พิมพ์:

```
$ mv a.out calculate
$ calculate
```

ไม่ว่ากรณีใด หลังจากคุณเริ่มทำงานโปรแกรม เคอร์เซอร์จะย้ายไปที่บรรทัด ด้านล่าง $ (พร้อมต์คำสั่ง) จากนั้น ป้อนตัวเลขและตัวดำเนินการเหมือนที่คุณต้องการบนเครื่องคิดเลข เมื่อคุณกดปุ่ม Enter โปรแกรมจะแสดงผลลัพธ์จากการดำเนินการ หลังจากคุณกำหนดค่า ให้แก่ตัวแปร ดังนี้ เคอร์เซอร์จะย้ายไปที่บรรทัดถัดไป

```
m=4 <enter>
_
```

เมื่อคุณใช้ตัวแปรในการคำนวณภายหลัง ตัวแปรจะ มีค่าที่กำหนด:

```
m+5 <enter>
9
_
```

ซอร์สโค้ดโปรแกรมวิเคราะห์คำ

ตัวอย่างต่อไปนี้แสดงเนื้อหาของไฟล์ **calc.yacc** ไฟล์นี้มีรายการในส่วนสามส่วนทั้งหมด ของไฟล์ไวยากรณ์ **yacc**: declarations, rules และ programs

```
%{
#include <stdio.h>

int regs[26];
int base;

%}

%start list

%union { int a; }


%token DIGIT LETTER

%left '|'
%left '&'
%left '+' '-'
%left '*' '/' '%'
%left UMINUS  /*supplies precedence for unary minus */

%%                   /* beginning of rules section */

list:                       /*empty */
         |
        list stat '\n'
         |
        list error '\n'
         {
           yyerrok;
         }
         ;
stat:    expr
         {
           printf("%d\n",$1);
         }
         |
         LETTER '=' expr
         {
           regs[$1.a] = $3.a;
         }
```

```
        ;

expr:   '(' expr ')'
        {
          $$ = $2;
        }
        |
        expr '*' expr
        {

          $$.a = $1.a * $3.a;
        }
        |
        expr '/' expr
        {
          $$.a = $1.a / $3.a;
        }
        |
        expr '%' expr
        {
          $$.a = $1.a % $3.a;
        }
        |
        expr '+' expr
        {
          $$.a = $1.a + $3.a;
        }
        |
        expr '-' expr
        {
          $$.a = $1.a - $3.a;
        }
        |
        expr '&' expr
        {
          $$.a = $1.a & $3.a;
        }
        |
        expr '|' expr
        {
          $$.a = $1.a | $3.a;
        }
        |

       '-' expr %prec UMINUS
        {
          $$.a = -$2.a;
        }
        |
        LETTER
        {
          $$.a = regs[$1.a];
        }
```

```
        |
        number
        ;

number:  DIGIT
        {
          $$ = $1;
          base = ($1.a==0) ? 8 : 10;
        }       |
        number DIGIT
        {
          $$.a = base * $1.a + $2.a;
        }
        ;

%%
main()
{
 return(yyparse());
}

yyerror(s)
char *s;
{
  fprintf( stderr, "%s\n" ,s);
}

yywrap()
{
  return(1);
}
```

ไฟล์มีส่วนต่อไปนี้:

- **ส่วนการประกาศ** ส่วนนี้มีรายการ ที่:
  - รวมไฟล์ส่วนหัว I/O มาตรฐาน
  - กำหนดตัวแปรโกลบอล
  - กำหนดกฎ `list` เป็นตำแหน่งที่จะเริ่มทำ การประมวลผล
  - กำหนดโทเค็นที่ใช้โดยโปรแกรมวิเคราะห์คำ
  - กำหนดตัวดำเนินการและการมาก่อน
- **ส่วนของกฎ** ส่วนกฎกำหนดกฎ ที่ใช้วิเคราะห์คำอินพุตสตรีม
  - ***%start*** – กำหนดว่า อินพุตทั้งหมดควรตรงกับ **stat**
  - ***%union*** – โดยดีฟอลต์ ค่าที่ส่งกลับ โดยการดำเนินการและตัววิเคราะห์ lexical จะเป็นจำนวนเต็ม **yacc** สามารถรองรับค่าของชนิดอื่น รวมถึงโครงสร้าง นอกจากนี้ **yacc** ยังติดตามชนิด และแทรกชื่อสมาชิกยูเนียนที่เหมาะสม เพื่อให้ผลการวิเคราะห์จะเป็นไปตามการตรวจสอบชนิดข้อมูล สแต็กค่า **yacc** ถูกประกาศให้เป็นยูเนียนของชนิดต่างๆ ของค่าที่ต้องการ ผู้ใช้ประกาศยูเนียน และชื่อสมาชิกยูเนียนที่เกี่ยวข้องให้แก่ แต่ละโทเค็น และสัญลักษณ์ที่ไม่ใช่เทอร์มินัลมีค่า เมื่อค่าถูกอ้างอิงผ่านคอนสตรัคชัน **$$** หรือ **$n**, **yacc** จะแทรกชื่อยูเนียนที่เหมาะสม เพื่อไม่ให้เกิดการแปลงที่ไม่ต้องการ

- *%type* – ทำให้การใช้สมาชิกของการประกาศ **%union** และให้ชนิดจำเพาะ สำหรับค่าที่เกี่ยวข้องกับแต่ละส่วนของไวยากรณ์
- *%toksn* – แสดงรายการโทเค็นที่มาจากเครื่องมือ lex ตามชนิดข้อมูล

• **ส่วนของโปรแกรม** ส่วนโปรแกรมมี รูทีนย่อยต่อไปนี้ เนื่องจากรูทีนย่อยเหล่านี้ถูกรวมในไฟล์ คุณไม่จำเป็นต้องใช้ไลบรารี **yacc** เมื่อประมวลผล ไฟล์นี้

รูทีนย่อย	คำอธิบาย
main	โปรแกรมหลักที่จำเป็นซึ่งเรียกใช้รูทีนย่อย **yyparse** ให้เริ่มทำงานโปรแกรม
yyerror(s)	รูทีนย่อยการจัดการข้อผิดพลาดนี้จะพิมพ์ข้อความแสดงข้อผิดพลาดทางไวยากรณ์ทั้งหมด
yywrap	รูทีนย่อยสรุปที่ส่งกลับค่า 1 เมื่อเกิดการสิ้นสุด อินพุต

ซอร์สโค้ดโปรแกรมวิเคราะห์คำ

ไฟล์นี้มีคำสั่ง include สำหรับอินพุตและเอาต์พุต มาตรฐาน รวมถึงสำหรับไฟล์ **y.tab.h** ถ้า คุณใช้คำสั่ง **–d** flag with the **yacc** โปรแกรม **yacc** จะสร้างไฟล์นั้นจากข้อมูลไฟล์ไวยากรณ์ **yacc** ไฟล์ **y.tab.h** มีนิยามสำหรับโทเค็นที่โปรแกรมวิเคราะห์คำใช้ นอกจากนั้น ไฟล์ **calc.lex** มีกฎที่จะใช้สร้าง โทเค็นเหล่านี้จากอินพุตสตรีม ต่อไปนี้คือเนื้อหาของไฟล์ **calc.lex**

```
%{

#include <stdio.h>
#include "y.tab.h"
int c;
%}
%%
" "       ;
[a-z]     {
            c = yytext[0];
            yylval.a = c - 'a';
            return(LETTER);
          }
[0-9]     {
            c = yytext[0];
            yylval.a = c - '0';
            return(DIGIT);
          }
[^a-z0-9\b]    {
                 c = yytext[0];
                 return(c);
               }
%%
```

คำสั่ง make

หัวข้อนี้ให้ข้อมูลเกี่ยวกับการททำการคอมไพล์ซ้ำ และการลิงก์กระบวนการใหม่ให้ง่ายขึ้นโดยใช้คำสั่ง **make**

ซึ่งอนุญาตให้คุณบันทึกความสัมพันธ์ที่เจาะจงระหว่างไฟล์เหล่านั้นเพียงครั้งเดียวเท่านั้น คุณสามารถ ใช้คำสั่ง **make** เพื่อดำเนินการอัพเดตงาน ทั้งหมดโดยอัตโนมัติ

ในโปรเจ็กต์ใดๆ โดยปกติคุณลิงก์โปรแกรมจากอ็อบเจ็กต์ไฟล์และไลบรารี จากนั้น หลังทำการแก้ไขไฟล์ต้นฉบับ คุณคอมไพล์บางส่วนของซอร์สใหม่และ ลิงก์โปรแกรมใหม่ได้บ่อยเท่าที่ต้องการ คำสั่ง **make** ช่วยในการดูแลรักษาชุดของโปรแกรม ที่โดย

ปกติเป็นของซอฟต์แวร์โปรเจ็กต์ที่เฉพาะ โดยการสร้างเวอร์ชันล่าสุด ของโปรแกรม คำสั่ง **make** เป็นประโยชน์ยิ่งสำหรับโปรแกรมมิงโปรเจ็กต์ขนาดกลาง คำสั่งไม่ได้ช่วยแก้ปัญหาการดูแลรักษา เวอร์ชันของซอร์สที่มีมากกว่าหนึ่งเวอร์ชันและการอธิบายโปรแกรมขนาดใหญ่ (โปรดดูที่คำสั่ง **sccs**)

การใช้คำสั่ง **make** เพื่อรักษา โปรแกรม คุณสามารถ:

- รวมคำสั่งสำหรับการสร้างโปรแกรมขนาดใหญ่ในไฟล์เดียว
- กำหนดแมโครเพื่อใช้ภายในไฟล์รายละเอียดคำสั่ง **make**
- ใช้คำสั่งเชลล์เพื่อกำหนดวิธีการสร้างไฟล์ หรือใช้คำสั่ง **make** เพื่อสร้างประเภทระดับต้นของไฟล์หลายประเภท
- สร้างไลบรารี

คำสั่ง **make** ต้องการ ไฟล์รายละเอียด ชื่อไฟล์ กฎที่ระบุเพื่อแจ้งให้คำสั่ง **make** ทราบวิธีสร้างประเภทไฟล์มาตรฐานหลายประเภท และการประทับเวลา ของไฟล์ไฟล์ระบบทั้งหมด

หลักการที่เกี่ยวข้อง:

“เครื่องมือและยูทิลีตี” ในหน้า 2
ส่วนนี้ให้ภาพรวมของเครื่องมือและยูทิลิตีที่คุณสามารถใช้เพื่อพัฒนาโปรแกรมภาษาที่คอมไพล์ด้วย C

การสร้างไฟล์รายละเอียด

คำสั่ง **make** ใช้ข้อมูล จากไฟล์รายละเอียด ที่คุณสร้าง เพื่อสร้างไฟล์ที่มีโปรแกรม ทั้งหมด ซึ่งต่อไปจะเรียกว่าไฟล์ *เป้าหมาย*

ไฟล์รายละเอียดแจ้งคำสั่ง **make** ให้ทราบวิธี สร้างไฟล์เป้าหมาย ซึ่งไฟล์เกี่ยวข้อง และความสัมพันธ์ที่มี ไปยังไฟล์อื่นๆ ในโพรซีเดอร์ ไฟล์รายละเอียดมี ข้อมูลต่อไปนี้:

- ชื่อไฟล์เป้าหมาย
- ชื่อไฟล์พาเรนต์ที่ประกอบเป็นไฟล์เป้าหมาย
- คำสั่งที่สร้างไฟล์เป้าหมายจากไฟล์พาเรนต์
- ข้อกำหนดของแมโครในไฟล์รายละเอียด
- กฎที่ผู้ใช้กำหนดสำหรับการสร้างไฟล์เป้าหมาย

โดยการตรวจสอบวันที่ของไฟล์พาเรนต์ คำสั่ง **make** พิจารณาว่าไฟล์ใดที่สร้างเพื่อรับ สำเนาล่าสุดของไฟล์เป้าหมาย ถ้าไฟล์พาเรนต์ใดๆ ถูกเปลี่ยนแปลงล่าสุดกว่า ไฟล์เป้าหมาย คำสั่ง **make** จะสร้างไฟล์ที่ได้รับผลจาก การเปลี่ยนแปลง รวมถึงไฟล์เป้าหมาย

ถ้าคุณตั้งค่าไฟล์รายละเอียดเป็น **makefile** หรือ **Makefile** และกำลังทำงานในไดเร็กทอรี ที่มีไฟล์รายละเอียด ให้พิมพ์คำสั่งต่อไปนี้เพื่ออัดเดตไฟล์เป้าหมายแรก และไฟล์พาเรนต์ของเป้าหมาย:

```
make
```

การอัพเดตเกิดขึ้นไม่ว่าจำนวนไฟล์ที่ถูกเปลี่ยนแปลง ตั้งแต่ครั้งล่าสุดที่คำสั่ง **make** ได้สร้าง ในกรณีส่วนใหญ่ ไฟล์รายละเอียดเขียนได้ง่าย และ ไม่เปลี่ยนแปลงบ่อย

ในการเก็บรักษาไฟล์รายละเอียดที่แตกต่างกันหลายๆ ไฟล์ในไดเร็กทอรี เดียวกัน ให้ตั้งชื่อให้ต่างกัน ดังนั้น ป้อน:

```
make -f Desc-File
```

โดยที่ *Desc-File* คือชื่อ ของไฟล์รายละเอียด

รูปแบบของรายการในไฟล์รายละเอียด make

รูปแบบทั่วไปของรายการคือ:

```
target1 [target2..]:[:] [parent1..][; command]...
[(tab) commands]
```

รายการภายในวงเล็บปีกกาเป็นทางเลือก เป้าหมายและพาเรนต์ เป็นชื่อไฟล์ (สตริงตัวอักษร ตัวเลข เครื่องหมายจุด และเครื่องหมายทับ) คำสั่ง **make** รู้จักอักขระ wildcard เช่น * (เครื่องหมายดอกจัน) และ ? (เครื่องหมายคำถาม) แต่ละบรรทัดในไฟล์รายละเอียดที่มีชื่อไฟล์เป้าหมาย ถูกเรียกว่า *บรรทัด dependency* บรรทัดที่ มีคำสั่งต้องเริ่มต้นด้วยอักขระแท็บ

> **หมายเหตุ:** คำสั่ง **make** ใช้ **$** (เครื่องหมายดอลล่าร์) เพื่อกำหนดแมโคร อย่าใช้ อักขระนั้นในชื่อไฟล์ของไฟล์เป้าหมาย หรือพาเรนต์ หรือในคำสั่ง ในไฟล์รายละเอียดเว้นแต่คุณกำลังใช้แมโครคำสั่ง **make** ที่กำหนดไว้แล้ว

ขึ้นต้นความคิดเห็นในไฟล์รายละเอียดด้วย # (เครื่องหมาย ปอนด์) คำสั่ง **make** ละเว้นอักขระ # และอักขระทั้งหมด ที่ตามหลังเครื่องหมายนั้น คำสั่ง **make** ยังละเว้นบรรทัด ว่าง

ยกเว้นบรรทัดความคิดเห็น คุณสามารถป้อนบรรทัดให้ยาว กว่าความยาวบรรทัดของอุปกรณ์อินพุต ในการไปที่บรรทัดบนบรรทัดถัดไป ให้ใส่\(backslash) ที่ท้ายบรรทัดเพื่อแสดงว่ามีต่อ

การใช้คำสั่งในไฟล์รายละเอียด make

คำสั่งคือสตริงของอักขระใดๆ ยกเว้น # (เครื่องหมาย ปอนด์) หรืออักขระขึ้นบรรทัดใหม่ คำสั่งสามารถใช้ # ต่อเมื่ออยู่ในเครื่องหมาย คำพูด

คำสั่งสามารถแสดงหลังเครื่องหมายเซมิโคลอนบนบรรทัด dependency หรือ บนบรรทัดที่เริ่มต้นด้วยแท็บที่ตามด้วยบรรทัด dependency ทันที

เมื่อทำการกำหนดลำดับคำสั่งสำหรับเป้าหมายที่เจาะจง ให้ระบุลำดับคำสั่งหนึ่งสำหรับแต่ละเป้าหมายในไฟล์รายละเอียด หรือให้แยกลำดับคำสั่งสำหรับชุดการขึ้นต่อกันพิเศษ อย่า ใช้ทั้งสอง

ในการใช้ลำดับคำสั่งหนึ่งสำหรับทุกๆ การใช้ไฟล์เป้าหมาย ให้ใช้ : (โคลอน) เดียวตามหลังชื่อเป้าหมายบนบรรทัด dependency ตัวอย่างเช่น:

```
test:       dependency list1...
        command list...
           .
           .
           .
test:       dependency list2...
```

กำหนดชื่อเป้าหมาย **test** ที่มีชุด ของไฟล์พาเรนต์และชุดของคำสั่งเพื่อสร้างไฟล์ ชื่อเป้าหมาย **test** สามารถแสดงในที่อื่นในไฟล์รายละเอียดที่มีรายการ dependency อื่น อย่างไรก็ตาม ชื่อนั้นไม่สามารถมีรายการคำสั่งอื่นใน ไฟล์รายละเอียด เมื่อไฟล์หนึ่งของไฟล์ที่ **test** ขึ้นต่อการเปลี่ยนแปลง คำสั่ง **make** รันคำสั่งในหนึ่งรายการคำสั่ง เพื่อสร้างไฟล์เป้าหมายชื่อ **test**

ในการระบุมากกว่าหนึ่งชุดของคำสั่งเพื่อสร้าง ไฟล์เป้าหมายเฉพาะ ให้ป้อนนิยาม dependency มากกว่าหนึ่งนิยาม แต่ละ dependency line ต้องมีชื่อเป้าหมาย ตามด้วย :: (สองโคลอน) รายการ dependency และรายการคำสั่งที่คำสั่ง **make** ใช้ถ้าไฟล์ใดๆ ในรายการ dependency เปลี่ยนแปลง ตัวอย่างเช่น:

```
test::      dependency list1...
         command list1...
test::      dependency list2...
         command list2...
```

กำหนดการประมวลผลสองตัวแยกกัน เพื่อสร้างไฟล์ เป้าหมาย **test** ถ้ามีไฟล์ใดๆ ใน dependency `list1` เปลี่ยนแปลง คำสั่ง **make** จะรัน command `list1` ถ้ามีไฟล์ใดๆ ใน dependency `list2` เปลี่ยนแปลง คำสั่ง **make** จะรัน command `list2` เพื่อเลี่ยงความขัดแย้ง ไฟล์พาเรนต์ไม่สามารถปรากฏในทั้ง dependency `list1` และ dependency `list2`

> **หมายเหตุ:** คำสั่ง **make** ส่งคำสั่งจากแต่ละบรรทัดบรรทัดคำสั่งไปยังเชลล์ใหม่ ขอให้ระมัดระวังเมื่อ ใช้คำสั่งที่มีความหมายเฉพาะภายในการประมวลผลเชลล์อย่างเดียว ตัวอย่าง **cd** และคำสั่งเชลล์ คำสั่ง **make** ละเว้นผลลัพธ์เหล่านี้ก่อนการรันคำสั่งบนบรรทัดถัดไป

ในการจัดกลุ่มคำสั่งเข้าด้วยกัน ใช้\(backslash) ที่ท้ายบรรทัดคำสั่ง จากนั้นคำสั่ง **make** ดำเนินการต่อจากบรรทัดนั้นไปยังบรรทัดถัดไปในไฟล์รายละเอียด เชลล์ ส่งทั้งสองบรรทัดเหล่านี้ไปยังเชลล์ใหม่เดี่ยว

การเรียกใช้คำสั่ง make จากไฟล์ รายละเอียด

การเรียกซ้อนการเรียกใช้คำสั่ง **make** ภายในไฟล์รายละเอียดคำสั่ง **make** ให้ใส่แมโคร **$(MAKE)** ในหนึ่งในบรรทัดคำสั่งในไฟล์

ถ้าแฟล็ก **–n** ถูกตั้งค่าเมื่อพบแมโคร **$(MAKE)** สำเนาใหม่ของคำสั่ง **make** ไม่ทำงานคำสั่งใดๆ เหล่านี้ ยกเว้นอีกแมโคร **$(MAKE)** ในการใช้คุณสมบัตินี้เพื่อทดสอบชุดของ ไฟล์รายละเอียดที่อธิบายโปรแกรม ให้ป้อน:

```
make -n
```

คำสั่ง **make** ไม่มีการดำเนินการ ใดๆ ของโปรแกรม อย่างไรก็ตาม จะเขียนขั้นตอนทั้งหมดที่จำเป็นสำหรับ สร้างโปรแกรม รวมถึงเอาต์พุตจากการเรียกใช้ระดับต่ำไปยังคำสั่ง **make**

การป้องกันไม่ให้คำสั่ง make หยุดทำงานเมื่อมีข้อผิดพลาด

โดยปกติคำสั่ง **make** จะหยุดทำงาน ถ้าโปรแกรมใดๆ ส่งกลับโค้ดระบุความผิดพลาดที่ไม่เป็นค่าศูนย์ บางโปรแกรมส่งกลับสถานะที่ ไม่มีความหมาย

ในการหลีกเลี่ยงคำสั่ง **make** มิให้หยุดทำงานเมื่อมีข้อผิดพลาด ให้ทำอย่างหนึ่งอย่างใดต่อไปนี้:

- ใช้แฟล็ก **–i** กับคำสั่ง **make** บนบรรทัดรับคำสั่ง
- ใส่ชื่อ fake target **.IGNORE** บนบรรทัด dependency ในไฟล์รายละเอียด เนื่องจาก **.IGNORE** ไม่ใช่ไฟล์เป้าหมายที่แท้จริง จึงถูกเรียกเป็นเป้าหมายปลอม ถ้า **.IGNORE** มีสิ่งที่จำเป็นต้องมี คำสั่ง **make** จะละเว้นข้อผิดพลาดที่เชื่อมโยง
- ใส่ – (เครื่องหมายลบ) ในตำแหน่งอักขระแรกของแต่ละบรรทัด ในไฟล์รายละเอียดที่คำสั่ง **make** ไม่ควร หยุดทำงานเมื่อพบข้อผิดพลาด

ตัวอย่างของไฟล์รายละเอียด

ตัวอย่าง โปรแกรมชื่อ **prog** จัดทำขึ้นโดยการคอมไพล์และลิงก์ไฟล์ภาษา C สามไฟล? **x.c, y.c** และ **z.c** ไฟล์ **x.c** และ **y.c** แบ่งใช้การประกาศบางอย่างในไฟล์ชื่อ `defs` ไฟล์ **z.c** ไม่แบ่งใช้การประกาศเหล่านั้น ต่อไปนี้คือตัวอย่าง ของไฟล์รายละเอียด ซึ่งสร้างโปรแกรม **prog** :

```
# Make prog from 3 object files
prog: x.o y.o z.o
# Use the cc program to make prog
    cc  x.o y.o z.o -o prog
# Make x.o from 2 other files
x.o:   x.c defs
# Use the cc program to make x.o
    cc -c x.c
# Make y.o from 2 other files
y.o: y.c defs
# Use the cc program to make y.o
    cc  -c y.c
# Make z.o from z.c
z.o:   z.c
# Use the cc program to make z.o
    cc  -c z.c
```

ถ้าไฟล์นี้ชื่อ **makefile** ป้อนคำสั่ง **make** เพื่ออัพเดตโปรแกรม **prog** หลังทำการ เปลี่ยนแปลงไปยังไฟล์ต้นฉบับใดๆ: **x.c, y.c, z.c** หรือ **defs**

การทำให้ไฟล์รายละเอียดง่ายขึ้น

ในการทำให้ไฟล์นี้ใช้ง่ายขึ้น ให้ใช้กฎภายใน ของโปรแกรม **make**

ตามข้อกำหนดการตั้งชื่อระบบไฟล์ คำสั่ง **make** รู้จักไฟล์ **.c** สามไฟล์ที่สอดคล้องกับไฟล์ **.o** ที่จำเป็น คำสั่งนี้ยังสามารถสร้างอ็อบเจ็กต์จากไฟล์ต้นฉบับ โดยการออกคำสั่ง **cc -c**

โดยยึดตามกฎภายในเหล่านี้ ไฟล์รายละเอียด จะเป็น:

```
# Make prog from 3 object files
prog:  x.o y.o z.o
# Use the cc program to make prog
    cc  x.o y.o z.o -o prog
# Use the file defs and the .c file
# when making x.o and y.o
x.o y.o:   defs
```

กฎภายในสำหรับโปรแกรม make

กฎภายในสำหรับคำสั่ง **make** อยู่ในไฟล์ที่คล้ายไฟล์รายละเอียด

เมื่อแฟล็ก **-r** ถูกระบุ คำสั่ง **make** จะไม่ใช้ไฟล์กฎภายใน คุณต้องกำหนดกฎที่จะสร้างไฟล์ ในไฟล์รายละเอียดของคุณ ไฟล์กฎภายในมีรายการคำต่อท้ายชื่อไฟล์ (เช่น **.o** หรือ **.a**) ที่ คำสั่ง **make** เข้าใช้และกฎที่แจ้ง ให้คำสั่ง **make** ทราบวิธีสร้างไฟล์ด้วยคำต่อท้ายหนึ่ง จากไฟล์ที่มีคำต่อท้ายอีกแบบหนึ่ง ถ้าคุณไม่ได้เปลี่ยนแปลงรายการ คำสั่ง **make** เข้าใจในคำต่อท้ายต่อไปนี้:

ส่วนเติม

ท้าย	คำอธิบาย
.a	ไลบรารีไฟล์เก็บถาวร
.C	ไฟล์ต้นฉบับ C++
.C\~	ไฟล์ Source Code Control System (SCCS) ที่มีไฟล์ต้นฉบับ C++
.c	ไฟล์ต้นฉบับ C
.c~	ไฟล์ SCCS ที่มีไฟล์ต้นฉบับ C
.f	ไฟล์ต้นฉบับ FORTRAN
.f~	ไฟล์ SCCS ที่มีไฟล์ต้นฉบับ FORTRAN
.h	ไฟล์ส่วนหัวภาษา C
.h~	ไฟล์ SCCS ที่มีส่วนหัวภาษา C
.l	ไวยากรณ์สำหรับซอร์ส **lex**
.l~	ไฟล์ SCCS ที่มีไวยากรณ์สำหรับซอร์ส **lex**
.o	อ็อบเจ็กต์ไฟล์
.s	ไฟล์ต้นฉบับแอสเซมเบลอร์
.s~	ไฟล์ SCCS ที่มีไฟล์ต้นฉบับแอสเซมเบลอร์
.sh	ไฟล์ต้นฉบับคำสั่งเชลล์
.sh~	ไฟล์ SCCS ที่มีไฟล์ต้นฉบับคำสั่งเชลล์
.y	ไวยากรณ์สำหรับซอร์ส **yacc-c**
.y~	ไฟล์ SCCS ที่มีไวยากรณ์สำหรับซอร์ส **yacc-c**

รายการคำต่อท้ายจะคล้ายกับรายการ dependency ในไฟล์รายละเอียด และตามชื่อเป้าหมายปลอมของ **.SUFFIXES** เนื่องจากคำสั่ง **make** โปรดดูที่ รายการคำต่อท้ายในลำดับจากซ้ายไปขวา ลำดับของรายการถือเป็นสิ่งสำคัญ

คำสั่ง **make** ใช้รายการแรกในรายการ ที่ตรงตามข้อกำหนดต่อไปนี้:

- รายการจับคู่อินพุตและข้อกำหนดคำต่อท้ายเอาต์พุต สำหรับไฟล์เป้าหมายและ dependency ปัจจุบัน
- รายการมีกฎที่กำหนดให้แก่รายการ

คำสั่ง **make** สร้าง ชื่อของกฎจากคำต่อท้ายทั้งสองของไฟล์ที่กฎกำหนด ตัวอย่างเช่น ชื่อของกฎเพื่อแปลงไฟล์ **.c** เป็นไฟล์ **.o** คือ **.c.o**

ในการเพิ่มคำต่อท้ายลงในรายการ ให้เพิ่มรายการ สำหรับชื่อเป้าหมายปลอมของ **.SUFFIXES** ในไฟล์ รายละเอียด สำหรับบรรทัด **.SUFFIXES** ที่ไม่มีคำต่อท้ายใดๆ ตามหลัง ชื่อเป้าหมายในไฟล์รายละเอียด คำสั่ง **make** จะลบรายการปัจจุบัน ในการเปลี่ยนแปลงลำดับของชื่อในรายการ ให้ลบ รายการปัจจุบันจากนั้นกำหนดชุดของค่าใหม่ให้แก่ **.SUFFIXES**

ตัวอย่างของไฟล์กฎดีฟอลต์

ตัวอย่างต่อไปนี้แสดงส่วนของไฟล์กฎ ดีฟอลต์:

```
# Define suffixes that make knows.
.SUFFIXES:  .o .C .C\~ .c .c~ .f .f~ .y .y~ .l .l~ .s .s~ .sh .sh~ .h .h~ .a
 #Begin macro definitions for
#internal macros
YACC=yacc
YFLAGS=
ASFLAGS=
LEX=lex
LFLAGS=
CC=cc
CCC=xlC
AS=as
CFLAGS=
CCFLAGS=
# End macro definitions for
```

```
# internal macros
# Create a .o file from a .c
# file with the cc program.
c.o:
        $(CC) $(CFLAGS) -c $<

# Create a .o file from
# a .s file with the assembler.
s.o:
        $(AS)$(ASFLAGS) -o $@ $<

.y.o:
# Use yacc to create an intermediate file
        $(YACC) $(YFLAGS) $<
# Use cc compiler
        $(CC) $(CFLAGS) -c y.tab.c
# Erase the intermediate file
        rm y.tab.c
# Move to target file
        mv y.tab.o $@.
.y.c:
# Use yacc to create an intermediate file
        $(YACC) $(YFLAGS) $<
# Move to target file
        mv y.tab.c $@
```

กฎ Single-suffix

คำสั่ง **make** มีชุดของ กฎ single-suffix เพื่อสร้างไฟล์ต้นฉบับโดยตรงในชื่อไฟล์เป้าหมาย และไม่มีคำต่อท้าย (ตัวอย่าง ไฟล์คำสั่ง)

คำสั่ง **make** ยังมีกฎในการเปลี่ยนแปลงไฟล์ต้นฉบับต่อไปนี้ด้วย คำต่อท้ายไปยังอ็อบเจ็กต์ไฟล์ที่ไม่มีคำต่อท้าย:

คำสั่ง	คำอธิบาย
.C:	จากไฟล์ต้นฉบับภาษา C++
.C\~:	จากไฟล์ต้นฉบับภาษา SCCS C++
.c:	จากไฟล์ต้นฉบับภาษา C
.c~:	จากไฟล์ต้นฉบับภาษา SCCS C++
.sh:	จากไฟล์เชลล์
.sh~:	จากไฟล์เชลล์ SCCS

ตัวอย่าง ถ้าไฟล์ต้นฉบับทั้งหมดที่ต้องการนั้นมี อยู่ในไดเร็กทอรีปัจจุบัน เพื่อดูแลรักษาโปรแกรม `cat` ให้ป้อน:

```
make cat
```

การใช้คำสั่ง make กับไลบรารีไฟล์เก็บถาวร

คำสั่ง **make** มีชุดของ กฎ single-suffix เพื่อสร้างไฟล์ต้นฉบับโดยตรงในชื่อไฟล์เป้าหมาย และไม่มีคำต่อท้าย (ตัวอย่าง ไฟล์คำสั่ง)

กฎภายใน สำหรับการเลือกไฟล์ต้นฉบับไปเป็นไฟล์ไลบรารี คือ:

กฎ	คำอธิบาย
.C.a	ซอร์ส C++ ที่จะเก็บถาวร
.C\~.a	ซอร์ส SCCS C++ ที่จะไฟล์เก็บถาวร
.c.a	ซอร์ส C ที่จะเก็บถาวร
.c~.a	ซอร์ส SCCS C ที่จะเก็บถาวร
.s~.a	ซอร์สแอสเซมเบลอร์ SCCS ที่จะเก็บถาวร
.f.a	ซอร์ส Fortran ที่จะเก็บถาวรsource to archive
.f~.a	ซอร์ส SCCS Fortran ที่จะเก็บถาวร

การกำหนดเงื่อนไขดีฟอลต์ในไฟล์รายละเอียด

เมื่อคำสั่ง **make** สร้าง ไฟล์เป้าหมายแต่ไม่สามารถพบคำสั่งในไฟล์รายละเอียดหรือกฎ ภายในเพื่อสร้างไฟล์ คำสั่งจะดูที่ไฟล์รายละเอียดเพื่อดูเงื่อนไขปกติ

เมื่อต้องการกำหนดคำสั่งที่คำสั่ง **make** ดำเนินการ ในกรณีนี้ใช้ชื่อเป้าหมาย **.DEFAULT** ในไฟล์รายละเอียด มีดังนี้:

```
.DEFAULT:
        command
        command
            .
            .
            .
```

เนื่องจาก **.DEFAULT** ไม่ใช่ไฟล์เป้าหมายจริง ถูกเรียกว่า *เป้าหมายปลอม* ใช้ชื่อเป้าหมายปลอม **.DEFAULT** สำหรับรูทีนการกู้ข้อผิดพลาด หรือสำหรับโพรซีเดอร์ทั่วไปเพื่อสร้างไฟล์ทั้งหมดในคำสั่งที่ไม่ได้ ถูกกำหนดโดยกฎภายในของโปรแกรม **make**

การรวมไฟล์อื่นในไฟล์รายละเอียด

รวมไฟล์อื่นนอกเหนือจากไฟล์รายละเอียดปัจจุบัน โดยใช้คำว่า **include** เป็นคำแรกบนบรรทัดใดๆ ในไฟล์รายละเอียด

ตามหลังคำด้วยช่องว่าง หรือแท็บ จากนั้นตามด้วย ชื่อไฟล์เพื่อให้คำสั่ง **make** รวมในการดำเนินการ

หมายเหตุ: สนับสนุนให้ใช้ได้หนึ่งไฟล์เท่านั้นสำหรับแต่ละคำสั่ง **include**

ตัวอย่างเช่น:

```
include /home/tom/temp
include /home/tom/sample
```

สั่งให้คำสั่ง **make** อ่านไฟล์ **temp** และ **sample** และไฟล์รายละเอียดปัจจุบันเพื่อสร้างไฟล์เป้าหมาย

เมื่อคุณใช้การรวม (include) ไฟล์ อย่าให้ซ้อนกัน มากกว่า 16 ระดับ

การกำหนดและการใช้แมโครในไฟล์คำอธิบาย

แมโคร คือชื่อ (หรือเลเบล) เพื่อใช้แทนชื่ออื่นๆ หลายชื่อ เป็นวิธีในการเขียนสตริงอักขระ ขนาดยาวโดยใช้เพียงชื่อที่สั้นลงชื่อเดียว

ในการกำหนดแมโคร:

1. เริ่มต้นบรรทัดใหม่ด้วยชื่อของแมโคร
2. ต่อท้ายชื่อด้วย = (เครื่องหมายเท่ากับ)

3. ด้านขวาของ = (เครื่องหมายเท่ากับ) ป้อนสตริง อักขระที่ชื่อแมโครแสดงแทน

นิยามแมโครสามารถมีช่องว่างก่อนและหลัง = (เครื่องหมายเท่ากับ) โดยไม่มีผลกระทบต่อผลลัพธ์ นิยามแมโคร ไม่สามารถมี : (โคลอน) หรือแท็บก่อน = (เครื่องหมายเท่ากับ)

ต่อไปนี้คือตัวอย่างของนิยามแมโคร:

```
# Macro -"2" has a value of "xyz"
2 = xyz

# Macro "abc" has a value of "-ll -ly"
abc = -ll -ly

# Macro "LIBES" has a null value
LIBES =
```

แมโครที่ตั้งชื่อ แต่ไม่ถูกกำหนด มีค่าเหมือนกับ สตริง null

การใช้แมโครในไฟล์รายละเอียด

หลังการกำหนดแมโครในไฟล์รายละเอียด ให้ใช้ แมโครในคำสั่งของไฟล์รายละเอียดโดยการใส่ $ (เครื่องหมายดอลล่าร์) หน้า ชื่อของแมโคร

ถ้าชื่อแมโครยาวเกินหนึ่งอักขระ ให้ใส่ ( ) (วงเล็บ) หรือ { } (วงเล็บปีกกา) ปิดชื่อแมโคร ต่อไปนี้คือ ตัวอย่างการใช้แมโคร:

```
$(CFLAGS)
$2
$(xy)
$Z
$(Z)
```

สองตัวอย่างสุดท้ายในรายการก่อนหน้ามี ผลเหมือนกัน

แฟรกเมนต์ต่อไปนี้แสดงวิธีกำหนดและใช้ บางแมโคร:

```
# OBJECTS is the 3 files x.o, y.o and
# z.o (previously compiled)
OBJECTS = x.o y.o z.o
# LIBES is the standard library
LIBES = -lc
# prog depends on x.o y.o and z.o
prog:  $(OBJECTS)
# Link and load the 3 files with
# the standard library to make prog
      cc $(OBJECTS) $(LIBES) -o prog
```

คำสั่ง **make** ที่ใช้ ไฟล์รายละเอียดนี้ลิงก์และโหลดสามอ็อบเจ็กต์ไฟล์ (**x.o, y.o** และ **z.o**) กับไลบรารี **libc.a**

นิยามแมโครที่ป้อนบนบรรทัดคำสั่งจะแทนที่ นิยามแมโครที่ซ้ำกันอื่นๆ ในไฟล์รายละเอียด ดังนั้น คำสั่งต่อไปนี้ โหลดไฟล์ที่มีไลบรารี **lex** (-11):

```
make "LIBES= -ll"
```

หมายเหตุ: เมื่อแมโครมีช่องว่าง และคุณป้อน ช่องว่างบนบรรทัดคำสั่ง ให้ใส่ " " (เครื่องหมายอัญประกาศคู่) รอบแมโคร ถ้าไม่มีเครื่องหมายอัญประกาศคู่ เชลล์จะแปลความหมายช่องว่างเหล่านั้นเป็น ตัวคั่นพารามิเตอร์และไม่ถูกนำมาเป็นส่วนหนึ่งของแมโคร

คำสั่ง **make** สามารถจัดการ กับการขยายการใช้แมโครซ้อนกันได้สูงสุดถึง 10 ระดับ โดยจากนิยามในตัวอย่าง ต่อไปนี้ นิพจน์ `$($(macro2))` จะหาค่า ได้เป็น `value1`:

```
macro1=value1

macro2=macro1
```

การหาค่าของแมโครเกิดขึ้นในแต่ละครั้งที่ แมโครถูกอ้างอิง โดยยังไม่ถูกประเมินค่าเมื่อกำหนด ถ้าแมโครถูกกำหนด แต่ไม่เคยใช้งาน แมโครก็จะไม่เคยถูกประเมินเพื่อหาค่า นี้เป็นสิ่งสำคัญมากถ้า แมโครถูกกำหนดค่าที่จะถูกแปลความหมายโดยเชลล์โดยเฉพาะ ถ้าค่าอาจมีการเปลี่ยนแปลง การประกาศตัวแปรเช่น:

```
OBJS = 'ls *.o'
```

อาจเปลี่ยนแปลงค่าถ้าถูกอ้างอิงในเวลาต่างกัน ระหว่างการประมวลผลเพื่อทำการสร้างหรือลบอ็อบเจ็กต์ไฟล์ ไม่สามารถเก็บค่า ของแมโคร **ls** ในตอนที่แมโคร **OBJS** ถูกกำหนดได้

แมโครภายใน

คำสั่ง **make** มีนิยามแมโคร ติดมาด้วยสำหรับใช้ในไฟล์รายละเอียด

แมโครเหล่านี้ช่วยระบุ ตัวแปรในไฟล์รายละเอียด คำสั่ง **make** แทนที่แมโครด้วยค่าใดค่าหนึ่งต่อไปนี้:

แมโคร	ค่า
$@	ชื่อของไฟล์เป้าหมายปัจจุบัน
$$@	ชื่อเลเบลบนบรรทัด dependency
$?	ชื่อของไฟล์ที่ถูกเปลี่ยนแปลงล่าสุดกว่าเป้าหมาย
$<	ชื่อไฟล์พาเรนต์ของไฟล์ที่ไม่ใช้แล้วที่เป็นสาเหตุให้สร้าง ไฟล์เป้าหมาย
$*	ชื่อของไฟล์พาเรนต์ปัจจุบันที่ไม่มีคำต่อท้าย
$%	ชื่อของสมาชิกไลบรารีไฟล์เก็บถาวร

ชื่อไฟล์เป้าหมาย

ถ้าแมโคร **$@** macro is อยู่ในลำดับคำสั่ง ในไฟล์รายละเอียด คำสั่ง **make** จะแทน สัญลักษณ์ด้วยชื่อเต็มของไฟล์เป้าหมายปัจจุบันก่อนส่งคำสั่ง ไปยังเชลล์เพื่อรัน คำสั่ง **make** แทน ค่าสัญลักษณ์ต่อเมื่อรันคำสั่งจากไฟล์รายละเอียดเพื่อสร้าง ไฟล์เป้าหมาย

ชื่อเลเบล

ถ้าแมโคร **$$@** อยู่บนบรรทัด dependency ในไฟล์รายละเอียด คำสั่ง **make** จะแทน สัญลักษณ์นี้ด้วยชื่อเลเบลที่อยู่ด้านซ้ายของโคลอนใน บรรทัด dependency ตัวอย่าง ถ้าประโยคต่อไปนี้ถูกรวมในบรรทัด dependency :

```
cat:    $$@.c
```

คำสั่ง **make** จะแปล เป็น:

```
cat:    cat.c
```

เมื่อคำสั่ง **make** หาค่า นิพจน์ใช้แมโครนี้เพื่อสร้างกลุ่มของไฟล์โดยที่แต่ละกลุ่มมี หนึ่งไฟล์ต้นฉบับเท่านั้น ตัวอย่าง ในการดูแลรักษาไดเร็กทอรีของคำสั่งระบบ ให้ใช้ไฟล์รายละเอียดที่คล้ายกับ:

```
# Define macro CMDS as a series
# of command names
CMDS = cat dd echo date cc cmp comm ar ld chown
# Each command depends on a .c file
$(CMDS):      $$@.c
# Create the new command set by compiling the out of
# date files ($?) to the target file name ($@)
         $(CC) -O $? -o $@
```

คำสั่ง **make** เปลี่ยนแมโคร **$$(@F)** เป็นส่วนไฟล์ของ **$@** เมื่อ รัน ตัวอย่าง ใช้สัญลักษณ์นี้เมื่อคงไดเร็กทอรี **usr/include** ไว้ขณะใช้ไฟล์รายละเอียดในอีกไดเร็กทอรีหนึ่ง ไฟล์รายละเอียดนั้นคล้ายกับตัวอย่างต่อไปนี้:

```
# Define directory name macro INCDIR
INCDIR = /usr/include
# Define a group of files in the directory
# with the macro name INCLUDES
INCLUDES = \
         $(INCDIR)/stdio.h \
         $(INCDIR)/pwd.h \
         $(INCDIR)/dir.h \
         $(INCDIR)/a.out.h \
# Each file in the list depends on a file
# of the same name in the current directory
$(INCLUDES):       $$(@F)
# Copy the younger files from the current
# directory to /usr/include
         cp $? $@
# Set the target files to read only status
         chmod 0444 $@
```

ไฟล์รายละเอียดก่อนหน้าจะสร้างไฟล์ในไดเร็กทอรี **/usr/include** เมื่อไฟล์ที่เกี่ยวข้องใน ไดเร็กทอรีปัจจุบันมีการเปลี่ยนแปลง

ไฟล์ที่เปลี่ยนแปลง

ถ้าแมโคร **$?** อยู่ในลำดับคำสั่ง ในไฟล์รายละเอียด คำสั่ง **make** จะแทน สัญลักษณ์ด้วยรายการไฟล์พาเรนต์ที่ถูกเปลี่ยนแปลงตั้งแต่ที่ไฟล์เป้าหมาย ถูกเปลี่ยนแปลงล่าสุด คำสั่ง **make** แทน ค่าสัญลักษณ์ต่อเมื่อรันคำสั่งจากไฟล์รายละเอียดเพื่อสร้างไฟล์เป้าหมาย

ไฟล์ที่ไม่ใช้แล้วไฟล์แรก

ถ้าแมโคร **$<** อยู่ใน ลำดับคำสั่งในไฟล์รายละเอียด คำสั่ง **make** จะแทนสัญลักษณ์ด้วยชื่อของไฟล์ที่เริ่มการสร้างไฟล์ ชื่อไฟล์ Out-of-Date คือชื่อของไฟล์พาเรนต์ที่การประทับเวลา ไม่ตรงกับการประทับเวลาของไฟล์เป้าหมาย ทำให้คำสั่ง **make** สร้างไฟล์เป้าหมายอีกครั้ง

นอกจากนั้น ใช้ตัวอักษร (**D** หรือ **F**) หลัง < (เครื่องหมายน้อยกว่า) เพื่อรับชื่อไดเร็กทอรี (**D**) หรือชื่อไฟล์ (**F**) ของไฟล์ที่ไม่ใช้ไฟล์แรก ตัวอย่าง ถ้าไฟล์ที่ไม่ใช้ไฟล์แรก คือ:

```
/home/linda/sample.c
```

ดังนั้นคำสั่ง **make** จะให้ ค่าต่อไปนี้:

```
$(<D)   =   /home/linda
$(<F)   =   sample.c
$<      =   /home/linda/sample.c
```

คำสั่ง **make** จะแทน สัญลักษณ์นี้ต่อเมื่อโปรแกรมรันคำสั่งจากกฎภายใน หรือจาก รายการ **.DEFAULT**

ส่วนนำหน้าชื่อไฟล์ปัจจุบัน

ถ้าแมโคร **$*** อยู่ในลำดับคำสั่ง ในไฟล์รายละเอียด คำสั่ง **make** จะแทน สัญลักษณ์ด้วยส่วนของชื่อไฟล์ (โดยไม่มีคำต่อท้าย) ของไฟล์พาเรนต์ ที่คำสั่ง **make** กำลังใช้เพื่อสร้าง ไฟล์เป้าหมายในขณะนี้ ตัวอย่าง ถ้าคำสั่ง **make** กำลังใช้ไฟล์:

```
test.c
```

ดังนั้นแมโคร **$*** จะแทน ชื่อไฟล์ `test`

นอกจากนั้น ใช้ตัวอักษร (**D** หรือ **F**) หลัง * (เครื่องหมายดอกจัน) เพื่อรับชื่อไดเร็กทอรี (**D**) หรือชื่อไฟล์ (**F**) ของ ไฟล์ปัจจุบัน

ตัวอย่าง คำสั่ง **make** ใช้หลายๆ ไฟล์ (ที่ระบุในไฟล์รายละเอียดหรือในกฎ ภายใน) เพื่อสร้างไฟล์เป้าหมาย เพียงไฟล์เดียวจาก ไฟล์เหล่านั้นเท่านั้น (ไฟล์ปัจจุบัน) ที่ใช้ในขณะใดๆ ถ้าไฟล์ปัจจุบันนั้นคือ:

```
/home/tom/sample.c
```

ดังนั้นคำสั่ง **make** จะ ให้ค่าต่อไปนี้สำหรับแมโคร:

```
$(*D)  =  /home/tom
$(*F)  =  sample
$*     =  /home/tom/sample
```

คำสั่ง **make** จะแทนค่า สัญลักษณ์นี้ต่อเมื่อรันคำสั่งจากกฎภายใน (หรือจากรายการ **.DEFAULT**) แต่ไม่ใช่เมื่อรันคำสั่งจาก ไฟล์ รายละเอียด

สมาชิกไลบรารีไฟล์เก็บถาวร

ถ้าแมโคร **$%** อยู่ในไฟล์รายละเอียด และไฟล์เป้าหมายเป็นสมาชิกไลบรารีไฟล์เก็บถาวร คำสั่ง **make** จะแทนสัญลักษณ์แม โครด้วยชื่อของสมาชิกไลบรารี ตัวอย่าง ถ้าไฟล์เป้าหมายคือ:

```
lib(file.o)
```

ดังนั้นคำสั่ง **make** แทน แมโคร **$%** ด้วยชื่อสมาชิก `file.o`

การเปลี่ยนนิยามแมโครในคำสั่ง

เมื่อแมโครในคำสั่งเชลล์ถูกกำหนดในไฟล์ รายละเอียด คุณสามารถเปลี่ยนค่าที่คำสั่ง **make** กำหนดให้แก่แมโครได้

ในการเปลี่ยนแปลงการกำหนดแมโคร ให้ใส่ : (โคลอน) หลังชื่อแมโคร ตามด้วยสตริงที่แทน รูปแบบ มีดังนี้:

```
$(macro:string1=string2)
```

โดยที่ `string1` คือ คำต่อท้ายหรือคำที่จะแทนในนิยามแมโครงและ `string2` คือคำต่อท้ายการแทนที่หรือคำ

เมื่อคำสั่ง **make** อ่าน แมโครและเริ่มการกำหนดค่าให้แก่แมโครตามนิยาม แมโคร คำสั่งจะแทนแต่ละ `string1` ใน นิยามแมโครด้วยค่าของ `string2` ตัวอย่าง ถ้าไฟล์รายละเอียดมีนิยามแมโคร:

```
FILES=test.o sample.o form.o defs
```

คุณสามารถแทนไฟล์ `form.o` ด้วยไฟล์ใหม่ `input.o` โดยใช้ แมโครในคำสั่งไฟล์รายละเอียด ดังนี้:

```
cc -o $(FILES:form.o=input.o)
```

คุณสามารถแทนคำต่อท้าย .o ทั้งหมดในแมโครด้วย .c ดังนี้:

```
cc -c $(FILES:.o=.c)
```

ค่าแมโครยังสามารถเปลี่ยนโดยใช้รูปแบบการจับคู่การแทนที่ ของฟอร์มต่อไปนี้:

```
$(macro: op%os= np%ns)
```

โดยที่ `op` คือคำนำหน้า (เก่า) ที่มีอยู่ `os` คือ คำต่อท้าย (เก่า) ที่มีอยู่ `np` คือคำนำหน้าใหม่ และ `ns` คือ คำต่อท้ายใหม่

`op`, `os`, `np` และ `ns` สามารถ เป็นสตริงอักขระอย่างน้อยศูนย์อักขระ รูปแบบที่จับคู่โดยเครื่องหมายเปอร์เซ็นต์ (%) บนด้านซ้ายของเครื่องหมายเท่ากับ ซึ่งเป็นสตริงอักขระอย่างน้อยศูนย์ อักขระ ถูกใช้กับ `np` และ `ns` เพื่อแทน ค่าแมโคร ตัวดำเนินการเครื่องหมายเปอร์เซ็นต์ (%) สามารถมีได้ไม่จำกัดจำนวน บนด้านขวาของเครื่องหมายเท่ากับ (=)

ตัวอย่างเช่น:

```
FOO=abc def
BAR=$(FOO:%=dir1/%.o dir1/%_cltn.o)
```

ตั้งค่าของ BAR เป็น `dir1/abc.o dir1/abc_cltn.o dir1/def.o dir1/def_cltn.o`

การเปลี่ยนค่าของแมโครในลักษณะนี้เป็นประโยชน์ เมื่อทำการดูแลรักษาไลบรารีไฟล์เก็บถาวร สำหรับข้อมูลเพิ่มเติม โปรดดูที่คำสั่ง **ar**

การสร้างไฟล์ปลายทางด้วยคำสั่ง make

ในการสร้างไฟล์ที่มีทั้งโปรแกรม เรียกว่าไฟล์ *เป้าหมาย* ซึ่งโปรแกรม **make** :

1. ค้นหาชื่อของไฟล์เป้าหมายในไฟล์รายละเอียดหรือใน คำสั่ง **make**
2. ทำให้แน่ใจว่าไฟล์ที่ไฟล์เป้าหมายต้องใช้ มีอยู่และทันสมัย
3. พิจารณาว่าไฟล์เป้าหมายทันสมัยหรือไม่ เทียบกับไฟล์ที่ต้องขึ้นกับ

ถ้าไฟล์เป้าหมายหรือหนึ่งในไฟล์พาเรนต์ไม่ทันสมัย คำสั่ง **make** จะสร้างไฟล์เป้าหมายโดยใช้ อย่างใดอย่างหนึ่งต่อไปนี้:

- คำสั่งจากไฟล์รายละเอียด
- กฎภายในเพื่อสร้างไฟล์ (ถ้ามี)
- กฎดีฟอลต์จากไฟล์รายละเอียด

ถ้าไฟล์ทั้งหมดในโพรซีเดอร์ทันสมัยเมื่อ รันโปรแกรม **make** คำสั่ง **make** จะแสดงข้อความเพื่อระบุว่าไฟล์นั้นทันสมัย จากนั้นหยุดทำงาน ถ้ามีบางไฟล์เปลี่ยนแปลง คำสั่ง **make** จะสร้างเฉพาะไฟล์เหล่านั้นที่ไม่ทันสมัย คำสั่งไม่สร้าง ไฟล์ที่เป็นค่าปัจจุบันอยู่แล้วใหม่

เมื่อคำสั่ง **make** รันคำสั่ง เพื่อสร้างไฟล์เป้าหมาย จะแทนค่าแมโครด้วยค่าของคำสั่ง เขียนแต่ละ บรรทัดคำสั่ง จากนั้นส่งคำสั่งไปยังสำเนาของเชลล์ใหม่

การใช้คำสั่ง make กับไฟล์ source code control system

คำสั่ง source code control system (SCCS) และระบบ ไฟล์ถูกใช้เพื่อควบคุมการเข้าถึงไฟล์ ติดตามผู้ที่เปลี่ยนแปลง ไฟล์ สาเหตุที่เปลี่ยนแปลง และสิ่งที่ถูกเปลี่ยนแปลงเป็นหลัก

ไฟล์ SCCS คือไฟล์ข้อความใดๆ ที่ควบคุมด้วยคำสั่ง SCCS การใช้คำสั่งที่มิใช่ SCCS เพื่อแก้ไขไฟล์ SCCS อาจทำให้ไฟล์ SCCS เสียหาย

ไฟล์ SCCS ทั้งหมดใช้คำนำหน้า **s.** เพื่อบ่งชี้ว่าไฟล์เหล่านั้นเป็นไฟล์ SCCS และไม่ใช่ไฟล์ข้อความปกติ คำสั่ง **make** ไม่รู้จักการอ้างถึงไปยังคำนำหน้า ของชื่อไฟล์ ดังนั้น อย่าอ้างอิงไฟล์ SCCS โดยตรงภายในไฟล์รายละเอียดของคำสั่ง **make** คำสั่ง **make** ใช้คำต่อท้ายที่ต่างกัน ~ (tilde) เพื่อแทนไฟล์ SCCS ดังนั้น **.c~.o** จะอ้างถึงกฎที่แปลงสภาพ ไฟล์ต้นฉบับภาษา C ของ SCCS C ไปเป็นอ็อบเจ็กต์ไฟล์ กฎภายในคือ:

```
.c~.o:
        $(GET) $(GFLAGS) -p  $<  >$*.c
        $(CC) $(CFLAGS) -c $*.c
        -rm -f $*.c
```

เครื่องหมาย ~ (tilde) ถูกเพิ่มไปยังคำต่อท้ายใดๆ ที่เปลี่ยน การค้นหาไฟล์เป็นการค้นหาชื่อไฟล์ SCCS ด้วยคำต่อท้ายจริงที่กำหนดโดย . (เครื่องหมายจุด) และอักขระทั้งหมดตั้งแต่ (แต่ไม่รวม) ~ (tilde) แมโคร **GFLAGS** ส่งค่าแฟล็กไปยัง SCCS เพื่อพิจารณาว่าจะใช้ เวอร์ชันไฟล์ SCCS ใด

คำสั่ง **make** รู้จัก คำต่อท้าย SCCS ต่อไปนี้:

ส่วนเติมท้าย	คำอธิบาย
.C\~	ซอร์ส C++
.c~	ซอร์ส **c**
.y~	ไวยากรณ์สำหรับซอร์ส **yacc**
.s~	ซอร์สแอสเซมเบลอร์
.sh~	เชลล์
.h~	ส่วนหัว
.f~	FORTRAN
.l~	ซอร์ส **lex**

คำสั่ง **make** มีกฎ ภายในสำหรับการเปลี่ยนไฟล์ SCCS ต่อไปนี้:

.C\~.a:

.C\~.c:

.C\~.o:

.c~:

.c~.a:

.c~.c:

.c~.o:

.f~:

.f~.a:

.f~.o:

.f~.f:

.h~.h:

.l~.o:

.s~.a:

.sh~:

.s~.o:

.y~.c:

.y~.o:

การทำความเข้าใจข้อควรพิจารณา makefile ใน Source Code Control Systems (SCCS)

ถ้าคุณระบุไฟล์รายละเอียด หรือไฟล์ชื่อ **makefile** หรือ **Makefile** อยู่ใน ไดเร็กทอรีปัจจุบัน คำสั่ง **make** จะไม่ค้นหาไฟล์รายละเอียด ภายใน SCCS

ถ้าไฟล์รายละเอียดไม่อยู่ในไดเร็กทอรีปัจจุบันและ คุณป้อนคำสั่ง **make** คำสั่ง **make** จะค้นหาไฟล์ SCCS ที่ชื่อ **s.makefile** หรือ **s.Makefile** ถ้าไฟล์ใดไฟล์หนึ่งแสดงอยู่ คำสั่ง **make** จะใช้คำสั่ง **get** เพื่อสั่ง SCCS ให้สร้างไฟล์รายละเอียดจากไฟล์ต้นฉบับนั้น เมื่อ SCCS สร้างไฟล์รายละเอียด คำสั่ง **make** จะใช้ไฟล์เป็นไฟล์รายละเอียดปกติ เมื่อคำสั่ง **make** ทำงานเสร็จเรียบร้อย คำสั่งจะลบไฟล์รายละเอียดที่สร้างขึ้น ออกจากไดเร็กทอรีปัจจุบัน

การระงับการแสดงรายละเอียดการเรียกค้น SCCS

คุณสามารถระงับการแสดงการเรียกค้น SCCS ดีฟอลต์ คำสั่งสำหรับการเรียกค้น ซอร์สไฟล์จาก SCCS สามารถระบุในกฎสำหรับเรียกข้อมูลพิเศษ *SCCS_GET* ในไฟล์รายละเอียด

ซึ่งจะระงับการเรียกค้นซอร์สดีฟอลต์ จาก SCCS

ตัวอย่างเช่น:

```
SCCS_GET:
        get -p $< > $*.c
```

การใช้คำสั่ง make กับไฟล์ที่ไม่มี source code control system (SCCS)

เริ่มทำงานคำสั่ง **make** จาก ไดเร็กทอรีที่มีไฟล์รายละเอียดสำหรับไฟล์ที่จะสร้าง

ชื่อ ตัวแปร *desc-file* แทนชื่อของไฟล์รายละเอียด จากนั้น ป้อนคำสั่ง:

```
make -f desc-file
```

ถ้าชื่อของไฟล์รายละเอียดเป็น **makefile** หรือ **Makefile** คุณไม่ต้องใช้แฟล็ก **–f** ป้อนนิยามแมโคร แฟล็ก ชื่อไฟล์รายละเอียด และ ชื่อไฟล์เป้าหมายพร้อมกับคำสั่ง **make** บนบรรทัดคำสั่ง ดังนี้:

```
make [flags] [macro definitions] [targets]
```

จากนั้นคำสั่ง **make** จะตรวจสอบ รายการบรรทัดคำสั่งเพื่อพิจาณาสิ่งที่ต้องทำ อันดับแรก ดูที่นิยามแมโครทั้งหมด บนบรรทัดคำสั่ง (รายการที่อยู่ในเครื่องหมายคำพูด และมี เครื่องหมายเท่ากับ) และกำหนดค่าให้ ถ้าคำสั่ง **make** พบนิยามสำหรับแมโครบนบรรทัดคำสั่งต่างไปจากนิยามสำหรับ แมโครนั้นในไฟล์รายละเอียด คำสั่งจะใช้นิยามจากบรรทัดคำสั่ง สำหรับแมโคร

ถัดไป คำสั่ง **make** จะดู ที่แฟล็ก

คำสั่ง **make** คาดว่า รายการในบรรทัดคำสั่งส่วนที่เหลือจะเป็นชื่อของไฟล์เป้าหมายที่จะสร้าง คำสั่งเชลล์ใดๆ ที่อยู่ใน back quotes ที่สร้างชื่อเป้าหมายจะ ดำเนินการโดยคำสั่ง **make** จากนั้นคำสั่ง **make** สร้างไฟล์เป้าหมายในลำดับจากซ้ายไปขวา เมื่อไม่มีชื่อไฟล์เป้าหมาย คำสั่ง **make** จะสร้างไฟล์ เป้าหมายแรกที่มีชื่อในไฟล์รายละเอียดที่ไม่ขึ้นต้นด้วยเครื่องหมายจุด ถ้ามีการระบุไฟล์รายละเอียดมากกว่าหนึ่งไฟล์ คำสั่ง **make** ค้นหาไฟล์รายละเอียดแรกสำหรับเป็นชื่อของไฟล์เป้าหมาย

การทำความเข้าใจวิธีที่คำสั่ง make ใช้งานตัวแปรสภาวะแวดล้อม

แต่ละครั้งที่คำสั่ง **make** รัน คำสั่งจะอ่านตัวแปรสภาวะแวดล้อมปัจจุบันและเพิ่มลงในแมโครที่กำหนด

การใช้แมโคร **MAKEFLAGS** หรือแมโคร **MFLAGS** ผู้ใช้สามารถระบุแฟล็กที่จะส่งไปยังคำสั่ง **make** ถ้าทั้งสองถูกตั้งค่า แมโคร **MAKEFLAGS** จะแทนที่แมโคร **MFLAGS** แฟล็กที่ระบุโดยใช้ ตัวแปรเหล่านี้จะถูกส่งไปยังคำสั่ง **make** พร้อมด้วย อ็อพชันบรรทัดคำสั่ง ในกรณีของการเรียกซ้ำคำสั่ง **make** โดยใช้แมโคร **$(MAKE)** ในไฟล์รายละเอียด คำสั่ง **make** จะส่งแฟล็กทั้งหมดกับแต่ละการร้องขอ

เมื่อคำสั่ง **make** รัน คำสั่งจะกำหนดนิยามแมโครในลำดับต่อไปนี้:

1. อ่านค่าตัวแปรสภาวะแวดล้อม **MAKEFLAGS**
   ถ้าตัวแปรสภาวะแวดล้อม **MAKEFLAGS** ไม่ มีอยู่หรือเป็น null คำสั่ง **make** จะตรวจหาค่าที่ไม่เป็น null ในตัวแปรสภาวะแวดล้อม **MFLAGS** ถ้าตัวแปรหนึ่ง ในตัวแปรเหล่านี้มีค่า คำสั่ง **make** จะถือว่า แต่ละตัวอักษรในค่านั้นเป็นแฟล็กอินพุต คำสั่ง **make** ใช้เหล่านี้แฟล็ก (ยกเว้นแฟล็ก **–f**, **–p** และ **–d** ซึ่งไม่สามารถตั้งค่า จากตัวแปรสภาวะแวดล้อม **MAKEFLAGS** หรือ **MFLAGS**) เพื่อพิจารณาเงื่อนไขการดำเนินการ
2. อ่านและตั้งค่าแฟล็กอินพุตจาก บรรทัดคำสั่ง บรรทัดคำสั่งเพิ่มในการตั้งค่าก่อนหน้าจากตัวแปรสภาวะแวดล้อม **MAKEFLAGS** หรือ **MFLAGS**
3. อ่านนิยามแมโครจากบรรทัดคำสั่ง คำสั่ง **make** ละเว้นการกำหนดค่าอื่นใด ให้แก่ชื่อเหล่านี้
4. อ่านนิยามแมโครภายใน
5. อ่านค่าสภาพแวดล้อม คำสั่ง **make** ถือตัวแปรสภาวะแวดล้อมเป็นนิยามแมโครและส่ง ไปยังโปรแกรมเชลล์อื่นๆ

การใช้คำสั่ง make ในโหมดรันแบบขนาน

โดยทั่วไป คำสั่ง **make** รันคำสั่งตามลำดับ ครั้งละหนึ่งเป้าหมายเท่านั้น จากนั้นรอให้คำสั่งดำเนินการเสร็จสิ้นก่อนการรัน ถัดไป

อย่างไรก็ตาม คำสั่ง **make** ยังสามารถรัน ในรันโหมดแบบขนาน ซึ่งสามารถรันงานพร้อมๆ กันได้หลายงาน เพื่อสร้างเป้าหมายที่เป็นอิสระต่อกัน

แฟล็ก **–j** บอกให้คำสั่ง **make** รันเป้าหมายที่เป็นอิสระต่อกันในเวลาเดียวกัน

ถ้าคุณตามหลังอ็อพชัน **–j** ด้วยเลขจำนวนเต็ม ดังนั้น เลขจำนวนเต็มจะระบุจำนวนงานที่เกิดขึ้นพร้อมกันสูงสุดที่สามารถทำได้ในการสร้างเป้าหมาย

ถ้าแฟล็ก **–j** ไม่ตามด้วยเลขจำนวนเต็ม ดังนั้น แสดงว่าไม่มีขีดจำกัดจำนวนงานที่จะถูกเรียกใช้เพื่อสร้างเป้าหมาย

ถ้าคำสั่ง **make** พบข้อผิดพลาดขณะ กำลังสร้างเป้าหมายในโหมดการเรียกทำงานแบบขนาน และไม่มีเมธอดที่ใช้ เพื่อข้ามข้อผิดพลาดนั้น ดังนั้นการเรียดใช้คำสั่ง เพื่อสร้างเป้าหมายนั้นจะหยุดทำงาน และคำสั่ง **make** จะรอให้ออกจากงานชายน์ที่ทำงานอยู่ก่อนดำเนินการให้เสร็จสมบูรณ์

เมื่อคุณกำลังรันมากกว่าหนึ่งงานในโหมดขนาน เอาต์พุตที่ งานสร้างขึ้นจะพิมพ์ออกบนหน้าจอขณะถูกสร้าง นี้อาจทำให้เกิดความสับสนสำหรับข้อความที่มาจากงานที่ต่างกันที่แสดงบนหน้าจอยกเว้นคุณระงับ การแสดงข้อความโดยใช้การเปลี่ยนทิศทาง หรือการให้คำสั่ง **make** ทำงานแบบไม่แสดงข้อความ

ภาพรวมเกี่ยวกับตัวประมวลผลแมโคร m4

หัวข้อนี้ให้ข้อมูลเกี่ยวกับตัวประมวลผลแมโคร **m4** ซึ่งเป็นตัวประมวลผลส่วนหน้าสำหรับภาษาโปรแกรมใดๆ ที่กำลังใช้ในสภาวะแวดล้อมระบบปฏิบัติการ

ที่จุดเริ่มต้นของโปรแกรม คุณสามารถกำหนดชื่อสัญลักษณ์ หรือค่าคงที่สัญลักษณ์เป็นสตริงของอักขระ จากนั้นคุณสามารถ ใช้ **m4** macro processor เพื่อแทนที่ชื่อสัญลักษณ์ที่ไม่มีเครื่องหมายคำพูดปิดรอบ ด้วย สตริงที่เกี่ยวข้อง นอกเหนือจากการแทนที่สตริงข้อความหนึ่งด้วยข้อความอื่น **m4** macro processor ยังมีคุณลักษณะดังต่อไปนี้:

- ความสามารถทางคณิตศาสตร์
- การจัดการไฟล์
- ส่วนขยายแมโครแบบมีเงื่อนไข
- ฟังก์ชัน String และ substring

m4 macro processor ประมวลผล สตริงของตัวอักษรและตัวเลขที่เรียกว่า *โทเค็น* **m4** macro processor อ่านแต่ละโทเค็นตัวอักษรและตัวเลขและระบุว่า เป็นชื่อของแมโครหรือไม่ จากนั้นโปรแกรมแทนที่ชื่อแมโคร ด้วยข้อความที่กำหนดไว้ของโปรแกรม และส่งสตริงผลลัพธ์กลับไปที่อินพุต เพื่อทำการสแกนซ้ำ คุณสามารถเรียกแมโครที่มีอาร์กิวเมนต์ ในกรณีซึ่งอาร์กิวเมนต์ ถูกรวบรวมและแทนที่ลงที่ด้านขวาในข้อความการกำหนด ก่อน ที่ข้อความการกำหนดถูกสแกนซ้ำ

m4 macro processor มี แมโครในตัวเช่น **define** คุณยังสามารถสร้าง แมโครใหม่ได้ด้วย แมโครในตัวและแบบผู้ใช้กำหนดทำงานแบบเดียวกัน

การใช้ตัวประมวลผลแมโคร m4

เมื่อต้องการใช้ **m4** macro processor ให้ป้อนคำสั่งดังต่อไปนี้:

```
m4 [file]
```

m4 macro processor ประมวลผล แต่ละอาร์กิวเมนต์ตามลำดับ ถ้าไม่มีอาร์กิวเมนต์หรือถ้าอาร์กิวเมนต์เป็น – (แดช), **m4** macro processor อ่าน อินพุตมาตรฐานเป็นอินพุตไฟล์ **m4** macro processor เขียนผลลัพธ์ไปที่เอาต์พุตมาตรฐาน ดังนั้น เพื่อเปลี่ยนทิศทางเอาต์พุตไปที่ ไฟล์เพื่อใช้ภายหลัง ให้ใช้คำสั่งเช่น:

```
m4 [file] >outputfile
```

การสร้างแมโครที่ผู้ใช้กำหนด

แมโคร	คำอธิบาย
define (*MacroName*, *Replacement***)**	กำหนดแมโครใหม่ *MacroName* ด้วยค่า *Replacement*

ตัวอย่างเช่น ถ้าคำสั่งดังต่อไปนี้อยู่ในโปรแกรม:

```
define(name, stuff)
```

m4 macro processor กำหนด สตริง `name` เป็น `stuff` เมื่อ สตริง `name` เกิดขึ้นในไฟล์โปรแกรม **m4** macro processor แทนที่สตริงด้วยสตริง `stuff` สตริง `name` ต้องเป็นตัวอักษรและตัวเลข ASCII และ ต้องเริ่มต้นด้วยตัวอักษรหรือเครื่องหมายขีดล่าง สตริง `stuff` เป็นข้อความ แต่ถ้าข้อความมีวงเล็บ จำนวนของวงเล็บเปิด หรือวงเล็บด้านซ้าย ต้องเท่ากับวงเล็บปิดหรือวงเล็บด้านขวา ใช้อักขระ **/** (slash) เพื่อกระจายข้อความสำหรับ `stuff` ในหลายบรรทัด

วงเล็บเปิด (ซ้าย) ต้องตามคำว่า **define** ตัวอย่างเช่น:

```
define(N, 100)
 . . .
if (i > N)
```

กำหนด `N` เป็น `100` และใช้ค่าคงที่สัญลักษณ์ `N` ในคำสั่ง **if**

การเรียกแมโครในโปรแกรมมีรูปแบบดังนี้:

```
name(arg1,arg2, . . . argn)
```

ชื่อแมโครรู้จักได้เฉพาะเมื่อถูกล้อมรอบ ด้วยอักขระที่ไม่ใช่ตัวเลขหรือตัวอักษร ในตัวอย่างดังต่อไปนี้ตัวแปร `NNN` ไม่ได้สัมพันธ์กับแมโครที่กำหนด `N`

```
define(N, 100)
 . . .
if (NNN > 100)
```

คุณสามารถกำหนดแมโครในรูปแบบที่ใช้ชื่ออื่น ตัวอย่างเช่น:

```
define(N, 100)
define(M, N)
```

กำหนดทั้ง `M` และ `N` เป็น `100` ถ้าคุณเปลี่ยน ข้อกำหนดของ `N` ในภายหลังและกำหนดค่าใหม่ `M` จะมีค่าเป็น `100` ไม่ใช่ `N`

m4 macro processor ขยาย ชื่อแมโครลงในข้อความการกำหนดเร็วที่สุดเท่าที่เป็นไปได้ สตริง `N` ถูกแทนที่ด้วย `100` จากนั้นสตริง `M` ถูกแทนที่ด้วย `100` เช่นกัน ผลลัพธ์โดยรวมเหมือนกับการใช้อินพุตดังต่อไปนี้เมื่อเริ่มต้น

```
define(M, 100)
```

ลำดับของข้อกำหนดสามารถเปลี่ยนสลับได้ ดังนี้:

```
define(M, N)
define(N, 100)
```

ขณะนี้ `M` ถูกกำหนดเป็น สตริง `N` ดังนั้นเมื่อค่าของ `M` ถูกร้องขอภายหลัง ผลลัพธ์คือค่าของ `N` ในเวลานั้น (เนื่องจาก `M` ถูกแทนที่โดย `N` ซึ่งถูกแทนที่โดย `100`)

การใช้อักขระเครื่องหมายคำพูด

เพื่อหน่วงเวลาส่วนขยายของอาร์กิวเมนต์ของ **define** ให้ปิดล้อมคำสั่งด้วยอักขระเครื่องหมายคำพูด ถ้าคุณไม่ได้ทำการเปลี่ยนแปลง อักขระเครื่องหมายคำพูดคือ ` และ ' (เครื่องหมายคำพูดเดี่ยวซ้ายและขวา) ข้อความที่ล้อมด้วย อักขระเครื่องหมายคำพูดไม่ถูกขยายในทันที แต่อักขระเครื่องหมายคำพูดถูก เอาออก ค่าของสตริงที่ปิดด้วยเครื่องหมายคำพูดเป็นสตริงที่ถูกเอาเครื่องหมายคำพูด ออก ถ้าอินพุตเป็น:

```
define(N, 100)
define(M, `N')
```

อักขระเครื่องหมายคำพูดรอบ N ถูกเอาออกขณะที่อาร์กิวเมนต์ถูกรวบรวม ผลลัพธ์ของการใช้ อักขระเครื่องหมายคำพูดคือเพื่อกำหนด M เป็นสตริง N ไม่ใช่ 100 กฎทั่วไป คือ **m4** macro processor ถอดอักขระ เครื่องหมายคำพูดออกหนึ่งระดับเสมอเมื่อมีการหาค่าบางอย่าง ซึ่งเป็น true แม้แต่ภายนอกแมโคร เมื่อต้องการทำให้คำ `define` แสดง ในเอาต์พุต ให้ป้อนคำในอักขระเครื่องหมายคำพูด ดังนี้:

```
`define' = 1;
```

ตัวอย่างอื่นของการใช้อักขระเครื่องหมายคำพูดคือการกำหนด N อีกครั้ง เมื่อต้องการกำหนด N อีกครั้ง ให้เลื่อนการประเมินค่าโดยการใส่ N ใน อักขระเครื่องหมายคำพูด ตัวอย่างเช่น:

```
define(N, 100)
. . .
define(`N', 200)
```

เพื่อป้องกันปัญหาไม่ให้เกิดขึ้น ให้ใส่เครื่องหมายคำพูดอาร์กิวเมนต์แรกของ แมโคร ตัวอย่างเช่น ส่วนต่อไปนี้ไม่ถูกกำหนดค่าใหม่ N:

```
define(N, 100)
. . .
define(N, 200)
```

N ในข้อกำหนดที่สอง ถูกแทนที่โดย 100 ผลลัพธ์เหมือนกับคำสั่งดังต่อไปนี้:

```
define(100, 200)
```

m4 macro processor ละเว้น คำสั่งนี้เนื่องจาก ละเว้นกำหนดได้เฉพาะชื่อ ไม่ใช่ตัวเลข

การเปลี่ยนอักขระเครื่องหมายคำพูด

อักขระเครื่องหมายคำพูดโดยปกติ ` และ ' (เครื่องหมายคำพูดเดี่ยว ซ้ายหรือขวา) ถ้าอักขระเหล่านี้ไม่สะดวกในการใช้ให้เปลี่ยนอักขระเครื่องหมายคำพูด ด้วยแมโครในตัวดังต่อไปนี้:

แมโคร	คำอธิบาย
changequote (*l*, *r*)	เปลี่ยนอักขระเครื่องหมายคำพูดซ้ายและขวาเป็นอักขระที่แสดง ด้วยตัวแปร *l* และ *r*

เมื่อต้องการคืนค่าอักขระเครื่องหมายคำพูดดั้งเดิมให้ใช้ **changequote** โดยไม่ต้องใส่อาร์กิวเมนต์ดังนี้:

```
changequote
```

อาร์กิวเมนต์

รูปแบบที่ง่ายที่สุดของการประมวลผลแมโครคือการแทนที่ สตริงหนึ่งด้วยสตริงอื่น (คงที่) อย่างไรก็ตาม แมโครมีอาร์กิวเมนต์ได้ เพื่อที่คุณสามารถใช้แมโครในที่ต่างกันได้ด้วยผลลัพธ์ที่ต่างกัน เพื่อระบุตำแหน่งอาร์กิวเมนต์ที่จะถูกใช้ภายในข้อความการแทนที่สำหรับ แมโคร (อาร์กิวเมนต์ที่สองของข้อกำหนด) ใช้สัญลักษณ์ **$***n* เพื่อระบุอาร์กิวเมนต์ *n*th เมื่อแมโครถูกใช้ **m4** macro processor แทนที่สัญลักษณ์ด้วยค่าของอาร์กิวเมนต์ที่ระบุ ตัวอย่างเช่น สัญลักษณ์ดังต่อไปนี้:

```
$2
```

หมายถึงอาร์กิวเมนต์ที่สองของแมโคร ดังนั้น ถ้าคุณกำหนดแมโครชื่อ bump ดังนี้:

```
define(bump, $1 = $1 + 1)
```

m4 macro processor สร้าง โค้ดเพื่อเพิ่มค่าอาร์กิวเมนต์แรกทีละ 1 คำสั่ง bump(x) เท่ากับ x = x + 1

แมโครมีจำนวนอาร์กิวเมนต์ได้ตามที่ต้องการ อย่างไรก็ตาม คุณสามารถเข้าถึงได้เพียงเก้าอาร์กิวเมนต์โดยใช้สัญลักษณ์ **$***n* ($1 ถึง $9) เมื่อต้องการเข้าถึงอาร์กิวเมนต์ผ่านเก้าอาร์กิวเมนต์ ให้ใช้แมโคร **shift**

แมโคร	**คำอธิบาย**
shift (*ParameterList***)**	ส่งกลับค่าทั้งหมดยกเว้นอิลิเมนต์แรกของ *ParameterList* เพื่อดำเนินการทำลาย shift ทางซ้ายของรายการ

แมโครนี้ลบอาร์กิวเมนต์แรกและกำหนดอาร์กิวเมนต์ที่เหลือใหม่ เป็นสัญลักษณ์ **$***n* symbols (อาร์กิวเมนต์ที่สอง เป็น $1, อาร์กิวเมนต์ที่สามเป็น $2 . . อาร์กิวเมนต์ที่สิบเป็น $9) การใช้แมโคร **shift** มากกว่าหนึ่งครั้งอนุญาตการเข้าถึงอาร์กิวเมนต์ทั้งหมด ที่ใช้กับแมโคร

แมโคร **$0** ส่งกลับชื่อ ของแมโคร อาร์กิวเมนต์ที่ไม่ได้กำหนดจะถูกแทนที่โดยสตริง null ดังนั้นคุณสามารถกำหนดแมโครที่เชื่อมอาร์กิวเมนต์ดังนี้:

```
define(cat, $1$2$3$4$5$6$7$8$9)
```

ดังนั้น:

```
cat(x, y, z)
```

เหมือนกับ:

```
xyz
```

อาร์กิวเมนต์ $4 ถึง $9 ในตัวอย่างนี้เป็น null เนื่องจากอาร์กิวเมนต์ ที่ตรงกันไม่ได้ถูกกำหนด

m4 macro processor ไม่ให้ความสำคัญ กับช่องว่างที่ไม่อยู่ในเครื่องหมายคำพูด แท็บ หรืออักขระบรรทัดใหม่ในอาร์กิวเมนต์ แต่เก็บค่า ช่องว่างอื่นทั้งหมด ดังนั้น:

```
define(a, b c)
```

กำหนด a เป็น b c

อาร์กิวเมนต์ถูกแยกโดยเครื่องหมายจุลภาค ใช้วงเล็บ เพื่อปิดล้อมอาร์กิวเมนต์ที่มีเครื่องหมายจุลภาค เพื่อที่เครื่องหมายจุลภาคจะไม่ จบอาร์กิวเมนต์ ตัวอย่างเช่น:

```
define(a, (b,c))
```

มีสองอาร์กิวเมนต์เท่านั้น อาร์กิวเมนต์แรกคือ a และทีสองคือ (b,c) เมื่อต้องการใช้เครื่องหมายจุลภาคหรือวงเล็บเดี่ยว ให้ปิดล้อมด้วยอักขระเครื่องหมายคำพูด

การใช้แมโคร m4 ที่กำหนดไว้ล่วงหน้า

m4 macro processor จัดเตรียมชุดของแมโคร ที่กำหนดไว้ล่วงหน้า ส่วนนี้อธิบายแมโครและการใช้งาน

การลบนิยามแมโคร

แมโคร	คำอธิบาย
undefine (`*MacroName*'**)**	ลบข้อกำหนดของแมโคร user-defined หรือ built-in (`*MacroName*')

ตัวอย่างเช่น:

```
undefine(`N')
```

ลบข้อกำหนดของ N หลังจากคุณลบแมโคร built-in ด้วยแมโคร **undefine** ดังนี้:

```
undefine(`define')
```

คุณไม่สามารถใช้ข้อกำหนดของแมโคร built-in ได้อีก

จำเป็นต้องมีเครื่องหมายคำพูดเดี่ยวในกรณีนี้เพื่อป้องกัน การแทนที่

การตรวจสอบสำหรับแมโครที่กำหนดไว้

แมโคร	คำอธิบาย
ifdef (`*MacroName*', *Argument1*, *Argument2***)**	ถ้าแมโคร *MacroName* ถูกกำหนด และไม่ได้กำหนดเป็น ศูนย์ ส่งกลับค่าของ *Argument1* มิฉะนั้น จะส่งกลับ *Argument2*

แมโคร **ifdef** อนุญาตสาม อาร์กิวเมนต์ ถ้าอาร์กิวเมนต์แรกถูกกำหนด ค่าของ **ifdef** คือ อาร์กิวเมนต์ที่สอง ถ้าอาร์กิวเมนต์แรกไม่ถูกกำหนด ค่าของ **ifdef** จะเป็นอาร์กิวเมนต์ที่สาม ถ้าไม่มีอาร์กิวเมนต์ที่สาม ค่า ของ **ifdef** จะเป็น null

การใช้คณิตศาสตร์เลขจำนวนเต็ม

m4 macro processor จัดเตรียม ฟังก์ชันในตัวดังต่อไปนี้ สำหรับการดำเนินการทางคณิตศาสตร์กับจำนวนเต็มเท่านั้น:

แมโคร	คำอธิบาย
incr (*Number***)**	ส่งกลับค่าของ *Number* + 1
decr (*Number***)**	ส่งกลับค่า *Number* – 1
eval	ประเมินนิพจน์เชิงคำนวณ

ดังนั้น เพื่อกำหนดตัวแปรมากกว่าค่า *Number* อยู่หนึ่ง ให้ใช้:

```
define(Number, 100)
define(Number1, `incr(Number)')
```

ซึ่งกำหนด Number1 มีค่ามากกว่าค่าปัจจุบันของ Number อยู่หนึ่ง

ฟังก์ชัน **eval** สามารถประเมิน ดังต่อไปนี้ที่มีตัวดำเนินการดังต่อไปนี้ (แสดงลำดับการมาก่อนจากมากไป หาน้อย) :

unary **+** และ **–**
** หรือ ^ (ยกกำลัง)
* **/** % (โมดูลัส)
+ –
== != < <= > >=
!(not)
& หรือ **&&** (โลจิคัล AND)
| หรือ || (โลจิคัล OR)

ใช้วงเล็บเพื่อจัดกลุ่มการดำเนินการเมื่อจำเป็น ตัวถูกดำเนินการทั้งหมดของนิพจน์ต้องเป็นตัวเลข ค่าตัวเลขของความสัมพันธ์ true (เช่น 1 > 0) คือ 1 และ false คือ 0 ความแม่นนำของฟังก์ชัน **eval** คือ 32 บิต

ตัวอย่างเช่น กำหนด M เป็น 2==N+1 โดยใช้ฟังก์ชัน **eval** ดังนี้:

```
define(N, 3)
define(M, `eval(2==N+1)')
```

นอกจากจะเป็นข้อความง่ายๆ ให้ใช้อักขระเครื่องหมายคำพูด รอบข้อความที่กำหนดแมโคร

จัดการไฟล์

เมื่อต้องการรวมไฟล์ใหม่ในอินพุต ใช้ฟังก์ชัน built-in **include**

แมโคร	คำอธิบาย
include (*File***)**	ส่งกลับเนื้อหาของไฟล์ *File*

ตัวอย่างเช่น:

```
include(FileName)
```

แทรกเนื้อหาของ FileName ในตำแหน่งของคำสั่ง **include**

ข้อผิดพลาดรุนแรงเกิดขึ้นถ้าไฟล์ที่ระบุในแมโคร **include** ไม่สามารถเข้าถึงได้ เพื่อหลีกเลี่ยงข้อผิดพลาดรุนแรง ให้ใช้รูปแบบทางเลือก แมโคร **sinclude** (silent include)

แมโคร	คำอธิบาย
sinclude (*File* **)**	ส่งกลับเนื้อหาของไฟล์ *File* แต่ไม่ รายงานข้อผิดพลาด ถ้าไม่สามารถเข้าถึง *File*

แมโคร **sinclude** (silent include) ไม่เขียน ข้อความ แต่ดำเนินต่อถ้าไม่สามารถเข้าถึงไฟล์ที่ระบุได้

การเปลี่ยนทิศทางเอาต์พุต

เอาต์พุตของ **m4** macro processor สามารถถูกเปลี่ยนทิศทางอีกครั้งไปที่ไฟล์ชั่วคราวระหว่างการประมวลผล และข้อมูลที่รวบรวม สามารถถูกเอาต์พุตบนคำสั่ง **m4** macro processor คงเก้าไฟล์ชั่วคราวที่เป็นไปได้ มีหมายเลข 1 ถึง 9 ถ้าคุณใช้ แมโคร built-in **divert**

แมโคร	คำอธิบาย
divert (*Number***)**	เปลี่ยนสตรีมเอาต์พุตไปที่ไฟล์ชั่วคราว *Number*

m4 macro processor เขียน เอาต์พุตทั้งหมดจากโปรแกรมหลังจากฟังก์ชัน **divert** ที่จุดสิ้นสุดของไฟล์ชั่วคราว *Number* เมื่อต้องการส่งกลับ เอาต์พุตไปที่จอแสดงผล ให้ใช้ฟังก์ชัน **divert** หรือ **divert(0)** ซึ่งดำเนิน การประมวลผลเอาต์พุตปกติต่อ

m4 macro processor เขียน เอาต์พุตที่เปลี่ยนทิศทางทั้งหมดไปที่ไฟล์ชั่วคราวในลำดับตัวเลข ที่ จุดสิ้นสุดของการประมวลผล **m4** macro processor ละเว้นเอาต์พุต ถ้าคุณเปลี่ยนทิศทางเอาต์พุตไปที่ไฟล์ชั่วคราวที่ไม่ใช่ 0 ถึง 9

เมื่อต้องการสำข้อมูลกลับจากไฟล์ชั่วคราวตาม ลำดับหมายเลข ให้ใช้แมโคร built-in **undivert**

แมโคร	คำอธิบาย
undivert (*Number1***, *Number2*...)**	ผนวกเนื้อหาของไฟล์ชั่วคราวที่ระบุไปที่ ไฟล์ชั่วคราวปัจจุบัน

เมื่อต้องการนำไฟล์ชั่วคราวที่เลือกกลับมาในลำดับที่ระบุ ให้ใช้แมโคร built-in **undivert** พร้อมกับแมโคร เมื่อใช้แมโคร **undivert, m4** macro processor ละเว้น ไฟล์ชั่วคราวที่ถูกกู้คืนและไม่ค้นหาข้อมูลที่กู้คืน สำหรับแมโคร

ค่าของแมโคร **undivert** ไม่ใช่ข้อความที่เปลี่ยนทิศทาง

คุณสามารถใช้แมโคร **divnum** เพื่อระบุว่าไฟล์ชั่วคราวใด ถูกใช้งานอยู่

แมโคร	คำอธิบาย
divnum	ส่งกลับค่าของไฟล์ชั่วคราวที่ใช้งานอยู่ขณะนี้

ถ้าคุณไม่ได้เปลี่ยนแปลงไฟล์เอาต์พุตด้วยแมโคร **divert, m4** macro processor นำเอาต์พุตทั้งหมดไปไว้ในไฟล์ชั่วคราว ชื่อ 0

การใช้โปรแกรมระบบในโปรแกรม

คุณสามารถรันโปรแกรมในระบบปฏิบัติการจาก โปรแกรมโดยใช้แมโคร built-in **syscmd** ตัวอย่าง คำสั่งดังต่อไปนี้รัน โปรแกรม **date**:

```
syscmd(date)
```

การใช้ unique file Nnames

ใช้แมโคร built-in **maketemp** เพื่อสร้างชื่อไฟล์เฉพาะจากโปรแกรม

แมโคร	คำอธิบาย
maketemp (*String...nnnnn...String***)**	สร้างชื่อไฟล์ไม่ซ้ำโดยแทนที่อักขระ *nnnnn* ในสตริงอาร์กิวเมนต์ด้วย process ID ปัจจุบัน

ตัวอย่างเช่น คำสั่ง:

```
maketemp(myfilennnnn)
```

m4 macro processor ส่งกลับ สตริงซึ่งคือ `myfile` ต่อกันกับ process ID ใช้สตริงนี้เพื่อตั้งชื่อไฟล์ชั่วคราว

การใช้นิพจน์แบบมีเงื่อนไข

การหาค่านิพจน์เงื่อนไขอนุญาตการกำหนดเวลาการประมวลผล ของนิพจน์แมโคร

นิพจน์	คำอธิบาย
ifelse (*String1*, *String2*, *Argument1*, *Argument2*)	ถ้า *String1* ตรงกับ *String2* ส่งกลับค่าของ *Argument1* มิฉะนั้น ส่งกลับ *Argument2*

แมโคร built-in **ifelse** ทำการ ทดสอบเงื่อนไข ในรูปแบบง่ายที่สุด:

```
ifelse(a, b, c, d)
```

เปรียบเทียบสองสตริง a และ b

ถ้า a และ b เหมือนกัน แมโคร built-in **ifelse** ส่งกลับสตริง c ถ้าไม่เหมือนกัน ส่งกลับสตริง d ตัวอย่างเช่น คุณสามารถกำหนดแมโครชื่อ compare เพื่อเทียบสองตริงและส่งกลับ yes ถ้ามีค่าเหมือนกัน หรือ no ถ้าค่า ต่างกันดังนี้:

```
define(compare, `ifelse($1, $2, yes, no)')
```

อักขระเครื่องหมายคำพูดป้องการการหาค่าแมโคร **ifelse** ไม่เห็นเกิดขึ้นก่อน ถ้าไม่มีอาร์กิวเมนต์ที่สี่ แมโครจะถือว่าว่างเปล่า

แมโคร **ifelse** มีอาร์กิวเมนต์เท่าใดก็ได้ เพื่อจัดเตรียมรูปแบบจำกัดของความสามารถการตัดสินใจ แบบหลายทางเลือก ตัวอย่างเช่น:

```
ifelse(a, b, c, d, e, f, g)
```

คำสั่งนี้โดยโลจิคัลเหมือนกับส่วนของโค้ด ดังต่อไปนี้:

```
if(a == b) x = c;
else if(d == e) x = f;
else x = g;
return(x);
```

ถ้าอาร์กิวเมนต์สุดท้ายถูกละเว้น ผลลัพธ์จะเป็น null, ดังนั้น:

```
ifelse(a, b, c)
```

คือ c ถ้า a ตรงกับ b และ null ถ้าเป็นอย่างอื่น

การจัดการกับสตริง

แมโครในส่วนนี้อนุญาตให้คุณแปลงสตริงอินพุตไปเป็น สตริงเอาต์พุต

แมโคร	คำอธิบาย
len	ส่งกลับความยาวเป็นไบต์ของสตริงที่เป็นอาร์กิวเมนต์

ดังนั้น:

```
len(abcdef)
```

คือ 6 และ:

```
len((a,b))
```

คือ 5

แมโคร	คำอธิบาย
dlen	ส่งกลับความยาวของอักขระที่แสดงผลได้ในสตริง

อักขระที่สร้างจากโค้ด 2 ไบต์ถูกสดงเป็นหนึ่งอักขระ ดังนั้น ถ้าสตริงมีขนาด 2 ไบต์เป็นอักขระที่สนับสนุนอักขระนานาชาติ ผลลัพธ์ของ **dlen** จะต่างจากผลของ **len**

แมโคร	คำอธิบาย
substr (*String*, *Position*, *Length***)**	ส่งกลับ substring ของ *String* ที่เริ่มต้นที่ หมายเลขอักขระ *Position* และมีความยาวอักขระ *Length*

การใช้อินพุต **substr (***s*, *i*, *n***)** ส่งกลับซับสตริงของ *s* ซึ่งเริ่มต้นที่ตำแหน่ง *i*th (เริ่มที่ศูนย์) และมีความยาวอักขระ *n* ถ้า *n* ถูกเว้นไว้ ส่วนที่เหลือของสตริงจะถูกส่งกลับ ตัวอย่างเช่นฟังก์ชัน:

```
substr(`now is the time',1)
```

ส่งกลับสตริงดังต่อไปนี้:

```
ow is the time
```

แมโคร	คำอธิบาย
index (*String1*, *String2***)**	ส่งกลับตำแหน่งอักขระใน *String1* ซึ่ง *String2* เริ่มต้น (เริ่มต้นด้วยหมายเลข อักขระ 0) หรือ -1 ถ้า *String1* ไม่มี *String2*

สำหรับแมโคร built-in **substr** จุดเริ่มต้นสำหรับ สตริงคือ 0

แมโคร	คำอธิบาย
translit (*String*, *Set1*, *Set2***)**	ค้นหา *String* สำหรับอักขระที่อยู่ใน *Set1* ถ้าพบ การเปลี่ยนแปลง (ถ่ายทอดตัวอักษร) อักขระที่ตรงกับอักขระใน *Set2*

มีรูปแบบทั่วไปดังนี้:

```
translit(s, f, t)
```

ซึ่งปรับเปลี่ยน s โดยแทนที่ อักขระที่พบใน f โดยอักขระที่ตรงกัน ของ t ตัวอย่างเช่น ฟังก์ชัน:

```
translit(`little', aeiou, 12345)
```

แทนที่สระด้วยหลักที่ตรงกัน และส่งคือค่าดังต่อไปนี้:

```
l3ttl2
```

ถ้า t สั้นกว่า f อักขระที่ไม่มีรายการใน t จะถูกลบ ถ้า t ไม่มีอยู่เลย อักขระจาก f จะถูกลบจาก s ดังนั้น:

```
translit(`little', aeiou)
```

ลบสระออกจากสตริง `little` และค่าส่งกลับเป็นดังนี้:

```
lttl
```

แมโคร	คำอธิบาย
dnl	ลบอักขระทั้งหมดที่ตามมา จนถึงอักขระ ขึ้นบรรทัดใหม่

ใช้แมโครนี้เพื่อลบบรรทัดว่าง ตัวอย่างเช่นฟังก์ชัน:

```
define(N, 100)
define(M, 200)
define(L, 300)
```

ผลลัพธ์บรรทัดใหม่ที่ตำแหน่งสุดท้ายของแต่ละบรรทัด ซึ่งไม่ได้เป็นส่วนหนึ่งของข้อกำหนด อักขระขึ้นบรรทัดใหม่เหล่านี้ถูกส่งผ่านไปที่ เอาต์พุต เพื่อลบบรรทัดใหม่ ให้เพิ่มแมโคร built-in **dnl** ให้กับแต่ละบรรทัด

```
define(N, 100) dnl
define(M, 200) dnl
define(L, 300) dnl
```

การดีบักแมโคร M4

แมโครในส่วนนี้อนุญาตให้คุณรายงานข้อผิดพลาดและประมวลผล ข้อมูล

แมโคร	คำอธิบาย
errprint (*String*)	เขียนอาร์กิวเมนต์ (*String*) ลงไฟล์ข้อผิดพลาด มาตรฐาน

ตัวอย่างเช่น:

```
errprint (`error')
```

แมโคร	คำอธิบาย
dumpdef (`*MacroName*'...)	ดัมพ์ชื่อและข้อกำหนดปัจจุบันของเทอมที่มีชื่อเป็นอาร์กิวเมนต์ (`*MacroName*'...)

ถ้าคุณไม่ได้กำหนดอาร์กิวเมนต์แมโคร **dumpdef** พิมพ์ชื่อและข้อกำหนดปัจจุบันทั้งหมด โปรดอย่าลืมใส่เครื่องหมายคำพูดให้กับชื่อ

แมโคร l m4 เพิ่มเติม

รายการของแมโคร **m4** เพิ่มเติมพร้อมกับคำอธิบายสั้นๆ ดังนี้:

แมโคร	คำอธิบาย
changecom (*l*, *r*)	เปลี่ยนอักขระข้อคิดเห็นซ้ายและขวาเป็นอักขระที่แสดง ด้วยตัวแปร *l* และ *r*
defn (*MacroName*)	ส่งกลับข้อกำหนดที่มีเครื่องหมายคำพูดของ *MacroName*
en (*String*)	ส่งกลับจำนวนของอักขระใน *String*
m4exit (*Code*)	ออกจาก **m4** macro processor ด้วยโค้ดส่งกลับ *Code*
m4wrap (*MacroName*)	รันแมโคร *MacroName* ที่จุดสิ้นสุดของ **m4** macro processor
popdef (*MacroName*)	แทนที่ข้อกำหนดปัจจุบันของ *MacroName* ด้วย ข้อกำหนดก่อนหน้าที่บันทึกด้วยแมโคร **pushdef**
pushdef (*MacroName*, *Replacement*)	บันทึกข้อกำหนดปัจจุบันของ *MacroName* จากนั้น กำหนด *MacroName* เป็น *Replacement*
sysval	รับรหัสที่ส่งกลับจากการใช้ครั้งล่าสุดของแมโคร **syscmd**
traceoff (*MacroList*)	ปิดการติดตามสำหรับแมโครทั้งหมดใน *MacroList* ถ้า *MacroList* เป็น null ปิดการติดตามทั้งหมด

แมโคร	คำอธิบาย
traceon (*MacroName***)**	เปิดการติดตามสำหรับแมโคร *MacroName* ถ้า *MacroName* เป็น null ให้เปิดการติดตามสำหรับแมโครทั้งหมด

Object data manager

Object Data Manager (ODM) คือตัวจัดการข้อมูลสำหรับเก็บ ข้อมูลระบบ ข้อมูลถูกเก็บและรักษาเป็นอ็อบเจ็กต์ ที่มีคุณลักษณะที่สัมพันธ์

คุณยังสามารถใช้ ODM เพื่อจัดการข้อมูลสำหรับโปรแกรม แอ็พพลิเคชันprograms.

ข้อมูลระบบที่จัดการโดย ODM รวมถึง:
- ข้อมูลการกำหนดค่าอุปกรณ์
- แสดงข้อมูลสำหรับ SMIT (เมนู, ตัวเลือก และกล่องโต้ตอบ)
- ข้อมูลสำคัญสำหรับการติดตั้งและโพรซีเดอร์อัพเดต
- ข้อมูลการกำหนดค่าการสื่อสาร
- ข้อมูลทรัพยากรระบบ

คุณสามารถ สร้าง เพิ่ม ล็อก เก็บ เปลี่ยน รับค่า แสดง ลบ และทิ้ง อ็อบเจ็กต์และคลาสอ็อบเจ็กต์ด้วย ODM คำสั่ง ODM จัดเตรียม อินเตอร์เฟสบรรทัดคำสั่งให้กับฟังก์ชันเหล่านี้ รูทีนย่อย ODM เข้าถึงฟังก์ชันเหล่านี้ จากภายในโปรแกรมแอ็พพลิเคชัน

บางคลาสอ็อบเจ็กต์ไม่มีมาพร้อมกับระบบ คลาสอ็อบเจ็กต์เหล่านี้มีการกล่าวถึงในเอกสารสำหรับผลิตภัณฑ์ระบบ ที่จัดเตรียมคลาสอ็อบเจ็กต์

หัวข้อนี้มีหัวข้อย่อยต่อไปนี้:
- ODM Object Classes และ Objects Storage
- ODM Descriptor
- ODM Object Searches
- คำสั่งและรูทีนย่อย ODM
- ตัวอย่าง ODM โค้ดและเอาต์พุต

คลาสอ็อบเจ็กต์ ODM และหน่วยเก็บข้อมูลอ็อบเจ็กต์

คอมโพเนนต์พื้นฐานของ ODM คือคลาสอ็อบเจ็กต์และ อ็อบเจ็กต์ เมื่อต้องการ จัดการกับคลาสอ็อบเจ็กต์และอ็อบเจ็กต์ คุณใช้คำสั่งและรูทีนย่อย ODM โดยเฉพาะ คุณใช้คุณลักษณะ การสร้างและ การเพิ่มของอินเตอร์เฟสเหล่านี้ในการสร้างคลาสอ็อบเจ็กต์และอ็อบเจ็กต์ สำหรับพื้นที่จัดเก็บข้อมูลและการจัดการข้อมูลของคุณเอง

คำศัพท์	คำอธิบาย
คลาสอ็อบเจ็กต์	กลุ่มอ็อบเจ็กต์ที่มีข้อกำหนดเหมือนกัน คลาสอ็อบเจ็กต์ประกอบด้วย หนึ่ง descriptors หรือมากกว่านั้น เหมือนกับตาราง แต่ละคลาสอ็อบเจ็กต์ที่คุณสร้าง ด้วยคำสั่ง **odmcreate** หรือรูทีนย่อย **odm_create_class** ถูกเก็บไว้ในไฟล์เป็นนิยามภาษา C ของอาร์เรย์ ของ structures
อ็อบเจ็กต์	สมาชิกของคลาสอ็อบเจ็กต์ เป็นเอนทิตีที่ต้องการพื้นที่จัดเก็บข้อมูล และการจัดการข้อมูล เหมือนกับเร็กคอร์ดเชิงตรรกะในฐานข้อมูล แต่ละอ็อบเจ็กต์ที่คุณเพิ่มให้กับคลาสอ็อบเจ็กต์ด้วยคำสั่ง **odmadd** หรือรูทีนย่อย **odm_add_obj** ถูกเก็บเป็นโครงสร้างภาษา C ในไฟล์เดียวกัน คุณกำหนดไดเร็กทอรีซึ่งเก็บไฟล์นี้ เมื่อคุณสร้างคลาสอ็อบเจ็กต์

คลาสอ็อบเจ็กต์โดยแนวคิดเหมือนกับอาร์เรย์ของ structures ที่มี แต่ละอ็อบเจ็กต์เป็น structure ที่เป็นอิลิเมนต์ของอาร์เรย์ ค่าถูกนำมาสัมพันธ์กับ descriptors ของอ็อบเจ็กต์ เมื่ออ็อบเจ็กต์ถูกเพิ่ม ให้กับคลาสอ็อบเจ็กต์ descriptors ของอ็อบเจ็กต์และค่าที่เกี่ยวข้อง สามารถถูกค้นหาและเปลี่ยนแปลงได้ด้วยหน่วยบริการ ODM

ต่อไปนี้เป็นตัวอย่างของการจัดการคลาสอ็อบเจ็กต์และอ็อบเจ็กต์

1. เมื่อต้องการสร้างคลาสอ็อบเจ็กต์ที่ชื่อ `Fictional_Characters` ให้ป้อน:

   ```
   class Fictional_Characters {
           char    Story_Star[20];
           char    Birthday[20];
           short   Age;
           char     Friend[20];
   };
   ```

   ในตัวอย่างนี้คลาสอ็อบเจ็กต์ `Fictional_Characters` มีสี่ descriptors: `Story_Star`, `Birthday`, และ `Friend` ซึ่งมีชนิด descriptor เป็น character และมีความยามสูงสุด 20 อักขระ; และ `Age` ซึ่งมีชนิด descriptor เป็น short เมื่อต้องการสร้างไฟล์คลาสอ็อบเจ็กต์ที่ต้องการโดย ODM คุณประมวลผลไฟล์นี้ ด้วยคำสั่ง **odmcreate** หรือรูทีนย่อย **odm_create_class**
2. เมื่อคุณสร้างคลาสอ็อบเจ็กต์ คุณสามารถ เพิ่มอ็อบเจ็กต์ให้กับคลาสโดยใช้คำสั่ง **odmadd** หรือ รูทีนย่อย **odm_add_obj** ตัวอย่างเช่น ป้อนโค้ดดังต่อไปนี้ ด้วยคำสั่ง **odmadd** เพื่อเพิ่มอ็อบเจ็กต์ `Cinderella` และ `Snow White` ให้กับ คลาสอ็อบเจ็กต์ `Fictional_Characters` ตามด้วย ค่าสำหรับ descriptors ที่สืบทอด:

   ```
   Fictional_Characters:
           Story_Star      = "Cinderella"
           Birthday = "Once upon a time"
           Age              = 19
           Friend          = "mice"

   Fictional_Characters:
           Story_Star      = "Snow White"
           Birthday = "Once upon a time"
           Age             = 18
           Friend          = "Fairy Godmother"
   ```

   ตาราง Fictional_Characters แสดงภาพแนวคิดของคลาสอ็อบเจ็กต์ `Fictional_Characters` พร้อมกับสองอ็อบเจ็กต์ที่เพิ่ม `Cinderella` และ `Snow White`

ตารางที่ 71. Fictional Characters

Story Star (char)	Birthday (char)	Age (short)	Friend (char)
Cinderella	Once upon a time	19	Mice
Snow White	Once upon a time	18	Fairy Godmother

```
Retrieved data for 'Story_Star = "Cinderella"'
     Cinderella:
          Birthday         = Once upon a time
          Age              = 19
          Friend = Mice
```

3. หลังจากสร้างคลาสอ็อบเจ็กต์ `Fictional_Characters` และเพิ่ม อ็อบเจ็กต์ `Cinderella` และ `Snow White` แล้ว ข้อมูลที่ถูกดึงสำหรับ `'Story_Star = "Cinderella"'` คือ:

```
Cinderella:
        Birthday         = Once upon a time
        Age              = 19
        Friend          = mice
```

การใช้คำสั่ง ODM

เมื่อใช้คำสั่ง **odmcreate** หรือ **odmdrop** เพื่อสร้างหรือลบคลาสอ็อบเจ็กต์ ให้ระบุไดเร็กทอรีซึ่งไฟล์ข้อกำหนด คลาสจะถูกเข้าถึง โดยใช้หนึ่งในคำสั่งดังต่อไปนี้:

1. เก็บไฟล์ในไดเร็กทอรีเริ่มต้นที่ระบุโดย **$ODMDIR**, ซึ่งคือไดเร็กทอรี **/etc/objrepos**
2. ใช้คำสั่ง **export** เพื่อ ตั้งค่าตัวแปรสภาวะแวดล้อม **ODMDIR** เพื่อระบุ ไดเร็กทอรีสำหรับจัดเก็บ
3. ใช้คำสั่ง **unset** เพื่อ unset ตัวแปรสภาวะแวดล้อม **ODMDIR** และคำสั่ง **cd** เพื่อเปลี่ยนไดเร็กทอรีปัจจุบันไปเป็นไดเร็กทอรีทีคุณ ต้องการเก็บคลาสอ็อบเจ็กต์ จากนั้นรันคำสั่ง ODM ในไดเร็กทอรีนั้น ไฟล์ กำหนดคลาสอ็อบเจ็กต์จะถูกเก็บในไดเร็กทอรีปัจจุบัน

เมื่อใช้คำสั่ง **odmdelete, odmadd, odmchange, odmshow,** หรือ **odmget** เพื่อทำงานกับคลาสและอ็อบเจ็กต์ ให้ระบุไดเร็กทอรีที่มีคลาสอ็อบเจกต์โดยใช้หนึ่งในคำสั่งดังต่อไปนี้:

1. ทำงานกับคลาสอ็อบเจ็กต์ในไดเร็กทอรีเริ่มต้นที่ระบุโดย **$ODMDIR** ซึ่งคือไดเร็กทอรี **/etc/objrepos**
2. ใช้คำสั่ง **export** เพื่อ ตั้งค่าตัวแปรสภาวะแวดล้อม **ODMDIR** เพื่อระบุ ไดเร็กทอรี
3. จากบรรทัดคำสั่งให้ใช้คำสั่ง **export** เพื่อ ตั้งค่าตัวแปรสภาวะแวดล้อม **ODMPATH** เป็นสตริงที่มี รายการ colon-separated ของไดเร็กทอรีที่จะถูกค้นหาคลาสอ็อบเจ็กต์ ตัวอย่างเช่น:

   ```
   $ export ODMPATH = /usr/lib/objrepos:/tmp/myrepos
   ```

   ไดเร็กทอรี ใน **$ODMPATH** ถูกค้นหาเฉพาะเมื่อ ไดเร็กทอรีที่ระบุโดย **$ODMDIR** ไม่มีใน คลาสอ็อบเจ็กต์

การสร้างคลาสอ็อบเจ็กต์

> **ข้อควรใส่ใจ:** การทำการเปลี่ยนแปลง กับไฟล์ที่กำหนดคลาสแฟ้มระบบและอ็อบเจ็กต์สามารถมีผลให้เกิด ปัญหากับระบบ ปรึกษาผู้ดูแลระบบของคุณก่อนการใช้ไดเร็กทอรี **/usr/lib/objrepos** เป็นไดเร็กทอรีที่เก็บคลาสอ็อบเจ็กต์ และอ็อบเจ็กต์

1. สร้างข้อกำหนดสำหรับหนึ่งคลาสอ็อบเจ็กต์หรือมากกว่านั้น ในไฟล์ ASCII “ตัวอย่างโค้ดและเอาต์พุต ODM” ในหน้า 631 แสดงไฟล์ ASCII ที่มี ข้อกำหนดคลาสอ็อบเจ็กต์
2. ระบุไดเร็กทอรีซึ่งจะเก็บ อ็อบเจ็กต์ที่สร้าง

″ODM Object Class and Object Storage″ กล่าวถึงเงื่อนไขที่ใช้ในเวลาสร้าง object-class เพื่อการ กำหนดไดเร็กทอรีที่เก็บคลาสอ็อบเจ็กต์และอ็อบเจ็กต์ที่สร้าง คลาสอ็อบเจ็กต์และอ็อบเจ็กต์ระบบส่วนใหญ่ถูกเก็บในไดเร็กทอรี **/usr/lib/objrepos**

สร้างคลาสอ็อบเจ็กต์ว่างโดยรันคำสั่ง **odmcreate** ด้วยไฟล์ ASCII ของข้อกำหนดคลาสอ็อบเจ็กต์ ที่ระบุเป็นไฟล์อินพุต *ClassDescriptionFile*

การเพิ่มอ็อบเจ็กต์เข้ากับคลาสอ็อบเจ็กต์

> **ข้อควรใส่ใจ:** การทำการเปลี่ยนแปลง กับไฟล์ที่กำหนดคลาสแฟ้มระบบและอ็อบเจ็กต์สามารถมีผลให้เกิด ปัญหากับระบบ ปรึกษาผู้ดูแลระบบของคุณก่อนการใช้ไดเร็กทอรี **/usr/lib/objrepos** เป็นไดเร็กทอรีที่เก็บคลาสอ็อบเจ็กต์ และอ็อบเจ็กต์

1. สร้างคลาสอ็อบเจ็กต์ให้กับอ็อบเจ็กต์ซึ่งจะ ถูกเพิ่ม โปรดดูที่ การสร้างคลาสอ็อบเจ็กต์ สำหรับคำแนะนำ เกี่ยวกับการสร้างคลาสอ็อบเจ็กต์
2. สร้างข้อกำหนดสำหรับหนึ่งอ็อบเจ็กต์หรือมากกว่านั้น “ตัวอย่างโค้ดและเอาต์พุต ODM” ในหน้า 631แสดงไฟล์ ASCII ที่มีหลายข้อกำหนดอ็อบเจ็กต์
3. ระบุไดเร็กทอรีซึ่งมีคลาสอ็อบเจ็กต์ ซึ่งอ็อบเจ็กต์ที่สร้างจะถูกเก็บ

   ″ODM Object Class and Object Storage″ กล่าวถึงเงื่อนไขที่ใช้ในเวลาสร้างคลาสอ็อบเจ็กต์เพื่อการ กำหนดไดเร็กทอรีที่เก็บคลาสอ็อบเจ็กต์และอ็อบเจ็กต์ที่สร้าง คลาสอ็อบเจ็กต์และอ็อบเจ็กต์ระบบส่วนใหญ่ถูกเก็บในไดเรกทอรี **/usr/lib/objrepos**
4. เพิ่มอ็อบเจ็กต์ใหกับคลาสอ็อบเจ็กต์โดยรันคำสั่ง **odmadd** ด้วยไฟล์ ASCII ของข้อกำหนดอ็อบเจ็กต์ ที่ระบุเป็นไฟล์อินพุต *InputFile*

การล็อกคลาสอ็อบเจ็กต์

ODM ไม่ได้ล็อกคลาสอ็อบเจ็กต์หรืออ็อบเจ็กต์อย่างเป็นนัย การทำงานร่วมกันของการล็อกและการปลดล็อกเป็นความรับผิดชอบของแอ็พพลิเคชัน ที่เข้าถึงคลาสอ็อบเจ็กต์ อย่างไรก็ตาม ODM มีรูทีนย่อย **odm_lock** และ **odm_unlock** ในการควบคุมการล็อกและปลดล็อกคลาสอ็อบเจ็กต์โดยโปรแกรมแอ็พพลิเคชัน

รูทีนย่อย	คำอธิบาย
odm_lock	ประมวลผลสตริงที่เป็นชื่อพาธและสามารถแยกแยะไฟล์คลาส อ็อบเจ็กต์หรือไดเร็กทอรีของคลาสอ็อบเจ็กต์ ซึ่งส่งกลับตัวระบุล็อก และตั้งค่าแฟล็กเพื่อบ่งชี้คลาสอ็อบเจ็กต์ที่ระบุหรือคลาสที่กำหนด โดยชื่อพาธถูกใช้งานอยู่

เมื่อรูทีนย่อย **odm_lock** ตั้งค่าแฟล็กล็อก รูทีนจะไม่ปิดใช้งานการใช้คลาสอ็อบเจ็กต์โดยกระบวนการอื่น ถ้าการชนกันของการใช้งานเป็นปัญหาที่เกิดขึ้นได้ โปรแกรมแอ็พพลิเคชันควรรอ จนกว่า จะได้รับล็อกบนคลาสก่อนการใช้คลาส

แอ็พพลิเคชันอื่นไม่สามารถรับล็อกบนชื่อพาธเดียวกัน ขณะที่การล็อกมีผลอยู่ อย่างไรก็ตาม ล็อกบนชื่อไดเร็กทอรีไม่ได้ป้องกัน แอ็พพลิเคชันอื่นในการรับล็อกบนไดเร็กทอรีย่อยหรือ ไฟล์ภายในไดเร็กทอรีนั้น

เมื่อต้องการปลดล็อกคลาสอ็อบเจ็กต์ที่ล็อก ให้ใช้ **odm_unlock** ที่เรียกกับตัวระบุล็อกที่ส่งกลับโดยรูทีนย่อย **odm_lock**

การเก็บคลาสอ็อบเจ็กต์และอ็อบเจ็กต์

แต่ละคลาสอ็อบเจ็กต์ที่คุณสร้าง ด้วยคำสั่ง **odmcreate** หรือรูทีนย่อย **odm_create_class** ถูกเก็บไว้ในไฟล์เป็นนิยามภาษา C ของอาร์เรย์ ของ structures แต่ละอ็อบเจ็กต์ที่คุณเพิ่มให้กับคลาสอ็อบเจ็กต์ด้วยคำสั่ง **odmadd** หรือรูทีนย่อย **odm_add_obj** ถูกเก็บเป็นโครงสร้างภาษา C ในไฟล์เดียวกัน

คุณกำหนดไดเรกทอรีซึ่งเก็บไฟล์นี้ เมื่อคุณสร้างคลาสอ็อบเจ็กต์

เมธอดของพื้นที่จัดเก็บข้อมูลต่างกันไปตามลักษณะว่า คำสั่งหรือรูทีนย่อยถูกใช้เพื่อสร้างคลาสอ็อบเจ็กต์และอ็อบเจ็กต์

> **ข้อควรใส่ใจ:** การทำการเปลี่ยนแปลง กับไฟล์ที่กำหนดคลาสแฟ้มระบบและอ็อบเจ็กต์สามารถมีผลให้เกิด ปัญหากับระบบ ปรึกษาผู้ดูแลระบบของคุณก่อนการใช้ไดเร็กทอรี **/usr/lib/objrepos** เป็นไดเร็กทอรีที่เก็บคลาสอ็อบเจ็กต์ และอ็อบเจ็กต์

การใช้รูทีนย่อย odm_create_class หรือ odm_add_obj

รูทีนย่อย **odm_create_class** หรือ **odm_add_obj** ถูกใช้ เพื่อสร้างคลาสอ็อบเจ็กต์และอ็อบเจ็กต์:

- ถ้ามีข้อกำหนดที่เจาะจงสำหรับแอ็พพลิเคชันของคุณ ในการเก็บคลาสอ็อบเจ็กต์ ที่ไม่เป็นไปตามที่ระบุโดยตัวแปรสภาวะแวดล้อม **ODMDIR** ให้ใช้รูทีนย่อย **odm_set_path** เพื่อรีเซ็ตพาธ ขอแนะนำให้คุณใช้รูทีนย่อยนี้ในการตั้งค่า พาธพื้นที่จัดเก็บข้อมูลที่ชัดเจน เมื่อใดก็ตามที่สร้างคลาสอ็อบเจ็กต์หรืออ็อบเจ็กต์ จากแอ็พพลิเคชัน

  OR
- ก่อนรันแอ็พพลิเคชันของคุณ ให้ใช้คำสั่ง **set** จากบรรทัดคำสั่งเพื่อตั้งค่าตัวแปรสภาวะแวดล้อม **ODMDIR** เพื่อระบุไดเร็กทอรีสำหรับพื้นที่จัดเก็บข้อมูล

  OR
- เก็บไฟล์ในที่เก็บอ็อบเจ็กต์ที่ใช้ เพื่อเก็บคลาสอ็อบเจ็กต์ระบบต่างๆ ไดเร็กทอรี **/usr/lib/objrepos**

ODM Descriptors

Object Data Manager (ODM) descriptor เป็นแนวคิดเหมือน กับตัวแปรที่มีชื่อและชนิด เมื่อมีการสร้างคลาสอ็อบเจ็กต์ descriptors จะถูกกำหนดเหมือนชื่อตัวแปรพร้อมกับชนิด ODM descriptor ที่เกี่ยวข้อง เมื่ออ็อบเจ็กต์ถูกเพิ่มให้กับคลาสอ็อบเจ็กต์ จะได้รับสำเนาของ descriptors ทั้งหมดของคลาสอ็อบเจ็กต์ ค่ายังถูกเชื่อมโยงกับอ็อบเจ็กต์ descriptors ที่ได้กล่าวไปแล้ว

ODM สนับสนุนชนิด descriptor หลายแบบ:

Descriptor	นิยาม
terminal descriptor	กำหนดชนิดข้อมูล อักขระหรือตัวเลข
link descriptor	กำหนดความสัมพันธ์ระหว่างคลาสอ็อบเจ็กต์
method descriptor	กำหนดการดำเนินการหรือเมธอดสำหรับอ็อบเจ็กต์

ใช้ descriptors ของอ็อบเจ็กต์และค่าที่เกี่ยวข้องเพื่อกำหนด เกณฑ์สำหรับเรียกข้อมูลแต่ละอ็อบเจ็กต์จากคลาสอ็อบเจ็กต์ จัดรูปแบบเกณฑ์การเลือกที่คุณผ่านไปยัง ODM ตามที่กำหนดไว้ใน ODM Object Searches อย่าใช้ **binary** terminal descriptor ในเกณฑ์การค้นหา เนื่องจากความยาวที่กำหนดไม่ได้

ODM terminal descriptors

Terminal descriptors กำหนด ชนิดข้อมูลพื้นฐานที่สุดที่ใช้โดย ODM terminal descriptor โดยปกติแล้วคือ ตัวแปรที่กำหนดด้วยชนิด ODM terminal descriptor ชนิด terminal descriptor ที่จัดเตรียมโดย ODM มีดังนี้:

Descriptor	นิยาม
short	ระบุตัวเลข 2-ไบต์แบบมีเครื่องหมาย
long	ระบุตัวเลข 4-ไบต์แบบมีเครื่องหมาย
ulong	ระบุตัวเลข 4-ไบต์แบบไม่มีเครื่องหมาย
binary	ระบุบิตสตริงความยาวคงที่ ชนิด binary terminal descriptor ถูกกำหนดโดยผู้ใช้เมื่อสร้าง ODM ชนิด binary terminal descriptor ไม่สามารถใช้ได้ในเกณฑ์การเลือก
char	ระบุความยาวคงที่ สตริงที่ปิดท้ายด้วย null
vchar	ระบุความตัวแปร สตริงที่ปิดท้ายด้วย null ชนิด **vchar** terminal descriptor สามารถถูกใช้ในเกณฑ์การเลือก
long64/ODM_LONG_LONG/int64	ระบุตัวเลข 8-ไบต์แบบมีเครื่องหมาย
ulong64/ODM_ULONG_LONG/uint64	ระบุตัวเลข 8-ไบต์แบบไม่มีเครื่องหมาย

ODM link descriptor

ODM *link descriptor* สร้าง ความสัมพันธ์ระหว่างคลาสอ็อบเจ็กต์และอ็อบเจ็กต์ใน คลาสอ็อบเจ็กต์อื่น link descriptor เป็นตัวแปรที่กำหนดด้วยชนิด ODM link descriptor

ตัวอย่างเช่น โค้ดดังต่อไปนี้สามารถถูกประมวลผลโดย ODM สร้างหน่วยบริการเพื่อสร้างคลาสอ็อบเจ็กต์ **Friend_Table** และ **Fictional_Characters**:

```
class Friend_Table {
        char     Friend_of[20];
        char     Friend[20];
};

  class Fictional_Characters {
        char    Story_Star[20];
        char    Birthday[20];
        short   Age;
        link    Friend_Table Friend_Table Friend_of Friends_of;
};
```

คลาสอ็อบเจ็กต์ **Fictional_Characters** ใช้ link descriptor เพื่อสร้าง **Friends_of** descriptors link ไปที่คลาสอ็อบเจ็กต์ Friend_Table เพื่อแยกแยะ link, **Friends_of** descriptor เรียกข้อมูลอ็อบเจ็กต์ ในคลาสอ็อบเจ็กต์ **Friend_Table** ที่ข้อมูลตรงกับใน **Friend_of** descriptors link descriptor ใน คลาสอ็อบเจ็กต์ **Fictional_Characters** กำหนดคลาสอ็อบเจ็กต์ ที่เชื่อมโยงไปที่ (**Friend_Table**), descriptor ถูกเชื่อมโยง ไปที่ (**Friend_of**), และชื่อของ link descriptor (**Friends_of**) ในคลาสอ็อบเจ็กต์ **Fictional_Characters**

โค้ดดังต่อไปนี้สามารถถูกใช้เพื่อเพิ่มอ็อบเจ็กต์ให้กับ คลาสอ็อบเจ็กต์ **Fictional_Characters** และ **Friend_Table**:

```
Fictional_Characters:
        Story_Star      = "Cinderella"
        Birthday = "Once upon a time"
        Age              = 19
        Friends_of      = "Cinderella"
```

```
Fictional_Characters:
        Story_Star      = "Snow White"
        Birthday = "Once upon a time"
        Age             = 18
        Friends_of      = "Snow White"

Friend_Table:
        Friend_of = "Cinderella"
        Friend          = "Fairy Godmother"

Friend_Table:
        Friend_of = "Cinderella"
        Friend          = "mice"

Friend_Table:
        Friend_of       = "Snow White"
        Friend          = "Sneezy"

Friend_Table:
        Friend_of       = "Snow White"
        Friend          = "Sleepy"

Friend_Table:
        Friend_of = "Cinderella"
        Friend          = "Prince"

Friend_Table:
        Friend_of       = "Snow White"
        Friend          = "Happy"
```

ตารางดังต่อไปนี้แสดงภาพแนวคิดของ คลาสอ็อบเจ็กต์ Fictional_Characters และ Friend_Table อ็อบเจ็กต์ที่เพิ่มให้กับคลาส และความสัมพันธ์ของการเชื่อมโยง ระหว่างคลาส

Story_Star (char)	Birthday (char)	Age (short)	Friends_of (link)
Cinderella	Once upon a time	19	Cinderella
Snow White	Once upon a time	18	Snow White

```
Retrieved data for 'Story_Star = "Cinderella"
     Cinderella:
          Birthday         = Once upon a time
          Age              = 19
          Friends_of       = Cinderella
          Friend_of        = Cinderella
```

มีการเชื่อมโยงโดยตรงระหว่างคอลัมน์ "Friends_of" และ "Friend_of" ของสองตาราง ตารางดังต่อไปนี้ จัดเตรียมภาพแนวคิดของความสัมพันธ์ของการเชื่อมโยงระหว่างสอง คลาสอ็อบเจ็กต์

Friend_of (char)	Friend (char)
Cinderella	Fairy Godmother
Cinderella	mice
Snow White	Sneezy
Snow White	Sleepy
Cinderella	Prince
Snow White	Happy

หลังจากคลาสอ็อบเจ็กต์ **Fictional_Characters** และ **Friend_Table** ถูกสร้างและอ็อบเจ็กต์ ถูกเพิ่ม ข้อมูลที่เรียกคืนสำหรับ `Story_Star = 'Cinderella'` จะเป็น:

```
Cinderella:
        Birthday         = Once upon a time
        Age              = 19
        Friends_of       = Cinderella
        Friend_of        = Cinderella
```

เมื่อต้องการดูความสัมพันธ์ที่ขยายระหว่างคลาสอ็อบเจ็กต์ที่เชื่อมโยง ให้ใช้คำสั่ง **odmget** บน คลาสอ็อบเจ็กต์ **Friend_Table** ข้อมูลที่เรียกคืน สำหรับคลาสอ็อบเจ็กต์ `Friend_of = 'Cinderella'` จะเป็น:

```
Friend_Table:
        Friend_Of = "Cinderella"
        Friend          = "Fairy Godmother"

Friend_Table:
        Friend_of = "Cinderella"
        Friend= "mice"

Friend_Table:
        Friend_of = "Cinderella"
        Friend          = "Prince"
```

ODM method descriptor

ODM *method descriptor* ให้ ข้อกำหนดของคลาสอ็อบเจ็กต์กับอ็อบเจ็กต์ที่สามารถมีเมธอดหรือการดำเนินการ ที่เชื่อมโยง เมธอด descriptor เป็นตัวแปรที่กำหนดด้วยชนิด ODM method descriptor

การดำเนินการหรือค่า method descriptor สำหรับอ็อบเจ็กต์ เป็นสตริงอักขระที่เป็นได้ทั้ง คำสั่ง โปรแกรม หรือสคริปต์ shell ที่รัน โดยการเรียกเมธอด เมธอดหรือการดำเนินการอื่นสามารถถูกกำหนดสำหรับ แต่ละอ็อบเจ็กต์ในคลาสอ็อบเจ็กต์ ตัวการดำเนินการไม่ได้เป็นส่วนหนึ่งของ ODM การดำเนินการถูกกำหนดและโค้ดโดยแอ็พพลิเคชันโปรแกรมเมอร์

เมธอดสำหรับอ็อบเจ็กต์ถูกเรียกโดยการเรียกไปที่รูทีนย่อย **odm_run_method** การเรียกเมธอดเป็นเหตุการณ์ซิงโครนัส ทำให้การดำเนินการ ODM หยุดชั่วคราว จนกว่าการดำเนินการจะสมบูรณ์

ตัวอย่างเช่น โค้ดดังต่อไปนี้สามารถถูกอินพุตให้กับ ODM สร้างหน่วยบริการเพื่อสร้างคลาสอ็อบเจ็กต์ **Supporting_Cast_Ratings**:

```
class Supporting_Cast_Ratings {
        char    Others[20];
        short   Dexterity;
        short   Speed;
        short   Strength;
        method  Do_This;
};
```

ในตัวอย่าง Do_This descriptor เป็น descriptor เมธอดที่กำหนดสำหรับคลาสอ็อบเจ็กต์ **Supporting_Cast_Ratings** ค่าของ descriptor เมธอด สามารถเป็นสตริงที่ระบุ คำสั่ง โปรแกรม หรือสคริปต์ shell สำหรับการเรียก ใช้ในอนาคตโดยรูทีนย่อย **odm_run_method**

โค้ดดังต่อไปนี้เป็นตัวอย่างวิธีเพิ่มอ็อบเจ็กต์ ให้กับ คลาสอ็อบเจ็กต์**Supporting_Cast_Ratings**:

```
Supporting_Cast_Ratings:
        Others          = "Sleepy"
        Dexterity       = 1
        Speed           = 1
        Strength        = 3
        Do_This   = "echo Sleepy has speed of 1"

Supporting_Cast_Ratings:
        Others          = "Fairy Godmother"
        Dexterity       = 10
        Speed           = 10
        Strength        = 10
        Do_This   = "odmget -q "Others='Fairy Godmother'" Supporting_Cast_Ratings"
```

ตารางดังต่อไปนี้แสดงภาพแนวคิดของ คลาสอ็อบเจ็กต์ Supporting_Cast_Ratings ที่มี descriptor เมธอด Do_This และการดำเนินการที่สัมพันธ์ กับแต่ละอ็อบเจ็กต์ในคลาส

Others (char)	Dexterity (short)	Speed (short)	Stength (short)	Do_This (method)
Sleepy	1	1	3	echo Sleepy has speed of 1
Fairy Godmother	10	10	10	odmget –q "Others='Fairy Godmother'"Supporting_Cast_Ratings"

```
odm_run_method  run of Sleepy's method displays
(using echo):
"Sleepy has speed of 1"
```

หลังจากคลาสอ็อบเจ็กต์ Supporting_Cast_Ratings ถูกสร้างและอ็อบเจ็กต์ถูกเพิ่ม การเรียก (โดย รูทีนย่อย **odm_run_method**) ของเมธอดที่กำหนด สำหรับ Sleepy จะทำให้คำสั่ง **echo** แสดง:

```
Sleepy has speed of 1
```

ODM object searches

รูทีน ODM จำนวนมากที่อ็อบเจ็กต์หนึ่งหรือมากกว่านั้น ในคลาสอ็อบเจ็กต์ที่ระบุถูกเลือกเพื่อประมวลผล คุณสามารถรวมเกณฑ์การค้นหา ในรูปแบบของ qualifiers เมื่อคุณเลือกอ็อบเจ็กต์ที่มีรูทีน

qualifier

พารามิเตอร์สตริงที่ปิดท้ายด้วย null ในการเรียกรูทีนย่อย ODM ซึ่งให้ เกณฑ์ที่เหมาะสมสำหรับอ็อบเจ็กต์ที่จะเรียกข้อมูล

ชื่อ descriptor และเกณฑ์ความถูกต้องที่ระบุโดยพารามิเตอร์นี้ กำหนดอ็อบเจ็กต์ในคลาสอ็อบเจ็กต์ที่จะถูกเลือกสำหรับการประมวลผลภายหลัง แต่ละ qualifier มีหนึ่ง *เพรดิเคต* หรือมากกว่านั้นที่เชื่อมต่อ กับตัวดำเนินการเชิงตรรกะ แต่ละเพรดิเคตประกอบด้วยชื่อ descriptor, comparison operator และค่าคงที่

qualifier กับสามเพรดิเคตที่รวมกันโดยสองตัวดำเนินการเชิงตรรกะ ดังนี้:

```
SUPPNO=30 AND (PARTNO>0 AND PARTNO<101)
```

ในตัวอย่างนี้ สตริงทั้งหมดถูกพิจารณาเป็น qualifier สมาเพรดีเคตคือ `SUPPNO=30`, `PARTNO>0` และ `PARTNO<101` และใช้ตัวดำเนินการเชิงตรรกะ AND เพื่อ รวมเพรดีเคต ในเพรดิเคต แรก `SUPPNO` คือชื่อของ descriptor ในอ็อบเจ็กต์ = (เครื่องหมายเท่ากับ) คือ comparison operator และ `30` คือค่าคงที่เทียบกับ ค่าของ descriptor ที่ถูกเปรียบเทียบ

แต่ละเพรดิเคตระบุการทดสอบที่ใช้กับ descriptor ที่ถูกกำหนดสำหรับแต่ละอ็อบเจ็กต์ในคลาสอ็อบเจ็กต์ การทดสอบคือการเปรียบเทียบ ระหว่างค่าของ descriptor ของอ็อบเจ็กต์และค่าคงที่ที่ระบุ เพรดิเคตแรกในตัวอย่างแสดง = (เท่ากับ) เปรียบเทียบระหว่าง ค่าของ descriptor (`SUPPNO`) และค่าคงที่ (`30`)

ส่วนของ qualifier ภายในวงเล็บ

```
PARTNO>0 AND PARTNO<101
```

มีสองเพรดิเคตที่รวมโดยตัวดำเนินการเชิงตรรกะ AND `PARTNO` descriptor ถูกทดสอบสำหรับ ค่าที่มากกว่า `0` ในเพรดิเคตแรก จากนั้นทดสอบสำหรับค่าที่น้อยกว่า `101` ในเพรดิเคตที่สอง จากนั้นสอง เพรดิเคตถูกเชื่อมกันทางตรรกะ เพื่อกำหนดค่าสำหรับส่วนของ qualifier ตัวอย่างเช่น ถ้า `PARTNO` เป็นชื่อ descriptor สำหรับหมายเลขชิ้นส่วนในบริษัทคลังสินค้า ดังนั้นชิ้นส่วนนี้ของ qualifier จะกำหนดการเลือกสำหรับผลิตภัณฑ์ทั้งหมดที่มีหมายเลขชิ้นส่วนมากกว่า `0` และ น้อยกว่า `101`

ในตัวอย่างอื่น, qualifier:

```
lname='Smith' AND Company.Dept='099' AND Salary<2500
```

สามารถถูกใช้เพื่อเลือกทุกคน (ใน ODM อ็อบเจ็กต์ทั้งหมด) ที่มีนามสกุล Smith ซึ่งอยู่ใน Department `099` และมีเงินเดือนน้อยกว่า `$2500` กรุณาสังเกตว่าชื่อ `Dept` descriptor ถูกระบุด้วย คลาสอ็อบเจ็กต์ `Company` เพื่อสร้าง descriptor เฉพาะ

ชื่อ Descriptor ใน ODM predicates

ใน ODM ชื่อ descriptor ไม่จะเป็นต้องเป็นชื่อเฉพาะ คุณสามารถชื่อ descriptor ในคลาสอ็อบเจ็กต์มากกว่าหนึ่งคลาสอ็อบเจ็กต์ ในกรณีนี้ คุณระบุชื่อคลาสอ็อบเจ็กต์ตามด้วยชื่อ descriptor ในเพรดิเคตเพื่อสร้างการอ้างอิงเฉพาะไปที่ descriptor

ตัวดำเนินการเปรียบเทียบใน ODM predicates

ข้อมูลต่อไปนี้เป็น comparison operators ที่ใช้ได้:

ตัวดำเนินการ	นิยาม
=	เท่ากับ
!=	ไม่เท่ากับ
>	มากกว่า
>=	มากกว่าหรือเท่ากับ
<	น้อยกว่า
<=	น้อยกว่าหรือเท่ากับ
LIKE	เหมือนกัน ค้นหารูปแบบในข้อมูลสตริงอักขระ

การเปรียบเทียบสามารถทำได้เฉพาะระหว่าง ชนิดข้อมูลที่เข้ากันได้

ตัวดำเนินการเปรียบเทียบ LIKE

ตัวดำเนินการ LIKE ทำให้สามารถค้นหารูปแบบ ภายในชนิด char descriptor ตัวอย่างเช่น เพรดิเคต:

```
NAME LIKE 'ANNE'
```

กำหนดการค้นหาสำหรับค่า ANNE ใน NAME descriptor ในแต่ละอ็อบเจ็กต์ในคลาสอ็อบเจ็กต์ที่ระบุ ในกรณีนี้ ตัวอย่างเท่ากับ:

```
NAME = 'ANNE'
```

คุณยังสามารถใช้ตัวดำเนินการ LIKE กับ อักขระและระเบียบ การจับคู่รูปแบบดังต่อไปนี้:

- ใช้ ? (เครื่องหมายคำถาม) เพื่อแสดงอักขระ เดี่ยว ตัวอย่างเพรดิเคต:

  ```
  NAME LIKE '?A?'
  ```

  กำหนดการค้นหาสำหรับอักขระสามอักขระที่มี *A* เป็นอักขระที่สองในค่าของ NAME descriptor ของอ็อบเจ็กต์ ค่า descriptor PAM, DAN, และ PAT ตรงกับเกณฑ์การค้นหานี้
- ใช้ * (เครื่องหมายดอกจัน) เพื่อแสดงสตริง ศูนย์หรือมากกว่านั้น ตัวอย่างเพรดิเคต:

  ```
  NAME LIKE '*ANNE*'
  ```

  กำหนดการค้นหาสำหรับสตริงที่มีค่า ANNE ใน NAME descriptor ของอ็อบเจ็กต์ ค่า descriptor LIZANNE, ANNETTE และ ANNE ทั้งหมดตรงกับ เกณฑ์การค้นหานี้
- ใช้ [] (วงเล็บเหลี่ยม) เพื่อจับคู่กับอักขระ ที่อยู่ภายในวงเล็บ ตัวอย่างเพรดิเคต:

  ```
  NAME LIKE '[ST]*'
  ```

  กำหนดการค้นหาสำหรับค่า descriptor ที่เริ่มต้นด้วย *S* หรือ *T* ใน NAME descriptor ของอ็อบเจ็กต์
  ใช้ – (เครื่องหมายลบ) เพื่อระบุช่วงของอักขระ ตัวอย่างเพรดิเคต:

  ```
  NAME LIKE '[AD-GST]*'
  ```

  กำหนดการค้นหา สำหรับค่า descriptor ที่เริ่มต้นด้วยอักขระ *A*, *D*, *E*, *F*, *G*, *S*, หรือ *T*
- ใช้ [!] (เครื่องหมายตกใจในวงเล็บเหลี่ยม) เพื่อจับคู่อักขระเดี่ยว ยกเว้นที่ปิดด้วยวงเล็บ ตัวอย่างเพรดิเคต:

  ```
  NAME LIKE '[!ST]*'
  ```

  กำหนด การค้นหาสำหรับค่า descriptor ยกเว้นที่เริ่มต้นด้วย *S* หรือ *T* ใน NAME descriptor ของอ็อบเจ็กต์
  คุณสามารถใช้อักขระและระเบียบ การจับคู่รูปแบบโดยใช้สตริงรวมกันได้ทั้งหมด

ค่าคงที่ใน ODM predicates

ค่าคงที่ที่ระบุเป็นได้ทั้งค่าคงที่ตัวเลข หรือค่าคงที่สตริงอักขระ:

1. ค่าคงที่ตัวเลขใน ODM predicates ประกอบด้วยเครื่องหมายเป็นทางเลือกที่ตามด้วย ตัวเลข (มีหรือไม่มีจุดทศนิยม) เป็นทางเลือกสามารถใช้การยกกำลัง ที่ทำเครื่องหมายโดยตัวอักษณ *E* หรือ *e* ถ้าใช้ ตัวอักษร *E* หรือ *e* ต้อง ตามด้วยเลขกำลังที่กำหนดเครื่องหมายได้
   ค่าคงที่ ตัวเลขที่ใช้ได้มีดังนี้:

   ```
   2            2.545   0.5   -2e5   2.11E0
   +4.555e-10   4E0     -10    999   +42
   ```

   เลขกำลัง E0 สามารถใช้เพื่อระบุว่าไม่มี การยกกำลัง

2.
   ค่าคงที่สตริงอักขระต้องถูกปิดด้วย เครื่องหมายคำพูดเดี่ยว:

   ```
   'smith'   '91'
   ```

   ค่าคงที่สตริงอักขระทั้งหมดพิจารณา ว่ามีความยาวไม่คงที่ เมื่อต้องการแสดงเครื่องหมายคำพูดเดี่ยวภายในค่าคงที่สตริง ให้ใช้เครื่องหมายคำพูดเดี่ยว ตัวอย่างเช่น:

   ```
   'DON''T GO'
   ```

   ถูกแปลเป็น:

   ```
   DON'T GO
   ```

ตัวดำเนินการเชิงตรรกะ AND สำหรับเพรดิเคต

ตัวดำเนินการเชิงตรรกะ AND ใช้ได้กับเพรดิเคต ใช้ AND หรือ and สำหรับตัวดำเนินการเชิงตรรกะ AND

ตัวดำเนินการเชิงตรรกะ AND เชื่อมต่อสองเพรดิเคตหรือมากกว่านั้น ตัวอย่าง qualifier:

```
predicate1 AND predicate2 AND predicate3
```

ระบุ `predicate1` ต่อกันทางตรรกะกับ `predicate2` ตามด้วยผลลัพธ์ซึ่งคือการเชื่อมต่อกันทางตรรกะกับ `predicate3`

คำสั่งและรูทีนย่อย ODM

คุณสามารถ สร้าง เพิ่ม เปลี่ยน เรียกข้อมูล แสดง ลบ และเอาอ็อบเจ็กต์ และคลาสอ็อบเจ็กต์ออก ด้วย ODM คุณป้อนคำสั่ง ODM บนบรรทัดคำสั่ง

คุณสามารถใส่รูทีนย่อย ODM ในโปรแกรมภาษา C เพื่อจัดการอ็อบเจ็กต์และ คลาสอ็อบเจ็กต์ รูทีนย่อย ODM ส่งกลับค่า –1 ถ้ารูทีนย่อย ทำงานไม่สำเร็จ ข้อวินิจฉัยระบุความผิดพลาดเฉพาะถูกส่งกลับเป็นตัวแปรภายนอก **odmerrno** (กำหนดไว้ใน **odmi.h** include file) ค่าคงที่ข้อวินิจฉัยความผิดพลาด ODM ยังรวมอยู่ใน **odmi.h** include file เช่นกัน

> **หมายเหตุ:** ถ้าคุณกำลังเขียนโปรแกรมภาษา C โดยใช้รูทีนย่อย ให้ใช้ตัวเลือก:
> **-binitfini:__odm_initfini_init:__odm_initfini_fini**

คำสั่ง

คำสั่ง ODM มีดังนี้:

คำสั่ง	คำอธิบาย
odmadd	เพิ่มอ็อบเจ็กต์ให้กับคลาสอ็อบเจ็กต์ คำสั่ง **odmadd** ใช้ไฟล์ ASCII stanza เป็นอินพุตและสร้างคลาสอ็อบเจ็กต์ด้วยอ็อบเจ็กต์ ที่พบในไฟล์ stanza
odmchange	เปลี่ยนอ็อบเจ็กต์จำเพาะในคลาสอ็อบเจ็กต์ที่ระบุ
odmcreate	สร้างคลาสอ็อบเจ็กต์ว่างเปล่า คำสั่ง **odmcreate** ใช้ ไฟล์ ASCII อธิบายคลาสอ็อบเจ็กต์เป็นอินพุตและสร้างไฟล์ภาษา C **.h** และ **.c** ที่จะถูกใช้โดย อ็อบเจ็กต์การเข้าถึงแอ็พพลิเคชันในคลาสอ็อบเจ็กต์เหล่านั้น
odmdelete	เอาอ็อบเจ็กต์ออกจากคลาสอ็อบเจ็กต์
odmdrop	เอาคลาสอ็อบเจ็กต์ทั้งหมดออก
odmshow	แสดงคำอธิบายของคลาสอ็อบเจ็กต์ คำสั่ง **odmshow** ใช้ชื่อคลาสอ็อบเจ็กต์เป็นอินพุตและนำข้อมูลคลาสอ็อบเจ็กต์ มาอยู่ในรูปแบบคำสั่ง **odmcreate**
odmget	เรียกข้อมูลอ็อบเจ็กต์จากคลาสอ็อบเจ็กต์และจัดข้อมูลอ็อบเจ็กต์ให้อยู่ในรูปแบบคำสั่ง **odmadd**

รูทีนย่อย

รูทีนย่อย ODM คือ:

รูทีนย่อย	คำอธิบาย
odm_add_obj	เพิ่มอ็อบเจ็กต์ใหม่ให้กับคลาสอ็อบเจ็กต์
odm_change_obj	เปลี่ยนเนื้อหาของอ็อบเจ็กต์
odm_close_class	ปิดคลาสอ็อบเจ็กต์
odm_create_class	สร้างคลาสอ็อบเจ็กต์ว่างเปล่า
odm_err_msg	เรียกข้อมูลสตริงข้อความ
odm_free_list	ทำหน่วยความจำที่จัดสรรสำหรับรูทีนย่อย **odm_get_list** ให้ว่าง
odm_get_by_id	เรียกข้อมูลอ็อบเจ็กต์โดยระบุ ID
odm_get_first	เรียกข้อมูลอ็อบเจ็กต์แรกที่ตรงกับเกณฑ์ที่ระบุใน คลาสอ็อบเจ็กต์
odm_get_list	เรียกข้อมูลรายการของอ็อบเจ็กต์ที่ตรงกับเกณฑ์ที่ระบุใน คลาสอ็อบเจ็กต์
odm_get_next	เรียกข้อมูลอ็อบเจ็กต์ถัดไปที่ตรงกับเกณฑ์ที่ระบุใน คลาสอ็อบเจ็กต์
odm_get_obj	เรียกข้อมูลอ็อบเจ็กต์ที่ตรงกับเกณฑ์ที่ระบุจาก คลาสอ็อบเจ็กต์
odm_initialize	กำหนดคาเริ่มต้นเซสชัน ODM
odm_lock	ล็อกคลาสอ็อบเจ็กต์หรือกลุ่มคลาส
odm_mount_class	เรียกข้อมูลโครงสร้างสัญลักษณ์ของคลาสสำหรับคลาสอ็อบเจ็กต์ที่ระบุ
odm_open_class	เปิดคลาสอ็อบเจ็กต์
odm_rm_by_id	ลบอ็อบเจ็กต์ตาม ID ที่ระบุ
odm_rm_obj	เอาอ็อบเจ็กต์ทั้งหมดที่ตรงกับเกณฑ์ที่ระบุออกจาก คลาสอ็อบเจ็กต์
odm_run_method	เรียกเมธอดสำหรับอ็อบเจ็กต์ที่ระบุ
odm_rm_class	เอาคลาสอ็อบเจ็กต์ออก
odm_set_path	ตั้งค่าพาธเริ่มต้นสำหรับการค้นหาคลาสอ็อบเจ็กต์
odm_unlock	ปลดล็อกคลาสอ็อบเจ็กต์หรือกลุ่มคลาส
odm_terminate	จบเซสชัน ODM

ตัวอย่างโค้ดและเอาต์พุต ODM

Fictional_Characters, Friend_Table, และ Enemy_Table Object Classes และตาราง Relationships แสดงคลาสอ็อบเจ็กต์และอ็อบเจ็กต์ที่สร้างโดยโค้ดตัวอย่างในส่วนนี้

ตารางที่ 72. Fictional_Characters

Story_Star (char)	Birthday (char)	Age (short)	Friends_of (link)	Enemies_of (link)	Do_This (method)
Cinderella	Once upon a time	19	Cinderella	Cinderella	echo Cleans House
Snow White	Once upon a time	18	Snow White	Snow White	echo Cleans House

ตารางที่ 73. Friend_Table

Friend_of (char)	Friend (char)
Cinderella	Fairy Godmother
Cinderella	mice
Snow White	Sneezy
Snow White	Sleepy
Cinderella	Prince
Snow White	Happy

ตารางที่ 74. Enemy_Table

Enemy_of (char)	Enemy (char)
Cinderella	midnight
Cinderella	Mean Stepmother
Snow White	Mean Stepmother

ตัวอย่างโค้ดอินพุต ODM และการสร้างคลาสอ็อบเจ็กต์

โค้ดตัวอย่างดังต่อไปนี้ในไฟล์ **MyObjects.cre** สร้างสามคลาสอ็อบเจ็กต์ เมื่อใช้เป็นไฟล์อินพุต ด้วยคำสั่ง **odmcreate**:

```
*      MyObjects.cre
*       An input file for ODM create utilities.
*       Creates three object classes:
*               Friend_Table
*               Enemy_Table
*               Fictional_Characters

class   Friend_Table {
        char     Friend_of[20];
        char     Friend[20];
};

class   Enemy_Table {
        char    Enemy_of[20];
        char    Enemy[20];
};

class   Fictional_Characters {
        char    Story_Star[20];
        char    Birthday[20];
        short   Age;
```

```
        link    Friend_Table Friend_Table Friend_of Friends_of;
        link    Enemy_Table Enemy_Table Enemy_of Enemies_of;
        method  Do_This;
};

* End of MyObjects.cre input file for ODM create utilities. *
```

คลาสอ็อบเจ็กต์ Fictional_Characters มีหก descriptor:

- Story_Star และ Birthday โดยมีชนิด descriptor เป็น char และ ความยาวสูงสุด 20 อักขระ
- Age มีชนิด descriptor เป็น short
- Friends_of และ Enemies_of ทั้งคู่มาจากคลาส link และเชื่อมโยงไปที่ คลาสอ็อบเจ็กต์ที่กำหนดก่อนหน้านี้

  หมายเหตุ: object class link ถูกทำซ้ำสองครั้ง
- Do_This มีชนิด descriptor เป็น method

เมื่อสร้างไฟล์คลาสอ็อบเจ็กต์ที่จำเป็นต่อ ODM ไฟล์มีโค้ดนี้ต้องถูกประมวลด้วยคำสั่ง **odmcreate**

ตัวอย่างเอาต์พุต ODM สำหรับนิยามคลาสอ็อบเจ็กต์

ประมวลผลโค้ดในไฟล์ **MyObjects.cre** ด้วยคำสั่ง **odmcreate** สร้าง structures ดังต่อไปนี้ในไฟล์ **.h**:

```
* MyObjects.h
* The file output from ODM processing of the MyObjects.cre input
* file. Defines structures for the three object classes:
*       Friend_Table
*       Enemy_Table
*       Fictional_Characters
#include <odmi.h>

struct  Friend_Table {
       long     _id;     * unique object id within object class *
       long     _reserved;  * reserved field *
       long     _scratch;   * extra field for application use *
       char     Friend_of[20];
       char     Friend[20];
};

#define Friend_Table_Descs 2
extern struct Class Friend_Table_CLASS[];
#define get_Friend_Table_list(a,b,c,d,e) (struct Friend_Table * )odm_get_list (a,b,c,d,e)

struct  Enemy_Table {
        long    _id;
        long    _reserved;
        long    _scratch;
        char    Enemy_of[20];
        char    Enemy[20];
};
#define Enemy_Table_Descs 2
extern struct Class Enemy_Table_CLASS[];
#define get_Enemy_Table_list(a,b,c,d,e) (struct Enemy_Table * )odm_get_list (a,b,c,d,e)

struct  Fictional_Characters {
        long    _id;
        long    _reserved;
```

```
        long    _scratch;
        char    Story_Star[20];
        char    Birthday[20];
        short   Age;
       struct  Friend_Table *Friends_of;    * link *
        struct  listinfo *Friends_of_info;   * link *
        char    Friends_of_Lvalue[20];       * link *
        struct  Enemy_Table *Enemies_of;     * link *
        struct  listinfo *Enemies_of_info;   * link *
        char    Enemies_of_Lvalue[20];       * link *
        char    Do_This[256];                * method *
};

#define Fictional_Characters_Descs 6

extern struct Class Fictional_Characters_CLASS[];
#define get_Fictional_Characters_list(a,b,c,d,e) (struct Fictional_Characters * )odm_get_list (a,b,c,d,e)

* End of MyObjects.h structure definition file output from ODM    * processing.
```

ตัวอย่างโค้ด ODM สำหรับการเพิ่มอ็อบเจ็กต์เข้ากับคลาสอ็อบเจ็กต์

โค้ดดังต่อไปนี้สามารถถูกประมวลผลโดยคำสั่ง **odmadd** เพื่อสร้างคลาสอ็อบเจ็กต์ที่สร้างโดย การประมวลผลไฟล์อินพุต **MyObjects.cre**:

```
* MyObjects.add
* An input file for ODM add utilities.
* Populates three created object classes:
*       Friend_Table
*       Enemy_Table
*       Fictional_Characters

Fictional_Characters:
Story_Star = "Cinderella" #a comment for the  MyObjects.add file.
Birthday        =       "Once upon a time"
Age             =       19
Friends_of      =       "Cinderella"
Enemies_of      =       "Cinderella"
Do_This =               "echo Cleans house"

Fictional_Characters:
Story_Star      =       "Snow White"
Birthday        =       "Once upon a time"
Age             =       18
Friends_of      =       "Snow White"
Enemies_of      =       "Snow White"
Do_This   =              "echo Cleans house"

Friend_Table:
Friend_of       =       "Cinderella"
Friend          =       "Fairy Godmother"

Friend_Table:
Friend_of       =       "Cinderella"
Friend          =       "mice"

Friend_Table:
Friend_of       =       "Snow White"
Friend          =       "Sneezy"
```

```
Friend_Table:
Friend_of       =       "Snow White"
Friend          =       "Sleepy"

Friend_Table:
Friend_of       =       "Cinderella"
Friend          =       "Prince"

Friend_Table:
Friend_of       =       "Snow White"
Friend          =       "Happy"

Enemy_Table:
Enemy_of        =       "Cinderella"
Enemy           =       "midnight"

Enemy_Table:
Enemy_of        =       "Cinderella"
Enemy           =       "Mean Stepmother"

Enemy_Table:
Enemy_of        =       "Snow White"
Enemy           =       "Mean Stepmother"

* End of MyObjects.add input file for ODM add utilities. *
```

หมายเหตุ: The * คอนเมนต์ (เครื่องหมายดอกจัน) หรือ # (เครื่องหมายปอนด์) ในตัวอย่างโค้ดก่อนหน้านี้ จะไม่อยู่ในอ็อบเจ็กต์ไฟล์ ถ้าจุดเริ่มต้นของ บรรทัดทำให้บรรทัดเป็นคอมเมนต์ คำสั่งจะไม่ไปอยู่ใน อ็อบเจ็กต์ไฟล์ คอมเมนต์จะถูกรวมไว้ในไฟล์และถือว่าเป็นสตริง ถ้าอยู่รวม ไว้ภายใน " " (เครื่องหมายคำพูดคู่)

Simultaneous multithreading

Simultaneous multithreading คือความสามารถของฟิสิคัลโพรเซสเซอร์เดี่ยวในการจัดส่งคำสั่งต่างๆ อย่างพร้อมเพียงกันจากบริบทของเธรดฮาร์ดแวร์มากกว่าหนึ่งเครื่อง เนื่องจาก มีเธรดฮาร์ดแวร์สองชุดต่อหนึ่งฟิสิคัลโพรเซสเซอร์ คำสั่งเพิ่มเติมสามารถรันได้ในเวลาเดียวกัน

Simultaneous multithreading อนุญาตให้คุณใช้ประโยชน์ของลักษณะ superscalar ของตัวประมวลผล โดยกำหนดตารางเวลาให้กับสองแอ็พพลิเคชันในเวลาเดียวกันบนตัวประมวลผลตัวเดียวกัน ไม่มีแอ็พพลิเคชันเดี่ยว ที่สามารถใช้โพรเซสเซอร์อย่างครบถ้วน

การรับประโยชน์จาก Simultaneous Multithreading

มีประโยชน์ในสภาพการใช้งานเชิงพาณิชย์ ซึ่งความเร็วในการทำธุรกรรมแต่ละรายการไม่ใช่สิ่งสำคัญเท่ากับจำนวนของธุรกรรมที่ถูกดำเนินการ Simultaneous multithreading ถูกคาดการณ์ว่าจะเพิ่มทรูพุตของเวิร์กโหลด ด้วยการเปลี่ยนชุดการทำงานให้ถี่ขึ้น เช่น เซิร์ฟเวอร์ฐานข้อมูลและเว็บเซิร์ฟเวอร์

เวิร์กโหลดที่มองเห็นประโยชน์ของ simultaneous multithreading มากที่สุดคือ เวิร์กโหลดที่มีจำนวน Cycles Per Instruction (CPI) สูง เวิร์กโหลดเหล่านี้ใช้โพรเซสเซอร์ และรีซอร์สหน่วยความจำที่ไม่เพียงพอ CPIs ขนาดใหญ่จะมีสาเหตุมาจากอัตราการขาดหายของแคชที่สูง จากชุดการทำงานที่มีขนาดใหญ่ เวิร์กโหลดในเชิงพาณิชย์ขนาดใหญ่จะขึ้นอยู่กับ เธรดของฮาร์ดแวร์สองตัวแบ่งใช้คำสั่งหรือข้อมูล หรือเธรดของฮาร์ดแวร์ที่แยกออกโดยสมบูรณ์โดยปกติแล้ว เวิร์กโหลดเชิงพาณิชย์จะมี

คุณสมบัตินี้ เวิร์กโหลดที่แบ่งใช้คำสั่งหรือข้อมูล ซึ่งประกอบด้วยเวิร์กโหลดเหล่านี้ที่รันในระบบปฏิบัติการ หรือภายในแอ็พพลิเคชันเดี่ยว อาจมองเห็นผลประโยชน์ที่เพิ่มขึ้นจาก simultaneous multithreading

เวิร์กโหลดที่ไม่มีประโยชน์มากจาก simultaneous multithreading จะเป็นเวิร์กโหลดที่เธรดของซอฟต์แวร์หลักแต่ละเธรด ใช้จำนวนของรีซอร์สขนาดใหญ่ในตัวประมวลผลหรือหน่วยความจำ ตัวอย่างเช่น เวิร์กโหลดที่เป็นเลขทศนิยมจะได้รับเพียงเล็กน้อยจาก simultaneous multithreading และเป็นเวิร์กโหมดเดียวที่สูญเสียผลการทำงาน เวิร์กโหลดเหล่านี้จะใช้หน่วยทศนิยมหรือแบนด์วิธหน่วยความจำ อย่างใดอย่างหนึ่ง เวิร์กโหลดที่มี CPI ต่ำและอัตราแคชที่หายไปที่มีขนาดต่ำ อาจมองเห็นประโยชน์ได้เพียงเล็กน้อย

การวัดจะใช้พาร์ติชันเฉพาะงานด้วยเวิร์กโหลดเชิงพาณิชย์ที่บ่งชี้การเพิ่มขึ้น 25%–40% ในทรูพุต Simultaneous multithreading ควรช่วยในการประมวลผลพาร์ติชัน ที่มีการประมวลผลแบบแบ่งใช้ เธรดพิเศษได้ให้การสนับสนุนแก่พาร์ติชันหลังจากที่ simultaneous multithreading ถูกจัดส่ง เนื่องจากพาร์ติชันจะกู้คืนชุดการทำงานได้เร็วกว่า ถัดมา เธรดจะดำเนินการเช่นเดียวกับที่ทำในพาร์ติชันเฉพาะงาน แม้ว่า เธรดอาจจะโต้ตอบตามกลไก simultaneous multithreading จะดำเนินการได้ดีที่สุด เมื่อผลการทำงานของแคชจะแย่ที่สุด

การตั้งค่าโหมดโดยใช้คำสั่ง smtctl

AIX อนุญาตให้คุณควบคุมโหมดของพาร์ติชันสำหรับ simultaneous multithreading ด้วยคำสั่ง **smtctl** ด้วยคำสั่งนี้ คุณสามารถเปิดหรือปิดระบบ simultaneous multithreading แบบกว้างๆ ได้ทันทีหรือในครั้งถัดไปที่ระบบบูต โหมด simultaneous multithreading มีอยู่ระหว่างที่บูตระบบ ตามค่าดีฟอลต์ AIX เปิดใช้งาน simultaneous multithreading

ไวยากรณ์สำหรับคำสั่ง **smtctl** จะเป็นดังนี้:

```
smtctl [ -m { off  | on } [ { -boot | -now } ] ]
```

สำหรับข้อมูลเพิ่มเติม โปรดดูคำสั่ง **smtctl** ใน *Commands Reference, Volume 5*

Hardware Management Console Configuration สำหรับ Simultaneous Multithreading

เมื่อคุณปรับแต่งพาร์ติชันของตัวประมวลผลแบบแบ่งใช้ที่ Hardware Management Console (HMC) คุณจะระบุจำนวนต่ำสุง จำนวนที่กำหนดไว้ และจำนวนสูงสุด ของตัวประมวลผลเสมือน สำหรับพาร์ติชันเฉพาะงาน คุณระบุชนิดของพารามิเตอร์ที่เหมือนกัน แต่ความหมายของตัวประมวลผลจะต่างกัน สำหรับพาร์ติชันเฉพาะงาน ตัวประมวลผลจะถูกเรียกว่าตัวประมวลผลเสมอ

แบบจำลองการแบ่งพาร์ติชันทั้งสองแบบต้องการให้คุณระบุช่วงของตัวประมวลผล ที่ควบคุมการบูตและการกำหนดรันไทม์ของตัวประมวลผลให้กับพาร์ติชัน หากเป็นไปได้ ค่าติดตั้งตัวประมวลผลที่ต้องการจะถูกให้สิทธิเมื่อระบบเริ่มต้นขึ้น หากเป็นไปได้ POWER Hypervisor™ จะเลือกค่าอื่น ตามการอ้างอิงชุดของรีซอร์สที่พร้อมใช้งานซึ่งมีค่ามากกว่า หรือเท่ากับค่าต่ำสุด

จำนวนของตัวประมวลผลที่ระบุที่ HMC จะส่งผลกระทบจำนวนของตัวประมวลผลแบบโลจิคัลที่ AIX จัดสรรไว้ ถ้าพาร์ติชันมีความสามารถของ simultaneous multithreading, AIX ที่จัดสรรค่าเป็นสองเท่าของตัวประมวลผลสูงสุดเท่าที่ตัวประมวลผลแบบโลจิคัลจำนวนมากมีอยู่ เนื่องจากมีเธรดของฮาร์ดแวร์สองตัวต่อหนึ่งตัวประมวลผลและ AIX ปรับแต่งเธรดของฮาร์ดแวร์ตามตัวประมวลผลแบบโลจิคัลที่แยกออก ซึ่งจะอนุญาตให้ AIX เปิดใช้งานหรือปิดใช้งาน simultaneous multithreading โดยไม่ต้องรีบูตพาร์ติชัน

Dynamic Logical Partitioning สำหรับ Simultaneous Multithreading

ขณะที่พาร์ติชันกำลังรัน คุณสามารถเปลี่ยนจำนวนของตัวประมวลผล ที่ได้กำหนดให้กับพาร์ติชันผ่านโพรซีเดอร์ Dynamic Logical Partitioning (DLPAR) ที่ HMC คุณสามารถเพิ่มหรือถอดตัวประมวลผลภายในข้อจำกัดของช่วงของตัวประมวลผลที่กำหนดไว้สำหรับพาร์ติชัน เมื่อเพิ่มตัวประมวลผลให้กับพาร์ติชันที่เปิดใช้งาน simultaneous multithreading, AIX ให้เริ่มต้นเธรดของฮาร์ดแวร์ และตัวประมวลผลแบบโลจิคัลสองตัวจะนำไปสู่การออนไลน์ เมื่อตัวประมวลผลถูกถอดออกจากพาร์ติชัน ที่เปิดใช้งาน simultaneous multithreading แล้ว AIX เธรดของฮาร์ดแวร์ทั้งสองตัวและตัวประมวลผลแบบโลจิคัลทั้งสองตัวจะออฟไลน์

เหตุการณ์ DLPAR สองเหตุการณ์จะถูกสร้างขึ้นเมื่อเปิดใช้งาน simultaneous multithreading เหตุการณ์หนึ่งเหตุการณ์ จะถูกสร้างขึ้นสำหรับแต่ละตัวประมวลผลแบบโลจิคัลที่ถูกเพิ่มหรือลบออก API สำหรับสคริปต์ DLPAR จะอ้างอิงตามตัวประมวลผลแบบโลจิคัล ดังนั้นจำนวนของเหตุการณ์ DLPAR จะขนานกับการเพิ่มหรือถอดตัวประมวลผลแบบโลจิคัล ถ้าไม่ได้เปิดใช้ simultaneous multithreading ในพาร์ติชัน จะมีเหตุการณ์ DLPAR เพียงเหตุการณ์เดียว AIX จะแปลคำร้องขอ DLPAR ที่ส่งจาก HMC ไปยังจำนวนของเหตุการณ์ DLPAR ที่เหมาะสมที่แสดงถึงแอ็พพลิเคชันที่รับรู้โดย DLPAR

Micro-Partitioning® และ Simultaneous Multithreading

POWER Hypervisor™ จะบันทึกและเรียกข้อมูลสถานะของตัวประมวลผลที่จำเป็นทั้งหมด ขณะจองหรือจัดส่งตัวประมวลผลเสมือน สำหรับตัวประมวลผลที่เปิดใช้งานสำหรับ simultaneous multithreading นี่หมายความถึงบริบทของเธรดที่แอ็คทีฟสองตัว เธรดของฮาร์ดแวร์แต่ละตัว จะสนับสนุนตัวประมวลผลแบบโลจิคัลที่แบ่งแยกโดย AIX สำหรับเหตุผลนี้ พาร์ติชันเฉพาะงานที่ถูกสร้างด้วยตัวประมวลผลแบบฟิสิคัลจะถูกปรับแต่งโดย AIX ซึ่งเป็นตัวประมวลผลแบบโลจิคัล 2 ทิศทาง เนื่องจากนี่คือพาร์ติชันแบบอิสระ พาร์ติชันแบบแบ่งใช้ด้วยตัวประมวลผลเสมือนสองตัวซึ่งถูกปรับแต่งโดย AIX ซึ่งเป็นตัวประมวลผลแบบโลจิคัล 4 ทิศทาง และพาร์ติชันแบบแบ่งใช้ด้วยตัวประมวลผลเสมือนทั้งหมดสี่ตัวจะถูกปรับแต่งโดย AIX ซึ่งเป็นตัวประมวลผลแบบโลจิคัล 8 ทิศทาง เธรดที่จับคู่แล้วยังถูกกำหนดตารางเวลาพร้อมกัน ในเวลาเดียวกันกับใน พาร์ติชันเดียวกัน

ความสามารถของตัวประมวลผลแบบแบ่งใช้จะถูกจัดส่งอยู่ในรูปของตัวประมวลผลแบบฟิสิคัลทั้งหมด ซึ่ง simultaneous multithreading, AIX ไม่ได้ปรับแต่งพาร์ติชันที่มีตัวประมวลผลเสมือน 4 ทิศทางด้วย 200 ยูนิตของตัวประมวลผลทั้งหมดให้เป็นพาร์ติชันที่มีตัวประมวลผลแบบโลจิคัล 4 ทิศทาง ซึ่งตัวประมวลผลแบบฟิสิคัลแต่ละตัวมีกำลัง 50% ของตัวประมวลผลแบบฟิสิคัล ด้วย simultaneous multithreading พาร์ติชันที่มีตัวประมวลผลแบบโลจิคัล 4 ทิศทางจะกลายเป็น พาร์ติชันที่มีตัวประมวลผลแบบโลจิคัล 8 ทิศทาง ซึ่งตัวประมวลผลแต่ละตัวจะมีกำลัง 25% ของตัวประมวลผลแบบฟิสิคัล อย่างไรก็ตาม ด้วย simultaneous multithreading เวลาแฝงที่เกี่ยวข้อง ซึ่งโดยปกติจะเชื่อมโยงกับความสามารถในการแบ่งส่วนของตัวประมวลผลแบบเสมือน จะไม่ใช้กับเธรดเป็นแนวตรง เนื่องจากเธรดทั้งสองจะถูกจัดส่งพร้อมกัน เธรดเหล่านั้นจะแอ็คทีฟในช่วงเวลา 50% ของหน้าต่างการจัดส่ง และจะแบ่งใช้ตัวประมวลผลแบบฟิสิคัลเพื่อให้ได้มาซึ่ง 25% ของโลจิคัล นั่นหมายความว่า ตัวประมวลผลแบบโลจิคัลแต่ละตัวจะสามารถอินเตอร์รัปต์ได้สองครั้ง ตราบเท่าที่ความสามารถของแต่ละตัวจะอนุญาต

ลำดับความสำคัญของเธรดฮาร์ดแวร์

ตัวประมวลผลอนุญาตให้ใช้ลำดับความสำคัญที่ต้องกำหนดค่าให้กับ เธรดฮาร์ดแวร์ ความแตกต่างในลำดับความสำคัญระหว่างเธรดแบบ sibling จะเป็นตัวกำหนดอัตราของการถอดรหัสฟิสิคัลโพรเซสเซอร์ ในสล็อตที่แบ่งส่วนให้กับแต่ละเธรด สล็อตที่มีจำนวนมาก จะทำให้ผลการทำงานเธรดดีขึ้น โดยปกติแล้ว AIX จะรักษาเธรดแบบ sibling ที่มีลำดับความสำคัญที่เหมือนกัน แต่เพิ่มหรือลดลำดับความสำคัญในตำแหน่งหลัก เพื่อออปติไมซ์ผลการทำงาน ตัวอย่างเช่น AIX มีลำดับความสำคัญของเธรดที่ต่ำกว่า เมื่อเธรดกำลังทำงานที่ไม่ใช่สภาวะแวดล้อมจริงที่ปรับขึ้นลงในลูปที่ว่าง หรือบนการล็อกเคอร์เนล ลำดับความสำคัญของเธรดจะถูกเพิ่มขึ้น เมื่อเธรดกำลังพักการล็อกเคอร์เนลที่สำคัญ การปรับลำดับความสำคัญเหล่านี้จะไม่คงอยู่ในโหมดผู้ใช้ AIX ไม่ได้พิจารณาลำดับความสำคัญที่จัดส่งของเธรดของซอฟต์แวร์ เมื่อเธรดกำลังเลือกลำดับความสำคัญของเธรดของฮาร์ดแวร์

การทำงานจะถูกแจกจ่ายระหว่างเธรดหลัก ก่อนที่งานจะถูกจัดส่งไปยังเธรดสำรอง ผลการทำงานของเธรดจะดีที่สุด เมื่อคู่ของเธรดไม่ได้ทำงาน เธรดที่อยู่ติดกันจะยังคงถูกพิจารณาในเวลาที่ไม่ได้ทำงาน และในการสร้างดุลยภาพในการโหลดคิวเป็นช่วงเวลา

การแบ่งโลจิคัลพาร์ติชันแบบไดนามิก

Partitioning your system is similar to partitioning a hard disk drive. When you partition a hard disk drive, you divide a single physical hard disk drive so that the operating system recognizes it as a number of separate logical hard disk drives.

บนแต่ละส่วนเหล่านี้ซึ่งจะเรียกว่า พาร์ติชัน คุณสามารถติดตั้งระบบปฏิบัติการ และใช้แต่ละพาร์ติชันได้ตามที่คุณต้องการทำกับระบบฟิสิคัลแยก

พาร์ติชันแบบโลจิคัล (LPAR) คือการแบ่ง ตัวประมวลผล หน่วยความจำ และรีซอร์สของฮาร์ดแวร์ของเครื่องคอมพิวเตอร์ออกเป็นหลาย สภาวะแวดล้อม โดยที่แต่ละสภาวะแวดล้อมสามารถดำเนินงานแยกเป็นอิสระสำหรับ ระบบปฏิบัติการ และแอ็พพลิเคชันของตน จำนวนของพาร์ติชันแบบโลจิคัลที่สามารถสร้างได้ นั้นขึ้นอยู่กับระบบ โดยทั่วไป พาร์ติชันถูกใช้ในวัตถุประสงค์ที่แตกต่างกัน เช่นการดำเนินการฐานข้อมูล การดำเนินการไคลเอ็นต์/เซิร์ฟเวอร์ การดำเนินการเว็บเซิร์ฟเวอร์ สภาวะแวดล้อมการทดสอบ และสภาวะแวดล้อมที่ทำงานจริง แต่ละพาร์ติชันสามารถสื่อสาร กับพาร์ติชันอื่นได้เหมือนกับว่าแต่ละพาร์ติชันอยู่บนเครื่องแยกกัน

Dynamic logical partitioning (DLPAR) ช่วยให้ มีความสามารถในการรวมเข้าหรือแยกรีซอร์สของระบบที่ได้รับการจัดการในแบบโลจิคัล กับระบบปฏิบัติการของพาร์ติชันแบบโลจิคัลโดยไม่ต้องบูตใหม่ บางคุณลักษณะ ของ DLPAR ได้แก่:

- คุณลักษณะ *Capacity Upgrade on Demand* (*CUoD*) ของ IBM System p ซึ่งให้ลูกค้าเรียกทำงานตัวประมวลผลที่ติดตั้งไว้แล้ว แต่ไม่แอ็คทีฟ ตามการเปลี่ยนแปลงข้อกำหนดรีซอร์ส
- คุณลักษณะ Dynamic Processor Deallocation ของเซิร์ฟเวอร์ IBM System p5® และบนบางโมเดล SMP Dynamic Processor Deallocation จะเปิดให้ ตัวประมวลผลออฟไลน์แบบไดนามิก เมื่อเกินเส้นแบ่งภายในของ ข้อผิดพลาดที่กู้คืนได้ DLPAR ปรับปรุงคุณลักษณะ Dynamic Processor Deallocation โดยการให้ตัวประมวลผลที่ไม่แอ็คทีฟ ถ้ามีอยู่ ถูกแทนที่สำหรับ ตัวประมวลผลที่ถูกสงสัยว่ามีข้อบกพร่อง การสลับแบบออนไลน์นี้ไม่มีผล ต่อแอ็พพลิเคชันหรือส่วนขยายเคอร์เนล
- DLPAR เปิดให้สามารถทำการจัดการเวิร์กโหลดข้ามพาร์ติชัน ซึ่ง สำคัญอย่างยิ่งสำหรับการรวมเซิร์ฟเวอร์เข้าด้วยกันเพื่อให้สามารถใช้ จัดการรีซอร์สระบบข้ามพาร์ติชัน

การร้องขอ DLPAR ถูกสร้างจากการร้องขอเพื่อเพิ่มและลบออกอย่างง่าย ที่ถูกกำหนดให้แก่พาร์ติชันแบบโลจิคัล ผู้ใช้สามารถเรียกทำงาน คำสั่งเหล่านี้เป็นการร้องขอเพื่อย้ายที่ Hardware Management Console (HMC) ซึ่งจะจัดการ การดำเนินการ DLPAR ทั้งหมด การดำเนินการ DLPAR ถูกเปิดใช้งานโดยเฟิร์มแวร์ System p และ AIX

ข้อมูลที่เกี่ยวข้อง:

cpupstat

drmgr

dr_reconfig

reconfig

DLPAR-safe และโปรแกรม aware

โปรแกรม DLPAR-safe จะไม่ล้มเหลวจากผลของการดำเนินการ DLPAR

ประสิทธิภาพอาจได้รับผลเสียเมื่อ ทรัพยากรถูกเอาออกและอาจไม่สเกลกับ ทรัพยากรใหม่เพิ่มเติม แต่โปรแกรมยังคงทำงานตามที่คาดไว้โดยข้อเท็จจริง โปรแกรม DLPAR-safe สามารถป้องกันการดำเนินการ DLPAR จากความสำเร็จเนื่องจากมีข้อมูลที่ไม่เป็นอิสระที่ระบบปฏิบัติการหมดเวลาที่จะ ปฏิบัติตาม

โปรแกรม *DLPAR-aware* เป็นโปรแกรมที่มีโค้ด DLPAR ที่ถูกกำหนดเพื่อปรับ การใช้ทรัพยากรระบบตามความสามารถจริงของระบบแปรตามเวลา ซึ่งทำให้สำเร็จได้ด้วยวิธี ดังต่อไปนี้:

- โดยการสุ่มข้อมูลระบบเป็นระยะเพื่อค้นการการเปลี่ยนใน โทโพโลยีของระบบ
- โดยการรีจิสเตอร์โค้ดจำเพาะแอ็พพลิเคชันที่ถูกแจ้งเตือนเมื่อ การเปลี่ยนแปลงเกิดขึ้นกับแบบแผนการจัดวางระบบ

โปรแกรม *DLPAR-aware* ต้อง ถูกออกแบบ ที่ระดับต่ำสุด เพื่อหลีกเลี่ยงการนำมาซึ่งเงื่อนไขที่อาจทำให้การดำเนินการ DLPAR ล้มเหลว ที่ค่าสูงสุด โปรแกรม DLPAR-aware ถูกคำนึงถึง ประสิทธิภาพและการปรับสเกลได้ นี่เป็นงานที่ซับซ้อนมากขึ้นเนื่องจาก บัฟเฟอร์อาจจำเป็นต้องถูกใช้หมดและถูกปรับขนาดเพื่อรักษาระดับที่คาดไว้ของ ประสิทธิภาพเมื่อหน่วยความจำถูกเพิ่มหรือเอาออก นอกจากนี้ จำนวนเธรด ต้องถูกปรับแบบไดนามิกเพื่อคำนวณการเปลี่ยนแปลงจำนวนออนไลน์โพรเซสเซอร์ การปรับ thread-based เหล่านี้ไม่จำเป็นต้องจำกัดที่ การตัดสินแบบ processor-based ตัวอย่างเช่น วิธีที่ดีที่สุดในการลดการใช้หน่วยความจำ ในโปรแกรม Java อาจเป็นการลดจำนวนเธรด เนื่องจากเป็นการลดจำนวน อ็อบเจ็กต์ที่ใช้งานอยู่ที่จำเป็นต้องถูกประมวลผลโดย garbage collector ของ Java Virtual Machine

แอ็พพลิเคชันส่วนใหญ่เป็น DLPAR-safe โดยค่าเริ่มต้น

การทำให้โปแกรม DLPAR-safe

ชนิดของข้อผิดพลาดดังต่อไปนี้ แสดงความเข้ากันได้ของไบนารี สามารถนำมาโดย DLPAR:

หมายเหตุ: ข้อผิดพลาดเหล่านี้ เป็นผลของการเพิ่มโพรเซสเซอร์

- ถ้าโปรแกรมมีโค้ดที่ถูกออปติไมซ์สำหรับระบบ uniprocessor และ จำนวนของโพรเซสเซอร์ในพาร์ติชันถูกเพิ่มขึ้นจากหนึ่งเป็นสอง โปรแกรม ที่ทำการตรวจสอบรันไทม์อาจใช้พาธที่ไม่ได้คาดไว้ ถ้าโพรเซสเซอร์ถูกเพิ่ม ระหว่างที่มีการตรวจสอบดังกล่าว ปัญหานี้ยังสามารถเกิดขึ้นได้ในโปรแกรม ที่นำการล็อกแบบพื้นฐานมาใช้เอง แต่ใช้เทคนิคการ uniprocessor serialization; นั่นคือไม่ได้รวมคำสั่ง sync และ isync การใช้คำสั่งเหล่านี้ยังจำเป็นสำหรับการแก้ไขตัวเอง และโค้ดที่สร้างและดังนั้นจำเป็นกับระบบ DLPAR-enabled ให้แน่ใจว่าได้ตรวจสอบตรรกะ uniprocessor-based โปรแกรมที่ทำ uniprocessor assertions ต้องรวมตรรกะที่ระบุจำนวนออนไลน์โพรเซสเซอร์

  โปรแกรมสามารถระบุจำนวนของออนไลน์โพรเซสเซอร์โดย:

  - การโหลด ฟิลด์ *_system_configuration.ncpus*
  - *var.v_ncpus*
  - การใช้การเรียกระบบ **sysconf** กับ แฟล็ก **_SC_NPROCESSORS_ONLN**
- โปรแกรมที่ทำดัชนีข้อมูลตามหมายเลขโพรเซสเซอร์ที่ปกติใช้การเรียกระบบ **mycpu** เพื่อกำหนดการระบุของโพรเซสเซอร์กระทำการ ปัจจุบัน เพื่อทำดัชนีลงในโครงสร้างข้อมูล ปัญหา เกิดขึ้นได้เมื่อโพรเซสเซอร์ใหม่ถูกเพิ่มเนื่องจากพาธไปยังข้อมูล อาจถูกกำหนดค่าหรือจัดสรรไม่ถูกต้อง โปรแกรมที่จัดสรรล่วงหน้ารายการ processor-based โดยใช้จำนวนของออนไลน์ CPU เสียหาย เนื่องจาก ค่านี้เปลี่ยนแปลงด้วย DLPAR

หลีกเลี่ยงปัญหานี้โดยทำการจัดสรรล่วงหน้าข้อมูล processor-based โดยใช้ค่าจำนวนโพรเซสเซอร์สูงสุดที่เป็นไปได้ ที่สามารถนำมาออนไลน์พร้อมกัน ระบบปฏิบัติการถูกกล่าวได้ว่า สามารถถูกกำหนดให้สนับสนุนจำนวนโพรเซสเซอร์สูงสุด *N* โพรเซสเซอร์ ไม่ใช่โพรเซสเซอร์ *N* โพรเซสเซอร์ที่ใช้งานอยู่ ในเวลาที่กำหนด ค่าจำนวนสูงสุดของโพรเซสเซอร์เป็นค่าคงที่ ขณะที่จำนวนออนไลน์ โพรเซสเซอร์ถูกเพิ่มขึ้นและลดลงเมื่อโพรเซสเซอร์ถูกนำมาออนไลน์ และออฟไลน์ เมื่อพาร์ติชันถูกสร้าง ค่าน้อยที่สุด ค่าที่ต้องการ และ จำนวนสูงสุดของโพรเซสเซอร์ถูกระบุ ค่าสูงสุดถูกแสดงใน ตัวแปรดังต่อไปนี้:

- *_system_configuration.max_ncpus*
- *_system_configuration.original_ncpus*
- *var.v_ncpus_cfg*
- **sysconf** (**_SC_NPROCESSORS_CONF**)

ตัวแปร *_system_configuration.original_ncpus* และ *var.v_ncpus_cfg* เป็นตัวแปรที่มีอยู่ก่อน บนระบบ DLPAR-enabled ตัวแปรแสดงค่าสูงสุด บนระบบที่ไม่ได้เปิดใช้งานสำหรับ DLPAR ค่าถูกกำหนดโดยจำนวนของโพรเซสเซอร์ที่ถูกกำหนดค่าตอนบูต เครื่อง ทั้งสองแบบแสดงค่าสูงสุดเชิงแนวคิดที่มีการสนับสนุนได้ แม้ว่า โพรเซสเซอร์อาจถูกออฟไลน์โดย Dynamic Processor Deallocation การใช้ ฟิลด์ที่มีอยู่ก่อนหน้าเหล่านี้เป็นข้อแนะนำสำหรับแอ็พพลิเคชันที่ถูกสร้าง บน AIX 4.3 เนื่องจากเป็นการอำนวยความสะดวกในการใช้ไบนารีเดียวกันบน AIX 4.3 และภายหลัง ถ้าแอ็พพลิเคชันต้องการการกำหนดค่าเริ่มต้นรันไทม์ของข้อมูล processor-based แอ็พพลิเคชันสามารถรีจิสเตอร์ตัวจัดการ DLPAR ที่ถูกเรียกก่อนโพรเซสเซอร์ ถูกเพิ่ม

การทำให้โปรแกรม DLPAR-aware

โปรแกรม *DLPAR-aware* คือโปรแกรมที่ถูกออกแบบเพื่อจดจำและปรับแบบไดนามิกต่อการเปลี่ยนแปลงใน การกำหนดค่าระบบ โค้ดนี้ไม่จำเป็นต้องลงทะเบียนไปที่โมเดลการรับรู้ DLPAR แต่สามารถถูกสร้างในรูปแบบที่ธรรมดากว่าของมอนิเตอร์ทรัพยากรระบบ ที่สุ่มข้อมูลระบบเป็นประจำเพื่อค้นหาการเปลี่ยนแปลง ในการกำหนดค่าระบบ วิธีนี้สามารถนำมาใช้เพื่อบรรลุเป้าหมาย performance-related ที่จำกัด แต่เนื่องจากไม่ได้เป็นการผูกพันกับ DLPAR มาก จึงไม่สามารถถูกใช้ได้อย่างมีประสิทธิภาพในการจัดการการเปลี่ยนแปลงขนาดใหญ่ กับการกำหนดค่าระบบ ตัวอย่างเช่น โมเดลการสุ่มตัวอย่างอาจไม่เหมาะสำหรับ ระบบที่สนับสนุนโพรเซสเซอร์ hot plug เนื่องจากยูนิต hot-pluggable อาจ ไม่ถูกประกอบด้วยโพรเซสเซอร์และแผงหน่วยความจำหลายชุด และไม่สามารถถูกใช้ เพื่อจัดการความไม่เป็นอิสระของ application-specific เช่นการเชื่อมโยงโพรเซสเซอร์ ที่จะเป็นต้องแยกแยะก่อนเหตุการณ์เอาโพรเซสเซอร์ DLPAR ออก เริ่มขึ้น

ชนิดของแอ็พพลิเคชันดังต่อไปนี้สามารถใช้ประโยชน์จาก เทคโนโลยี DLPAR:

- แอ็พพลิเคชันที่ถูกออกแบบให้สเกลกับการกำหนดค่าระบบ รวมถึง:
  - การตรวจจับจำนวนออนไลน์โพรเซสเซอร์หรือขนาดของหน่วยความจำฟิสิคัล เมื่อแอ็พพลิเคชันเริ่มการทำงาน
  - สามารถกำหนดได้จากภายนอกเพื่อสเกลขึ้นกับการกำหนดค่าที่คาดการณ์ ของโพรเซสเซอร์และหน่วยความจำ ซึ่งโดยปกติแปลเป็นการใช้จำนวน เธรดสูงสุด ขนาดบัฟเฟอร์สูงสุด หรือจำนวนหน่วยความจำที่ pin สูงสุด
- แอ็พพลิเคชันที่รับรู้ถึงจำนวนของออนไลน์โพรเซสเซอร์และ จำนวนรวมของหน่วยความจำระบบ ประกอบด้วยประเภทแอ็พพลิเคชันดังต่อไปนี้:
  - Performance monitors
  - Debugging tools
  - System crash tools
  - Workload managers
  - License managers

หมายเหตุ: ไม่ใช่ license managers ทั้งหมดที่ใช้ได้กับ DLPAR โดยเฉพาะ user-based license managers

- แอ็พพลิเคชันที่ pin ข้อมูล ข้อความ หรือสแต็กของแอ็พพลิเคชันโดยใช้การเรียกระบบ **plock**
- แอ็พพลิเคชันที่ใช้ System V Shared Memory Segments กับ **PinvOption** (**SHM_PIN**)
- แอ็พพลิเคชันที่เชื่อมโยงโพรเซสเซอร์โดยใช้การเรียกระบบ **bindprocessor**

Dynamic logical partitioning ของเพจหน่วยความจำขนาดใหญ่ไม่ได้รับการสนับสนุน จำนวน หน่วยความจำที่ถูกจัดสรรล่วงหน้าไปที่พูลเพจขนาดใหญ่สามารถมีผล กระทบทรัพยากรกับความสามารถ DLPAR ของพาร์ติชันที่เกี่ยวกับหน่วยความจำขอบเขตหน่วยความจำที่มีเพจขนาดใหญ่ไม่สามารถถูกเอาออกได้ ดังนั้น ต้องการ แอ็พพลิเคชันอาจต้องการจัดเตรียมตัวเลือกเพื่อไม่ใช้เพจขนาดใหญ่

การทำให้โปรแกรม DLPAR-aware โดยใช้ DLPAR APIs

อินเตอร์เฟสแอ็พพลิเคชันถูกจัดเตรียมเพื่อสร้าง โปรแกรม DLPAR-aware สัญญาณ **SIGRECONFIG** ถูกส่งไปที่แอ็พพลิเคชันที่เฟสต่างๆ ของ dynamic logical partitioning ระบบย่อย DLPAR กำหนดเฟส check, pre และ post สำหรับการดำเนินการปกติ แอ็พพลิเคชัน สามารถคอยสัญญาณนี้และใช้การเรียกระบบ DLPAR-supported เมื่อต้องการเรียนรู้เพิ่มเติมเกี่ยวกับการดำเนินการที่กระทำอยู่และการใช้การดำเนินการที่จำเป็น

หมายเหตุ: เมื่อใช้สัญญาณ แอ็พพลิเคชันอาจบล็อกสัญญาณโดยไม่เจตนา หรือโหลดบนระบบอาจขัดขวางเธรดไม่ให้รันได้ตามเวลาที่กำหนด ในกรณีของสัญญาณ ระบบจะรอเป็นเวลาสั้นๆ ซึ่งฟังก์ชันที่ผู้ใช้ระบุหมดเวลาใช้งาน และดำเนินต่อ ไปยังเฟสถัดไป เป็นการไม่เหมาะสมที่จะรอไปตลอด เนื่องจากเธรด non-privileged rogue อาจขัดขวางการดำเนินการ DLPAR ทั้งหมดไม่ให้เกิดขึ้น

เรื่องการส่งสัญญาณตามเวลาสามารถถูกจัดการโดยแอ็พพลิเคชัน โดยการควบคุมมาสก์สัญญาณและการจัดลำดับความสำคัญ โค้ด DLPAR-aware สามารถถูกนำมารวมได้โดยตรงในอัลกอริธึม และตัวจัดการสัญญาณ สามารถถูกส่งต่อข้ามหลายไลบรารีที่แบ่งใช้ เพื่อที่การแจ้งเตือนสามารถ ถูกรวมในแบบมอดูลาร์มากขึ้น

เมื่อต้องการรวมเหตุการณ์ DLPAR โดยใช้ APIs ให้ปฏิบัติดังต่อไปนี้:

1. จับสัญญาณ **SIGRECONFIG** โดยใช้การเรียกระบบ **sigaction** การดำเนินการเริ่มต้น คือละเว้นสัญญาณ
2. ควบคุมมาสก์สัญญาณในอย่างน้อยหนึ่งเธรดเพื่อที่สัญญาณ สามารถส่งมอบได้ตามเวลาจริง
3. ตรวจสอบว่าการจัดลำดับความสำคัญสำหรับเธรดเพื่อรับ สัญญาณมีความเหมาะสม เพื่อที่เธรดจะรันได้อย่างรวดเร็วหลังจากสัญญาณ ได้ถูกส่ง
4. รันการเรียกระบบ **dr_reconfig** เพื่อรับชนิดของทรัพยากร ชนิดการดำเนินการ เฟสของเหตุการณ์ และ ข้อมูลอื่นที่สัมพันธ์กับการร้องขอปัจจุบัน

หมายเหตุ: การเรียกระบบ **dr_reconfig** ถูกใช้ภายในตัวจัดการ สัญญาณเพื่อกำหนดลักษณะของการร้องขอ DLPAR

การจัดการการขึ้นต่อกัน DLPAR ของแอ็พพลิเคชัน

การร้องขอเพื่อลบ DLPAR อาจล้มเหลวได้หลายสาเหตุ โดยส่วนมาก เกิดจากรีซอร์สไม่ว่าง หรือขณะนี้มีรีซอร์สระบบไม่เพียงพอสำหรับการใช้เพื่อ ดำเนินการร้องขอให้เสร็จสมบูรณ์

ในกรณีเหล่านี้ รีซอร์สถูกให้คงอยู่ในสถานะปกติ เหมือนว่าเหตุการณ์ DLPAR ไม่เคย เกิดขึ้น

สาเหตุหลักของ *การลบ ตัวประมวลผล* ล้มเหลวคือการโยงตัวประมวลผล ระบบปฏิบัติการไม่สามารถละเลย การโยงตัวประมวลผลและดำเนินการ DLPAR ต่อไปได้ หรือแอ็พพลิเคชัน อาจไม่สามารถทำงานต่อได้อย่างเหมาะสม เพื่อให้แน่ใจว่าปัญหานี้ไม่เกิดขึ้น ให้รีลีสการโยง สร้างการโยงใหม่ หรือจบการทำงานแอ็พพลิเคชัน กระบวนการ ที่ระบุ หรือเธรดที่ได้รับผลกระทบคือฟังก์ชันของประเภทของการโยงที่ นำมาใช้

สาเหตุหลัก ของ *การลบหน่วยความจำ* ล้มเหลวคือไม่มีหน่วยความจำที่กำหนด เพียงพอในระบบเพื่อที่จะดำเนินการร้องขอให้เสร็จสมบูรณ์ได้ นี้เป็นปัญหาระดับระบบ และไม่จำเป็นต้องส่งผลต่อแอ็พพลิเคชันที่ระบุ ถ้าหน้า ในส่วนหน่วยความจำที่จะถูกลบออกมีหน้าที่ถูกกำหนด เนื้อหาของหน้าต้อง ถูกโอนย้ายไปยังหน้าอื่นที่ถูกกำหนด ขณะที่ทำการดูแลรักษาการแสดงเสมือนใน การแม็พแบบฟิสิคัลไว้ ความล้มเหลวเกิดขึ้นเมื่อไม่มีหน่วยความจำที่สามารถกำหนดได้ เพียงพอในระบบเพื่อช่วยในการโอนย้ายของข้อมูลที่ถูกกำหนดใน ส่วนที่จะถูกลบออก เพื่อให้แน่ใจว่าปัญหานี้ไม่เกิดขึ้น ให้ลดระบบของหน่วยความจำ ที่ถูกกำหนดในระบบลง ซึ่งสามารถทำได้โดยการ destroy เซ็กเมนต์หน่วยความจำที่แบ่งใช้ที่ถูกกำหนด ยุติการทำงานโปรแกรมที่มีการเรียกใช้ระบบ **plock** หรือการลบ **plock** บนโปรแกรม

สาเหตุหลัก ของ *การถอนสล็อต PCI ออก* ล้มเหลวคืออะแด็ปเตอร์ ที่อยู่ในสล็อตไม่ว่าง โปรดทราบว่าการขึ้นต่อกันของอุปกรณ์จะไม่ถูกติดตาม ตัวอย่าง การขึ้นต่อกันของอุปกรณ์อาจขยายจากสล็อตหนึ่งไปที่หนึ่งในอุปกรณ์ต่อไปนี้: อะแด็ปเตอร์ อุปกรณ์ กลุ่มวอลุ่ม โลจิคัลวอลุ่ม ระบบไฟล์ หรือไฟล์ ในกรณีนี้ แก้ไขปัญหาการขึ้นต่อกันด้วยตนเองโดยการหยุดทำงานแอ็พพลิเคชันที่เกี่ยวข้อง เลิกการเม้าท์ระบบไฟล์ และการปิดใช้กลุ่มวอลุ่ม

หลักการที่เกี่ยวข้อง:

"การเชื่อมโยงโพรเซสเซอร์"

แอ็พพลิเคชันสามารถเชื่อมต่อโพรเซสเซอร์โดยใช้การเรียกของระบบ **bindprocessor** การเรียกของระบบนี้มีสมมุติฐานว่ารูปแบบการกำหนดหมายเลขโพรเซสเซอร์ เริ่มต้นด้วยศูนย์ (0) และสิ้นสุดด้วย *N*–1, โดยที่ *N* คือจำนวนของออนไลน์ CPU

การเชื่อมโยงโพรเซสเซอร์

แอ็พพลิเคชันสามารถเชื่อมต่อโพรเซสเซอร์โดยใช้การเรียกของระบบ **bindprocessor** การเรียกของระบบนี้มีสมมุติฐานว่ารูปแบบการกำหนดหมายเลขโพรเซสเซอร์ เริ่มต้นด้วยศูนย์ (0) และสิ้นสุดด้วย *N*–1, โดยที่ *N* คือจำนวนของออนไลน์ CPU

N ถูกระบุ ทางโปรแกรมโดยการอ่านตัวแปรระบบ **_system_configuration.ncpus** ขณะที่โพรเซสเซอร์ถูกเพิ่มและเอาออก ตัวแปรนี้ ถูกเพิ่มค่าและลดค่าโดยใช้ dynamic logical partitioning

กรุณาสังเกตว่า รูปแบบการกำหนดหมายเลขไม่รวมช่องว่าง โพรเซสเซอร์ถูกเพิ่มให้กับตำแหน่ง *N*th และถูกเอาออกจากตำแหน่ง *N*th–1 เสมอ รูปแบบการกำหนดหมายเลขที่ใช้โดยการเรียกระบบ **bindprocessor** ไม่สามารถถูกใช้เพื่อเชื่อมต่อกับโลจิคัลโพรเซสเซอร์ เนื่องจากโพรเซสเซอร์สามารถถูกเอาออกและไม่มีการเปลี่ยน ค่าในรูปแบบการกำหนดหมายเลขเนื่องจาก *N*th–1 CPU ถูกยกเลิกการจัดสรรเสมอ ด้วยเหตุนี้ ตัวระบุที่ใช้โดยการเรียกระบบ **bindprocessor** ถูกเรียกว่า *bind CPU IDs*

การเปลี่ยนแปลง กับตัวแปรระบบ **_system_configuration.ncpus** มีสิ่งที่เกี่ยวข้องดังต่อไปนี้:

- แอ็พพลิเคชันต้องถูกเตรียมการเพื่อรับข้อผิดพลาดจาก การเรียกระบบ **bindprocessor** ถ้าโพรเซสเซอร์สุดท้าย ถูกเอาออกหลังจากแอ็พพลิเคชันทำการอ่านตัวแปร เงื่อนไขข้อผิดพลาดนี้ ถูกนำมาใช้ครั้งแรกโดย Dynamic Processor Deallocation (การยกเลิกการจัดสรรรันไทม์ของ โพรเซสเซอร์ที่บกพร่อง)
- แอ็พพลิเคชันที่ถูกออกแบบให้สเกลตามจำนวนโพรเซสเซอร์ ต้องอ่านตัวแปรระบบ **_system_configuration.ncpus** ใหม่เมื่อจำนวนของโพรเซสเซอร์เปลี่ยนแปลง

แอ็พพลิเคชัน สามารถเชื่อมโยงกับชุดของโพรเซสเซอร์ได้เช่นกัน โดยใช้คุณลักษณะของ Workload Manager (WLM) ที่เรียกว่า *Software Partitioning* โดยมีสมมุติฐานว่ารูปแบบ การกำหนดหมายเลขมาจากโลจิคัล CPU IDs, ซึ่งเริ่มต้นด้วยศูนย์ (0) และ สิ้นสุดด้วย *N*–1 อย่างไรก็ตาม *N* ใน กรณีนี้คือจำนวนโพรเซสเซอร์สูงสุดที่สามารถรองรับได้ทางสถาปัตยกรรม โดยพาร์ติชัน รูปแบบการกำหนดหมายเลขมีผลทั้งกับโพรเซสเซอร์ออนไลน์และออฟไลน์

ดังนั้นเป็นสิ่งสำคัญที่ต้อง สนใจชนิดของการเชื่อมต่อที่ถูกใช้ เพื่อที่วิธีแก้ไขที่ถูกต้อง จะสามารถนำมาใช้ได้เมื่อเอาโพรเซสเซอร์ออก คำสั่ง **bindprocessor** สามารถถูกใช้เพื่อกำหนดจำนวนของออนไลน์โพรเซสเซอร์ คำสั่ง **ps** สามารถถูกใช้เพื่อระบุกระบวนการและเธรดที่ถูก เชื่อมโยงกับโพรเซสเซอร์ออนไลน์ล่าสุด หลังจากเป้าหมายได้ถูกระบุ คำสั่ง **bindprocessor** สามารถถูกใช้อีกครั้งเพื่อ กำหนดสิ่งที่แนบใหม่

ส่วนเพิ่มเติมที่สัมพันธ์กับ WLM สามารถถูกจำแนกได้โดย การระบุพาร์ติชันซอฟต์แวร์ที่ทำให้เกิดปัญหา เพื่อแก้ปัญหาส่วนเพิ่มเติม เหล่านี้ ให้ทำดังต่อไปนี้:

หมายเหตุ: ตารางเวลาระบบที่เชื่อมต่อ งานของโพรเซสเซอร์ ออฟไลน์หรือออฟไลน์ค้างอยู่ ดังนั้นไม่จำเป็นต้องมีการเปลี่ยนแปลง ถ้าซอฟต์แวร์พาร์ติชันมีออนไลน์ CPU อื่น

1. ใช้คำสั่ง **lsrset** เพื่อดูชุดของซอฟต์แวร์พาร์ติชัน ที่ถูกใช้โดย WLM
2. ระบุซอฟต์แวร์พาร์ติชันโดยใช้คำสั่ง **lsclass**
3. ระบุชุดของคลาสที่ใช้ซอฟต์แวร์พาร์ติชันเหล่านี้โดยใช้ คำสั่ง **chclass**
4. จำแนกคลาสระบบอีกครั้งโดยใช้คำสั่ง **wlmctrl**

ที่จุดนี้ ข้อกำหนดคลาสใหม่มีผล และระบบย้ายงานที่เชื่อมโยง โดยอัตโนมัติออกจากโลจิคัลโพรเซสเซอร์ที่กำลังถูกเอาออก

หลักการที่เกี่ยวข้อง:

"การจัดการการขึ้นต่อกัน DLPAR ของแอ็พพลิเคชัน" ในหน้า 641
การร้องขอเพื่อลบ DLPAR อาจล้มเหลวได้หลายสาเหตุ โดยส่วนมาก เกิดจากรีซอร์สไม่ว่าง หรือขณะนี้มีรีซอร์สระบบไม่เพียงพอสำหรับการใช้เพื่อ ดำเนินการร้องขอให้เสร็จสมบูรณ์

การรวมการดำเนินการ DLPAR ลงในแอ็พพลิเคชัน

การดำเนินการ DLPAR สามารถถูกรวมไว้ในแอ็พพลิเคชัน ในวิธีดังต่อไปนี้:

- การใช้วิธี script-based ซึ่งผู้ใช้ติดตั้งชุดของสคริปต์ DLPAR ลงในไดเร็กทอรี สคริปต์เหล่านี้ถูกเรียกขณะการดำเนินการ DLPAR ถูกรัน สคริปต์ถูกออกแบบเพื่อการกำหนดค่าแอ็พพลิเคชันซ้ำภายนอก
- การใช้สัญญาณ **SIGRECONFIG** ซึ่งถูกใช้เพื่อจับสัญญาณของทุกกระบวนการที่ได้รีจิสเตอร์ เมธอด สัญญาณถือว่าแอ็พพลิเคชันได้ถูกโค้ดเพื่อจับสัญญาณ และตัวจัดการสัญญาณนั้นจะกำหนดค่าแอ็พพลิเคชันใหม่ ตัวจัดการสัญญาณ เรียกอินเตอร์เฟสเพื่อกำหนดลักษณะของการดำเนินการ DLPAR

เมธอดทั้งสองเป็นไปตามโครงสร้างระดับสูงเหมือนกัน เมธอดทั้งสอง สามารถถูกใช้เพื่อจัดเตรียมการสนับสนุนสำหรับ DLPAR, แม้ว่าเฉพาะกลไก script-based เท่านั้นที่สามารถถูกใช้เพื่อจัดการส่วนเพิ่มเติม DLPAR ที่สัมพันธ์กับซอฟต์แวร์พาร์ติชัน Workload Manager (ชุดโพรเซสเซอร์) ไม่มี API สัมพันธ์กับ Workload Manager ดังนั้น การใช้ตัวจัดการสัญญาณไม่เหมาะสำหรับการจัดการกับข้อจำกัดการจัดตารางเวลา Workload Manager-imposed ตัวแอ็พพลิเคชันเองไม่ใช่ Workload Manager-aware ในกรณีนี้ ผู้ดูแลระบบอาจต้องการจัดเตรียมสคริปต์ที่เรียก คำสั่ง Workload Manager เพื่อจัดการการโต้ตอบ DLPAR กับ Workload Manager

การตัดสินว่าจะใช้เมธอดใดควรอยู่บนพื้นฐานที่ว่าระบบหรือ ตรรกะจำเพาะทรัพยากรถูกนำมาใช้ในแอ็พพลิเคชันอย่างไร ถ้าแอ็พพลิเคชัน ถูกกำหนดภายนอกให้ใช้จำนวนเธรดที่เจาะจงหรือเพื่อกำหนดขนาดบัฟเฟอร์ให้ใช้แนวทาง script-based ถ้าแอ็พพลิเคชันรับรู้การกำหนดค่าของระบบโดยตรง และใช้ข้อมูลนี้ตามลำดับ ให้ใช้ แนวทาง signal-based

ตัวการดำเนินการ DLPAR เองถูกแบ่งเป็นเฟสดังต่อไปนี้:

- **เฟส check**

  เฟส check ถูกเรียกก่อนและเปิดใช้งานแอ็พพลิเคชันเพื่อทำให้การร้องขอ DLPAR ปัจจุบัน ล้มเหลว ก่อนที่ภาวะในระบบจะถูกเปลี่ยน ตัวอย่างเช่น เฟส check ควรถูก ใช้โดยตัวจัดการไลเซนส์ CPU-based เพื่อทำให้การรวมโพรเซสเซอร์ใหม่ล้มเหลว ถ้า CPU ที่เพิ่มนั้นทำให้จำนวนโพรเซสเซอร์ในระบบเกิน จำนวนของโพรเซสเซอร์ที่ได้ไลเซนส์ และสามารถถูกใช้เพื่อรักษาความปลอดภัย DLPAR ของโปรแกรมที่ไม่ใช่ DLPAR-safe ในกรณีหลัง ต้องมีการพิจารณา เซอร์วิสที่จัดเตรียมโดยแอ็พพลิเคชัน เนื่องจากอาจเป็นการ ดีกว่าที่จะหยุดโปรแกรม, ทำตามการร้องขอให้สมบูรณ์ จากนั้นรีสตาร์ทโปรแกรม
- **เฟส pre** และ **เฟส post**

  เฟส pre และ *เฟส post* ถูกจัดเตรียมเพื่อหยุดโปรแกรม ทำตามการร้องขอให้สมบูรณ์ จากนั้นรีสตาร์ทโปรแกรม

ระบบพยายามทำให้แน่ใจว่าโค้ดการตรวจสอบทั้งหมดใน สื่อต่างกันถูกดำเนินการในระดับระบบทั้งหมดก่อนที่ เหตุการณ์ DLPAR ดำเนินไปยังเฟสถัดไป

การดำเนินการที่ทำโดยสคริปต์ DLPAR

แอ็พพลิเคชันสคริปต์ถูกเริ่มทำงานสำหรับทั้งการดำเนินการเพิ่ม และลบออก

เมื่อเอาทรัพยากรออก, สคริปต์ถูกจัดเตรียมเพื่อแก้ปัญหาเงื่อนไข ที่เกิดจากแอ็พพลิเคชันที่ป้องกันไม่ให้เอาทรัพยากรออก การเชื่อมโยงของโพรเซสเซอร์และการขาด pinnable memory อาจทำให้การร้องขอการเอาทรัพยากรออกล้มเหลว ชุดของคำสั่ง ถูกจัดเตรียมเพื่อระบุสถานการณ์เหล่านี้ ดังนั้นสคริปต์สามารถถูก เขียนเพื่อแก้ปัญหา

เพื่อระบุและแยกแยะความไม่เป็นอิสระของ DLPAR, สามารถใช้คำสั่งดังต่อไปนี้:

- คำสั่ง **ps** แสดงสิ่งที่แนบของ bindprocessor และสถานะการเรียกระบบ **plock** ที่ ระดับกระบวนการ
- คำสั่ง **bindprocessor** แสดง ออนไลน์โพรเซสเซอร์และสร้างสิ่งที่แนบใหม่
- คำสั่ง **kill** ส่งสัญญาณไปที่ กระบวนการ
- คำสั่ง **ipcs** แสดงเซ็กเมนต์ shared-memory ที่พินที่ระดับกระบวนการ
- คำสั่ง **lsrset** แสดงโพรเซสเซอร์ เซ็ต
- คำสั่ง **lsclass** แสดงคลาส Workload Manager ซึ่งอาจรวมอยู่ในโพรเซสเซอร์เซ็ต
- คำสั่ง **chclass** ถูกใช้เพื่อเปลี่ยน ข้อกำหนดคลาส

สคริปต์ยังสามารถถูกใช้ในเรื่องการปรับสเกลได้และประสิทธิภาพ ทั่วไป เมื่อทรัพยากรถูกเอาออก คุณสามารถลดจำนวนเธรด ที่ถูกใช้หรือขนาดของบัฟเฟอร์แอ็พพลิเคชัน เมื่อมีการเพิ่ม ทรัพยากร คุณสามารถเพิ่มพารามิเตอร์เหล่านี้ได้ คุณสามารถจัดเตรียมคำสั่ง ซึ่งสามารถถูกใช้เพื่อทำการปรับค่าเหล่านี้แบบไดนามิก ซึ่งสามารถ ถูกทริกเกอร์ได้โดยสคริปต์เหล่านี้ ติดตั้งสคริปต์เพื่อเริ่มทำงานคำสั่ง เหล่านี้ภายในบริบทของการดำเนินการ DLPAR

โครงสร้างระดับสูงของสคริปต์ DLPAR

ส่วนนี้จัดเตรียมภาพรวมของสคริปต์ ซึ่งสามารถเป็น สคริปต์ Perl, สคริปต์ shell หรือคำสั่ง แอ็พพลิเคชันสคริปต์จำเป็น ในการจัดเตรียมคำสั่งดังต่อไปนี้:

- **scriptinfo**

  ระบุเวอร์ชัน, วันที่ และผู้ค้าของสคริปต์ ซึ่งถูกเรียกเมื่อสคริปต์ถูกติดตั้ง

- **register**

  ระบบทรัพยากรที่จัดการโดย สคริปต์ ถ้าสคริปต์ส่งคืนชื่อรีซอร์ส *cpu*, *mem*, *capacity* หรือ *var_weight* สคริปต์ถูกเริ่มทำงาน โดยอัตโนมัติเมื่อ DLPAR พยายาม กำหนดคอนฟิกตัวประมวลผล หน่วยความจำ ความจุที่ถือสิทธิ หรือน้ำหนักตัวแปร ใหม่ คำสั่งรีจิสเตอร์ถูกเรียกเมื่อ สคริปต์ถูกติดตั้งกับระบบย่อย DLPAR

- **usage** *resource_name*

  ส่งกลับ ข้อมูลที่อธิบายวิธีที่ทรัพยากรถูกใช้โดยแอ็พพลิเคชัน คำอธิบายควรสัมพันธ์กัน ดังนั้นผู้ใช้สามารถกำหนด ว่าจะติดตั้งหรือถอนการติดตั้งสคริปต์ ซึ่งควรระบุ ความสามารถของซอฟต์แวร์ของแอ็พพลิเคชันที่ได้รับผลกระทบ คำสั่ง **usage** ถูกเรียกสำหรับแต่ละทรัพยากรที่ถูก ระบุโดยคำสั่ง **register**

- **checkrelease** *resource_name*

  ระบุ ว่าระบบย่อย DLPAR ควรดำเนินต่อหรือไม่กับการเอา ทรัพยากรที่มีชื่อออก สคริปต์อาจบ่งชี้ว่าทรัพยากรไม่ควรถูกเอาออก ถ้าแอ็พพลิเคชันไม่ใช่ DLPAR-aware และ แอ็พพลิเคชันมีความสำคัญกับการทำงานของระบบ

- **prerelease** *resource_name*

  กำหนดค่าใหม่ หยุดทำงานชั่วคราว หรือสิ้นสุดแอ็พพลิเคชัน เพื่อการถือครอง ทรัพยากรที่กำหนดชื่อถูกรีลีส

- **postrelease** *resource_name*

  ดำเนินต่อ หรือรีสตาร์ทแอ็พพลิเคชัน

- **undoprerelease** *resource_name*

  เริ่มทำงานถ้า พบข้อผิดพลาด และรีซอร์สไม่ถูกรีลีส

- **checkacquire** *resource_name*

  บ่งชี้ ว่าระบบย่อย DLPAR ควรดำเนินต่อด้วยการเพิ่มเติม ทรัพยากรหรือไม่ ซึ่งอาจถูกใช้โดยตัวจัดการไลเซนส์เพื่อป้องกันการเพิ่มเติม ทรัพยากรใหม่ ตัวอย่างเช่น cpu จนกว่าทรัพยากรจะได้รับไลเซนส์

- **preacquire** *resource_name*

  ใช้ เพื่อเตรียมการสำหรับการเพิ่มเติมทรัพยากร

- **undopreacquire** *resource_name*

  เริ่มทำงานถ้าพบ ข้อผิดพลาดในเฟส preacquire หรือเมื่อเกิด เหตุการณ์ขึ้น

- **postacquire** *resource_name*

  ดำเนินต่อ หรือเริ่มการทำงานแอ็พพลิเคชัน

- **preaccevent** *resource_name*

  ใช้ เพื่อจัดเตรียมการอัพเดต DLPAR

- **postaccevent** *resource_name*

  ดำเนินต่อ หรือเริ่มการทำงานแอ็พพลิเคชัน

- **undopreaccevent** *resource_name*

  เริ่มทำงาน ถ้าพบข้อผิดพลาดในเฟส the **preaccevent** หรือเมื่อ เกิดเหตุการณ์ขึ้น

- **pretopolgyupdate** *resource_name*

ใช้เพื่อเตรียมการสำหรับการอัพเดตโทโพโลยีของระบบ

- **postopolgyupdate** *resource_name*

  ดำเนินต่อ หรือเริ่มการทำงานแอ็พพลิเคชัน

การติดตั้งสคริปต์แอ็พพลิเคชันโดยใช้คำสั่ง drmgr

คำสั่ง **drmgr** คง ฐานข้อมูลภายในของข้อมูลสคริปต์ที่ติดตั้ง ข้อมูล นี้ถูกรวบรวมเมื่อระบบถูกบูตและถูกรีเฟชร เมื่อสคริปต์ใหม่ถูกติดตั้งหรือถอนการติดตั้ง ข้อมูลถูก สืบทอดจากคำสั่ง **scriptinfo, register,** และ **usage** การติดตั้งของสคริปต์ ถูกสนับสนุนผ่านคำสั่ง **drmgr** ซึ่ง คัดลอกสคริปต์ที่กำหนดชื่อกับที่เก็บสคริปต์ ซึ่งสามารถเข้า ถึงได้ภายหลัง ตำแหน่งเริ่มต้นสำหรับที่เก็บนี้คือ **/usr/lib/dr/scripts/all** ภายในพาร์ติชันเวิร์กโหลด ตำแหน่งที่เก็บสคริปต์เริ่มต้นคือ **/var/dr/scripts** คุณสามารถระบุตำแหน่งอื่นสำหรับที่เก็บนี้ เพื่อกำหนด เครื่องกับสคริปต์ที่ถูกใช้ ให้ระบุชื่อโฮสต์เป้าหมาย เมื่อติดตั้งสคริปต์

เพื่อระบุตำแหน่งของที่เก็บฐาน ใช้คำสั่ง ดังต่อไปนี้:

```
drmgr -R base_directory_path
```

เพื่อติดตั้งสคริปต์ ใช้คำสั่งดังต่อไปนี้:

```
drmgr -i script_name [-f] [-w mins] [-D hostname]
```

มีการกำหนดแฟล็กดังต่อไปนี้:

- แฟล็ก **–i** ถูกใช้เพื่อตั้งชื่อสคริปต์
- แฟล็ก **–f** ต้องถูกใช้เพื่อแทนที่สคริปต์ที่ รีจิสเตอร์
- แฟล็ก **–w** ถูกใช้เพื่อระบุจำนวนของ นาทีที่สคริปต์ถูกคาดว่าจะดำเนินการ ข้อมูลนี้ถูกจัดเตรียม เป็นตัวเลือกแทนที่กับค่าที่ระบุโดยผู้ค้า
- แฟล็ก **–D** ถูกใช้เพื่อรีจิสเตอร์สคริปต์ ที่จะถูกใช้บนโฮสต์

เพื่อถอนการติดตั้งสคริปต์ ใช้คำสั่งดังต่อไปนี้:

```
drmgr -u script_name [-D hostname]
```

มีการกำหนดแฟล็กดังต่อไปนี้:

- แฟล็ก **–u** ถูกใช้เพื่อระบุสคริปต์ ที่ควรถูกถอนการติดตั้ง
- แฟล็ก **–D** ถูกใช้เพื่อถอนการติดตั้งสคริปต์ ที่ถูกรีจิสเตอร์สำหรับไดเร็กทอรี

เพื่อแสดงข้อมูลเกี่ยวกับสคริปต์ที่ ได้ถูกติดตั้งแล้ว ใช้คำสั่งดังต่อไปนี้:

```
drmgr -l
```

หลักการตั้งชื่อสำหรับสคริปต์

ขอแนะนำให้ชื่อสคริปต์ถูกสร้างจากชื่อผู้ค่า และระบบย่อยที่ถูกควบคุม ผู้ดูแลระบบ ควรตั้งชื่อสคริปต์ด้วยส่วนเติมหน้า *sysadmin* ตัวอย่างเช่น ผู้ดูแลระบบซึ่งต้องการจัดเตรียมสคริปต์ เพื่อควบคุมการกำหนดค่า Workload Manager อาจตั้งชื่อสคริปต์ว่า `sysadmin_wlm`

สภาวะแวดล้อมการเรียกทำงานสคริปต์และพารามิเตอร์อินพุต

สคริปต์ถูกเริ่มทำงานโดยมีสภาวะแวดล้อมการเรียกทำงานต่อไปนี้:

- Process UID ถูกตั้งค่าเป็น UID ของสคริปต์
- Process GID ถูกตั้งค่าเป็น GID ของสคริปต์
- ตัวแปรสภาวะแวดล้อม **PATH** ถูกตั้งค่า เป็นไดเร็กทอรี **/usr/bin:/etc:/usr/sbin**
- ตัวแปรสภาวะแวดล้อม **LANG** อาจหรืออาจ จะไม่ถูกตั้งค่า
- ไดเร็กทอรีการทำงาน Current ถูกตั้งค่าเป็น **/tmp**
- อาร์กิวเมนต์คำสั่งและตัวแปรสภาวะแวดล้อมถูกใช้เพื่ออธิบาย เหตุการณ์ DLPAR

สคริปต์ได้รับพารามิเตอร์อินพุตผ่านอาร์กิวเมนต์คำสั่งและ ตัวแปรสภาวะแวดล้อม และจัดเตรียมเอาต์พุตโดยเขียนคู่ *name=value* ไปที่เอาต์พุตมาตรฐาน โดยที่คู่ *name=value* ถูกคั่นด้วยบรรทัดใหม่ *name* ถูกกำหนด เป็นชื่อของรายการข้อมูลส่งกลับที่คาดไว้ และ *value* คือค่าที่สัมพันธ์กับรายการข้อมูล สตริงข้อความต้องถูกปิดด้วยวงเล็บ ตัวอย่างเช่น `DR_ERROR="text"` ตัวแปรสภาวะแวดล้อมทั้งหมดและคู่ *name=value* ต้องเริ่มด้วย `DR_` ซึ่งถูกสำรองไว้สำหรับสื่อสารกับ แอ็พพลิเคชันสคริปต์

สคริปต์ใช้คู่ตัวแปรสภาวะแวดล้อม **DR_ERROR** *name=value* เพื่อจัดเตรียมการอธิบายข้อผิดพลาด

คุณสามารถตรวจสอบอาร์กิวเมนต์คำสั่งกับสคริปต์เพื่อกำหนด เฟสของการดำเนินการ DLPAR ชนิดของการดำเนินการ และชนิดของทรัพยากรที่เป็นประเด็นของการร้องขอ DLPAR ที่ค้างอยู่ ตัวอย่างเช่น ถ้าอาร์กิวเมนต์คำสั่งสคริปต์คือ `checkrelease mem` ดังนั้นเฟสคือ `check` การดำเนินการคือ `remove` และชนิดของทรัพยากรคือ `memory` ทรัพยากรที่ระบุซึ่งเกี่ยวข้อง สามารถถูกระบุโดยการตรวจสอบตัวแปรสภาวะแวดล้อม

ตัวแปรสภาวะแวดล้อมดังต่อไปนี้ ถูกตั้งค่าสำหรับการเพิ่มและการเอาหน่วยความจำออก:

หมายเหตุ: ในคำอธิบาย ดังต่อไปนี้ หนึ่งเฟรมเท่ากับ 4 KB

- **DR_FREE_FRAMES**=`0xFFFFFFFF`
  จำนวนของ เฟรมที่ว่างขณะนี้ในระบบ ในรูปแบบเลขฐานสิบหก
- **DR_MEM_SIZE_COMPLETED**=*n*
  จำนวน เมกะไบต์ที่เพิ่มหรือเอาออกสำเร็จ ในรูปแบบ เลขฐานสิบ
- **DR_MEM_SIZE_REQUEST**=*n*
  ขนาด หน่วยความจำที่ร้องขอเป็นเมกะไบต์ ในรูปแบบเลขฐานสิบ
- **DR_PINNABLE_FRAMES**=`0xFFFFFFFF`
  จำนวน รวมของเฟรม pinnable ที่มีอยู่ในระบบในรูปแบบ เลขฐานสิบหก พารามิเตอร์นี้จัดเตรียมข้อมูลที่มีค่า เมื่อเอาหน่วยความจำออก ซึ่งสามารถถูกใช้เพื่อระบุเวลาที่ระบบ ใกล้ถึงขีดจำกัดของหน่วยความจำ pinnable ซึ่งเป็นสาเหตุหลักของ ความล้มเหลวในการร้องขอการเอาหน่วยความจำออก
- **DR_TOTAL_FRAMES**=`0xFFFFFFFF`
  จำนวน รวมของเฟรมที่มีอยู่ในระบบในรูปแบบเลขฐานสิบหก

ตัวแปรสภาวะแวดล้อมดังต่อไปนี้ ถูกตั้งค่าสำหรับการเพิ่มและการเอาโพรเซสเซอร์ออก:

- **DR_BCPUID**=*N*

CPU ID การเชื่อมโยงของโพรเซสเซอร์ที่ถูกเพิ่มหรือเอาออกในรูปแบบ เลขฐานสิบ การผนวก **bindprocessor** กับโพรเซสเซอร์นี้ ไม่จำเป็นต้องหมายถึงว่าการผนวกต้องถูกยกเลิก ค่านี้เป็นจริงต่อเมื่อเป็นตัวประมวลผลที่ *N* ในระบบเนื่องจากตัวประมวลผลที่ *N* เป็นตัว ที่ถูกลบออกในการดำเนินการลบของ Central Processing Unit (CPU) เสมอ Bind ID ตามปกติจะเรียงลำดับ โดยเริ่มจาก 0 ถึง *N* และ ถูกใช้เพื่อระบุเฉพาะออนไลน์โพรเซสเซอร์ ใช้คำสั่ง **bindprocessor** เพื่อกำหนดจำนวนออนไลน์ CPU

- **DR_LCPUID**=*N*

  Central Processor Unit (CPU) ID โลจิคัลของตัวประมวลผลที่กำลังถูกเพิ่มหรือลบออกในรูปแบบ ทศนิยม

ตัวแปรสภาวะแวดล้อมดังต่อไปนี้ถูกตั้งค่าสำหรับ Micro-Partitioning

DR_CPU_CAPACITY=N
เปอร์เซ็นต์พาร์ติชันของโพรเซสเซอร์ฟิสิคัลที่ที่แบ่งใช้

DR_VAR_WEIGHT=N
ลำดับความสำคัญความสัมพันธ์ของพาร์ติชันเพื่อกำหนดวิธีจัดสรร วงรอบ pool idle ที่แบ่งใช้

DR_CPU_CAPACITY_DELTA=N
ความแตกต่างระหว่างค่าปัจจุบันของเปอร์เซ็นต์พาร์ติชัน ของฟิสิคัลโพรเซสเซอร์ที่แบ่งใช้ และค่าซึ่งจะถูกเปลี่ยนแปลง เมื่อการดำเนินการนี้สมบูรณ์

DR_VAR_WEIGHT_DELTA=N
ความแตกต่างระหว่างค่าปัจจุบันน้ำหนักตัวแปรของพาร์ติชัน และค่าซึ่งจะถูกเปลี่ยนแปลง เมื่อการดำเนินการนี้สมบูรณ์

ตัวดำเนินการ สามารถแสดงข้อมูลเกี่ยวกับการร้องขอ DLPAR ปัจจุบัน โดยใช้ระดับรายละเอียดที่ HMC เพื่อสังเกตุเมื่อเหตุการณ์ เกิดขึ้น พารามิเตอร์นี้ถูกระบุกับสคริปต์ โดยใช้ตัวแปรสภาวะแวดล้อม **DR_DETAIL_LEVEL**=*N* โดยที่ *N* สามารถมีช่วงจาก 0 ถึง 5 ค่าดีฟอลต์ เป็นศูนย์ (0) และหมายถึงไม่มีข้อมูล ค่า หนึ่ง (1) ถูกสำรองไว้สำหรับระบบปฏิบัติการและถูกใช้เพื่อแสดง โฟลว์ระดับสูง ระดับที่เหลือ (2-5) สามารถถูกใช้โดยสคริปต์ เพื่อให้ข้อมูลพร้อมกับสมมุติฐานว่าตัวเลขที่มากกว่าให้รายละเอียดมากกว่า

สคริปต์ให้ข้อมูลรายละเอียดโดยเขียนคู่ *name=value* ดังต่อไปนี้ไปที่เอาต์พุตมาตรฐาน:

name=value pair	**คำอธิบาย**
DR_LOG_ERR=*message*	บันทึกข้อความพร้อมกับระดบ syslog ของตัวแปรสภาวะแวดล้อม **LOG_ERR**
DR_LOG_WARNING=*message*	บันทึกข้อความพร้อมกับระดบ syslog ของตัวแปรสภาวะแวดล้อม **LOG_WARNING**
DR_LOG_INFO=*message*	บันทึกข้อความพร้อมกับระดบ syslog ของตัวแปรสภาวะแวดล้อม **LOG_INFO**
DR_LOG_EMERG=*message*	บันทึกข้อความพร้อมกับระดบ syslog ของตัวแปรสภาวะแวดล้อม **LOG_EMERG**
DR_LOG_DEBUG=*message*	บันทึกข้อความพร้อมกับระดบ syslog ของตัวแปรสภาวะแวดล้อม **LOG_DEBUG**

นอกจากนี้ ตัวดำเนินการ ยังสามารถตั้งค่าบันทึกของข้อมูลที่ถูกรักษาไว้โดยใช้หน่วยบริการ **syslog** ในกรณีที่ข้อมูลด้านบนถูกส่งไปที่หน่วยบริการนั้นเช่นกัน คุณต้องกำหนดค่าหน่วยบริการ **syslog** ในกรณีนี้

DLPAR คำสั่งสคริปต์

ส่วนนี้อธิบายคำสั่งสคริปต์สำหรับ DLPAR:

`scriptinfo`
ให้ข้อมูลเกี่ยวกับสคริปต์ที่ติดตั้ง เช่นวันที่สร้างและทรัพยากร

`register`
เริ่มทำงานเพื่อรวบรวมรายการรีซอร์สที่ถูกจัดการโดย สคริปต์ คำสั่ง **drmgr** ใช้รายการเหล่านี้เพื่อเริ่มทำงานสคริปต์ ตามชนิดของรีซอร์สที่กำลังถูกกำหนดคอนฟิกใหม่

`การใช้`
จัดเตรียมสตริงที่ผู้ใช้อ่านได้ อธิบายเซอร์วิสที่จัดเตรียมโดยทรัพยากรมีชื่อที่กำหนด บริบทของข้อความควร ช่วยเหลือผู้ใช้ตัดสินความไม่ชัดเจนเกี่ยวกับแอ็พพลิเคชันและ เซอร์วิสเมื่อทรัพยากรที่กำหนดชื่อถูกกำหนดค่าใหม่ คำสั่งนี้เริ่มทำงาน เมื่อสคริปต์ถูกติดตั้ง และข้อมูลที่ระบุโดยคำสั่ง นี้จะถูกดูแลรักษาอยู่ในฐานข้อมูลภายในที่ใช้โดยคำสั่ง **drmgr** แสดงข้อมูล โดยใช้ตัวเลือกรายการ *-l* ของคำสั่ง **drmgr**

`checkrelease`
เมื่อเอาทรัพยากรออกคำสั่ง **drmgr** เข้าถึงผลของการเอาทรัพยากรออก ซึ่งรวมถึง การกระทำการของสคริปต์ DLPAR ที่นำไปใช้กับคำสั่ง **checkrelease** แต่ละสคริปต์ DLPAR ในทางกลับกันจะสามารถประเมิน คุณสมบัติเฉพาะของแอ็พพลิเคชันและแจ้งกับคำสั่ง **drmgr** ที่กำลังใช้รหัสที่ส่งกลับของสคริปต์ว่าการเอาทรัพยากรออก จะมีผลกับแอ็พพลิเคชันที่เกี่ยวข้องหรือไม่ ถ้าพบว่าการเอาทรัพยากรออก สามารถกระทำได้อย่างปลอดภัย สถานะจบการทำงานแจ้งว่าสำเร็จถูกส่งกลับ ถ้าแอ็พพลิเคชันอยู่ในสภาวะ ที่รีซอร์สเกิดภาวะวิกฤตในการเรียกทำงาน และไม่สามารถกำหนดคอนฟิกใหม่ได้ โดยไม่มีการขัดจังหวะการเรียกทำงานของแอ็พพลิเคชัน ดังนั้นสคริปต์ ระบุว่ารีซอร์สไม่ควรถูกลบโดยการส่งคืน ข้อผิดพลาด เมื่อตัวเลือก *FORCE* ถูกระบุโดยผู้ใช้ ซึ่งใช้กับการดำเนินการ DLPAR ทั้งหมดรวมถึงเฟส, คำสั่ง **drmgr** ข้ามคำสั่ง **checkrelease** และเริ่มต้นด้วย คำสั่ง **prerelease**

`prerelease`
ก่อนที่ทรัพยากรจะถูกรีลีส สคริปต์ DLPAR ถูกกำหนดให้ช่วยในการรีลีสทรัพยากรที่กำหนดชื่อ โดยลดหรือขจัดการใช้ทรัพยากรออกจากแอ็พพลิเคชัน อย่างไรก็ตาม ถ้าสคริปต์ตรวจพบว่าทรัพยากรไม่สามารถถูกรีลีส จากแอ็พพลิเคชัน สคริปต์ควรแจ้งว่าทรัพยากรจะไม่ถูกเอาออก จากแอ็พพลิเคชันโดยส่งกลับข้อผิดพลาด นี่ไม่ได้เป็นการ ป้องกันระบบจากการพยายามเอาทรัพยากรออกใน โหมด forced หรือ non-forced ของการกระทำการ และสคริปต์จะ ถูกเรียกในเฟสหลัง ไม่ว่าจะมีการดำเนินการหรือไม่โดย คำสั่ง **prerelease** การดำเนินการ โดยระบบปฏิบัติการปลอดภัย ถ้าทรัพยากรไม่สามารถถูกเอาออก เรียบร้อย การดำเนินการจะล้มเหลว

สคริปต์ DLPAR ถูก คาดหมายในการบันทึก การดำเนินการที่ทำโดยคำสั่ง **prerelease** แบบภายใน เพื่อที่การดำเนินการสามารถถูกคืนค่า ในเฟสหลัง ควรมีข้อผิดพลาดเกิดขึ้น ซึ่งยังสามารถถูกจัดการ ในเฟสหลัง ถ้ามีการสร้าง rediscovery แอ็พพลิเคชันอาจ จำเป็นต้องรับการวัดผลอย่างแม่นยำ ถ้ามีการระบุตัวเลือก *force*

`postrelease`
หลังจากรีซอร์สถูกรีลีสเรียบร้อย คำสั่ง **postrelease** สำหรับแต่ละสคริปต์ DLPAR ที่ติดตั้ง จะเริ่มทำงาน แต่ละสคริปต์ DLPAR ดำเนินการประมวลผล ภายหลัง ที่จำเป็นระหว่างขั้นตอนนี้ แอ็พพลิเคชันที่หยุดทำงานไป ควรถูกรีสตาร์ท

โปรแกรมที่เรียกจะละเว้นข้อผิดพลาด ที่รายงานโดยคำสั่ง **postrelease** และการดำเนินการจะถูกพิจารณาว่าสำเร็จ แม้ว่าการบ่งชี้ ถึงข้อผิดพลาดที่อาจเกิดขึ้นจะถูกรายงานไปที่ผู้ใช้ เช่นกัน ข้อความตัวแปรสภาวะแวดล้อม **DR_ERROR** ถูกจัดเตรียมสำหรับจุดประสงค์นี้ ดังนั้นข้อความควรระบุแอ็พพลิเคชัน ที่ไม่ได้ถูกกำหนดค่าใหม่อย่างถูกต้อง

`undoprerelease`
หลังจากคำสั่ง **prerelease** ถูกส่งออกมาโดยคำสั่ง **drmgr** ไปที่สคริปต์ DLPAR ถ้าคำสั่ง **drmgr** ล้มเหลวในการเอาออก หรือ รีลีสทรัพยากร คำสั่งจะพยายามเปลี่ยนกลับไปเป็นภาวะเดิม เนื่องจาก เป็นส่วนหนึ่งของกระบวนการนี้ คำสั่ง **drmgr**

จะ ส่งคำสั่ง **undoprerelease** ไปที่สคริปต์ DLPAR คำสั่ง **undoprerelease** จะเริ่มทำงานต่อเมื่อสคริปต์ที่เรียกใช้ก่อนหน้านี้ เพื่อรีลีส รีซอร์สในการร้องขอ DLPAR ปัจจุบัน ในกรณีนี้ สคริปต์ ควรยกเลิกการดำเนินการทั้งหมดที่กระทำโดยคำสั่ง **prerelease** ของสคริปต์ สุดท้าย สคริปต์อาจจำเป็นต้องทำเอกสาร การดำเนินการของสคริปต์ หรือจัดเตรียมความสามารถในการค้นหา ภาวะของระบบอีกครั้ง และกำหนดค่าแอ็พพลิเคชันใหม่ เพื่อให้มีผล๑ เหตุการณ์ DLPAR ไม่เคยเกิดขึ้น

`checkacquire`

คำสั่งนี้เป็นคำสั่ง DLPAR script-based แรกที่ถูกเรียกในลำดับ acquire-new-resource ถ้า ถูกเรียกสำหรับและสคริปต์ที่ติดตั้งซึ่งก่อนหน้านี้แจ้งว่า สนับสนุนชนิดของทรัพยากรที่กำลังถูกเพิ่ม หนึ่ง ในจุดประสงค์หลักของเฟส checkacquire คือเพื่อเปิดใช้งานตัวจัดการไลเซนส์ ตามแบบโพรเซสเซอร์ ซึ่งอาจต้องการทำให้การเพิ่มโพรเซสเซอร์ล้มเหลว คำสั่ง **checkacquire** ถูกเริ่มทำงานเสมอ ไม่ว่าค่าของตัวแปรสภาวะแวดล้อม **FORCE** เป็น ค่าใด และโปรแกรมที่เรียกใช้จะให้ค่าส่งคืนใดในสคริปต์ ผู้ใช้ไม่สามารถบังคับการเพิ่มโพรเซสเซอร์ใหม่ได้ ถ้าสคริปต์ หรือโปรแกรม DLPAR-aware ทำให้การดำเนินการ DLPAR ในเฟสตรวจสอบล้มเหลว

ตัวแปรสภาวะแวดล้อม **FORCE** ไม่ได้ใช้กับคำสั่ง **checkacquire** จริงๆ แม้ว่าจะใช้กับเฟสอื่น ในเฟส preacquire, คำสั่งบ่งชี้ระยะเวลาที่สคริปต์ควรทำงานเมื่อกำหนดค่าแอ็พพลิเคชันใหม่ ตัวเลือก *force* สามารถถูกใช้โดยสคริปต์ เพื่อควบคุมนโยบาย ซึ่งแอ็พพลิเคชันถูกหยุดและเริ่มใหม่ เหมือนกับเมื่อทรัพยากรถูกรีลีส ซึ่งส่วนใหญ่เป็น DLPAR-safe

`preacquire`

สมมุติว่าไม่มีข้อผิดพลาดถูกรายงานในเฟส checkacquire ระบบทำงานต่อไปยังเฟส preacquire ซึ่ง ชุดของสคริปต์เดิมถูกเรียกเพื่อเตรียมการสำหรับการรับ ทรัพยากรใหม่ ซึ่งถูกสนับสนุนผ่านคำสั่ง **preacquire** แต่ละสคริปต์ถูกเรียก ก่อนที่ระบบจะทำการรวม ทรัพยากรจริงๆ นอกจากว่ามีการรายงานข้อผิดพลาดและตัวแปรสภาวะแวดล้อม **FORCE** ไม่ถูกระบุโดย ผู้ใช้ ถ้าตัวแปรสภาวะแวดล้อม **FORCE** ถูกระบุ ระบบทำงานต่อไปที่ขั้นการรวม ไม่ว่า รหัสที่ส่งกลับของสคริปต์จะเป็นอะไร ไม่มีการตรวจพบข้อผิดพลาดเมื่อตัวแปรสภาวะแวดล้อม **FORCE** ถูกระบุ เนื่องจากข้อผิดพลาดทั้งหมด หลีกเลี่ยงได้โดยยกเลิกการกำหนดค่าแอ็พพลิเคชัน ซึ่งเป็น การปฏิบัติที่ยอมรับเมื่อมีการระบุตัวแปรสภาวะแวดล้อม **FORCE** ถ้าพบข้อผิดพลาดและตัวแปรสภาวะแวดล้อม **FORCE** ไม่ถูกระบุ ระบบจะทำงานต่อไปที่เฟส undopreacquire แต่เฉพาะสคริปต์ที่ ดำเนินการก่อนหน้านี้เท่านั้นในเฟสปัจจุบันที่จะถูกรันซ้ำ ระหว่างเฟสหลัง สคริปต์ถูกกำหนดให้ทำการดำเนินการกู้คืน

`undopreacquire`

เฟส undopreacquire ถูกจัดเตรียมเพื่อที่ สคริปต์สามารถทำการดำเนินการกู้คืนได้ ถ้าสคริปต์ถูกเรียก ในเฟส undopreacquire สคริปต์ถือได้ว่าได้ดำเนิน คำสั่ง **preacquire** เสร็จสมบูรณ์

`postacquire`

คำสั่ง **postacquire** ถูกดำเนินการหลังจากทรัพยากรถูกรวมเข้ากับระบบ สำเร็จ แต่ละสคริปต์ DLPAR ที่ก่อนหน้านี้ถูกเรียก ในเฟส check และ pre ถูกเรียกอีกครั้ง คำสั่งนี้ถูกใช้เพื่อ รวมทรัพยากรใหม่เข้ากับแอ็พพลิเคชัน ตัวอย่างเช่น แอ็พพลิเคชันอาจต้องการสร้างเธรดใหม่, ขยายบัฟเฟอร์ หรือแอ็พพลิเคชันอาจจำเป็นต้องถูกรีสตาร์ท ถ้าก่อนหน้านี้ หยุดทำงาน

`checkmigrate`

คำสั่งนี้เป็นคำสั่ง DLPAR ที่ใช้สคริปต์คำสั่งแรกที่ถูกเรียกใช้ในลำดับของการโอนย้ายระบบ ถ้า ถูกเรียกสำหรับและสคริปต์ที่ติดตั้งซึ่งก่อนหน้านี้แจ้งว่า สนับสนุนชนิดของทรัพยากรที่กำลังถูกเพิ่ม คำสั่ง **checkmigrate** ถูกเริ่มทำงานเสมอ ไม่ว่าค่าของตัวแปรสภาวะแวดล้อม **FORCE** เป็น ค่าใด และโปรแกรมที่เรียกใช้จะให้ค่าส่งคืนใดในสคริปต์ ผู้ใช้ไม่สามารถการโอนย้ายพาร์ติชันได้ ถ้าสคริปต์หรือโปรแกรม DLPAR-aware ทำให้การดำเนินการ DLPAR ในขั้นตอนการตรวจสอบล้มเหลว

premigrate

สมมติว่าไม่มีข้อผิดพลาดรายงานในเฟส **checkmigrate** ระบบจะไปยังเฟส **premigrate** โดยที่มีสคริปต์ ชุดเดียวกันถูกเริ่มทำงานเพื่อจัดเตรียมพาร์ติชันที่จะเกิดขึ้น สคริปต์แต่ละสคริปต์เหล่านี้จะถูกเรียกใช้ ก่อนที่ระบบจะพยายามโอนย้ายพาร์ติชันจริงๆ ระบบจะดำเนินการขั้นตอนการโอนย้าย โดยไม่สนใจสถานะโค้ดส่งคืนของสคริปต์ ถ้าพบข้อผิดพลาด ระบบจะดำเนินขั้นตอน **undopremigrate** แต่จะรันสคริปต์ที่ถูกเรียกทำงานก่อนหน้านี้ในขั้นตอนปัจจุบันใหม่ ระหว่างเฟสหลัง สคริปต์ถูกกำหนดให้ทำการดำเนินการกู้คืน

undopremigrate

ขั้นตอน **undopremigrate** ถูกจัดเตรียมเพื่อให้สคริปต์สามารถดำเนินดารกู้คืนได้ ถ้าสคริปต์ถูกเรียกใช้ในขั้นตอน **undopremigrate** สคริปต์ถือได้ว่าได้ใช้คำสั่ง **premigrate** เสร็จสมบูรณ์

postmigrate

คำสั่ง **postmigrate** จะถูกเรียกทำงานหลังจากพาร์ติชันถูกโอนย้ายเรียบร้อยแล้ว สคริปต์ DLPAR แต่ละสคริปต์ที่ถูกเรียกใช้ก่อนหน้านี้ในขั้นตอนการตรวจสอบและการเตรียมการจะถูกเรียกใช้อีกครั้ง

pretopologyupdate

คำสั่ง **pretopologyupdate** จะถูกเรียกทำงานก่อนการดำเนินการที่จะมีผลกับทอพอโลยีของพาร์ติชัน เช้น การเพิ่มหรือถอดตัวประมวลผลหรือหน่วยความจำ คำสั่งนี้จะแจ้งว่ามีการดำเนินการเกี่ยวกับทอพอโลยีและไม่สามารถล้มเหลว สคริปต์จะดำเนินการขั้นตอนการรวม โดยไม่สนใจสถานะโค้ดส่งคืนของสคริปต์

posttopologyupdate

คำสั่ง **posttopologyupdate** จะถูกเรียกทำงานหลังจากพาร์ติชันมีการดำเนินการเกี่ยวกับทอพอโลยีเรียบร้อยแล้ว สคริปต์ DLPAR แต่ละสคริปต์ที่ถูกเรียกใช้ก่อนหน้านี้จะถูกเรียกใช้อีกครั้ง

checkhibernate

คำสั่งนี้เป็นคำสั่ง DLPAR ที่ใช้สคริปต์คำสั่งแรกที่ถูกเรียกใช้ในลำดับของการ hibernation ถ้า ถูกเรียกสำหรับและสคริปต์ที่ติดตั้งซึ่งก่อนหน้านี้แจ้งว่า สนับสนุนชนิดของทรัพยากรที่กำลังถูกเพิ่ม คำสั่ง **checkhibernate** ถูกเริ่มทำงานเสมอ ไม่ว่าค่าของตัวแปรสภาวะแวดล้อม **FORCE** เป็น ค่าใด และโปรแกรมที่เรียกใช้จะให้ค่าส่งคืนใดในสคริปต์ ผู้ใช้ไม่สามารถบังคับการ hibernation ของพาร์ติชันได้ ถ้าสคริปต์หรือโปรแกรม DLPAR–aware ทำให้การดำเนินการ DLPAR ในขั้นตอนการตรวจสอบล้มเหลว

prehibernate

สมมติว่าไม่มีข้อผิดพลาดรายงานในเฟส **checkhibernate** ระบบจะไปยังเฟส **prehibernate** โดยที่มีสคริปต์ ชุดเดียวกันถูกเริ่มทำงานเพื่อจัดเตรียมพาร์ติชันที่จะเกิดขึ้น สคริปต์แต่ละสคริปต์เหล่านี้จะถูกเรียกใช้ ก่อนที่ระบบจะพยายาม hibernate พาร์ติชันจริงๆ ระบบจะดำเนินการขั้นตอนการ hibernate โดยไม่สนใจสถานะโค้ดส่งคืนของสคริปต์ ถ้าพบข้อผิดพลาด ระบบจะดำเนินขั้นตอน **undohibernate** แต่จะรันสคริปต์ที่ถูกเรียกทำงานก่อนหน้านี้ในขั้นตอนปัจจุบันใหม่ ระหว่างเฟสหลัง สคริปต์ถูกกำหนดให้ทำการดำเนินการกู้คืน

undohibernate

ขั้นตอน **undohibernate** ถูกจัดเตรียมเพื่อให้สคริปต์สามารถดำเนินดารกู้คืนได้ ถ้าสคริปต์ถูกเรียกใช้ในขั้นตอน **checkhibernate** สคริปต์ถือได้ว่าได้ใช้คำสั่ง **checkhibernate** เสร็จสมบูรณ์

posthibernate

คำสั่ง **posthibernate** จะถูกเรียกทำงานหลังจากพาร์ติชันถูก hibernate เรียบร้อยแล้ว แต่ละสคริปต์ DLPAR ที่ก่อนหน้านี้ถูกเรียก ในเฟส check และ pre ถูกเรียกอีกครั้ง

preaccevent
: คำสั่งนี้เป็นคำสั่ง DLPAR script-based แรก ที่ถูกเรียกใช้ในลำดับตัวเร่งความเร็วการเข้ารหัส DLPAR ซึ่ง ถูกเรียกใช้สำหรับสคริปต์ที่ถูกติดตั้งซึ่งก่อนหน้านี้ระบุว่า สนับสนุนชนิดของรีซอร์สที่กำลังถูกเพิ่มหรือรีลีส ซึ่งไม่เป็นที่รู้จักในขณะที่เกิดเหตุการณ์นี้ถ้าการดำเนินการต่อไปนี้ จะถูกเพิ่มหรือรีลีสของตัวเร่งความเร็วการเข้ารหัส การดำเนินการดังกล่าวจะถูกเตรียมไว้ระหว่างเพส post ใดเฟสหนึ่งต่อไปนี้

postaccevent
: คำสั่ง **postaccevent** ถูกเรียกทำงานหลังจาก รีซอร์สถูกประมวลผลโดยระบบ สคริปต์ DLPAR แต่ละสคริปต์ที่ ถูกเรียกใช้ก่อนหน้านี้ในเหส pre จะถูกเรียกใช้อีกครั้ง คำสั่งนี้ใช้เพื่อ รวมสถานะรีซอร์สใหม่ลงในแอ็พพลิเคชัน

undoaccevent
: เฟส **undoaccevent** ถูกจัดเตรียมเพื่อให้ สคริปต์สามารถดำเนินการกู้คืนได้ ถ้าสคริปต์ถูกเรียกใช้ใน เฟส **undoaccevent** แสดงว่าสคริปต์ทำคำสั่ง **preaccevent** เสร็จสมบูรณ์

การทำให้ส่วนขยายเคอร์เนล DLPAR-aware

เหมือนกับแอ็พพลิเคชัน ส่วนขยายเคอร์เนลส่วนใหญ่เป็น DLPAR-safe โดยค่าเริ่มต้น

อย่างไรก็ตามบางส่วนให้ความสำคัญกับการกำหนดค่าระบบ และอาจจำเป็นต้องถูกรีจิสเตอร์ด้วยระบบย่อย DLPAR บางส่วนขยายเคอร์เนลพาร์ติชันข้อมูลตามโพรเซสเซอร์ไลน์ สร้างเธรดจากจำนวนของออนไลน์โพรเซสเซอร์ หรือจัดเตรียมพูลบัฟเฟอร์หน่วยความจำที่พินขนาดใหญ่ ส่วนขยายเคอร์เนลเหล่านี้ต้องถูก แจ้งเมื่อโทโพโลยีระบบเปลี่ยนแปลง กลไกและการดำเนินการ ที่จำเป็นต้องทำควบคู่กันกับแอ็พพลิเคชัน DLPAR-aware

การลงทะเบียนตัวจัดการการกำหนดค่าใหม่

เซอร์วิสเคอร์เนลดังต่อไปนี้ถูกจัดเตรียมเพื่อรีจิสเตอร์และยกเลิก การรีจิสเตอร์ตัวจัดการการกำหนดค่าใหม่:

```
#include sys/dr.h

int reconfig_register(int (*handler)(void *, void *, int, dr_info_t *),
                      int actions, void * h_arg, ulong *h_token, char *name);

void reconfig_unregister(ulong h_token);

int (*handler)(void *event, void *h_arg, unsigned long long req, void *resource_info);

void reconfig_unregister(ulong h_token);

int reconfig_register_ext (int (*handler)(void *, void *, unsigned long long, dr_info_t *),
unsigned long long actions, void * h_arg, ulong *h_token, char *name);

int (*handler)(void *event, void *h_arg, unsigned long long req, void *resource_info);

kerrno_t reconfig_register_list(int (*handler)(void *, void *, dr_kevent_t, void *),
dr_kevent_t event_list[], size_t list_size, void *h_arg, ulong *h_token, char *name);

int (*handler)(void *event, void *h_arg, dr_kevent_t event_in_prog, void *resource_info);
```

หมายเหตุ: คุณได้รับการสนับสนุนให้ใช้เคอร์เนลเซอร์วิส **reconfig_register_list** เซอร์วิสนี้ สนับสนุนเหตุการณ์ที่จะแจ้งเตือนเคอร์เนลส่วนขยายมากขึ้น เซอร์วิส เคอร์เนลก่อนหน้า (**reconfig_register** และ **reconfig_register_ext**) ถูกจำกัดที่ 32 และ 64 เหตุการณ์ ตามลำดับทีให้ส่วนขยายเคอร์เนลที่ใช้เซอร์วิสนี้ ไม่สามารถพอร์ตไปที่ระบบในอนาคตที่สนับสนุนมากกว่า 32 และ 64 เหตุการณ์

พารามิเตอร์ สำหรับรูทีนย่อย **reconfig_register reconfig_register_ext** และ **reconfig_register_list** มีดังนี้:

- พารามิเตอร์ *handler* เป็นส่วนขยายเคอร์เนล ที่ถูกเรียก
- พารามิเตอร์ *actions* อนุญาตให้ส่วนขยาย เคอร์เนลระบุเหตุการณ์ที่ต้องการการแจ้งเตือน สำหรับรายการ ของเหตุการณ์ โปรดดูที่เซอร์วิสเคอร์เนล **reconfig_register, reconfig_register_ext** และ **reconfig_unregister**
- พารามิเตอร์ *h_arg* ถูก ระบุโดยส่วนขยายเคอร์เนล ที่จำไว้โดยเคอร์เนลตามด้วย ฟังก์ชัน descriptor สำหรับตัวจัดการ แล้วส่งไปที่ ตัวจัดการเมื่อถูกเรียก ซึ่งไม่ได้ถูกใช้โดยตรงโดยเคอร์เนล แต่เพื่อสนับสนุนส่วนขยายเคอร์เนลที่จัดการหลายอะแด็ปเตอร์อินสแตนซ์ในทางปฏิบัติ พารามิเตอร์ชี้ไปที่ บล็อกควบคุมอะแด็ปเตอร์
- พารามิเตอร์ *h_token* เป็นเอาต์พุตพารามิเตอร์และมีไว้เพื่อถูกใช้ เมื่อตัวจัดการ ถูกยกเลิกการรีจิสเตอร์
- พารามิเตอร์ *name* ถูกจัดเตรียมสำหรับจุดประสงค์ด้านข้อมูลและสามารถถูกรวมไว้ภายใน รายการบันทึกข้อผิดพลาด ถ้าไดรเวอร์ส่งกลับข้อผิดพลาด ถูกจัดเตรียมโดย ส่วนขยายเคอร์เนลและควรถูกจำกัดที่ 15 อักขระ ASCII
- พารามิเตอร์ *event_list* เป็นอาร์เรย์ของค่า dr_kevent_t ซึ่งส่วนขยายเคอร์เนลควรถูก แจ้งเมื่อเหตุการณ์เกิดขึ้น สำหรับรายการของเหตุการณ์ที่กำหนด โปรดดูที่ เซอร์วิสเคอร์เนล **reconfig_register_list**
- พารามิเตอร์ *list_size* คือขนาดของหน่วยความจำที่ใช้โดยพารามิเตอร์ *event_list*

ฟังก์ชัน **reconfig_register** และ **reconfig_register_ext** ส่งกลับ 0 เมื่อสำเร็จ และค่า `errno` ที่เหมาะสมถ้าเป็นกรณีอื่น

ฟังก์ชัน **reconfig_unregister** ถูกเรียกเพื่อเอาตัวจัดการที่ติดตั้งก่อนหน้านี้ออก

ฟังก์ชัน **reconfig_register, reconfig_register_ext,** และ **reconfig_unregister** สามารถเรียก ได้ในสภาพแวดล้อมกระบวนการเท่านั้น

ถ้าส่วนขยายเคอร์เนลรีจิสเตอร์สำหรับ pre-phase ขอแนะนำ ให้รีจิสเตอร์สำหรับเฟส check เพื่อหลีกเลี่ยงการไม่ได้กำหนดค่าบาง ส่วนของระบบเมื่อเอาทรัพยากรออก

ตัวจัดการการกำหนดค่าใหม่

อินเตอร์เฟส ที่ตัวจัดการการกำหนดค่าใหม่ใช้กับเซอร์วิสเคอร์เนล **reconfig_register_list** มีดังนี้:

```
Int (*handler)(void *event, void *h_arg, dr_kevent_t event_in_prog, void *resource_info);
```

พารามิเตอร์ที่ส่งไปให้ตัวจัดการการกำหนดค่าใหม่มีดังนี้:

- พารามิเตอร์ *event* ถูกส่งไปที่ตัวจัดการและเพื่อใช้เฉพาะเมื่อมีการเรียก รูทีนย่อย **reconfig_handler_complete**
- พารามิเตอร์ *h_arg* ถูกระบุมีเมื่อมีการรีจิสเตอร์โดยตัวจัดการ
- พารามิเตอร์ *event_in_prog* บ่งชี้การดำเนินการ DLPAR ที่ดำเนินการโดยตัวจัดการ สำหรับรายการของเหตุการณ์โปรดดูที่เซอร์วิสเคอร์เนล **reconfig_register_list**
- พารามิเตอร์ *resource_info* ระบุ ข้อมูลจำเพาะทรัพยากรสำหรับการร้องขอ DLPAR ปัจจุบัน ถ้าการร้องขอเป็น processor-based ดังนั้นข้อมูล *resource_info* ถูกจัดเตรียมผ่านโครงสร้าง *dri_cpu* ถ้าการร้องขอ เป็นแบบ memory-based, โครงสร้าง *dri_mem* จะถูกใช้ บนพาร์ติชัน Micro-Partitioning ถ้าการร้องขอเป็นแบบ processor-capacity ข้อมูล *resource_info* ถูกจัดเตรียม ผ่านโครงสร้าง *dri_cpu_capacity* สำหรับข้อมูลเพิ่มเติม และสำหรับรูปแบบของโครงสร้าง *dri_cpu_capacity* โปรดอ้างอิงที่ reconfig Kernel Service

```
struct dri_cpu {
       cpu_t          lcpu;          /* Logical CPU Id of target CPU */
       cpu_t          bcpu;          /* Bind Id of target CPU       */
```

```
};

struct dri_mem {
        size64_t        req_memsz_change;   /* user requested mem size  */
        size64_t        sys_memsz;          /* system mem size at start */
        size64_t        act_memsz_change;   /* mem added/removed so far */
        rpn64_t         sys_free_frames;    /* Number of free frames */
        rpn64_t         sys_pinnable_frames;/* Number of pinnable frames */
        rpn64_t         sys_total_frames;   /* Total number of frames */
        unsigned long long lmb_addr;        /* start addr of logical memory block */
        size64_t        lmb_size;           /* Size of logical memory block being added */
};
```

ถ้าการร้องขอ DLPAR ปัจจุบันเป็นการย้ายพาร์ติชัน ตัวจัดการ จัดเตรียมข้อมูล *resource_info* ให้กับส่วนขยาย เคอร์เนลข้อมูล *resource_info* แต่ส่วนขยาย เคอร์เนลไม่จำเป็นต้องเข้าถึงเนื้อหาของข้อมูล *resource_info* เนื่องจากข้อมูลนี้ไม่ถูกใช้โดยส่วนขยายเคอร์เนล

ตัวจัดการการกำหนดค่าใหม่ถูกเรียกในสภาพแวดล้อมกระบวนการ

ส่วนขยายเคอร์เนลยึดตามข้อมูลดังต่อไปนี้:

- ทรัพยากรชนิดเดียวเท่านั้นที่ถูกกำหนดค่าหรือเอาออก ได้ในหนึ่งครั้ง
- ต้องไม่ระบุหลายโพรเซสเซอร์พร้อมกัน อย่างไรก็ตาม ส่วนขยายเคอร์เนลควรถูกโค้ดเพื่อสนับสนุน การเพิ่มหรือการเอาออก ของบล็อกหน่วยความจำโลจิคัลหลายบล็อก คุณสามารถเริ่มการร้องขอ เพื่อเพิ่มหรือเอาหน่วยความจำเป็นกิกะไบต์ออกได้

เฟส check จัดเตรียม ความสามารถสำหรับแอ็พพลิเคชัน DLPAR-aware applications และ ส่วนขยายเคอร์เนลเพื่อตอบสนองกับการร้องของของผู้ใช้ก่อนที่จะถูกนำไปใช้ ดังนั้น ตัวจัดการส่วนขยายเคอร์เนล check-phase ถูกเรียกครั้งเดียว แม้ว่าการร้องขออาจส่งไปที่บล็อกหน่วยความจำโลจิคัลหลายบล็อก ไม่เหมือนกับเฟส check, pre-phase เฟส post และเฟส post-error ถูกใช้กับระดับบล็อกหน่วยความจำโลจิคัล ซึ่งต่างกัน สำหรับการแจ้งเตือนแอ็พพลิเคชัน โดยที่ pre-phase เฟส post หรือ ที่เป็นทางเลือก เฟส post-error ถูกเรียกหนึ่งครั้งสำหรับแต่ละการร้องขอ ของผู้ใช้ไม่ขึ้นกับจำนวนบล็อกหน่วยความจำโลจิคัล ความแตกต่างอีกอย่างคือ เฟส post-error สำหรับส่วนขยายเคอร์เนล ถูกใช้เมื่อการดำเนินการของบล็อกหน่วยความจำโลจิคัลล้มเหลว โดยที่ เฟส post-error สำหรับแอ็พพลิเคชันถูกใช้เมื่อการดำเนินการ ซึ่งในกรณีนี้คือการร้องขอของผู้ใช้ ล้มเหลว

โดยทั่วไป ระหว่าง เฟส check ส่วนขยายเคอร์เนลตรวจสอบสถานะของตัวเองเพื่อกำหนด ว่าสามารถทำตามการร้องขอ DLPAR ที่จะมีเข้ามาหรือไม่ ถ้าการดำเนินการนี้ไม่สามารถถูกจัดการ หรือถ้าอาจมีผลเสียหาย ร้ายแรงต่อการดำเนินการอย่างถูกต้องของส่วนขยาย ดังนั้นตัวจัดการส่งกลับ **DR_FAIL** หรือไม่แล้วตัวจัดการจะส่งกลับ **DR_SUCCESS**

ระหว่างเฟส pre-remove ส่วนขยายเคอร์เนลพยายาม เอาส่วนเพิ่มเติมออก ที่อาจมีบนทรัพยากรที่กำหนด ตัวอย่างคือไดรเวอร์ที่รักษาพูลบัฟเฟอร์ per-processor ไดรเวอร์อาจทำเครื่องหมายพูลบัฟเฟอร์ที่เกี่ยวข้องเป็น รอการลบ ดังนั้นการร้องขอใหม่จะไม่ถูกจัดสรรจากบัฟเฟอร์ในที่สุด ถูก ใช้จนหมดและอาจถูกฟรี รายการอื่นที่ต้องถูกพิจารณาใน เฟส pre-remove คือตัวจับเวลาและเธรดที่เชื่อมต่อ ซึ่งจำเป็น ต้องถูกหยุดและจบการทำงาน ตามลำดับ ทางเลือก เธรดที่เชื่อมต่อ สามารถถูกยกเลิกได้

ระหว่างเฟส post-remove ส่วนขยายเคอร์เนลพยายามฟรีทรัพยากร ผ่าน garbage collection โดยถือว่าทรัพยากรนั้นถูกเอาออก แล้วจริงๆ ถ้าไม่เป็นเช่นนั้น ตัวจับเวลาและเธรดต้องถูกสร้างใหม่ การร้องขอ **DR_resource_POST_ERROR** ถูกใช้ เพื่อแสดงว่ามีข้อผิดพลาดเกิดขึ้น

ระหว่างเฟส pre-add ส่วนขยายเคอร์เนลต้อง pre-initialize พาธข้อมูลที่ขึ้นอยู่กับทรัพยากรใหม่ ดังนั้นเมื่อ ทรัพยากรใหม่ถูกกำหนดค่า ข้อมูลก็พร้อมใช้งาน ระบบไม่การันตี ว่าทรัพยากรจะไม่ถูกใช้ก่อนที่ตัวจัดการ ถูกเรียกอีกครั้งในเฟส post

ระหว่างเฟส post-add ส่วนขยายเคอร์เนลสามารถยึดถือว่า ทรัพยากรได้ถูกเพิ่มอย่างถูกต้องและสามารถใช้ได้ เฟสนี้ เป็นจุดที่สะดวกในการเริ่มเธรดเชื่อมต่อ ตัวจับเวลาตารางเวลา และ เพิ่มขนาดของบัฟเฟอร์

ส่วนขยายเคอร์เนลยังสามารถถูกแจ้งเตือนการเอาออกหรือการเพิ่ม หน่วยความจำในแบบ per-operation เหมือนกับแอ็พพลิเคชัน โดยการ รีจิสเตอร์หนึ่งชนิดการแจ้งเตือน **_OP_** หรือมากกว่านั้น ซึ่งเปิดใช้งานให้ส่วนขยายเคอร์เนลทำการปรับเปลี่ยนการใช้ทรัพยากรในการ ตอบสนองกับการดำเนินการ DR หน่วยความจำเพียงหนึ่งครั้งต่อการดำเนินการ ไม่ใช่หนึ่งครั้งต่อ logical memory block (LMB)

การแจ้งเตือน **DR_MEM_REMOVE_OP_PRE** ถูกส่ง ก่อนหน่วยความจำถูกเอาออก ตัวจัดการการกำหนดค่าใหม่สามารถเริ่มการปรับ ทรัพยากร โดยคาดว่ามีการเอาหน่วยความจำออกในครั้งนี้ การแจ้งเตือน **DR_MEM_REMOVE_OP_POST** และ **DR_MEM_ADD_OP_POST** ถูกส่งหลังจากการเอาหน่วยความจำออก หรือการดำเนินการเพิ่ม ตามลำดับ ไม่ว่า การดำเนินการจะล้มเหลวหรือไม่ ถ้าการดำเนินการล้มเหลว **act_memsz_change** เป็น 0

ถ้าเป็นไปได้ ภายในไม่กี่วินาที ตัวจัดการการกำหนดค่าใหม่ ส่งกลับ **DR_SUCCESS** เพื่อบ่งชี้การกำหนดค่าใหม่สำเร็จ หรือ **DR_FAIL** เพื่อบ่งชี้ความล้มเหลว ถ้าจำเป็นต้อง ใช้เวลาเพิ่มขึ้น ตัวจัดการจะส่งกลับ **DR_WAIT**

ตัวจัดการ DR ส่วนขยาย

ถ้าส่วนขยาย เคอร์เนลคาดว่าการดำเนินการต้องใช้เวลานาน นั่นคือหลายวินาที ตัวจัดการส่งกลับ **DR_WAIT** ไปที่ ผู้เรียก แต่ดำเนินการกับการร้องขอต่อแบบอะซิงโครนัส ในกรณีดังต่อไปนี้ ตัวจัดการบ่งชี้ว่า ได้ทำตามการร้องขอสมบูรณ์แล้วโดยเรียกรูทีน **reconfig_handler_complete**

```
void reconfig_handler_complete(void *event, int rc);
```

พารามิเตอร์ *event* เหมือนกับพารามิเตอร์ ที่ถูกส่งไปที่ตัวจัดการ เมื่อถูกเรียกโดยเคอร์เนล พารามิเตอร์ *rc* ต้องถูกตั้งค่าเป็น **DR_SUCCESS** หรือ **DR_FAIL** เพื่อบ่งชี้ สถานะความสมบูรณ์ของตัวจัดการ

เซอร์วิสเคอร์เนล **reconfig_handler_complete** สามารถถูกเรียกในสภาพแวดล้อม กระบวนการหรืออินเตอร์รัปต์

การใช้ xmemdma kernel service

บนระบบที่มีความสามารถของ DLPAR เช่นการเอา หน่วยความจำออก แบบไดนามิก เรียกไปที่เซอร์วิสเคอร์เนล **xmemdma** โดยไม่มีแฟล็ก **XMEM_DR_SAFE** มีผลใน หน่วยความจำที่ระบุถูกแฟล็กเป็นไม่สามารถเอาออกได้ นี่เป็นการทำ เพื่อการันตีความสมบูรณ์ของระบบ เนื่องจากระบบไม่รู้ว่า ตัวเรียกต้องการใช้แอดเดรสหน่วยความจำจริง ที่ถูกส่งกลับอย่างไร การดำเนินการเอาหน่วยความจำไดนามิกออกยังเป็น ไปได้สำหรับหน่วยความจำอื่น แต่ไม่ใช่สำหรับหน่วยความจำที่การเรียก **xmemdma** ระบุ

ถ้าตัวเรียกกำลังใช้แอดเดรสหน่วยความจำจริงสำหรับจุดประสงค์ ทางข้อมูลเท่านั้น เช่นเพื่อติดตามข้อมูลบัฟเฟอร์หรือดีบัก ดังนั้น ตัวเรียกสามารถตั้งค่าแฟล็ก **XMEM_DR_SAFE** นี่เป็น การบ่งชี้กับระบบที่แอดเดรสหน่วยความจำจริงสามารถถูกเปิดเผย กับตัวเรียกโดยไม่มีความเสี่ยงในเรื่องข้อมูลเสียหาย เมื่อแฟล็กนี้มีอยู่ ระบบจะยังคงอนุญาตให้หน่วยความจำที่ระบุถูกเอาออก แบบไดนามิก

ถ้าตัวเรียกกำลังใช้แอดเดรสหน่วยความจำจริงเพื่อดำเนินการ เข้าถึงข้อมูลจริง โดยการปิดการแปลข้อมูลหรือทำการโหลด CPU หรือเก็บการเข้าถึงไปยังหน่วยความจำจริง หรือโดยการโปรแกรมมิง คอนโทรลเลอร์ direct memory access (DMA) เพื่อเป้าหมายหน่วยความจำจริงแฟล็ก **XMEM_DR_SAFE** ต้องถูกตั้งค่า ถ้าแฟล็กถูกตั้งค่า data integrity ของระบบอาจมีความเสี่ยง เมื่อหน่วยความจำ ถูกเอาออกแบบไดนามิก สำหรับข้อมูลเกี่ยวกับการแปลงส่วนขยายเคอร์เนล ที่ใช้แอดเดรสหน่วยความจำจริงในวิธีนี้ให้เป็น DLPAR-aware, โปรดติดต่อตัวแทนบริการ IBM ของคุณ

สำหรับข้อมูลเพิ่มเติม โปรดดูที่เซอร์วิสเคอร์เนล **xmemdma**

การควบคุมการแจ้งเตือน DLPAR หน่วยความจำสำหรับแอ็พพลิเคชัน

การเพิ่มหรือการเอาหน่วยความจำออกแบบไดนามิกจาก LPAR ที่รัน หลายโปรแกรม DLPAR-aware สามารถทำให้มีความขัดแย้งสำหรับทรัพยากร โดยค่าเริ่มต้น แต่ละโปรแกรมได้รับการแจ้งเท่ากันเกี่ยวกับการเปลี่ยนแปลงทรัพยากร ตัวอย่างเช่น ถ้าหน่วยความจำ 1 GB ถูกเอาออกจาก LPAR ที่รันสองโปรแกรม DR-aware ดังนั้นโดยค่าเริ่มต้นแต่ละโปรแกรมถูกแจ้งว่าหน่วยความจำ 1 GB ได้ถูกเอาออก เนื่องจากสองโปรแกรมไม่รับรู้ถึงกัน โปรแกรมทั้งสองจะลดการใช้หน่วยความจำลง 1 GB, นำไปสู่ความไม่มีประสิทธิภาพ ปัญหาเรื่องประสิทธิภาพเหมือนกันนี้ เกิดขึ้นได้เช่นกันเมื่อหน่วยความจำใหม่ถูกเพิ่ม

เพื่อแก้ปัญหานี้ AIX อนุญาตให้สคริปต์แอ็พพลิเคชัน ถูกติดตั้งด้วยเปอร์เซ็นต์แฟคเตอร์ที่บ่งชี้เปอร์เซ็นต์ ของการเปลี่ยนแปลงทรัพยากรหน่วยความจำจริง จากนั้นระบบแจ้ง แอ็พพลิเคชันในเหตุการณ์ของหน่วยความจำ DLPAR ขณะติดตั้งสคริปต์แอ็พพลิเคชัน โดยใช้คำสั่ง **drmgr** คุณสามารถระบุ เปอร์เซ็นต์แฟคเตอร์นี้โดยใช้คู่ **DR_MEM_PERCENT name=value** สคริปต์แอ็พพลิเคชันจำเป็นต้องเอาต์พุตคู่ name=value นี้เมื่อถูกเรียกโดยคำสั่ง **drmgr** ด้วยคำสั่งย่อย **scriptinfo** ค่าต้องเป็นจำนวนเต็มระหว่าง 1 ถึง 100 ค่าใดที่อยู่นอกช่วงนี้ถูกละเว้น และจะใช้ค่าดีฟอลต์ ซึ่งคือ 100 นอกจากนี้ คุณสามารถตั้งค่าคู่ **name=value** นี้เป็น ตัวแปรสภาวะแวดล้อมขณะทำการติดตั้ง ระหว่างการติดตั้ง ค่าจากตัวแปรสภาวะแวดล้อม ถ้าตั้งค่า จะเขียนทับ ค่าที่จัดเตรียมโดยสคริปต์แอ็พพลิเคชัน

เช่นเดียวกัน, ในแอ็พพลิเคชันใช้ตัวจัดการสัญญาณ **SIGRECONFIG** และการเรียกระบบ **dr_reconfig( )** คุณสามารถ ควบคุมการแจ้งเตือน DLPAR หน่วยความจำโดยการตั้งค่าคู่ **DR_MEM_PERCENT name=value** เป็นตัวแปรสภาวะแวดล้อมก่อนแอ็พพลิเคชัน เริ่มรัน อย่างไรก็ตาม ค่านี้ไม่สามารถถูกเปลี่ยนแปลงโดยไม่มีการ รีสตาร์ทแอ็พพลิเคชัน

ข้อมูลโปรแกรม sed

โปรแกรม **sed** คือเท็กซ์เอดิเตอร์ที่มีการทำงานคล้ายกับ **ed** ในเอดิเตอร์รายบรรทัด

ซึ่งไม่เหมือนกับ **ed** อย่างไรก็ตาม โปรแกรม **sed** จะดำเนินการแก้ไขโดยไม่ต้องโต้ตอบ กับบุคคลที่ร้องขอการแก้ไข

การจัดการกับสตริงด้วย sed

โปรแกรม **sed** ทำการแก้ไขโดยไม่มีการโต้ตอบ กับผู้ที่ร้องขอการแก้ไข

เมธอดของการดำเนินการนี้อนุญาต **sed** ให้ทำดังต่อไปนี้:

- แก้ไขไฟล์ขนาดใหญ่มาก
- ทำการแก้ไขที่ซับซ้อนหลายครั้งโดยไม่ต้องการ การพิมพ์ใหม่และการจัดตำแหน่งเคอร์เซอร์ มาก (เหมือนที่ตัวแก้ไขแบบโต้ตอบ ทำ)
- ทำการเปลี่ยนแปลงโกลบอลในหนึ่งอินพุต แบบส่งผ่าน

ตัวแก้ไขคงบรรทัดของไว้ที่ถูกแก้ไขอยู่เพียงไม่กี่บรรทัด ในหน่วยความจำต่อหนึ่งครั้ง และไม่ใช้ไฟล์ชั่วคราว ดังนั้น ไฟล์ที่จะถูกแก้ไขจะมีขนาดเท่าใดก็ได้ ตราบที่มีพื้นที่สำหรับ ทั้งไฟล์อินพุตและไฟล์เอาต์พุตในระบบไฟล์

หลักการที่เกี่ยวข้อง:

"เครื่องมือและยูทิลีตี" ในหน้า 2

ส่วนนี้ให้ภาพรวมของเครื่องมือและยูทิลิตีที่คุณสามารถใช้เพื่อพัฒนาโปรแกรมภาษาที่คอมไพล์ด้วย C

การเริ่มต้นเอดิเตอร์

ตัวอย่างดังต่อไปนี้ ไฟล์คำอธิบายสามารถคง โปรแกรม **make** ไว้

แต่ละคำสั่งในไฟล์คำสั่งต้องอยู่บนบรรทัดแยกกัน เมื่อไฟล์คำสั่งถูกสร้าง ให้ป้อนคำสั่งดังต่อไปนี้บนบรรทัดคำสั่ง:

```
sed -fCommandFile >Output <Input
```

ในคำสั่งนี้พารามิเตอร์มีความหมายดังนี้:

พารามิเตอร์	**นิยาม**
CommandFile	ชื่อของไฟล์ที่มีคำสั่งการแก้ไข
เอาต์พุต	ชื่อของไฟล์เพื่อเก็บเอาต์พุตที่แก้ไข
อินพุต	ชื่อของไฟล์ที่จะถูกแก้ไข

จากนั้นโปรแกรม **sed** ทำการเปลี่ยนแปลง และเขียนข้อมูลที่เปลี่ยนแปลงลงในไฟล์เอาต์พุต เนื้อหาของ ไฟล์อินพุตไม่ถูกเปลี่ยนแปลง

sed ทำงานอย่างไร

โปรแกรม **sed** เป็นตัวแก้ไขสตรีม ที่รับอินพุตจากอินพุตมาตรฐาน เปลี่ยนอินพุตนั้นเป็น ถูกกำหนดทิศทางโดยคำสั่งในไฟล์คำสั่ง และเขียนสตรีมผลลัพธ์ไปที่ เอาต์พุตมาตรฐาน

ถ้าคุณไม่ได้จัดเตรียมไฟล์คำสั่งและไม่ได้ใช้แฟล็ก กับคำสั่ง **sed** โปรแกรม **sed** คัดลอกอินพุตมาตรฐานไปที่เอาต์พุตมาตรฐาน โดยไม่มี การเปลี่ยนแปลง อินพุตของโปรแกรมมาจากสองแหล่งข้อมูล:

โปรแกรม	**คำอธิบาย**
สตรีมอินพุต	สตรีมของอักขระ ASCII จากไฟล์หนึ่งไฟล์หรือมากกว่านั้นหรือ ที่ป้อนโดยตรงจากแป้นพิมพ์ สตรีมนี้เป็นข้อมูลที่จะถูกแก้ไข
คำสั่ง	ชุดของแอดเดรสและคำสั่งที่เกี่ยวข้องที่จะถูกดำเนินการ ใน รูปทั่วไปดังต่อไปนี้: `[Line1 [,Line2] ] command [argument]` พารามิเตอร์ *Line1* และ *Line2* คือแอดเดรสที่เรียก แอดเดรสสามารถ เป็นรูปแบบเพื่อจับคู่ในสตรีมอินพุต หรือหมายเลขบรรทัดใน สตรีมอินพุต

คุณยังสามารถป้อนคำสั่งการแก้ไขตามด้วยคำสั่ง **sed** โดยใช้แฟล็ก **-e**

เมื่อแก้ไข **sed** คำสั่งจะ อ่านสตรีมอินพุตครั้งละหนึ่งบรรทัดลงในพื้นที่ในหน่วยความจำที่เรียกว่า pattern space เมื่อบรรทัดของข้อมูลอยู่ใน **sed** อ่าน ไฟล์คำสั่งและพยายามจับคู่แอดเดรสในไฟล์คำสั่งด้วย อักขระใน pattern space ถ้าพบแอดเดรสที่ตรงกับข้อมูลใน pattern space **sed** จะดำเนินคำสั่ง ที่สัมพันธ์กับแอดเดรสบนส่วนของ pattern space ที่ตรงกับ แอดเดรส ผลของคำสั่งเปลี่ยนเนื้อหาของ pattern space และดังนั้นกลายเป็นอินพุตสำหรับคำสั่งดังต่อไปนี้ทั้งหมด

เมื่อ **sed** พยายามจับคู่ แอดเดรสทั้งหมดในไฟล์คำสั่ง ด้วยเนื้อหาของ pattern space, sed เขียนเนื้อหาสุดท้ายของ pattern space ลงในเอาต์พุตมาตรฐาน จากนั้น อ่านบรรทัดอินพุตใหม่จากอินพุตมาตรฐานและเริ่มกระบวนการใหม่อีกครั้ง ที่จุดเริ่มต้นของไฟล์คำสั่ง

บางคำสั่งการแก้ไขเปลี่ยนวิธีที่กระบวนการทำงาน

Flags ที่ใช้กับคำสั่ง **sed** ยังสามารถเปลี่ยนการดำเนินการของคำสั่ง

การใช้นิพจน์ทั่วไป

นิพจน์ทั่วไปคือสตริงที่มีอักขระตามตัวอักษร อักขระการจับคู่รูปแบบ และหรือตัวดำเนินการที่กำหนดชุด ของสตริงที่เป็นไปได้หนึ่งสตริงหรือมากกว่านั้น

ตัวแก้ไขสตรีมใช้ชุดของอักขระการ จับคู่รูปแบบที่ต่างจากอักขระการจับคู่รูปแบบของ shell แต่ เหมือนกับตัวแก้ไขบรรทัด **ed**

การใช้สรุปคำสั่ง sed

คำสั่ง **sed** ทั้งหมดเป็นอักขระ เดี่ยวบวกกับพารามิเตอร์ เช่นหมายเลขบรรทัดหรือสตริงข้อความ

คำสั่ง ที่สรุปทางด้านล่างเปลี่ยนบรรทัดใน pattern space

สัญลักษณ์ดังต่อไปนี้ถูกใช้ในไดอะแกรมไวยากรณ์:

สัญลักษณ์	**ความหมาย**
[ ]	วงเล็บเหลี่ยมปิดส่วนทางเลือกของคำสั่ง
ตัวเอียง	พารามิเตอร์ในตัวเองแสดงชื่อทั่วไปสำหรับชื่อที่คุณ ป้อน ตัวอย่างเช่น *FileName* แสดงพารามิเตอร์ ที่คุณแทนที่ด้วยชื่อของไฟล์จริง
Line1	สัญลักษณ์นี้เป็นหมายเลขบรรทัดหรือนิพจน์ทั่วไปเพื่อจับคู่ซึ่ง กำหนดจุดเริ่มต้นสำหรับการใช้คำสั่งการแก้ไข
Line2	สัญลักษณ์นี้เป็นหมายเลขบรรทัดหรือนิพจน์ทั่วไปเพื่อจับคู ซึ่งกำหนดจุดสิ้นสุดเพื่อหยุดการใช้คำสั่งการแก้ไข

การจัดการบรรทัด

ส่วนนี้อธิบายการจัดการบรรทัด

ฟังก์ชัน	**ไวยากรณ์/คำอธิบาย**
ผนวกบรรทัด	[*Line1*]**a****n***Text* เขียนบรรทัดที่มีใน *Text* เพื่อเอาต์พุตสตรีมหลังจาก *Line1* คำสั่ง **a** ต้องอยู่ที่ท้ายบรรทัด
เปลี่ยนบรรทัด	[*Line1* [,*Line2*]]**c****n***Text* ลบบรรทัดที่ระบุโดย *Line1* และ *Line2* ตามที่คำสั่ง *ลบบรรทัด* ทำ จากนั้น คำสั่งเขียน *Text* ไปที่สตรีมเอาต์พุตในตำแหน่งของ บรรทัดที่ลบ
ลบบรรทัด	[*Line1* [,*Line2*]]**d** เอาบรรทัดออกจากสตรีมอินพุต และไม่คัดลอกบรรทัดไปที่สตรีมเอาต์พุต บรรทัดที่ไม่ถูกคัดลอกเริ่มที่ หมายเลขบรรทัด *Line1* บรรทัดถัดไปที่คัดลอกไปที่สตรีมเอาต์พุต คือหมายเลขบรรทัด *Line2* + 1 ถ้าคุณระบุเฉพาะ หนึ่งหมายเลขบรรทัด เฉพะบรรทัดนั้นเท่านั้นที่ไม่ถูกคัดลอก ถ้าคุณไม่ได้ระบุ หมายเลขบรรทัด บรรทัดถัดไปไม่ถูกคัดลอก คุณไม่สามารถดำเนินฟังก์ชันอื่น บนบรรทัดที่ไม่ถูกคัดลอกไปที่เอาต์พุต
แทรกบรรทัด	[*Line1*] i **n***Text* เขียนบรรทัดที่มีใน *Text* ไปที่สตรีมเอาต์พุตก่อน *Line1* คำสั่ง **i** ต้องอยู่ที่ท้ายบรรทัด

ฟังก์ชัน	ไวยากรณ์/คำอธิบาย
บรรทัดถัดไป	[*Line1* [,*Line2*]]**n** อ่านบรรทัดถัดไป หรือกลุ่ม ของบรรทัดจาก *Line1* ไปที่ *Line2* ลงใน pattern space เนื้อหาปัจจุบันของ pattern space ถูกเขียน ไปที่เอาต์พุตถ้ายังไม่ได้ถูกลบ

การแทนที่

ส่วนนี้อธิบายการแทนค่า

ฟังก์ชัน	ไวยากรณ์/คำอธิบาย
การแทนที่รูปแบบ	[*Line1* [,*Line2*]] **s**/*Pattern*/*String*/*Flags* ค้นหาบรรทัดที่ระบุสำหรับชุดของอักขระที่ตรงกับนิพจน์ทั่วไป ที่กำหนดใน *Pattern* เมื่อพบคำสั่งจะแทนที่ชุดของอักขระนั้นด้วยชุดของอักขระที่ระบุ โดย *String*

อินพุตและเอาต์พุต

ส่วนนี้อธิบายอินพุตและเอาต์พุต

ฟังก์ชัน	ไวยากรณ์/คำอธิบาย
พิมพ์บรรทัด	[*Line1* [,*Line2*]] **p** เขียนบรรทัดที่ระบุไปที่ STDOUT ที่จุดในกระบวนการแก้ไขที่คำสั่ง **p** เกิดขึ้น
เขียนบรรทัด	[*Line1* [,*Line2*]]**w** *FileName* เขียนบรรทัดที่ระบุไปที่ *FileName* ที่ตำแหน่ง ในกระบวนการแก้ไขซึ่งคำสั่ง **w** เกิดขึ้น ถ้า *FileName* มีอยู่จะถูกเขียนทับ; มิฉะนั้นจะถูกสร้าง ไฟล์ต่างกันสูงสุด 10 ไฟล์สามารถถูกเรียกเป็น ไฟล์อินพุตหรือเอาต์พุตในกระบวนการแก้ไขทั้งหมด ใส่หนึ่งช่องว่าง ระหว่าง **w** และ *FileName*
อ่านไฟล์	[*Line1*]**r** *FileName* อ่าน *FileName* และ ผนวกเนื้อหาหลังจากบรรทัดที่ระบุโดย *Line1* ใส่หนึ่งช่องว่างระหว่าง **r** และ *FileName* ถ้า *FileName* ไม่สามารถเปิดได้ คำสั่งจะอ่านไฟล์เป็นไฟล์ null โดยไม่มี การระบุข้อผิดพลาด

การจับคู่ข้ามบรรทัด

ส่วนนี้อธิบายการจับคู่ข้ามบรรทัด

ฟังก์ชัน	ไวยากรณ์/คำอธิบาย
รวมบรรทัดถัดไป	[*Line1* [,*Line2*]]**N** รวมบรรทัดอินพุตที่ระบุ เข้าด้วยกัน แยกบรรทัดโดยใส่อักขระบรรทัดใหม่ รูปแบบที่ตรง สามารถขยายข้ามบรรทัดใหม่
ลบบรรทัดแรกของ pattern space	[*Line1* [,*Line2*]]**D** ลบข้อความทั้งหมดใน pattern space จนถึงและรวมอักขระบรรทัดใหม่แรก ถ้ามีเพียงบรรทัดเดียว ใน pattern space, คำสั่งจะอ่านบรรทัดอื่น เริ่มต้นรายการของคำสั่งการแก้ไข อีกครั้งจากจุดเริ่มต้น
พิมพ์บรรทัดแรกของ pattern space	[*Line1* [,*Line2*]]**P** พิมพ์ข้อความทั้งหมดใน pattern space จนถึงและรวมอักขระบรรทัดใหม่แรกไปที่ STDOUT

Pick up และ put down

ส่วนนี้อธิบายเกี่ยวกับ pick up และ put down

ฟังก์ชัน	ไวยากรณ์/คำอธิบาย
เลือกการคัดลอก	[*Line1* [,*Line2*]]**h** คัดลอกเนื้อหาของ pattern space ที่ระบุโดย *Line1* และ *Line2* ถ้ามี ไปที่พื้นที่พักข้อมูล
เลือกการคัดลอก, ผนวก	[*Line1* [,*Line2*]]**H** คัดลอกเนื้อหาของ pattern space ที่ระบุโดย *Line1* และ *Line2* ถ้ามีไปที่พื้นที่พักข้อมูล และผนวกต่อท้าย เนื้อหาก่อนหน้าของพื้นที่พักข้อมูล
วางการคัดลอก	[*Line1* [,*Line2*]]**g** คัดลอกเนื้อหาของพื้นที่ พักข้อมูลไปที่ pattern space ที่ระบุโดย *Line1* และ *Line2* ถ้ามี เนื้อหาก่อนหน้าของ pattern space ถูกทำลาย
วางการคัดลอก ผนวก	[*Line1* [,*Line2*]]**G** คัดลอกเนื้อหาของพื้นที่ พักข้อมูลไปที่ท้ายของ pattern space ที่ระบุโดย *Line1* และ *Line2* ถ้ามี เนื้อหาก่อนหน้าของ pattern space จะไม่เปลี่ยนแปลง อักขระบรรทัดใหม่แยกเนื้อหา ก่อนหน้าจากข้อความที่ผนวก
แลกเปลี่ยนการคัดลอก	[*Line1* [,*Line2*]]**x** แลกเปลี่ยนเนื้อหาของพื้นที่ พักข้อมูลด้วยเนื้อหาของ pattern space ที่ระบุโดย *Line1* และ *Line2* ถ้ามี

การควบคุม

ส่วนนี้อธิบาย pick up และ put down

ฟังก์ชัน	ไวยากรณ์/คำอธิบาย
การปฏิเสธ	[*Line1* [,*Line2*]]**!** **!** (เครื่องหมาย ตกใจ) ใช้คำสั่งที่ตามมาบนบรรทัดเดียวกันกับส่วนของ ไฟล์อินพุตที่ ได้ เลือกโดย *Line1* และ *Line2*

กลุ่มคำสั่ง	[*Line1* [,*Line2*]]{
	
	คำสั่งที่จัดกลุ่ม
	
	}
	
	{ (ปีกกาซ้าย) และ } (ปีกกาขวา) ปิดรอบ ชุดคำสั่งที่จะถูกใช้เป็นชุดกับบรรทัดอินพุตที่เลือกโดย *Line1* และ *Line2* คำสั่งแรกใน ชุดสามารถอยู่บนบรรทัดเดียวกันหรือบนบรรทัดที่ต่อจากปีกกาซ้าย ปีกกาขวาต้องอยู่บนบรรทัดต่อจากคำสั่ง คุณสามารถซ้อนกลุ่มภายในกลุ่ม

เลเบล	:*เลเบล* ทำเครื่องหมายตำแหน่ง ในสตรีมของคำสั่งการแก้ไขที่จะถูกใช้เป็นปลายทางของแต่ละสาขา สัญลักษณ์ *Label* เป็นสตริงยาวได้ถึง 8 ไบต์ แต่ละ *Label* ในสตรีมการแก้ไขต้องต่างจาก *Label*อื่น
แยกสาขาไปที่เลเบล, ไม่มีเงื่อนไข	[*Line1* [,*Line2*]]**b***Label* แยกสาขาไปที่จุดในสตรีมการแก้ไขที่ระบุโดย *Label* และดำเนินกระบวนการบรรทัดอินพุตปัจจุบันต่อด้วย คำสั่งต่อจาก *Label* ถ้า *Label* เป็น, แยกสาขาไปที่จุดสิ้นสุดของสตรีมการแก้ไข ซึ่งมีผลให้อ่านบรรทัด อินพุตใหม่และเริ่มสตรีมการแก้ไขอีกครั้ง สตริง *Label* ต้องแสดงเป็น *Label* ในสตรีม การแก้ไข
ทดสอบและแยกสาขา	[*Line1* [,*Line2*]]**t***Label* ถ้าการแทนที่สำเร็จกระทำขึ้นบนบรรทัดอินพุตปัจจุบัน แยกสาขาไปที่ *Label* ถ้าไม่มีการแทนที่ ไม่ต้องทำอะไร ล้างแฟล็กที่ระบุว่ามีการแทนที่เกิดขึ้น แฟล็กนี้ถูกล้าง ที่จุดเริ่มต้นของแต่ละบรรทัดอินพุตใหม่
wait	[*Line1*]**q** หยุดการแก้ไขตามลำดับโดยการเขียนบรรทัดปัจจุบันไปที่เอาต์พุต เขียนการทดสอบที่ผนวกหรืออ่านไปที่เอาต์พุต และหยุดการทำงานตัวแก้ไข
ค้นหายหมายเลขบรรทัด	[*Line1*]**=** เขียนหมายเลขบรรทัดที่ตรงกับ *Line1*ไปที่เอาต์พุตมาตรฐาน

การใช้ข้อความในคำสั่ง

คำสั่ง append, insert และ change ที่เกี่ยวกับบรรทัดทั้งหมดใช้ สตริงข้อความที่กำหนดให้ในการเพิ่มข้อมูลให้กับสตรีมเอาต์พุต

สตริงข้อความเป็นไปตาม กฎดังต่อไปนี้:

- มีความยาวหนึ่งบรรทัดหรือมากกว่าได้
- แต่ละ\n (new-line character) ภายใน *Text* ต้องมีอักขระ\เพิ่มเติมนำหน้า (\\n)
- สตริง *Text* ที่ลงท้ายด้วยการขึ้นบรรทัดใหม่ที่ไม่มี อักขระ\เพิ่มเติมนำหน้า (\n)

- เมื่อคำสั่งแทรกสตริง *Text* สตริง:
  - ถูกเขียนไปที่สตรีมเอาต์พุตเสมอ แม้ว่า สิ่งที่คำสั่งอื่นดำเนินการกับบรรทัด ซึ่งทำให้มีการแทรกข้อมูล
  - ไม่ถูกสแกนเพื่อหาแอดเดรสที่ตรงกัน
  - ไม่ได้รับผลจากคำสั่งแก้ไขอื่น
  - ไม่มีผลกับตัวนับหมายเลขบรรทัด

การใช้การแทนที่สตริง

คำสั่ง **s** ดำเนินการแทนที่ สตริงในบรรทัดที่ระบุในไฟล์อินพุต

ถ้าคำสั่งพบ ชุดของอักขระในไฟล์อินพุตที่ตรงกับนิพจน์ทั่วไป *Pattern* คำสั่งจะแทนที่ชุดของอักขระด้วยชุดของ อักขระที่ระบุใน *String*

พารามิเตอร์ *String* เป็น ชุดตามตัวอักษรของอักขระ (ตัวเลข ตัวอักษรและสัญลักษณ์) สัญลักษณ์พิเศษสองสัญลักษณ์ สามารถใช้ได้ใน *String*:

สัญลักษณ์	การใช้
&	สัญลักษณ์นี้ใน *String* ถูกแทนที่โดย ชุดของอักขระในบรรทัดอินพุตที่ตรงกับ *Pattern* ตัวอย่างเช่น, คำสั่ง:

```
s/boy/&s/
```

ระบุกับ **sed** ให้ค้นหารูปแบบ boy ในบรรทัดอินพุต และคัดลอกรูปแบบไปที่เอาต์พุตด้วย s ที่ผนวก ดังนั้นมีการเปลี่ยนแปลงบรรทัดอินพุต:

จาก: The boy look at the game.

เป็น: The boys look at the game.

สัญลักษณ์	การใช้
\d	**d** คือตัวเลขหลักเดียว สัญลักษณ์นี้ใน *String* ถูกแทนที่โดยชุดของอักขระในบรรทัดอินพุตที่ตรงกับ ซับสตริง **d**th ใน *Pattern* ซับสตริงเริ่มด้วยอักขระ\(และสิ้นสุดด้วยอักขระ\) ตัวอยางเช่น คำสั่ง: `s/\(stu\)\(dy\)/\1r\2/` **จาก:** The study chair **เป็น:** The sturdy chair

ตัวอักษรที่แสดงเป็นแฟล็กเปลี่ยนการแทนที่ ดังนี้:

สัญลักษณ์	การใช้
g	แทนที่ *String* สำหรับอินสแตนซ์ทั้งหมดของ *Pattern* ในบรรทัดที่ระบุ อักขระใน *String* ไม่ถูกสแกนสำหรับการจับคู่ของ *Pattern* หลัง จากถูกแทรก ตัวอย่างเช่นคำสั่ง: `s/r/R/g` เปลี่ยน: **จาก:** the red round rock **เป็น:** the Red Round Rock
p	พิมพ์ (ไปที่ STDOUT) บรรทัดที่ *Pattern* มีการจับคู่สำเร็จ
w *FileName*	เขียนไปที่ *FileName* ด้วย ข้อมูลบรรทัดที่ตรงกับ *Pattern* ถ้า *FileName* มีอยู่จะถูกเขียนทับ;มิฉะนั้นจะถูกสร้าง ไฟล์ต่างกันสูงสุด 10 ไฟล์สามารถถูกเรียกเป็น ไฟล์อินพุตหรือเอาต์พุตในกระบวนการแก้ไขทั้งหมด ใส่หนึ่งช่องว่าง ระหว่าง **w** และ *FileName*

ไลบรารีแบบแบ่งใช้และหน่วยความจำแบบแบ่งใช้

หัวข้อนี้ให้ข้อมูลเกี่ยวกับโปรแกรมอำนวยความสะดวก ระบบปฏิบัติการที่มีให้สำหรับการแบ่งใช้ไลบรารี และการจัดสรรหน่วยความจำ

ระบบปฏิบัติการมีตัวช่วยสร้าง และใช้ไลบรารีที่แบ่งใช้ซึ่งเชื่อมโยงถึงกันแบบไดนามิก Dynamic binding อนุญาตให้ใช้สัญลักษณ์ภายนอกที่อ้างอิงในโค้ดผู้ใช้ และที่ได้กำหนดไว้ในไลบรารีที่แบ่งใช้เพื่อให้โหลดเดอร์นำมาใช้แก้ไข ณ เวลารันไทม์

โค้ดไลบรารีที่แบ่งใช้ไม่มีอยู่ในอิมเมจที่สามารถเรียกทำงานได้บนดิสก์โค้ดที่แบ่งใช้จะถูกโหลดลงในหน่วยความจำเพียงครั้งเดียวในเซ็กเมนต์ไลบรารีที่แบ่งใช้ และแบ่งใช้โดยโพรเซสเซอร์ทั้งหมดที่อ้างอิงถึงโค้ดนั้น ข้อดีของไลบรารีที่บ่งใช้คือ :

- ใช้พื้นที่ดิสก์ที่น้อยลง เนื่องจากโค้ดไลบรารีที่แบ่งใช้ไม่ได้สอดแทรกอยู่ในโปรแกรมเรียกทำงาน
- ใช้หน่วยความจำที่น้อยลง เนื่องจากโค้ดไลบรารีที่แบ่งใช้ไม่ได้โหลดเพียงแค่ครั้งเดียว
- เวลาที่ใช้ในการโหลดลดน้อยลง เนื่องจากโค้ดไลบรารีที่แบ่งใช้อาจมีอยู่แล้วในหน่วยความจำ
- ปรับปรุงผลการทำงานให้ดีขึ้น เนื่องจากมีเพจที่บกพร่องอยู่เพียงเล็กน้อยที่ถูกสร้างขึ้น เมื่อโค้ดไลบรารีที่แบ่งใช้มีอยู่แล้วในหน่วยความจำ อย่างไรก็ตาม ผลการทำงานให้ดีขึ้นนี้จะมีต้นทุนในการเรียกใช้รูทีนไลบรารีที่แบ่งใช้ของหนึ่งในแปดคำสั่ง

สัญลักษณ์ที่กำหนดไว้ในโค้ดไลบรารีที่แบ่งใช้ซึ่งถูกทำให้พร้อมใช้งาน สำหรับอ้างอิงโมดูลต้องถูกเอ็กซ์พอร์ตอย่างชัดเจนโดยใช้เอ็กซ์พอร์ตไฟล์ เว้นแต่ว่าคุณได้ใช้อ็อพชัน –bexpall ในบรรทัดแรกของไฟล์ จะมีชื่อพาธของไลบรารีที่แบ่งใช้ บรรทัดที่ตามมา จะมีสัญลักษณ์ที่ต้องการเอ็กซ์พอร์ต

ข้อมูลที่เกี่ยวข้อง:

ar

as

dump

ipcs

ipcrm

id

pagesize

rtl_enable

update

vmstat

XCOFF

อ็อบเจ็กต์แบบแบ่งใช้และการลิงก์รันไทม์

ตามค่าดีฟอลต์แล้ว โปรแกรมจะถูกลิงก์ ดังนั้น การอ้างอิงถึงสัญลักษณ์ที่ถูกอิมพอร์ตจากอ็อบเจ็กต์แบบแบ่งใช้จะผูกกับนิยามในเวลาที่โหลด

สิ่งนี้จะเกิดขึ้นจริง แม้ว่าโปรแกรมหรืออ็อบเจ็กแบบแบ่งใช้อื่นๆ ที่จำเป็นต้องมีในโปรแกรมจะนิยามสัญลักษณ์เดียวกัน

ตัวลิงก์รันไทม์
อ็อบเจ็กต์แบบแบ่งใช้ที่อนุญาตให้สัญลักษณ์ทำการลิงก์ใหม่ สำหรับโปรแกรมที่ลิงก์อย่างเหมาะสม

คุณรวมตัวลิงก์รันไทม์ในโปรแกรม โดยลิงก์โปรแกรมกับอ็อพชัน **-brtl** อ็อบชันนี้มีผลกระทบต่อไปนี้ :

- การอ้างอิงถึงตัวลิงก์รันไทม์ จะถูกเพิ่มลงในโปรแกรมของคุณ เมื่อการประมวลโปรแกรมเริ่มต้นขึ้น โค้ดเริ่มต้นทำงาน (**/lib/crt0.o**) จะเรียกตัวลิงก์รันไทม์ก่อนที่ฟังก์ชันหลักจะถูกเรียก
- ไฟล์อินพุตที่แบ่งใช้อ็อบเจ็กต์จะถูกแสดงเป็นส่วนที่ต้องพึ่งพาของโปรแกรมของคุณ ในส่วนของโหลดเดอร์ของโปรแกรมอ็อบเจ็กต์แบบแบ่งใช้จะถูกแสดงตามลำดับเดียวกับที่ระบุไว้บน บรรทัดรับคำสั่ง ซึ่งทำให้โหลดเดอร์ของระบบโหลดอ็อบเจ็กต์แบบแบ่งใช้เหล่านี้ทั้งหมด เพื่อให้ตัวลิงก์รันไทม์สามารถใช้นิยามเหล่านั้นได้ ถ้าไม่ได้ใช้อ็อพชัน **-brtl** อ็อบเจ็กต์แบบแบ่งใช้ที่ไม่ได้อ้างอิงโดยโปรแกรมจะไม่ถูกแสดง แม้ว่าจะได้จัดเตรียมนิยามที่อาจต้องการโดยอ็อบเจ็กต์แบบแบ่งใช้อื่นๆ ซึ่งจะถูกใช้โดยโปรแกรม
- อ็อบเจ็กต์แบบแบ่งใช้นี้ที่มีไฟล์เก็บถาวร จะถูกแสดงขึ้นหากไฟล์เก็บถาวรนั้นระบุการโหลดแบบอัตโนมัติ สำหรับสมาชิกของอ็อบเจ็กต์แบบแบ่งใช้ คุณระบุการโหลดแบบอัตโนมัติสำหรับสมาชิกที่เก็บถาวร **foo.o** โดยสร้างไฟล์ด้วยบรรทัดต่อไปนี้:

  ```
  # autoload
  #! (foo.o)
  ```

  และเพิ่มไฟล์ที่เป็นสมาชิกให้กับไฟล์เก็บถาวรนั้น
- ในโหมดแบบไดนามิก ไฟล์อินพุตที่ระบุด้วยแฟล็ก **-l** อาจลงท้ายด้วย **.so** พร้อมกับใน **.a** นั่นคือ การอ้างอิงถึง **-lfoo** จะถูกตอบสนองโดย **libfoo.so** หรือ **libfoo.a** อันดับแรกที่พบในไดเร็กทอรีที่ต้องการค้นหา โหมดแบบไดนามิกจะได้รับผลกระทบตามค่าดีฟอลต์ ยกเว้นจะใช้อ็อพชัน **-bstatic**

ตัวลิงก์รันไทม์จะเลียนแบบลักษณะการทำงานของคำสั่ง **ld** เมื่อใช้การลิงก์แบบสแตติก ยกเว้นว่าสัญลักษณ์ที่เอ็กซ์พอร์ตสามารถ ใช้เพื่อแก้ไขสัญลักษณ์เท่านั้น แม้ว่าเมื่อใช้การลิงก์รันไทม์ ตัวโหลดของระบบต้องสามารถโหลดและแก้ปัญหาการอ้างอิงสัญลักษณ์ทั้งหมด ในโปรแกรมหลักและโมดูลใดๆ ที่ต้องอ้างอิง ดังนั้น ถ้านิยามถูกลบออกจากโมดูลแล้ว และโปรแกรมหลักมีการอ้างอิงถึงนิยามนี้ โปรแกรมจะไม่ถูกประมวลผล แม้ว่า นิยามอื่นๆ สำหรับสัญลักษณ์มีอยู่ในโมดูลอื่นๆ

ตัวลิงก์รันไทม์สามารถโยงการอ้างอิงทั้งหมดกับสัญลักษณ์ที่อิมพอร์ต จากโมดูลอื่นอีกครั้ง การอ้างอิงถึงสัญลักษณ์ที่กำหนดไว้ในโมดูลเดียวกันกับ การอ้างอิง สามารถโยงอีกครั้งได้เฉพาะถ้าโมดูลนั้นถูกสร้างด้วยการลิงก์แบบรันไทม์ ที่เปิดใช้งานสำหรับสัญลักษณ์นั้น

โมดูลแบบแบ่งใช้จะถูกจัดส่งมาพร้อมกับ AIX 4.2 หรือสูงกว่า เพื่อเปิดใช้งานการลิงก์รันไทม์สำหรับตัวแปรที่เอ็กซ์พอร์ตแล้ว ส่วนใหญ่ การลิงก์รันไทม์ สำหรับฟังก์ชันมีการเปิดใช้งานสำหรับฟังก์ชันที่เรียกผ่านตัวชี้ฟังก์ชัน เท่านั้น ตัวอย่างเช่น ตามที่จัดส่ง การเรียกรูทีนย่อย **malloc** ภายในอ็อบเจ็กต์ที่แบ่งใช้ **shr.o** ใน **/lib/libc.a** ไม่สามารถผูกใหม่ได้ แม้ว่านิยามของ **malloc** มีอยู่ในโปรแกรมหลัก หรือโมดูลแบบแบ่งใช้อื่นๆ คุณสามารถลิงก์โมดูลแบบแบ่งใช้ที่จัดส่งมาส่วนใหญ่ เพื่อเปิดใช้งานการลิงก์รันไทม์สำหรับฟังก์ชันและตัวแปร โดยรันคำสั่ง **rtl_enable**

การดำเนินการของตัวลิงก์รันไทม์

โปรแกรมหลักจะถูกโหลดและแก้ปัญหาโดยโหลดเดอร์ของระบบ ด้วยวิธีการปกติ ถ้าโปรแกรมที่สามารถเรียกทำงานได้ไม่สามารถโหลดได้ด้วยเหตุผลใดๆ รูทีนย่อย **exec( )** จะล้มเหลว และตัวลิงก์รันไทม์ จะไม่ถูกเรียกใช้งานเลย ถ้าโปรแกรมหลักโหลดเสร็จเรียบร้อยแล้ว การควบคุม จะส่งผ่านไปยังตัวลิงก์รันไทม์ ซึ่งจะโยงสัญลักษณ์อีกครั้งดังอธิบาย ด้านล่าง เมื่อตัวลิงก์รันไทม์เสร็จสมบูรณ์แล้ว จะมีการเรียก รูทีนการเริ่มต้น ถ้าเหมาะสม จากนั้น จะเรียกฟังก์ชันหลัก

ตัวลิงก์รันไทม์จะประมวลผลโมดูลในการค้นหาในแนวกว้าง ซึ่งเริ่มต้นด้วยความสามารถในการประมวลผลหลักและดำเนินการต่อ ด้วยการพึ่งพาความสามารถในการประมวลผลหลัก ตามลำดับของโมดูลที่ต้องพึ่งพา ซึ่งแสดงอยู่ในส่วนของโหลดเดอร์ของแต่ละโมดูล ลำดับนี้ยังถูกนำมาใช้ ขณะค้นหาการนิยามอินสแตนซ์ของสัญลักษณ์ "การกำหนดอินสแตนซ์" ของสัญลักษณ์คือ อินสแตนซ์แรกของสัญลักษณ์ แต่มีข้อยกเว้นอยู่ด้วยกันสองข้อ ถ้า อินสแตนซ์แรกของสัญลักษณ์ไม่ใช่อินสแตนซ์ที่ไม่สามารถแก้ปัญหาได้ อิมพอร์ตที่รออยู่ หรือไม่มีการนิยามอินสแตนซ์อยู่ ถ้าอินสแตนซ์แรกคือ สัญลักษณ์ BSS (นั่นคือ ชนิด **XTY_CM** ซึ่งบ่งชี้ถึงตัวแปรที่ไม่ได้กำหนดค่าเริ่มต้นไว้) และมีอินสแตนซ์อื่นๆ ของสัญลักษณ์ที่ไม่ใช่สัญลักษณ์ BSS และไม่สามารถแก้ปัญหาได้ อิมพอร์ตที่รออยู่อินสแตนซ์แรกของสัญลักษณ์ คือการนิยามอินสแตนซ์

ส่วนของโหลดเดอร์ของโมดูลแต่ละตัวจะแสดงสัญลักษณ์ที่อิมพอร์ต ซึ่งจะถูกนิยามในโมดูลที่ระบุไว้อื่นๆ และสัญลักษณ์ที่เอ็กซ์พอร์ต ซึ่งถูกกำหนดอยู่ในโมดูล สัญลักษณ์ที่ถูกอิมพอร์ตและเอ็กซ์พอร์ตของถูกเรียกว่า การอิมพอร์ตแบบ "passed-through'' ดังนั้น เครื่องหมายจะปรากฏขึ้นตามที่ได้นิยามไว้ในโมดูลหนึ่ง แม้ว่า จะถูกนิยามไว้ในโมดูลอื่น

สัญลักษณ์ยังสามารถทำเครื่องหมายเป็น "อิมพอร์ตที่รออยู่" การอ้างอิงถึง สัญลักษณ์การอิมพอร์ตที่รออยู่จะไม่มีทางถูกโยงอีกครั้งโดยตัวลิงก์รันไทม์ การแก้ปัญหาสัญลักษณ์เหล่านี้จะถูกดำเนินการโดยโหลดเดอร์ของระบบ โดยการเรียก **loadbind( )** หรือโดยการโหลดโมดูลใหม่ด้วย **load( )** or **dlopen( )** อย่างใดอย่างหนึ่ง

การอ้างอิงถึงสัญลักษณ์ที่อิมพอร์ต (นอกเหนือจากการอิมพอร์ตที่รออยู่) สามารถผูกใหม่ได้เสมอ โหลดเดอร์ของระบบจะถูกแก้ปัญหาการอิมพอร์ตส่วนใหญ่ การอ้างอิงถึงสัญลักษณ์ที่อิมพอร์ตแต่ละครั้งจะถูกผูกขึ้นใหม่กับการนิยาม อินสแตนซ์ของสัญลักษณ์ ถ้าไม่มีการนิยามอินสแตนซ์อยู่ ข้อความแสดงความผิดพลาดจะถูกพิมพ์ ในข้อผิดพลาดมาตรฐาน นอกจากนี้ ถ้าการตรวจสอบชนิดสตริงแบบแฮชของสัญลักษณ์ที่อิมพอร์ต จะไม่ตรงกับสตริงแบบแฮชของการนิยามสัญลักษณ์ และข้อความแสดงความผิดพลาดจะถูกพิมพ์

การอ้างอิงถึงสัญลักษณ์ที่เอ็กซ์พอร์ตจะยังคงถูกผูกขึ้นใหม่กับนิยาม อินสแตนซ์ของการอ้างอิงเหล่านั้น ตราบเท่าที่การอ้างอิงปรากฏใน ตารางการจัดสรรใหม่ของส่วนของโหลดเดอร์ (การอิมพอร์ตแบบ passed-through จะถูกประมวลผลตามการอิมพอร์ตอื่นๆ ดังที่ได้กล่าวไว้ข้างต้น) ซึ่งขึ้นอยู่กับวิธีการลิงก์โมดูล การอ้างอิงบางส่วนกับสัญลักษณ์ที่เอ็กซ์พอร์ตจะถูกผูกไว้ ณ เวลาที่ลิงก์ และไม่สามารถผูกขึ้นใหม่ได้ เนื่องจากสัญลักษณ์ที่เอ็กซ์พอร์ตจะถูกนิยามอยู่ในโมดูลการเอ็กซ์พอร์ต การนิยามอินสแตนซ์ของสัญลักษณ์จะยังมีอยู่เสมอ ยกเว้นว่าอินสแตนซ์แรกคือ การอิมพอร์ตที่รออยู่ ดังนั้น ข้อผิดพลาดจะไม่เหมือนกัน แต่ยังคงอาจเกิดขึ้นได้ เมื่อการผูกใหม่จะเอ็กซ์พอร์ตสัญลักษณ์ เนื่องจากการอิมพอร์ต ข้อผิดพลาดจะถูกพิมพ์ หากการตรวจสอบชนิดสตริงแบบแฮชไม่ตรงกัน เมื่อสัญลักษณ์ถูกผูกไว้ใหม่

เมื่อใดก็ตามที่สัญลักษณ์ถูกผูกไว้ การพึ่งพาจะถูกเพิ่มจากโมดูลโดยใช้สัญลักษณ์ในโมดูลที่นิยามสัญลักษณ์ การพึ่งพานี้ จะป้องกันโมดูลจากการลบออกจากพื้นที่แอดเดรสก่อนกำหนด นี่คือสิ่งสำคัญ เมื่อโมดูลที่ถูกโหลดโดยรูทีนย่อย **dlopen** จะกำหนดสัญลักษณ์ที่ยังคงถูกใช้ เมื่อพยายามยกเลิกการโหลดโมดูลด้วยรูทีนย่อย **dlclose**

ตารางสัญลักษณ์ในส่วนของโหลดเดอร์จะไม่มีข้อมูลใดๆ เกี่ยวกับการจัดตำแหน่งหรือความยาวของสัญลักษณ์ ดังนั้น จึงไม่มีข้อผิดพลาดที่ตรวจพบ เมื่อสัญลักษณ์ถูกผูกใหม่กับอินสแตนซ์ที่มีความยาวที่สั้นมาก หรือที่ถูกจัดตำแหน่งอย่างไม่ถูกต้อง ข้อผิดพลาดในการประมวลผลอาจเกิดขึ้นในกรณีนี้

หลังจากประมวลผลโมดูลทั้งหมดแล้ว ตัวลิงก์รันไทม์จะเรียกรูทีนย่อย **exit** ถ้าข้อผิดพลาดของการลิงก์รันไทม์ใดๆ เกิดขึ้น และส่งผ่านโค้ดการออก 144 (0x90) หรือ การเรียกใช้งานยังคงเรียกการกำหนดค่าเริ่มต้นรูทีนหรือ **main( )**

การสร้างอ็อบเจ็กต์แบบแบ่งใช้ที่มีการเปิดใช้งานการลิงก์รันไทม์

หากต้องการสร้างอ็อบเจ็กต์แบบแบ่งใช้ที่เปิดใช้งานการลิงก์รันไทม์ คุณต้องลิงก์ด้วยแฟล็ก **-G** เมื่อแฟล็กนี้ถูกนำมาใช้ การดำเนินการต่อไปนี้จะเข้าแทนที่:

1. สัญลักษณ์ที่เอ็กซ์พอร์ตจะถูกกำหนดแอ็ตทริบิวต์ `nosymbolic` เพื่อให้การอ้างอิงถึงสัญลักษณ์ทั้งหมดสามารถกลับคืนมาโดยใช้ตัวลิงก์ รันไทม์
2. สัญลักษณ์ที่ไม่ได้นิยามไว้จะได้รับอนุญาตให้ใช้ (โปรดดูอ็อพชัน **-berok**) ดังนั้น สัญลักษณ์จะถูกทำเครื่องหมายว่ากำลังอิมพอร์ต จากชื่อโมดูลเชิงสัญลักษณ์ "..". สัญลักษณ์จะถูกอิมพอร์ตจาก "." ต้องถูกแก้ปัญหาโดยตัวลิงก์รันไทม์ก่อนที่จะสามารถนำมาใช้ได้ เนื่องจากโหลดเดอร์ของระบบ จะไม่แก้ไขสัญลักษณ์เหล่านี้
3. ไฟล์เอาต์พุตจะได้รับการกำหนดชนิดโมดูลที่เป็น `SRE` ถ้าระบุ อ็อพชัน **-bM:SRE**
4. อ็อบเจ็กต์แบบแบ่งใช้ทั้งหมดที่แสดงอยู่บนบรรทัดรับคำสั่ง จะถูกแสดงตามการพึ่งพาของโมดูลเอาต์พุต ด้วยวิธีเดียวกับที่อธิบายไว้ ขณะที่ลิงก์โปรแกรมด้วยอ็อพชัน **-brtl**
5. อ็อบเจ็กต์แบบแบ่งใช้ในไฟล์เก็บถาวรจะถูกแสดงถ้าอ็อบเจ็กต์เหล่านั้นมีแอ็ตทริบิวต์ `autoload`

การใช้อ็อพชัน **-berok** ซึ่งหมายความถึงแฟล็ก **-G** จะสามารถพรางข้อผิดพลาดที่สามารถตรวจพบได้ในเวลา ที่ลิงก์ ถ้าคุณต้องการนิยามสัญลักษณ์ที่อ้างอิงทั้งหมด ขณะที่ลิงก์โมดูล คุณควรใช้อ็อพชัน **-bernotok** หลังแฟล็ก **-G** ซึ่งอาจเป็นสาเหตุของข้อผิดพลาดที่ต้องถูกรายงาน สำหรับสัญลักษณ์ที่ไม่ได้นิยามไว้

ไลบรารีแบบแบ่งใช้และ lazy loading

โดยดีฟอลต์ เมื่อโมดูลถูกโหลด โหลดเดอร์ระบบ จะโหลดทั้งหมดที่ขึ้นกับโมดูลนั้นพร้อมกันโดยอัตโนมัติ การโหลด สิ่งที่ขึ้นต่อกันจะเกิดขึ้นเนื่องจากโมดูลถูกลิงก์ รายการของโมดูลที่ขึ้นต่อกัน ของโมดูลนั้นถูกบันทึกไว้ในส่วนโหลดเดอร์ของโมดูล

โมดูล	คำอธิบาย
dump -H	คำสั่งที่อนุญาตให้ดูรายการโมดูลที่ขึ้นต่อกัน
-blazy	ใน AIX 4.2.1 และใหม่กว่า อ็อพชันตัวลิงก์ที่ลิงก์โมดูลเพื่อให้ เฉพาะบางส่วนของสิ่งที่ขึ้นต่อกันเท่านั้นถูกโหลดเมื่อฟังก์ชันในโมดูล ถูกใช้งานเป็นครั้งแรก

เมื่อคุณใช้ lazy loading คุณสามารถปรับปรุงผลการทำงาน ของโปรแกรมได้ถ้าการขึ้นต่อการส่วนใหญ่ของโมดูลยังไม่เคยถูกใช้มาก่อนอย่างแท้จริง หรือ อีกนัยหนึ่ง ทุกฟังก์ชันเรียกใช้ไปยังโมดูลที่ถูกโหลดแบบเลซี่มีโอเวอร์เฮดเพิ่มประมาณ 7 คำสั่ง และการเรียกใช้ฟังก์ชันครั้งแรกจำเป็นต้อง ทำการโหลดโมดูลการกำหนดค่าและการแก้ไขการเรียกใช้ฟังก์ชัน ดังนั้น ถ้า โมดูลเรียกใช้ฟังก์ชันในการขึ้นต่อกันส่วนใหญ่ lazy loading อาจไม่ เหมาะสม

เมื่อฟังก์ชันที่กำหนดในโมดูลที่โหลดแบบเลซี่ ถูกเรียกใช้เป็นครั้งแรก มีความพยายามที่จะโหลดโมดูลการกำหนดค่า และค้นหาฟังก์ชันที่ต้องการ ถ้าไม่พบโมดูลหรือถ้าฟังก์ชัน ไม่ถูกเอ็กซ์พอร์ตโดยโมดูล ลักษณะการทำงานดีฟอลต์คือพิมพ์ข้อความแสดงความผิดพลาด ไปยังข้อผิดพลาดมาตรฐานและออกจากการทำงานโดยค่าโค้ดส่งกลับเป็น 1 แอ็พพลิเคชันสามารถ จัดหา handler ข้อผิดพลาดของตนโดยการเรียใช้ฟังก์ชัน **_lazySetErrorHandler** และจัดหาแอดเดรสของ handler ข้อผิดพลาด handler ข้อผิดพลาดถูกเรียกใช้ ด้วย 3 อาร์กิวเมนต์: ชื่อของโมดูล ชื่อของสัญลักษณ์ และค่า ข้อผิดพลาดที่ระบุสาเหตุของข้อผิดพลาด ถ้า handler ข้อผิดพลาดส่งค่ากลับ ค่าที่ ส่งกลับควรเป็นแอดเดรสของฟังก์ชันการแทนสำหรับฟังก์ชัน ที่ต้องการ ค่าที่ส่งกลับสำหรับ **_lazySetErrorHandler**เป็น NULL ถ้าไม่มี handler ข้อผิดพลาดออกจากการทำงาน และส่งกลับแอดเดรสของ handler ก่อนหน้า ถ้ามี handler

การใช้ lazy loading โดยปกติไม่เปลี่ยนลักษณะการทำงาน ของโปรแกรม แต่มีข้อยกเว้นบางอย่าง ประการแรก โปรแกรมใดๆ ที่ต้องขึ้นกับ ลำดับที่โมดูลถูกโหลดจะได้รับผลกระทบ เนื่องจากโมดูล อาจถูกโหลดในลำดับต่างกัน และบางโมดูลอาจไม่ถูกโหลด เลย

ประการที่สองโปรแกรมที่เปรียบเทียบตัวชี้ฟังก์ชัน อาจทำงานไม่ถูกต้องเมื่อใช้ lazy loading เนื่องจากฟังก์ชันเดียว สามารถมีได้หลายแอดเดรสโดยเฉพาะ ถ้าโมดูล A เรียกใช้ฟังก์ชัน `f' ในโมดูล B และถ้า lazy loading ของโมดูล B ถูกระบุเมื่อโมดูล A ถูก ลิงก์ ดังนั้นแอดเดรสของ `f' ที่คำนวณในโมดูล A จะแตกต่างจากแอดเดรส ของ `f' ที่คำนวณในโมดูลอื่นๆ ดังนั้นเมื่อคุณใช้ lazy loading สองค่าของตัวชี้ฟังก์ชัน อาจไม่เท่ากัน แม้ว่าจะชี้ไปยังฟังก์ชันเดียวกัน

ประการที่สาม ถ้าโมดูลใดๆ ถูกโหนดด้วยชื่อพาธสัมพัทธ์ และถ้าโปรแกรมเปลี่ยนแปลงไดเร็กทอรีที่ทำงาน โมดูลที่ขึ้นต่อกันอาจไม่พบเมื่อจำเป็นต้องถูกโหลด เมื่อคุณใช้ lazy loading คุณควรใช้ชื่อพาธสัมบูรณ์เท่านั้น เมื่ออ้างอิงโมดูที่ขึ้นต่อกัน ในตอนลิงก์

การตัดสินใจเปิดใช้งาน lazy loading กระทำในตอน ลิงก์ในลักษณะของโมดูลแต่ละโมดูล ในโปรแกรมเดี่ยว คุณสามารถผสมโมดูล ที่ใช้ lazy loading กับโมดูลที่ไม่ได้ใช้ เมื่อลิงก์โมดูลเดียว การอ้างอิงไปยังตัวแปรในโมดูลที่ขึ้นต่อกันจะกันมิให้โมดูลนั้นถูกโหลดแบบเลซี่ ถ้าการอ้างอิงทั้งหมดไปยังโมดูลเป็นการทำงานสัญลักษณ์ โมดูลที่ขึ้นต่อกันสามารถถูกโหลดแบบเลซี่

สิ่งอำนวยความสะดวก lazy loading สามารถใช้ในแอ็พพลิเคชันทั้งที่เป็นเธรด และที่ไม่เป็นเธรด

การติดตามการเรียกทำงาน Lazy loading

คุณลักษณะรันไทม์ที่จัดให้มีอนุญาตให้คุณ ดูวิวการโหลดเมื่อเกิดขึ้น ซึ่งทำได้โดยการใช้ตัวแปรสภาวะแวดล้อม **LDLAZYDEBUG** ค่าของตัวแปรนี้ เป็นตัวเลข ที่อยู่ในรูปฐานสิบ ฐานแปด (นำหน้าด้วย 0) หรือเลขฐานสิบหก (นำหน้าด้วย 0x) ที่ เป็นผลรวมของค่าต่อไปนี้อย่างน้อยหนึ่งค่า:

ตัวแปร	คำอธิบาย
1	แสดงข้อผิดพลาดการโหลดหรือการค้นหา ถ้าไม่พบ โมดูลที่ต้องการ ข้อความแสดงและ handler ข้อผิดพลาด lazy load ถูกเรียกใช้ ถ้าสัญลักษณ์ที่ร้องขอไม่มีอยู่ในโมดูลที่ถูกอ้างถึงซึ่งถูกโหลด ข้อความจะแสดงก่อนเรียกใช้ handler ข้อผิดพลาด
2	เขียนข้อความการติดตามไปยัง **stderr** แทน **stdout** โดยดีฟอลต์ ข้อความเหล่านี้ ถูกเขียนลงสตรีมไฟล์เอาต์พุตมาตรฐาน ค่านี้เลือกสตรีม ข้อผิดพลาดมาตรฐาน

ตัวแปร	คำอธิบาย
4	แสดงชื่อของโมดูลที่กำลังโหลด เมื่อต้องการใช้โมดูลใหม่แก้ปัญหาการเรียกใช้ฟังก์ชัน ชื่อของ โมดูลที่พบและโหลดจะแสดง การแสดงนี้เกิดขึ้นในการอ้างอิงครั้งแรก ไปยังฟังก์ชันภายในโมดูลนั้น นั่นคือ เมื่อโมดูลถูกโหลด จะยังคง พร้อมใช้งานสำหรับการอ้างอิงไปยังฟังก์ชันภายในโมดูลนั้นที่จะมีตามมา ไม่จำเป็น ต้องมีการดำเนินการโหลดเพิ่มเติม
8	แสดงชื่อของฟังก์ชันที่เรียกใช้ ชื่อของฟังก์ชันที่ต้องการ พร้อมกับชื่อของโมดูลจาก ฟังก์ชันที่คาดจะแสดง ข้อมูลนี้แสดงก่อนที่โมดูลจะถูกโหลด

พื้นที่ไลบรารีแบบแบ่งใช้ที่กำหนดชื่อ

โดยค่าเริ่มต้น AIX แบ่งใช้ไลบรารีระหว่างกระบวนการโดยใช้ชุดของเซ็กเมนต์โกลบอล ที่อ้างอิงเป็นพื้นที่ไลบรารีที่แบ่งใช้โกลบอล

สำหรับกระบวนการ 32-บิต พื้นที่นี้ประกอบด้วยหนึ่งเซ็กเมนต์สำหรับข้อความไลบรารีที่แบ่งใช้ (เซ็กเมนต์ 0xD) และหนึ่งเซ็กเมนต์สำหรับข้อมูลไลบรารี pre-relocated (เซ็กเมนต์ 0xF) การแบ่งใช้ ข้อความและข้อมูล pre-relocating เพื่อประสิทธิภาพบนระบบซึ่งกระบวนการ จำนวนมากใช้ไลบรารีที่แบ่งใช้ทั่วไป

เนื่องจากพื้นที่ไลบรารีที่แบ่งใช้โกลบอลเป็นทรัพยากรขนาดคงที่เดี่ยว ความพยายามในการแบ่งใช้ชุดของไลบรารีที่เกิดความจุของพื้นที่ ไม่สามารถทำได้ในสถานการณ์นี้ ส่วนของไลบรารีกระบวนการถูกโหลด แบบไพรเวต การโหลดไลบรารีแบบไพรเวต ตรงข้ามกับแบบแบ่งใช้ ใช้พื้นที่แอดเดรส ไพรเวตในกระบวนการและมีการร้องขอพื้นที่การเพจจำนวนมาก นำไปสู่ การลดลงของประสิทธิภาพระบบโดยรวม

เพื่อแก้ไขข้อจำกัดของพื้นที่ไลบรารีที่แบ่งใช้โกลบอลนี้ AIX 5.3 สนับสนุน *พื้นที่ไลบรารีที่แบ่งใช้ที่กำหนดชื่อ* ซึ่งมีประโยชน์ดังนี้:

- พื้นที่ไลบรารีที่แบ่งใช้ที่กำหนดชื่อแทนที่พื้นที่ไลบรารีที่แบ่งใช้โกลบอลสำหรับ กลุ่มของกระบวนการ
- พื้นที่ไลบรารีที่แบ่งใช้ที่กำหนดชื่อทำให้กลุ่มกระบวนการใช้ไลบรารีที่แบ่งใช้ที่มีได้เต็มที่ในตำแหน่งเดียวกันในพื้นที่ effective address เหมือนกับไลบรารีที่แบ่งใช้โกลบอล (เซ็กเมนต์ 0xD และ 0xF)
- คุณลักษณะพื้นที่ไลบรารีที่แบ่งใช้ที่กำหนดชื่อถูกเปิดใช้ งานผ่านตัวแปรสภาวะแวดล้อม **LDR_CNTRL** และไม่จำเป็นต้องเปลี่ยนแปลงไบนารีที่มีอยู่
- หลายพื้นที่ไลบรารีที่แบ่งใช้ที่กำหนดชื่อสามารถถูกแอ็คทีฟบนระบบพร้อมกัน
- กระบวนการระบุพื้นที่ไลบรารีที่แบ่งใช้ที่กำหนดชื่อด้วยชื่อเฉพาะ ชื่อนี้ถูกเลือกโดยกระบวนการที่ทำให้มีการสร้างพื้นที่
- พื้นที่ไลบรารีที่แบ่งใช้ที่กำหนดชื่อใช้ได้เฉพาะกระบวนการ 32-บิต

เนื่องจากการใช้พื้นที่ไลบรารีที่แบ่งใช้ที่กำหนดชื่อถูกจำกัดกับ กระบวนการที่ร้องขอ พื้นที่จะไม่ถูกใช้โดยกระบวนการที่ใช้พื้นที่ไลบรารีที่แบ่งใช้โกลบอล หรือพื้นที่ไลบรารีที่แบ่งใช้ที่กำหนดชื่ออื่น นี่เป็นการลด การแย่งชิงพื้นที่ในพื้นที่ไลบรารีที่แบ่งใช้ที่กำหนดชื่อ ซึ่งมีประโยชน์กับกระบวนการที่ใช้พื้นที่ กระบวนการเหล่านี้มีการใช้พื้นที่แอดเดรสไพรเวต ต่ำกว่าและมีความสามารถสูงกว่าในการแบ่งใช้ไลบรารีระหว่างตัวกระบวนการเอง การใช้พื้นที่ไลบรารีที่แบ่งใช้ที่กำหนดชื่อระหว่างกระบวนการที่ใช้ไลบรารีที่แบ่งใช้ทั่วไป สามารถออปติไมซ์การใช้พื้นที่แอดเดรสของกระบวนการและลดความต้องการ พื้นที่การเพจ มีผลให้ประสิทธิภาพระบบโดยรวมเพิ่มขึ้น

โมเดลหน่วยความจำทางเลือก (doubletext32)

นอกจากโมเดลหน่วยความจำพื้นที่ไลบรารีที่แบ่งใช้เริ่มต้น (หนึ่งเซ็กเมนต์ กำหนดให้กับข้อความไลบรารีที่แบ่งใช้และหนึ่งเซ็กเมนต์กำหนดให้กับข้อมูลไลบรารี pre-relocated) พื้นที่ไลบรารีที่แบ่งใช้ที่กำหนดชื่อสนับสนุโมเดลหน่วยความจำทางเลือก ที่กำหนดทั้งสองเซ็กเมนต์ให้กับข้อความไลบรารีที่แบ่งใช้ โมเดลนี้มีประโยชน์สำหรับ กลุ่มกระบวนการที่แบ่งใช้ข้อความไลบรารีมากกว่า 256 MB กรุณาสังเกตว่า เนื่องจากโมเดลหน่วยความจำทางเลือกนี้ไม่ได้ดำเนินการ pre-relocation ของข้อมูลไลบรารี มีอาจพบการลดลงของประสิทธิภาพบ้างระหว่างการโหลดโมดูล (สำหรับทั้ง ความขึ้นต่อกัน exec-time และโมดูลที่โหลดแบบไดนามิก) ดังนั้น ผลประโยชน์ ของประสิทธิภาพจริงของความจุข้อความไลบรารีที่แบ่งใช้ที่เพิ่มขึ้นควรถูก พิจารณาเป็นกรณีไป

อินเตอร์เฟส

เข้าถึง

กระบวนการร้องขอการใช้พื้นที่ไลบรารีที่แบ่งใช้ที่กำหนดชื่อโดยมีตัวแปรสภาวะแวดล้อม **LDR_CNTRL** พร้อมกับตัวเลือก **NAMEDSHLIB** ในสภาพแวดล้อมขณะรันไทม์ ไวยากรณ์ของตัวเลือกใหม่ เป็นดังนี้:

```
NAMEDSHLIB=name[,attribute][,attribute2]...[,attributeN]
```

สติงชื่อที่ใช้ได้คือสตริงที่ตรงกับนิพจน์ทั่วไป `[A-Za-z0-9_\.]+` (มีเฉพาะ ตัวอักษรและตัวเลข ขีดล่าง และอักขระ จุด)

สตริงชื่อที่ใช้ได้ต้องถูกปิดด้วยหนึ่งในอักขระดังต่อไปนี้:

- `@` (เครื่องหมาย at): ตัวคั่นสำหรับหลายตัวเลือก **LDR_CNTRL**
- `,` (เครื่องหมายจุลภาค): ตัวคั่นสำหรับแอ็ตทริบิวต์ **NAMEDSHLIB**
- `\0` (null): ตัวปิดท้ายของสตริงสภาพแวดล้อม **LDR_CNTRL**

ถ้ามีการระบุสตริงชื่อไม่ถูกต้อง ตัวเลือก **NAMEDSHLIB** ทั้งหมดจะถูกละเว้น ถ้ามีการระบุแอ็ตทริบิวต์ไม่ถูกต้องเฉพาะแอ็ตทริบิวต์นั้น จะถูกละเว้น ขณะนี้ มีหนึ่งแอ็ตทริบิวต์ที่สนับสนุน: **doubletext32**

ไม่มีข้อจำกัดการเข้าถึงสำหรับการใช้พื้นที่ไลบรารีที่แบ่งใช้ที่กำหนดชื่อ การร้องขอทั้งหมดสำหรับใช้พื้นที่จะได้รับอนุญาต

การสร้าง

ไม่มีอินเตอร์เฟสที่ชัดเจนในการสร้างพื้นที่ไลบรารีที่แบ่งใช้ที่กำหนดชื่อ เมื่อ กระบวนการร้องขอการใช้พื้นที่ไลบรารีที่แบ่งใช้ที่กำหนดชื่อที่ไม่มีอยู่ พื้นที่จะถูกสร้างโดยอัตโนมัติ

การขจัด

ระบบเอาไลบรารีที่ไม่ได้ใช้ออกจากพื้นที่ไลบรารีที่แบ่งใช้ที่กำหนดชื่อโดยใช้ กลไกเดียวกับที่ใช้กับพื้นที่ไลบรารีที่แบ่งใช้โกลบอล:

- การเอาไลบรารีที่ไม่ใช้ออกอัตโนมัติเกิดขึ้นเมื่อพื้นที่เต็ม
- การบังคับเอาไลบรารีที่ไม่ใช้ออกทำได้โดยใช้คำสั่ง **slibclean**

การทำลาย

ไม่มีอินเตอร์เฟสที่ชัดเจนใการทำลายพื้นที่ไลบรารีที่แบ่งใช้ที่กำหนดชื่อ เมื่อกระบวนการสุดท้ายที่ใช้พื้นที่ไลบรารีที่แบ่งใช้ที่กำหนดชื่อจบการทำงาน (usecount ของพื้นที่ลดลงถึงศูนย์) พื้นที่จะถูกทำลายโดยอัตโนมัติ

แอ็ตทริบิวต์

แอ็ตทริบิวต์ **NAMEDSHLIB** ถูกตรวจสอบโดยโหลดเดอร์ระบบ เฉพาะระหว่างการสร้างพื้นที่ไลบรารีที่แบ่งใช้ที่กำหนดชื่อ ดังนั้น การร้องขอ เพื่อใช้พื้นที่ไลบรารีที่แบ่งใช้ที่กำหนดชื่อไม่เข้มงวดในการที่ต้องระบุ แอ็ตทริบิวต์ที่ตรงกับที่ระบุเมื่อทำการสร้าง (การร้องขอจะ ไม่ล้มเหลว เนื่องจากการที่แอ็ตทริบิวต์ไม่ตรงกัน) อย่างไรก็ตาม เนื่องจากระบบทำลาย พื้นที่ไลบรารีที่แบ่งใช้ที่กำหนดชื่อที่ไม่ใช้โดยอัตโนมัติ เป็นแนวทางปฏิบัติที่ดีในการระบุ เสมอ แม้เมื่อคุณกำลังร้องขอการใช้พื้นที่ไลบรารีที่แบ่งใช้ที่กำหนดชื่อ ที่มีอยู่แล้ว

ตัวอย่าง

1. รันคู่ของแอ็พพลิเคชันโดยใช้พื้นที่ไลบรารีที่แบ่งใช้ที่ชื่อ *XYZ* โดยมีหนึ่งเซ็กเมนต์กำหนดให้กับข้อความไลบรารีที่แบ่งใช้ และหนึ่งเซ็กเมนต์กำหนดให้กับข้อมูลไลบรารีที่เปลี่ยนตำแหน่งล่วงหน้าโดยรันคำสั่ง ดังต่อไปนี้:

   ```
   $ export LDR_CNTRL=NAMEDSHLIB=XYZ
   $ xyz_app
   $ xyz_app2
   ```

2. รันคู่ของแอ็พพลิเคชันโดยใช้พื้นที่ไลบรารีที่แบ่งใช้ที่ชื่อ *more_shtext* โดยมีทั้งสองเซ็กเมนต์ที่กำหนดให้กับข้อความไลบรารีที่แบ่งใช้โดยรันคำสั่งดังต่อไปนี้:

   ```
   $ export LDR_CNTRL=NAMEDSHLIB=more_shtext,doubletext32
   $ mybigapp
   $ mybigapp2
   ```

กรสร้างไลบรารีแบบแบ่งใช้

ส่วนนี้อธิบายวิธีสร้างไลบรารีแบบแบ่งใช้

งานที่จำเป็นต้องมี

1. สร้างไฟล์ต้นฉบับหนึ่งไฟล์หรือมากกว่านั้นที่จะถูก คอมไพล์และเชื่อมโยงเพื่อสร้างไลบรารีที่แบ่งใช้ ไฟล์เหล่านี้มีสัญลักษณ์ที่ส่งออกที่ได้อ้างอิงในไฟล์ต้นฉบับอื่น ๆ

   สำหรับ ตัวอย่างในบทความนี้ สองไฟล์ต้นฉบับ `share1.c` และ `share2.c` ถูกใช้ ไฟล์ `share1.c` มีโค้ดดังต่อไปนี้:

   ```
   /************
     * share1.c: shared library source.
   *************/

   #include <stdio.h>

   void func1 ()
    {
         printf("func1 called\n");
    }

   void func2 ()
    {
         printf("func2 called\n");
    }
   ```

   ไฟล์ `share2.c`มีโค้ดดังต่อไปนี้:

```
/************
  * share2.c: shared library source.
*************/

void func3 ()
 {
      printf("func3 called\n");
 }
```

สัญลักษณ์ที่ส่งออกในไฟล์เหล่านี้คือ func1, func2 และ func3

2. สร้างไฟล์ต้นฉบับหลักที่อ้างอิงสัญลักษณ์ที่ส่งออกที่จะอยู่ในไลบรารีที่แบ่งใช้
   สำหรับตัวอย่างในบทความนี้ใช้ไฟล์ต้นฉบับหลักชื่อ main.c ไฟล์ main.c มีโค้ดดังต่อไปนี้:

```
/************
  * main.c: contains references to symbols defined
 * in share1.c and share2.c
*************/

#include <stdio.h>

    extern void func1 (),
                func2 (),
                func3 ();
main()
 {
                func1 ();
                func2 ();
                func3 ();
 }
```

3. สร้างไฟล์ส่งออกที่จำเป็นเพื่อส่งออก สัญลักษณ์ในไลบรารีที่แบ่งใช้ที่ถูกอ้างอิงโดยโมดูลอ็อบเจ็กต์ อื่น
   สำหรับตัวอย่างในบทความนี้ใช้ไฟล์ส่งออกชื่อ shrsub.exp ไฟล์ shrsub.exp มีโค้ดดังต่อไปนี้:

```
#! /home/sharelib/shrsub.o
* ด้านบนเป็นชื่อพาธแบบเต็มไปยังไฟล์อ็อบเจ็กต์ไลบรารีแบบแบ่งใช้
func1
func2
func3
```

บรรทัด #! มีความหมายเฉพาะเมื่อ ไฟล์ถูกใช้เป็นไฟล์นำเข้า ในกรณีนี้บรรทัด #! ระบุชื่อของไฟล์ไลบรารีที่แบ่งใช้ที่จะถูกใช้ ขณะรันไทม์

โพรซีเดอร์

1. คอมไพล์และเชื่อมโยงสองไฟล์ซอร์สโค้ดที่ จะถูกแบ่งใช้ (โพรซีเดอร์นี้ถือว่าคุณอยู่ในไดเร็กทอรี **/home/sharedlib**) เมื่อต้องการคอมไพล์และเชื่อมโยงไฟล์ต้นฉบับ ให้ป้อน คำสั่งดังต่อไปนี้:

```
cc -c share1.c
cc -c share2.c
cc -o shrsub.o share1.o share2.o -bE:shrsub.exp -bM:SRE -bnoentry
```

ซึ่งจะสร้างไลบรารีที่แบ่งใช้ชื่อ shrsub.o ในไดเร็กทอรี **/home/sharedlib**

`-bM:SRE flag`
ทำเครื่องหมายอ็อบเจ็กต์ไฟล์ผลลัพธ์ `shrsub.o` เป็น ไลบรารีที่แบ่งใช้ re-entrant
แต่ละกระบวนการที่ใช้โค้ดที่แบ่งใช้ได้รับสำเนาส่วนตัว ของข้อมูลในพื้นที่กระบวนการส่วนตัว

แฟล็ก
ตั้งค่าดัมมี entry point `_nostart` เพื่อ เขียนทับ entry point เริ่มต้น `_start`

`-bnoentry flag`
แจ้ง linkage editor ว่าไลบรารีที่แบ่งใช้ไม่มี entry point
ไลบรารีที่แบ่งใช้อาจมี entry point แต่ตัวโหลดระบบ ไม่ได้ใช้ entry point เมื่อไลบรารีที่แบ่งใช้ถูกโหลด

2. ใช้คำสั่งดังต่อไปนี้เพื่อนำไลบรารีที่แบ่งใช้ไปไว้ในไฟล์เก็บถาวร:

```
ar qv libsub.a shrsub.o
```

ขั้นตอนนี้เป็นทางเลือก การนำไลบรารีที่แบ่งใช้ ไปไว้ในไฟล์เก็บถาวรทำให้เป็นเรื่องง่ายกว่าในการระบุไลบรารีที่แบ่งใช้ เมื่อเชื่อมโยง โปรแกรมของคุณ เนื่องจากคุณสามารถใช้แฟล็ก **–l** และ **–L** กับคำสั่ง **ld**

3. คอมไพล์และเชื่อมโยงซอร์สโค้ดหลักด้วย ไลบรารีที่แบ่งใช้เพื่อสร้างไฟล์เรียกทำงาน (ขั้นตอนนี้ถือว่าไดเร็กทอรีทำงาน ปัจจุบัน ของคุณมีไฟล์ **main.c**) ใช้ คำสั่งดังต่อไปนี้:

```
cc -o main main.c -lsub -L/home/sharedlib
```

ถ้าไลบรารีที่แบ่งใช้ไม่ได้อยู่ในไฟล์เก็บถาวร ให้ใช้คำสั่ง:

```
cc -o main main.c /home/sharedlib/shrsub.o -L/home/sharedlib
```

โปรแกรม `main` ในขณะนี้เป็นโปรแกรมที่รันได้ สัญลักษณ์ `func1`, `func2` และ `func3` ได้ถูกทำเครื่องหมายสำหรับเรสโซลูชันที่เลื่อนไปเวลาโหลด ขณะ รันไทม์ ตัวโหลดระบบโหลดโมดูลลงในไลบรารีที่แบ่งใช้ (นอกจากว่า โมดูลถูกโหลดอยู่แล้ว) และวิเคราะห์การอ้างอิงแบบไดนามิก

หมายเหตุ: เมื่อสร้างไลบรารีที่แบ่งใช้จากอ็อบเจ็กต์ C++ คุณต้องใช้ชื่อสัญลักษณ์ C++ ที่ถูก mangle ในไฟล์เอ็กซ์พอร์ต อย่างไรก็ตาม คอมไพเลอร์ C++ ของคุณอาจจัดเตรียมอ็อพชันเพื่อสร้างไลบรารีที่แบ่งใช้สำหรับคุณ โปรดอ้างถึงเอกสารคู่มือคอมไพเลอร์ของคุณ สำหรับข้อมูลเพิ่มเติม

`-L flag`
เพิ่มไดเร็กทอรีที่ระบุ (ในกรณีนี้ `/home/sharedlib`) ให้กับพาธการค้นหาไลบรารี ซึ่งถูกบันทึกในส่วนโหลดเดอร์ของโปรแกรม

ขณะรันไทม์พาธการค้นหาไลบรารีถูกใช้เพื่อแจ้งโหลดเดอร์ถึงตำแหน่ง ของไลบรารีที่แบ่งใช้

ตัวแปรสภาวะแวดล้อม `LIBPATH`
รายการที่แยกด้วยโคลอนของพาธไดเร็กทอรีซึ่งสามารถใช้เพื่อ ระบุพาธการค้นหาไลบรารีที่ต่างกันได้ รูปแบบเหมือนกับตัวแปรสภาวะแวดล้อม **PATH**

ไดเร็กทอรีในรายการถูกค้นหาเพื่อแยกแยะการอ้างอิงไปที่ อ็อบเจ็กต์ที่แบ่งใช้ไลบรารี **/usr/lib** and **/lib** มีไลบรารีที่ใช้ร่วมกัน และโดยปกติควรถูกรวมไว้ในพาธการค้นหาไลบรารี ของคุณ

ภาพรวมพื้นที่แอดเดรสโปรแกรม

Base Operating System จัดเตรียมเซอร์วิส สำหรับโปรแกรมมิงการใช้หน่วยความจำของแอ็พพลิเคชันโปรแกรม

เครื่องมือพร้อมใช้งาน ในการช่วยเหลือ การจัดสรรหน่วยความจำ การแม็พหน่วยความจำและไฟล์ และการทำโปรไฟล์ การใช้หน่วยความจำของแอ็พพลิเคชัน จากเบื้องหลัง ส่วนนี้อธิบานสถาปัตยกรรม การจัดการหน่วยความจำของระบบและนโยบายการจัดการหน่วยความจำ

บทนำสถาปัตยกรรมหน่วยความจำระบบ

ระบบใช้แผนการการจัดการหน่วยความจำที่ใช้ ซอฟต์แวร์เพื่อเพิ่มความสามารถของฟิสิคัลซอฟต์แวร์ เนื่องจาก พื้นที่แอดเดรสไม่ได้ตรงตามหน่วยความจำจริงแบบหนึ่งต่อหนึ่ง พื้นที่แอดเดรส (และวิธีที่ระบบทำให้แอดเดรสตอบสนองกับหน่วยความจำจริง) เรียกว่า หน่วยความจำเสมือน

ระบบย่อยของเคอร์เนลและฮาร์ดแวร์ที่ทำงานร่วมกัน เพื่อแปลแอดเดรสเสมือนไปเป็นฟิสิคัลแอดเดรสทำให้เกิด ระบบย่อยการจัดการหน่วยความจำ การดำเนินการที่เคอร์เนลใช้เพื่อประกันว่ากระบวนการ แบ่งใช้หน่วยความจำหลักเป็นไปตามนโยบายการจัดการหน่วยความจำ ส่วนดังต่อไปนี้ อธิบายคุณลักษณะของระบบย่อยการจัดการหน่วยความจำ ในรายละเอียด

พื้นที่แอดเดรสฟิสิคัลของระบบ 64 บิต

ฮาร์ดแวร์จัดเตรียมช่วงต่อเนื่องของแอดเดรสหน่วยความจำเสมือน จาก 0x00000000000000000000 ถึง 0xFFFFFFFFFFFFFFFFFFFF สำหรับพื้นที่เข้าถึง พื้นที่รวมที่กำหนดแอดเดรสได้มีมากกว่า 1000000000000 เทราไบต์ คำสั่งการเข้าถึงหน่วยความจำสร้างแอดเดรส 64 บิต: 36 บิต เพื่อเลือกเซ็กเมนต์รีจิสเตอร์และ 28 บิตเพื่อให้ ออฟเซ็ตภายในเซ็กเมนต์ รูปแบบการกำหนดแอดเดรสจัดเตรียมการเข้าถึงได้มากกว่า 64 ล้านเซ็กเมนต์แต่ละเซ็กเมนต์มีข้อมูลได้ถึง 256M ไบต์ แต่ละเซ็กเมนต์รีจิสเตอร์มี segment ID 52 บิต ที่เป็นส่วนเติมหน้ากับ 28-บิตออฟเซ็ต ซึ่งร่วมกัน ฟอร์มแอดเดรสหน่วยความจำเสมือน ผลลัพธ์แอดเดรสเสมือน 80-บิตอ้างอิง กับพื้นที่หน่วยความจำเสมือน single, large, systemwide

พื้นที่กระบวนการคือพื้นที่แอดเดรส 64-บิต นั้นคือ โปรแกรมใช้ตัวชี้ 64-บิต อย่างไรก็ตาม แต่ละกระบวนการหรือ interrupt handler สามารถแอดเดรสได้เพียง พื้นที่หน่วยความจำเสมือน systemwide (เซ็กเมนต์) ซึ่ง segment IDs อยู่ในรีจิสเตอร์เซ็กเมนต์

การกำหนดแอดเดรสรีจิสเตอร์เซ็กเมนต์

เคอร์เนลระบบโหลดรีจิสเตอร์เซ็กเมนต์บางส่วนใน วิธีปกติสำหรับกระบวนการทั้งหมด การจัดหาความสามารถในการแอดเดรสหน่วยความจำถูกต้องการไปโดยปริยายโดยกระบวนการมากที่สุด รีจิสเตอร์เหล่านี้รวมถึงสองเคอร์เนลเซ็กเมนต์ และเซ็กเมนต์ shared-library และเซ็กเมนต์อุปกรณ์ I/O ที่ถูกแบ่งใช้โดย กระบวนการทั้งหมดและที่เนื้อหาเป็นแบบอ่านอย่างเดียวกับโปรแกรมที่ไม่ใช่เคอร์เนล ยังมี เซ็กเมนต์สำหรับการเรียกระบบ **exec** ของกระบวนการ ซึ่งถูกแบ่งใช้แบบอ่านอย่างเดียว กับกระบวนการอื่นที่ดำเนินการกับ โปรแกรมเดียวกัน เซ็กเมนต์ข้อมูล shared-library ที่มีข้อมูลไลบรารี read-write และเซ็กเมนต์ read-write มีเป็นแบบไพรเวทต่อกระบวนการ รีจิสเตอร์เซ็กเมนต์ ที่เหลืออาจถูกโหลดโดยใช้เทคนิคการแม็พหน่วยความจำเพื่อจัดเตรียม หน่วยความจำเพิ่มเติม หรือผ่านการเข้าถึงหน่วยความจำกับไฟล์ ตามสิทธิการเข้าถึงที่กำหนดโดยเคอร์เนล

การกำหนดแอดเดรส 32-บิตของระบบ และการเข้าถึงที่จัดเตรียม ผ่านความสามารถทางอ้อม ให้อินเตอร์เฟสกับแต่ละกระบวนการที่ ไม่ขึ้นกับขนาดจริงของพื้นที่หน่วยความจำเสมือน systemwide บาง รีจิสเตอร์เซ็กเมนต์ถูกแบ่งใช้โดยกระบวนการทั้งหมด บางส่วนโดยชุดย่อยของกระบวนการ และบางส่วนเข้าถึงได้เพียงกระบวนการเดียวเท่านั้น การแบ่งใช้ทำได้โดยอนุญาตให้สองกระบวนการหรือมากกว่านั้นโหลด segment ID เดียวกัน

พื้นที่การเพจ

เมื่อต้องการบรรจุพื้นที่หน่วยความจำเสมือนที่มี พื้นที่หน่วยความจำจริงที่จำกัด ระบบใช้หน่วยความจำจริงเป็นพื้นที่ใช้งาน และ เก็บข้อมูลและโปรแกรมที่ไม่ใช้งานที่ไม่ได้ถูกแม็พบนดิสก์ พื้นที่ของ ดิสก์ที่มีข้อมูลนี้เรียกว่าการเพจพื้นที่ว่าง เพจเป็น หน่วย ของหน่วยความจำเสมือนที่มีข้อมูล 4K ไบต์และสามารถถูกถ่ายโอน ระหว่างพื้นที่เก็บข้อมูลจริงและสำรอง เมื่อระบบ ต้องการข้อมูลหรือโปรแกรมใน พื้นที่เพจ ระบบจะ:

1. ค้นหาพื้นที่ของหน่วยความจำที่ไม่ได้ ที่ใช้งานอยู่
2. ตรวจสอบว่าสำเนาของข้อมูลหรือโปรแกรม ปัจจุบันจากพื้นที่หน่วยความจำนั้นอยู่ในการเพจพื้นที่ว่างบนดิสก์
3. อ่านโปรแกรมหรือข้อมูลใหม่จากการเพจพื้นที่ว่าง บนดิสก์ลงในพื้นที่หน่วยความจำที่ว่าง

นโยบายการจัดการหน่วยความจำ

การแปลแอดเดรส real-to-virtual และหน่วยบริการ หน่วยความจำเสมือนถูกจัดเตรียมให้กับระบบอย่างชัดเจนโดย Virtual Memory Manager (VMM) VMM มีการนำหน่วยความจำเสมือนมาใช้ อนุญาต การสร้างเซ็กเมนต์ที่ใหญ่กว่าหน่วยความจำฟิสิคัลที่มีอยู่ใน ระบบ ซึ่งทำได้โดยการรักษารายการของเพจว่างของหน่วยความจำจริง ที่ใช้ในการเรียกเพจที่จำเป็นต้องนำมาไว้ในหน่วยความจำ

ในบางครั้ง VMM ต้องเติมเพจบนรายการฟรีโดยเอาข้อมูลเพจปัจจุบันออกบางส่วนจากหน่วยความจำจริง กระบวนการ ย้ายข้อมูลระหว่างหน่วยความจำและดิสก์เมื่อข้อมูลมีความจำเป็น เรียกว่า "การเพจ" เพื่อสร้างการเพจ VMM ใช้อัลกอริธึม page-stealing ที่จัด หมวดหมู่เพจเป็นสามคลาส แต่ละคลาสมีรายการและเกณฑ์จบการทำงานเฉพาะ:

- การทำงานกับเพจหน่วยเก็บข้อมูล
- เพจไฟล์โลคัล
- เพจไฟล์รีโมต

โดยทั่วไป เพจการทำงานมีลำดับความสำคัญสูงสุด ตามด้วย เพจไฟล์โลคัล และเพจไฟล์รีโมต

นอกจากนี้ VMM ใช้เทคนิคเรียกว่า อัลกอริธึม clock เพื่อเลือกเพจที่จะถูกแทนที่ เทคนิคนี้ใช้ประโยชน์ ของบิตที่อ้างอิงสำหรับแต่ละเพจเป็นการบ่งชี้ว่าเพจไหนที่ถูกใช้ไม่นานนี้ (ถูกอ้างอิง) เมื่อรูทีน page-stealer ถูกเรียก จะวนผ่าน ตารางเฟรมเพจเพื่อตรวจสอบแต่ละบิตที่อ้างอิงของเพจ ถ้าเพจ ถูกยกเลิกอ้างอิงและนำมาใช้ได้ (นั้นคือ ไม่ถูก pin และตรงกับเกณฑ์ page-stealing อื่น) เพจจะถูกนำมาใช้และกำหนดไว้ในรายการฟรี เพจที่อ้างอิงอาจไม่ ถูกนำมาใช้ แต่บิตของเพจถูกรีเซ็ต "กำหนดอายุ" อย่างมีประสิทธิภาพต่อ การอ้างอิง เพื่อที่เพจอาจถูกนำมาใช้ครั้งถัดไปที่มีการใช้อัลกอริธึม page-stealing

การจัดสรรหน่วยความจำ

เวอร์ชัน 3 ของระบบปฏิบัติการใช้เทคนิคสล็อตการเพจ ที่หน่วงเวลาสำหรับพื้นที่จัดเก็บข้อมูลที่จัดสรรให้กับแอ็พพลิเคชัน ซึ่งหมายความว่าเมื่อ พื้นที่จัดเก็บข้อมูลถูกจัดสรรให้กับแอ็พพลิเคชันด้วยรูทีนย่อยเช่น **malloc** ไม่มีการเพจพื้นที่ว่างถูกกำหนดให้กับพื้นที่จัดเก็บข้อมูลนั้นจนกว่าพื้นที่จัดเก็บข้อมูล ถูกอ้างอิง

หลักการที่เกี่ยวข้อง:

"การทำความเข้าใจกับการแม็พหน่วยความจำ" ในหน้า 675
ความเร็วที่คำสั่งของแอ็พพลิเคชันจะประมวลผลบนระบบ จะแบ่งส่วนตามจำนวนของการดำเนินการสำหรับการเข้าถึงที่จำเป็น เพื่อขอรับข้อมูลภายนอกหน่วยความจำที่สามารถกำหนดแอดเดรสของโปรแกรมได้

"ข้อกำหนดการเขียนโปรแกรมพื้นที่การเพจ" ในหน้า 686
จำนวนของการเพจพื้นที่ว่างที่จำเป็นโดยแอ็พพลิเคชัน ขึ้นอยู่กับชนิดของกิจกรรมที่ดำเนินบนระบบ ถ้าการเพจพื้นที่ว่าง รันต่ำ การประมวลผลอาจสูญหาย

“การจัดสรรหน่วยความจำระบบโดยใช้ระบบย่อย malloc” ในหน้า 696
หน่วยความจำจะถูกจัดสรรให้กับแอ็พพลิเคชัน โดยใช้ระบบย่อย **malloc**

การทำความเข้าใจกับการแม็พหน่วยความจำ

ความเร็วที่คำสั่งของแอ็พพลิเคชันจะประมวลผลบนระบบ จะแบ่งส่วนตามจำนวนของการดำเนินการสำหรับการเข้าถึงที่จำเป็น เพื่อขอรับข้อมูลภายนอกหน่วยความจำที่สามารถกำหนดแอดเดรสของโปรแกรมได้

ระบบจะมีสองเมธอด สำหรับการลดการใช้งานทำรายการที่เชื่อมโยงกับการดำเนินการอ่าน และเขียนภายนอกเหล่านี้ คุณสามารถแม็พไฟล์ข้อมูลลงในพื้นที่แอดเดรส ของการประมวลผลได้ คุณยังสามารถแม็พการประมวลผลกับส่วนของหน่วยความจำที่ไม่ระบุชื่อ ซึ่งอาจแบ่งใช้โดยการทำการประมวลผลร่วมกัน

หน่วยความจำของไฟล์ที่แม็พจะจัดเตรียมกลไก สำหรับการประมวลผลเพื่อเข้าถึงไฟล์ด้วยข้อมูลไฟล์ที่รวมเข้าด้วยกันได้โดยตรง ในพื้นที่แอดเดรสของการประมวลผล การใช้ไฟล์ที่แม็พสามารถลดการย้ายข้อมูล I/O เนื่องจากข้อมูลไฟล์ไม่ได้ถูกทำสำเนาไปยังบัฟเฟอร์ข้อมูล แต่จะถูกทำโดยรูทีนย่อย **read** และ **write** เมื่อมีการประมวลผลที่มากกว่าหนึ่งกระบวนการแม็พไฟล์เดียวกัน เนื้อหาจะถูกแบ่งใช้ระหว่างการประมวลผลทั้งสองกระบวนการนั้น ซึ่งจะเตรียมกลไกสำหรับการใช้งานที่ต่ำด้วยการประมวลผลที่สามารถซิงโครไนซ์ และสื่อสารได้

ส่วนของหน่วยความจำที่แม็พยังถูกเรียกว่าพื้นที่หน่วยความจำที่แบ่งใช้ สามารถใช้เป็นพูลขนาดใหญ่สำหรับการแลกเปลี่ยนข้อมูลระหว่างการประมวลผล รูทีนที่พร้อมใช้งาน ไม่ได้ล็อกหรือควบคุมสิทธิในการเข้าถึงระหว่างการประมวลผล ดังนั้น การประมวลผลที่ใช้พื้นที่หน่วยความจำที่แบ่งใช้ต้องถูกตั้งค่าเมธอดการควบคุมสัญญาณและอุปกรณ์ส่งสัญญาณ เพื่อป้องกันการเข้าถึงที่ขัดแย้งกันและป้องกันการประมวลผลจากการเปลี่ยนแปลงข้อมูล ที่บุคคลอื่นกำลังใช้อยู่ พื้นที่หน่วยความจำที่แบ่งใช้สามารถนำมาใช้ให้เกิดประโยชน์ เมื่อจำนวนข้อมูลที่แลกเปลี่ยนกันระหว่างการประมวลผลมีขนาดใหญ่เกินกว่าที่จะถ่ายโอนด้วยข้อความ หรือเมื่อการประมวลผลจำนวนมากยังคงอยู่ในฐานข้อมูลที่มีขนาดใหญ่

ระบบจะจัดเตรียมสองเมธอดนี้ไว้สำหรับการแม็พไฟล์ และส่วนของหน่วยความจำที่ไม่ระบุ รูทีนต่อไปนี้ซึ่งรู้จักกันในนามของเซอร์วิส **shmat** จะถูกใช้เพื่อสร้าง และใช้เซ็กเมนต์หน่วยความจำที่แบ่งใช้จากโปรแกรม:

รูทีนย่อย	นิยาม
shmctl	ควบคุมการดำเนินการกับหน่วยความจำที่แบ่งใช้
shmget	ขอรับหรือสร้างเซ็กเมนต์หน่วยความจำที่แบ่งใช้
shmat	ผ่วงต่อเซ็กเมนต์หน่วยความจำที่แบ่งใช้จากการประมวลผล ไม่อนุญาตให้คุณ แม็พอุปกรณ์บล็อก
shmdt	ดึงเซ็กเมนต์หน่วยความจำที่แบ่งใช้จากการประมวลผล
mprotect	แก้ไขการป้องกันการเข้าถึงของช่วงของแอดเดรสที่ระบุ ภายในเซ็กเมนต์หน่วยความจำที่แบ่งใช้
disclaim	ลบการแม็พออกจากช่วงของแอดเดรสที่ระบุไว้ ภายในเซ็กเมนต์หน่วยความจำที่แบ่งใช้

รูทีนย่อย **ftok** จัดเตรียมคีย์ที่รูทีน **shmget** ใช้เพื่อสร้างเซ็กเมนต์ที่แบ่งใช้

ชุดของเซอร์วิสสำรองที่รู้จักกันในนามของเซอร์วิส **mmap** จะถูกใช้สำหรับการแม็พไฟล์ แม้ว่าอาจจะถูกใช้สำหรับการสร้างเซ็กเมนต์หน่วยความจำที่แบ่งใช้ด้วยก็ตาม

การดำเนินการทั้งหมดที่ถูกต้องสำหรับผลลัพธ์สำหรับหน่วยความจำที่ได้จาก mmap() ของไฟล์ จะใช้ได้กับผลลัพธ์หน่วยความจำที่ได้จาก mmap() ของอุปกรณ์บล็อก อุปกรณ์บล็อกคือไฟล์พิเศษ ที่จัดเตรียมการเข้าถึงไดรเวอร์อุปกรณ์ที่แสดงอินเตอร์เฟสบล็อก อินเตอร์เฟสบล็อกในไดรเวอร์อุปกรณ์ต้องการเข้าถึงข้อมูลในบล็อก ของขนาดที่กำหนดไว้ อินเตอร์เฟสจะถูกใช้สำหรับอุปกรณ์หน่วยเก็บข้อมูล

เซอร์วิส mmap ประกอบด้วยรูทีนย่อยต่อไปนี้:

รูทีนย่อย	นิยาม
madvise	แนะนำระบบของลักษณะการเพจที่คาดการณ์ไว้ของการประมวลผล
mincore	กำหนดตำแหน่งของหน่วยความจำเพจ
mmap	แม็พอ็อบเจ็กต์ไฟล์กับหน่วยความจำเสมือน อนุญาตให้คุณแม็พบล็อกอุปกรณ์ สำหรับหนึ่งการประมวลผลในหนึ่งครั้ง
mprotect	แก้ไขการป้องกันการเข้าถึงการแม็พหน่วยความจำ
msync	ซิงโครไนซ์ไฟล์ที่แม็พด้วยอุปกรณ์หน่วยเก็บ
munmap	ยกเลิกการแม็พส่วนของหน่วยความจำที่แม็พ

รูทีนย่อย **msem_init, msem_lock, msem_unlock, msem_remove, msleep** และ **mwakeup** จะจัดเตรียมการควบคุมสิทธิในการเข้าถึงสำหรับการประมวลผลที่แม็พโดยใช้เซอร์วิส **mmap**

โปรดอ้างถึงส่วนต่อไปนี้เพื่อเรียนรู้เพิ่มเติมเกี่ยวกับการแม็พหน่วยความจำ:

การเปรียบเทียบ mmap กับ shmat

สำหรับเซอร์วิส shmat ส่วนของพื้นที่แอดเดรสของการประมวลผลที่พร้อมใช้งานสำหรับการแม็พไฟล์กับเซอร์วิส mmap จะพึ่งพาการประมวลผลที่เป็นการประมวลผลแบบ 32 บิต หรือ การประมวลผลแบบ 64 บิต สำหรับการประมวลผลแบบ 32 ส่วนของพื้นที่แอดเดรสที่พร้อมใช้งาน สำหรับการแม็พประกอบด้วยแอดเดรสที่อยู่ในช่วง 0x30000000–0xCFFFFFFF สำหรับจำนวนทั้งหมด 2.5 กิกะไบต์ของพื้นที่แอดเดรส ส่วนของพื้นที่แอดเดรสที่พร้อมใช้สำหรับการแม็พไฟล์ ประกอบด้วยแอดเดรสที่อยู่ในช่วง 0x30000000–0xCFFFFFFF และ 0xE0000000-0xEFFFFFFF สำหรับจำนวนทั้งหมด 2.75 กิกะไบต์ของพื้นที่แอดเดรส ใน AIX 5.2 และเวอร์ชันถัดมา การประมวลผลแบบ 32 บิตจะรันด้วยแบบจำลองพื้นที่แอดเดรสที่มีขนาดใหญ่มาก ซึ่งมีช่วงตั้งแต่ 0x30000000–0xFFFFFFFF ที่พร้อมใช้งานสำหรับการแม็พ และมีจำนวนทั้งหมด 3.25GB ของพื้นที่แอดเดรส

ช่วงที่พร้อมใช้งานทั้งหมดภายในพื้นที่แอดเดรสของการประมวลผลแบบ 32 บิต จะพร้อมใช้งานสำหรับการแม็พตำแหน่งที่กำหนดไว้และตำแหน่งที่ผันแปร การแม็พตำแหน่งที่กำหนดไว้ จะเกิดขึ้นเมื่อแอ็พพลิเคชันระบุการแม็พที่อยู่ที่ตำแหน่งที่ได้กำหนดไว้ ภายในพื้นที่แอดเดรส การแม็พตำแหน่งที่ผันแปรจะเกิดขึ้น เมื่อแอ็พพลิเคชันระบุว่า ระบบควรพิจารณาตำแหน่งที่การแม็พควรจะถูกจัดวาง

สำหรับการประมวลผลแบบ 64 บิต ชุดของช่วงแอดเดรสสองชุดพร้อมกับพื้นที่แอดเดรสของการประมวลผล จะพร้อมใช้งานสำหรับการแม็พ **mmap** หรือ **shmat** ในตอนแรก จะประกอบด้วยช่วงเดี่ยว 0x07000000_00000000–0x07FFFFFF_FFFFFFFF ซึ่งพร้อมใช้งานสำหรับการแม็พทั้งตำแหน่งที่ได้กำหนดไว้และตำแหน่งที่ผันแปร ชุดของช่วงแอดเดรสสำรอง จะพร้อมใช้งานสำหรับการแม็พตำแหน่งที่กำหนดไว้โดยเฉพาะ และประกอบด้วยช่วงตั้งแต่ 0x30000000-0xCFFFFFFF, 0xE0000000-0xEFFFFFFF และ 0x10_00000000–0x06FFFFFF_FFFFFFFF ช่วงสุดท้ายของชุดนี้ ประกอบด้วย 0x10_00000000–0x06FFFFFF_FFFFFFFF ซึ่งจะพร้อมใช้งานกับโหลดเดอร์ของระบบ เพื่อพักข้อความโปรแกรม ข้อมูลและฮีป ดังนั้น ส่วนของช่วงที่ไม่ได้ใช้ จะพร้อมใช้งานสำหรับการแม็พตำแหน่งที่ได้กำหนดไว้

ทั้งเซอร์วิส **mmap** และ **shmat** จะจัดเตรียมความสามารถสำหรับการประมวลผลจำนวนมากเพื่อแม็พส่วนของอ็อบเจ็กต์ที่เหมือนกัน ดังนั้น จึงแบ่งใช้ความสามารถในการกำหนดแอดเดรสให้กับอ็อบเจ็กต์นั้น อย่างไรก็ตาม รูทีนย่อย **mmap** จะขยายความสามารถนี้ให้ใกล้เคียงกับที่จัดเตรียมไว้โดยรูทีนย่อย **shmat** ด้วยการอนุญาตให้ใช้จำนวนของการแม็พที่ไม่มีขีดจำกัดที่ต้องการสร้างขึ้น ขณะที่ความสามารถนี้เพิ่มจำนวนของการแม็พ ที่สนับสนุนต่ออ็อบเจ็กต์ไฟล์ หรือเซ็กเมนต์หน่วยความจำ ความสามารถนี้ยังรับรองแอ็พพลิเคชันที่มีการประมวลผลจำนวนมากแม็พข้อมูลไฟล์ที่เหมือนกัน ลงในพื้นที่แอดเดรส

รูทีนย่อย **mmap** จะจัดเตรียมแอดเดรสของอ็อบเจ็กต์ที่ไม่ซ้ำกันสำหรับการประมวลผลแต่ละกระบวนการที่แม็พกับอ็อบเจ็กต์ ซอฟต์แวร์จะบรรลุเป้าหมายนี้โดยจัดเตรียมการประมวลผลแต่ละกระบวนการด้วยแอดเดรสเสมือนที่ไม่ซ้ำกัน ซึ่งรู้จักกันในนามของนามแฝง รูทีนย่อย **shmat** อนุญาตให้การประมวลผล แบ่งใช้แอดเดรสของอ็อบเจ็กต์ที่แม็พ

เนื่องจากหนึ่งในนามแฝงที่มีอยู่สำหรับเพจที่กำหนดไว้ในอ็อบเจ็กต์เท่านั้น ที่มีการแปลแอดเดรสจริงในเวลาที่กำหนด เฉพาะหนึ่งในการแม็พ **mmap** เท่านั้นที่สามารถทำการอ้างอิงกับเพจนั้น โดยไม่เกิดข้อบกพร่องของเพจ การอ้างอิงกับเพจด้วยการแม็พที่แตกต่างกัน (และนามแฝงที่แตกต่างกันด้วย) ส่งผลทำให้ข้อบกพร่องของเพจที่เป็นสาเหตุทำให้การแปลแอดเดรสจริงมีอยู่ สำหรับเพจที่ไม่ได้ตรวจสอบความถูกต้อง ผลลัพธ์ที่ได้ การแปลใหม่ต้องถูกสร้างขึ้นสำหรับภายในนามแฝงอื่น การประมวลผลจะแบ่งใช้เพจ โดยการย้ายเพจเหล่านั้นระหว่างการแปลที่ต่างกัน

สำหรับแอ็พพลิเคชันที่มีการประมวลผลจำนวนมากแม็พกับข้อมูลไฟล์ที่เหมือนกัน ในพื้นที่แอดเดรสของแอ็พพลิเคชัน การสลับการประมวลผลอาจส่งผล ตรงกันข้ามกับผลการทำงาน ในกรณีเหล่านี้ รูทีนย่อย **shmat** อาจจัดเตรียมความสามารถในการแม็พไฟล์ที่มีประสิทธิภาพมากกว่า

หมายเหตุ: สำหรับระบบที่มีตัวประมวลผล PowerPC แอดเดรสเสมือนจำนวนมาก สามารถมีอยู่สำหรับแอดเดรสจริงที่เหมือนกัน แอดเดรสจริงสามารถกำหนดนามแฝง เพื่อให้แตกต่างจากแอดเดรสในการประมวลผลที่ต่างกันโดยไม่มีการสลับ เนื่องจากไม่มีการสลับ จึงไม่มีการลดระดับของผลการทำงานลง

ใช้เซอร์วิส **shmat** ภายใต้สถานการณ์ต่อไปนี้:

- สำหรับแอ็พพลิเคชันแบบ 32 บิต ไฟล์ตั้งแต่เจ็ดไฟล์ขึ้นไปจะถูกแม็พพร้อมกัน แต่ละไฟล์จะมีขนาดน้อยกว่า 256MB
- ขณะที่แม็พไฟล์ที่มีขนาดใหญ่กว่า 256MB
- ขณะที่แม็พส่วนของหน่วยความจำที่แบ่งใช้ต้องการแบ่งใช้ระหว่างการประมวลผลที่ไม่เกี่ยงข้องกัน (ไม่มีความสัมพันธ์แบบ parent-child)
- ขณะที่แม็พไฟล์ทั้งหมด

ใช้ **mmap** ภายใต้สถานการณ์ต่อไปนี้:

- ความสามารถในการเคลื่อนย้ายแอ็พพลิเคชันจะนำมาพิจารณา
- ไฟล์จำนวนมากถูกแม็พพร้อมกัน
- เฉพาะส่วนของไฟล์ที่ต้องการแม็พเท่านั้น
- การป้องกันระดับของเพจจำเป็นต้องตั้งค่าการแม็พ
- การแม็พแบบส่วนบุคคลเป็นสิ่งจำเป็น

ความสามารถของ " **shmat** ที่ขยายเพิ่ม" จะพร้อมใช้งานสำหรับแอ็พพลิเคชันแบบ 32 บิตด้วยพื้นที่แอดเดรสที่จำกัด ถ้าคุณกำหนดตัวแปรสภาวะแวดล้อม **EXTSHM=ON** ไว้ จากนั้น การประมวลผลในสภาวะแวดล้อมนั้น สามารถสร้างและดึงเซ็กเมนต์หน่วยความจำที่แบ่งใช้มากกว่าเจ็ดเซ็กเมนต์ การประมวลผลสามารถพ่วงกับเซ็กเมนต์เหล่านี้ภายในพื้นที่แอดเดรสสำหรับขนาดของเซ็กเมนต์ เซ็กเมนต์อื่นสามารถพ่วงต่อได้ที่จุดสิ้นสุดขงเซ็กเมนต์แรก ในส่วนที่มีขนาด 256 เมกะไบต์เหมือนกัน แอดเดรสที่การประมวลผลพ่วงต่อคือ ที่ขอบเขตของเพจ ซึ่งมี **SHMLBA_EXTSHM** หลายไบต์

ข้อจำกัดบางอย่างมีอยู่เกี่ยวกับการใช้คุณลักษณะ **shmat** ที่ขยายเพิ่ม ส่วนของหน่วยความจำที่แบ่งใช้เหล่านี้ไม่สามารถใช้เป็นบัฟเฟอร์ I/O ซึ่งการยกเลิกการตรึงบัฟเฟอร์ไว้จะเกิดขึ้นใน handler อินเตอร์รัปต์ ข้อจำกัดเกี่ยวกับการใช้ **shmat** ที่ขยายเพิ่มของบัฟเฟอร์ I/O จะเหมือนกับบัฟเฟอร์ **mmap**

ตัวแปรสภาวะแวดล้อมจัดเตรียมอ็อพชันของการประมวลผลแอ็พพลิเคชันด้วยการทำงานเพิ่มเติม โดยพ่วงต่อเซ็กเมนต์มากกว่า 11 เซ็กเมนต์ เมื่อ **EXTSHM=ON** หรือการเข้าถึงผลการทำงานที่สูงกว่ากับเซ็กเมนต์ 11 เซ็กเมนต์หรือน้อยกว่า เมื่อไม่ได้ตั้งค่าตัวแปรสภาวะแวดล้อมไว้ และอีกครั้ง ความสามารถของ "**shmat** ที่ขยายเพิ่ม" จะใช้ได้กับการประมวลผลแบบ 32 บิต

ข้อควรพิจารณาเกี่ยวกับความเข้ากันได้ของ mmap

เซอร์วิส mmap จะถูกระบุโดยมาตรฐานต่างๆ และการใช้เป็นอินเตอร์เฟสการแม็พไฟล์ของตัวเลือก ในการนำไปปฏิบัติของระบบปฏิบัติการอื่นๆ อย่างไรก็ตาม การนำไปปฏิบัติของระบบของรูทีนย่อย **mmap** อาจแตกต่างจากการนำไปปฏิบัติของระบบอื่นๆ รูทีนย่อย **mmap** จะรวมเข้ากับการปรับเปลี่ยนต่อไปนี้:

- การแม็พในพื้นที่การประมวลผลส่วนบุคคลไม่ได้รับการสนับสนุน
- การแม็พไม่สามารถยกเลิกการแม็พได้ การดำเนินการ **mmap** ที่ระบุ **MAP_FIXED** จะเกิดความล้มเหลว ถ้าการแม็พยังคงอยู่ ภายในช่วงที่ระบุ
- สำหรับการแม็พส่วนบุคคล ความหมายของการทำสำเนาเพื่อเขียนจะสร้างสำเนาของเพจ สำหรับการอ้างอิงการเขียนในครั้งแรก
- การแม็พ I/O หรืออุปกรณ์หน่วยความจำไม่ได้รับการสนับสนุน
- การแม็พอุปกรณ์อักขระหรือใช้ส่วนของ **mmap** เป็นบัฟเฟอร์สำหรับการดำเนินการอ่าน–เขียนไปยังอุปกรณ์อักขระไม่ได้รับการสนับสนุน
- รูทีนย่อย **madvise** ถูกจัดเตรียมไว้สำหรับความเข้ากันได้ เท่านั้น ระบบจะไม่ดำเนินการใดๆ กับขอแนะนำที่ระบุไว้
- รูทีนย่อย **mprotect** อนุญาตให้ส่วนที่ระบุ มีเพจที่ไม่มีการแม็พ ในการดำเนินการ เพจที่ไม่มีการแม็พจะถูกข้าม
- อ็อพชัน OSF/AES ที่ระบุเฉพาะสำหรับดีฟอลต์การแม็พ และสำหรับแฟล็ก **MAP_INHERIT**, **MAP_HASSEMAPHORE** และ **MAP_UNALIGNED** ไม่ได้รับการสนับสนุน

การใช้รูทีนย่อย semaphore

รูทีนย่อย **msem_init, msem_lock, msem_unlock, msem_remove, msleep** และ **mwakeup** ปรับเข้ามาตรฐานกับข้อกำหนดคุณสมบัติ OSF Application Environment ซึ่งจะจัดเตรียมตัวเลือกให้กับอินเตอร์เฟส IPC เช่น รูทีนย่อย **semget** และ **semop** ข้อได้เปรียบของการใช้ semaphores ประกอบด้วยเมธอด serialization ที่ได้ประสิทธิภาพ และการใช้งานที่ลดลง ซึ่งไม่ได้สร้างการเรียกของระบบในกรณีที่ ไม่มี contention สำหรับ semaphore

Semaphores ควรอยู่ในส่วนของหน่วยความจำที่แบ่งใช้ Semaphores จะถูกระบุด้วยโครงสร้าง **msemaphore** ค่าทั้งหมดที่อยู่ในโครงสร้าง **msemaphore** ควรได้รับผลมาจากการเรียกรูทีนย่อย **msem_init** ซึ่งการเรียกนี้อาจ หรืออาจไม่ทำตามลำดับของการเรียกรูทีนย่อย **msem_lock** หรือรูทีนย่อย **msem_unlock** ถ้าค่าของโครงสร้าง **msemaphore** ได้มาด้วยวิธีอื่น ผลลัพธ์ของรูทีนย่อย semaphore ไม่ถูกกำหนดไว้

แอดเดรสของโครงสร้าง **msemaphore** จะมีความสำคัญ คุณควรให้ความระมัดระวังโดยไม่แก้ไขแอดเดรสของโครงสร้าง ถ้าโครงสร้างมีค่าที่คัดลอกมาจากโครงสร้าง **msemaphore** ที่แอดเดรสอื่น ผลลัพธ์ของรูทีนย่อย semaphore จะไม่ถูกกำหนดไว้

รูทีนย่อย semaphore อาจรับรองประสิทธิภาพที่น้อยกว่า เมื่อโครงสร้าง semaphore มีอยู่ในส่วนของหน่วยความจำที่ไม่ได้ระบุไว้ ซึ่งสร้างด้วยรูทีนย่อย **mmap** โดยเฉพาะในกรณีนี้ คือที่ที่การประมวลผลจำนวนมากอ้างอิงถึง semaphores ตัวเดียวกัน ในตัวอย่างเหล่านี้โครงสร้าง semaphore ควรถูกจัดสรรส่วนของหน่วยความจำที่แบ่งใช้ที่มีไม่เพียงพอ ซึ่งสร้างด้วยรูทีนย่อย **shmget** และ **shmat**

การแม็พไฟล์ด้วยรูทีนย่อย shmat

การแม็พสามารถนำมาใช้เพื่อลดการใช้ที่เกี่ยวข้องกับการเขียน และการอ่านเนื้อหาของไฟล์ หากเนื้อหาของไฟล์ ถูกแม็พกับพื้นที่ของหน่วยความจำของผู้ใช้ ไฟล์อาจถูกจัดการ หากมีข้อมูลอยู่ในหน่วยความจำโดยใช้ตัวชี้ไปยังข้อมูลนั้นแทนการเรียกอินพุต/เอาต์พุต สำเนาของไฟล์บนดิสก์ยังใช้เป็นพื้นที่การเพจสำหรับไฟล์นั้น โดยบันทึกพื้นที่การเพจ

โปรแกรมสามารถใช้ไฟล์ปกติใดๆ เป็นไฟล์ข้อมูลที่แม็พ คุณยังสามารถขยายคุณลักษณะของไฟล์ข้อมูลที่แม็พให้กับไฟล์ที่มีอ็อบเจ็กต์โค้ด ที่สามารถเรียกทำงานได้และคอมไพล์ได้ เนื่องจากไฟล์ที่ถูกแม็พสามารถเข้าถึง ได้เร็วกว่าไฟล์ธรรมดา ระบบสามารถโหลดโปรแกรมได้ เร็วกว่าถ้าไฟล์อ็อบเจ็กต์ที่เรียกทำงานได้ถูกแม็พกับไฟล์

หากต้องการสร้างโปรแกรมให้เป็นไฟล์ที่สามารถเรียกทำงานได้ซึ่งแม็พแล้ว ให้คอมไพล์และลิงก์โปรแกรมโดยใช้แฟล็ก **-K** ด้วยคำสั่ง **cc** หรือ **ld** แฟล็ก **-K** จะบอกตัวเชื่อมโยงให้สร้างอ็อบเจ็กต์ไฟล์ด้วยรูปแบบการจัดตำแหน่งเพจ นั่นคือ แต่ละส่วนของอ็อบเจ็กต์ไฟล์จะเริ่มต้นบนขอบเขตของเพจ (แอดเดรสที่สามารถแบ่งออกได้ด้วยขนาด 2K ไบต์โดยไม่มีส่วนที่เหลือ) อ็อพชันนี้จะส่งผลทำให้มีพื้นที่ว่างบางส่วนในอ็อบเจ็กต์ไฟล์ แต่อนุญาตให้ไฟล์ที่สามารถเรียกทำงานได้ถูกแม็พกับหน่วยความจำ เมื่อระบบแม็พอ็อบเจ็กต์ไฟล์ในหน่วยความจำ ข้อความและข้อมูลจะถูกจัดการต่างกัน

ไฟล์ที่ถูกแม็พแบบ Copy-on-write

หากต้องการป้องกันการเปลี่ยนแปลงที่ทำกับไฟล์ที่แม็พไว้จากการปรากฏขึ้นในทันที ในไฟล์บนดิสก์ ให้แม็พไฟล์เป็นไฟล์ที่ทำสำเนาเพื่อเขียนเท่านั้น อ็อพชันนี้จะสร้างไฟล์แม็พพร้อมกับการเปลี่ยนแปลง ที่บันทึกไว้ในพื้นที่การเพจของระบบ แทนการทำสำเนาของไฟล์บนดิสก์ คุณต้องเลือกเพื่อเขียนการเปลี่ยนแปลงเหล่านี้ ลงในสำเนาบนดิสก์เพื่อบันทึกการเปลี่ยนแปลงมิฉะนั้น คุณจะสูญเสียการเปลี่ยนแปลงที่ได้ทำไว้ เมื่อปิดไฟล์

เนื่องจากการเปลี่ยนแปลงไม่ได้มีผลในทันที ในสำเนาของไฟล์ที่ผู้ใช้อื่นอาจเข้าถึง ให้ใช้ไฟล์ที่แม็พแบบทำสำเนาเพื่อเขียนระหว่างการประมวลผลที่ทำงานร่วมกับการประมวลผลอื่น

ระบบไม่ได้ตรวจหาจุดสิ้นสุดของไฟล์ที่แม็พไว้ด้วยรูทีนย่อย **shmat** ดังนั้น ถ้าโปรแกรมเขียนจนใกล้ถึงจุดสิ้นสุดของไฟล์ในไฟล์ที่แม็พ แบบทำสำเนาเพื่อเขียนโดยเก็บลงในเซ็กเมน๖หน่วยความจำที่สอดคล้องกัน (ซึ่งไฟล์ถูกแม็พอยู่) ไฟล์ที่มีอยู่จริงบนดิสก์ จะถูกขยายด้วยบล็อกที่มีค่าศูนย์ในการจัดเตรียมสำหรับข้อมูลใหม่ ถ้าโปรแกรมไม่ได้ใช้รูทีนย่อย **fsync** ก่อนที่จะปิดไฟล์ ข้อมูลที่เขียนลงให้ใกล้กับจุดสิ้นสุดของไฟล์ก่อนหน้านี้ จะไม่ถูกเขียนลงในดิสก์ ไฟล์จะปรากฏขึ้นด้วยขนาดที่ใหญ่กว่าแต่มีเฉพาะศูนย์ที่ถูกเพิ่มไว้ ดังนั้น ให้ใช้รูทีนย่อย **fsync** เสมอก่อนที่จะปิดไฟล์ที่แม็พแบบทำสำเนาเพื่อเขียน เพื่อสงวนข้อมูลที่เพิ่มหรือเปลี่ยนแปลง

การแม็พเซ็กเมนต์หน่วยความจำแบบแบ่งใช้กับรูทีนย่อย shmat

ระบบจะใช้เซ็กเมนต์หน่วยความจำที่แบ่งใช้ด้วยวิธีที่คล้ายกับการสร้าง และใช้ไฟล์ การกำหนดคำศัพท์ในการใช้หน่วยความจำที่แบ่งใช้ ซึ่งเกี่ยวข้องกับคำศัพท์ระบบไฟล์ที่คุ้นเคยมากกว่าเป็นสิ่งที่สำคัญ ในการทำความเข้าใจกับหน่วยความจำที่แบ่งใช้ รายการนิยามของคำศัพท์เกี่ยวกับหน่วยความจำที่แบ่งใช้มีดังนี้:

คำศัพท์	นิยาม
คีย์	identifier เฉพาะของเซ็กเมนต์ที่แบ่งใช้โดยเฉพาะ ซึ่งเชื่อมโยงกับเซ็กเมนต์ที่แบ่งใช้ ตราบนานเท่าที่เซ็กเมนต์ที่แบ่งใช้ยังคงอยู่ ในกรณีนี้จะคล้ายกับ *ชื่อไฟล์* ของไฟล์
shmid	identifier ที่กำหนดให้กับเซ็กเมนต์ที่แบ่งใช้ สำหรับใช้ภายในการประมวลผลโดยเฉพาะ ซึ่งจะคล้ายกับที่ใช้กับ *file descriptor* สำหรับไฟล์
attach	ระบุว่า การประมวลผลต้องพ่วงต่อเซ็กเมนต์ที่แบ่งใช้ หากต้องการใช้ การพ่วงต่อเซ็กเมนต์ที่แบ่งใช้จะคล้ายกับการเปิดไฟล์
detach	ระบุว่า การประมวลผลต้องดึงเซ็กเมนต์ที่แบ่งใช้ออกมา หากเสร็จสิ้นการใช้ การดึงเซ็กเมนต์ที่แบ่งใช้ออกมาจะคล้ายกับการปิดไฟล์

หลักการที่เกี่ยวข้อง:

“ภาพรวมพื้นที่แอดเดรสโปรแกรม” ในหน้า 672

Base Operating System จัดเตรียมเซอร์วิส สำหรับโปรแกรมมิงการใช้หน่วยความจำของแอ็พพลิเคชันโปรแกรม

“การสร้างเซ็กเมนต์หน่วยความจำแบบแบ่งใช้ด้วยรูทีนย่อย shmat” ในหน้า 685

ส่วนนี้อธิบายวิธีสร้างเซ็กเมนต์หน่วยความจำแบบแบ่งใช้ ด้วยรูทีนย่อย **shmat**

“ขีดจำกัดการสื่อสารระหว่างกระบวนการ”

หัวข้อนี้อธิบายขีดจำกัดระบบสำหรับกลไก Interprocess communication (IPC)

ขีดจำกัดการสื่อสารระหว่างกระบวนการ

หัวข้อนี้อธิบายขีดจำกัดระบบสำหรับกลไก Interprocess communication (IPC)

บนระบบ UNIX บางระบบ ผู้ดูแลระบบสามารถแก้ไขไฟล์ **/etc/master** และกำหนดข้อจำกัดสำหรับกลวิธี IPC (เซมาฟอร์ เซ็กเมนต์หน่วยความจำที่แบ่งใช้ และคิวข้อความ) ปัญหาที่เกิดจากวิธีนี้คือ ข้อจำกัดที่มากขึ้น ทำให้ระบบปฏิบัติการใช้หน่วยความจำมากยิ่งขึ้น และอาจส่งผลต่อผลการทำงานในทางตรงกันข้าม

AIX ใช้วิธีอื่นที่ต่างออกไป ใน AIX ขีดจำกัดบนถูก กำหนดสำหรับกลวิธี IPC ซึ่งไม่สามารถตั้งค่าได้ โครงสร้างข้อมูล IPC แต่ละโครงสร้างถูกจัดสรรและยกเลิกการจัดสรรเท่าที่จำเป็น ดังนั้นความต้องการใช้ หน่วยความจำจะขึ้นอยู่กับการใช้งานระบบปัจจุบันของกลวิธี IPC

ความแตกต่างของวิธีนี้บางครั้งทำให้ผู้ใช้ที่กำลังติดตั้ง และใช้ฐานข้อมูลสับสน ขีดจำกัดที่ทำให้เกิดความสับสนมากที่สุดคือ จำนวน เซ็กเมนต์หน่วยความจำที่แบ่งใช้สูงสุดที่สามารถถูกยึดติดในเวลาเดียวกันต่อหนึ่งกระบวนการ สำหรับการประมวลผล 64 บิต จำนวนเซ็กเมนต์หน่วยความจำที่แบ่งใช้สูงสุดคือ 268435456 สำหรับการประมวลผล 32 บิต จำนวนเซ็กเมนต์หน่วยความจำที่แบ่งใช้สูงสุดคือ 11 ยกเว้นใช้ความสามารถ **shmat** ที่ขยายเพิ่ม

ตารางต่อไปนี้สรุปขีดจำกัดเซมาฟอร์ของกลไก IPC

เซมาฟอร์	4.3.0	4.3.1	4.3.2	5.1	5.2	5.3	7.1
จำนวน IDs เซมาฟอร์สูงสุดสำหรับเคอร์เนล 32 บิต	4096	4096	131072	131072	131072	131072	N/A
จำนวน IDs เซมาฟอร์สูงสุดสำหรับเคอร์เนล 32 บิต	4096	4096	131072	131072	131072	1048576	1048576
เซมาฟอร์สูงสุดต่อหนึ่ง ID เซมาฟอร์	65535	65535	65535	65535	65535	65535	65535
การดำเนินการสูงสุดต่อการเรียกใช้ semop	1024	1024	1024	1024	1024	1024	1024
รายการเลิกทำสูงสุดต่อหนึ่งกระบวนการ	1024	1024	1024	1024	1024	1024	1024

เซมาฟอร์	4.3.0	4.3.1	4.3.2	5.1	5.2	5.3	7.1
ขนาดเป็นจำนวนไบต์ของโครงสร้างการเลิกทำ	8208	8208	8208	8208	8208	8208	8208
ค่าสูงสุดของเซมาฟอร์	32767	32767	32767	32767	32767	32767	32767
ค่าสูงสุดในการปรับเปลี่ยนตอนออก	16384	16384	16384	16384	16384	16384	16384

ตารางต่อไปนี้สรุปขีดจำกัดคิวข้อความของกลไก IPC

คิวข้อความ	4.3.0	4.3.1	4.3.2	5.1	5.2	5.3	7.1
ขนาดข้อความสูงสุด	4 MB	4 MB	4 MB	4 MB	4 MB	4 MB	4 MB
ไบต์สูงสุดบนคิว	4 MB	4 MB	4 MB	4 MB	4 MB	4 MB	4 MB
จำนวนสูงสุดของ IDs คิวข้อความสำหรับเคอร์เนล 32 บิต	4096	4096	131072	131072	131072	131072	131072
จำนวนสูงสุดของ IDs คิวข้อความสำหรับเคอร์เนล 32 บิต	4096	4096	131072	131072	131072	1048576	1048576
จำนวนข้อความสูงสุดต่อหนึ่ง ID คิว	524288	524288	524288	524288	524288	524288	524288

ตารางต่อไปนี้สรุปขีดจำกัดของหน่วยความจำที่แบ่งใช้ของกลไก IPC

หน่วยความจำที่แบ่งใช้	4.3.0	4.3.1	4.3.2	5.1	5.2	5.3	7.1
ขนาดเซ็กเมนต์สูงสุด (การประมวลผล 32 บิต)	256 MB	2 GB	2 GB	2 GB	2 GB	2 GB	2 GB
ขนาดเซ็กเมนต์สูงสุด (การประมวลผล 32 บิต)	256 MB	2 GB	2 GB	64 GB	1 TB	1 TB	N/A
ขนาดเซ็กเมนต์สูงสุด (การประมวลผล 64 บิต) สำหรับเคอร์เนล 64 บิต	256 MB	2 GB	2 GB	64 GB	1 TB	32 TB	32 TB
ขนาดเซ็กเมนต์ต่ำสุด	1	1	1	1	1	1	1
จำนวนสูงสุดของ IDs หน่วยความจำที่แบ่งใช้ (เคอร์เนล 32 บิต)	4096	4096	131072	131072	131072	131072	131072

หน่วยความจำที่แบ่งใช้	4.3.0	4.3.1	4.3.2	5.1	5.2	5.3	7.1
จำนวนสูงสุดของ IDs หน่วยความจำที่แบ่งใช้ (เคอร์เนล 32 บิต)	4096	4096	131072	131072	131072	1048576	1048576
จำนวนเซ็กเมนต์สูงสุดต่อหนึ่งการประมวลผล (การประมวลผล 32 บิต)	11	11	11	11	11	11	11
จำนวนเซ็กเมนต์สูงสุดต่อหนึ่งการประมวลผล (การประมวลผล 32 บิต)	268435456	268435456	268435456	268435456	268435456	268435456	268435456

หมายเหตุ: สำหรับการประมวลผล 32 บิต จำนวนเซ็กเมนต์สูงสุดต่อหนึ่งการประมวลผล ถูกจำกัดโดยขนาดของพื้นที่แอดเดรสเท่านั้นเมื่อใช้ความสามารถ **shmat** capability ที่ขยายเพิ่ม

ข้อจำกัด IPC บน AIX4.3

- สำหรับเซมาฟอร์และคิวข้อความ ตารางแสดงขีดจำกัดระบบ
- สำหรับหน่วยความจำที่แบ่งใช้ ขนาดเซ็กเมนต์หน่วยความจำที่แบ่งใช้สูงสุดคือ 256GB
- สำหรับหน่วยความจำที่แบ่งใช้ที่ไม่มีความสามารถ **shmat** ที่ขยายเพิ่ม:
  - การประมวลผลสามารถผูกกับสูงสุดแบ่งใช้ที่แบ่งใช้สูงสุด 11 เซ็กเมนต์
- สำหรับหน่วยความจำที่แบ่งใช้ที่มีความสามารถ **shmat** ที่ขยายเพิ่ม:
  - เมื่อเซ็กเมนต์หน่วยความจำที่แบ่งใช้ถูกยึดเข้า ขนาดจะถูกปัดเศษเป็นจำนวนผลคูณ ของ 4096 ไบต์
  - การประมวลผลสามารถยึดกับเซ็กเมนต์หน่วยความจำที่แบ่งใช้ได้เท่าที่ต้องการที่จะพอดีกับ พื้นที่แอดเดรสที่ใช้ได้ ขนาดพื้นที่แอดเดรสที่ใช้ได้สูงสุดคือ 11 เซ็กเมนต์ หรือ 11 เท่าของ 256 MB
- ความสามารถ **shmat** ที่ขยายเพิ่มถูกใช้ถ้าตัวแปรสภาวะแวดล้อม **EXTSHM** มีค่า ON เมื่อการประมวลผลเริ่ม การทำงาน
- พื้นที่แอดเดรสที่พร้อมใช้สำหรับการยึดกับเซ็กเมนต์หน่วยความจำที่แบ่งใช้จะถูกลดขนาด หากใช้โมเดลพื้นที่แอดเดรสขนาดใหญ่ หรือขนาดใหญ่มาก

ข้อจำกัด IPC บน AIX 4.3.1

- ขนาดสูงสุดของเซ็กเมนต์หน่วยความจำที่หน่วยความจำเพิ่มจาก 256 MB เป็น 2 GB เมื่อมีการยึดกับเซ็กเมนต์หน่วยความจำที่แบ่งใช้ที่มีขนาดใหญ่กว่า 256 ขนาด จะถูกปัดเศษเป็นจำนวนผลคูณของ 256 MB แม้ว่ากำลังใช้งานความสามารถ **shmat** ที่ขยายเพิ่ม

ข้อจำกัด IPC บน AIX 4.3.2

- จำนวนสูงสุดของคิวข้อความ IDs เซมาฟอร์ และเซมาฟอร์หน่วยความจำ ที่แบ่งใช้คือ 131072
- จำนวนสูงสุดของข้อความต่อหนึ่งคิวคือ 524288

ข้อจำกัด IPC บน AIX 5.1

- ขนาดสูงสุดของเซ็กเมนต์หน่วยความจำที่แบ่งใช้สำหรับการประมวลผล 64 บิต คือ 64 GB การประมวลผล 32 บิตไม่สามารถยึดกับเซ็กเมนต์หน่วยความจำที่แบ่งใช้ที่มีขนาดใหญ่กว่า 2 GB

ข้อจำกัด IPC บน AIX 5.2

- ขนาดสูงสุดของเซ็กเมนต์หน่วยความจำที่แบ่งใช้สำหรับการประมวลผล 64 บิต คือ 1 TB การประมวลผล 32 บิตไม่สามารถยึดกับเซ็กเมนต์หน่วยความจำที่แบ่งใช้ที่มีขนาดใหญ่กว่า 2 GB
- แอ็พพลิเคชัน 32 บิตสามารถใช้ความสามารถ **shmat** เพื่อให้มีมากกว่า 11 เซ็กเมนต์เมื่อใช้โมเดลพื้นที่แอดเดรสขนาดใหญ่มาก โดยไม่มีการใช้ **shmat** ที่ขยายเพิ่ม สำหรับข้อมูลเพิ่มเติม เกี่ยวกับโมเดลพื้นที่แอดเดรสขนาดใหญ่มาก
- แอ็พพลิเคชันสามารถเคียวรีดูขีดจำกัด IPC บนระบบได้โดยใช้การเรียกใช้ระบบ **vmgetinfo**

ข้อจำกัด IPC บน AIX 5.3

- ขนาดสูงสุดของเซ็กเมนต์หน่วยความจำที่แบ่งใช้สำหรับการประมวลผล 64 บิต คือ 32 TB การประมวลผล 32 บิตไม่สามารถยึดกับเซ็กเมนต์หน่วยความจำที่แบ่งใช้ที่มีขนาดใหญ่กว่า 2 GB
- แอ็พพลิเคชัน 32 บิตสามารถใช้ความสามารถ **shmat** เพื่อให้มีมากกว่า 11 เซ็กเมนต์เมื่อใช้โมเดลพื้นที่แอดเดรสขนาดใหญ่มาก โดยไม่มีการใช้ **shmat** ที่ขยายเพิ่ม สำหรับข้อมูลเพิ่มเติม เกี่ยวกับโมเดลพื้นที่แอดเดรสขนาดใหญ่มาก
- แอ็พพลิเคชันสามารถเคียวรีดูขีดจำกัด IPC บนระบบได้โดยใช้การเรียกใช้ระบบ **vmgetinfo**

ข้อจำกัด IPC บน AIX 6.1

ไม่สนับสนุนการใช้เคอร์เนล 32 บิตบน AIX 6.1 อีกต่อไป ค่าอื่นๆ ทั้งหมดจะมีค่าเหมือนกัน

หลักการที่เกี่ยวข้อง:

"การทำความเข้าใจกับการแม็พหน่วยความจำ" ในหน้า 675

ความเร็วที่คำสั่งของแอ็พพลิเคชันจะประมวลผลบนระบบ จะแบ่งส่วนตามจำนวนของการดำเนินการสำหรับการเข้าถึงที่จำเป็น เพื่อขอรับข้อมูลภายนอกหน่วยความจำที่สามารถกำหนดแอดเดรสของโปรแกรมได้

การสร้างไฟล์ข้อมูลที่แม็พกับรูทีนย่อย shmat

ส่วนนี้อธิบายวิธีสร้างไฟล์ข้อมูลที่แม็พ โดยใช้รูทีนย่อย **shmat**

เงื่อนไขสิ่งที่จำเป็นต้องมี

ไฟล์ที่จะถูกแม็พเป็นไฟล์ธรรมดา

โพรซีเดอร์

การสร้างไฟล์ข้อมูลที่แม็พเป็นกระบวนการสองขั้นตอน ขั้นแรก คุณสร้างไฟล์ที่แม็พ จากนั้นเนื่องจากรูทีนย่อย **shmat** ไม่ได้จัดเตรียมไว้ คุณต้องโปรแกรมเมธอดสำหรับการตรวจจับ จุดสิ้นสุดของไฟล์ที่แม็พ

1. เมื่อต้องการสร้างไฟล์ข้อมูลที่แม็พ:
   a. เปิด (หรือสร้าง) ไฟล์และบันทึกไฟล์ descriptor:

```
if( ( fildes = open( filename , 2 ) ) < 0 )
{
        printf( "cannot open file\n" );
        exit(1);
}
```

b. แม็พไฟล์กับเซ็กเมนต์ด้วยรูทีนย่อย **shmat**:

```
file_ptr=shmat (fildes, 0, SHM_MAP);
```

ค่าคงที่ SHM_MAP ถูกกำหนดในไฟล์ **/usr/include/sys/shm.h** ค่าคงที่นี้บ่งชี้ว่าไฟล์เป็น ไฟล์ที่แม็พ รวมไฟล์นี้และไฟล์ header หน่วยความจำที่แบ่งใช้อื่นในโปรแกรม ด้วย directives ดังต่อไปนี้:

```
#include <sys/shm.h>
```

2. เมื่อต้องการตรวจจับจุดสิ้นสุดของไฟล์ที่แม็พ:

a. ใช้รูทีนย่อย **lseek** เพื่อไปที่จุดสิ้นสุดของไฟล์:

```
eof = file_ptr + lseek(fildes, 0, 2);
```

ตัวอย่างนี้ตั้งค่า eof เป็น แอดเดรสที่ 1 ไบต์หลังจากจุดสิ้นสุดไฟล์ ใช้ค่านี้เป็นตัวทำเครื่องหมาย end-of-file ในโปรแกรม

b. ใช้ file_ptr เป็นตัวชี้ไปยังจุดเริ่มต้นของไฟล์ข้อมูล และเข้าถึงข้อมูลเหมือนกับข้อมูลอยู่ในหน่วยความจำ:

```
while ( file_ptr < eof)
{
      .
      .
      .
      (references to file using file_ptr)
}
```

หมายเหตุ: รูทีนย่อย **read** และ **write** ยังทำงานกับ ไฟล์ที่แม็พและสร้างข้อมูลเหมือนกับเมื่อตัวชี้ถูกใช้เพื่อเข้าถึงข้อมูล

c. ปิดไฟล์เมื่อโปรแกรมเสร็จสิ้น การทำงานกับไฟล์:

```
close( fildes );
```

การสร้างไฟล์ข้อมูลที่แม็พ copy-on-write ด้วยรูทีนย่อย shmat

ส่วนนี้อธิบายวิธีสร้างไฟล์ข้อมูลที่แม็พแบบ copy-on-write ด้วยรูทีนย่อย **shmat**

เงื่อนไขสิ่งที่จำเป็นต้องมี

ไฟล์ที่จะถูกแม็พเป็นไฟล์ธรรมดา

โพรซีเดอร์

1. เปิด (หรือสร้าง) ไฟล์และบันทึกไฟล์ descriptor:

```
if( ( fildes = open( filename , 2 ) ) < 0 )
{
        printf( "cannot open file\n" );
        exit(1);
}
```

2. แม็พไฟล์กับเซ็กเมนต์แบบ copy-on-write, ด้วยรูทีนย่อย **shmat**:

```
file_ptr = shmat( fildes, 0, SHM_COPY );
```

ค่าคงที่ SHM_COPY ถูกกำหนดในไฟล์ **/usr/include/sys/shm.h** ค่าคงที่นี้บ่งชี้ว่าไฟล์เป็นไฟล์ที่แม็พแบบ copy-on-write รวมไฟล์ header นี้และไฟล์ header หน่วยความจำที่แบ่งใช้ในโปรแกรม ด้วย directives ดังต่อไปนี้:

```
#include <sys/shm.h>
```

3. ใช้ file_ptr เป็นตัวชี้ไปยังจุดเริ่มต้นของไลฟ์ข้อมูล และเข้าถึงข้อมูลเหมือนกับข้อมูลในหน่วยความจำ

```
while ( file_ptr < eof)
{
      .
      .
      .
      (references to file using file_ptr)
}
```

4. ใช้รูทีนย่อย **fsync** เพื่อเขียนการเปลี่ยนแปลงกับสำเนาของไฟล์ ลงบนดิสก์เพื่อบันทึกการเปลี่ยนแปลง:

```
fsync( fildes );
```

5. ปิดไฟล์เมื่อโปรแกรมเสร็จสิ้น การทำงานกับไฟล์:

```
close( fildes );
```

การสร้างเซ็กเมนต์หน่วยความจำแบบแบ่งใช้ด้วยรูทีนย่อย shmat

ส่วนนี้อธิบายวิธีสร้างเซ็กเมนต์หน่วยความจำแบบแบ่งใช้ ด้วยรูทีนย่อย **shmat**

งานหรือเงื่อนไขที่จำเป็นต้องมี

ไม่มี

โพรซีเดอร์

1. สร้างคีย์เพื่อระบุเฉพาะเซ็กเมนต์ที่ แบ่งใช้ ใช้รูทีนย่อย **ftok** เพื่อสร้างคีย์ ตัวอย่างเช่น เพื่อสร้างคีย์ mykey ใช้ project ID ของ R ที่มีใน ตัวแปร proj (type **char**) และชื่อไฟล์ null_file, ใช้คำสั่ง ดังนี้:

```
mykey = ftok( null_file, proj );
```

2. หรือ:
   - สร้างเซ็กเมนต์หน่วยความจำที่แบ่งใช้ด้วยรูทีนย่อย **shmget** ตัวอย่างเช่น เพื่อสร้างเซ็กเมนต์ที่แบ่งใช้ ที่มีขนาด 4096 ไบต์และกำหนด **shmid** ให้กับ ตัวแปร integer mem_id, ใช้คำสั่งดังนี้:

     ```
     mem_id = shmget(mykey, 4096, IPC_CREAT | o666 );
     ```

   - รับเซ็กเมนต์ที่แบ่งใช้ที่สร้างก่อนหน้านี้ ด้วยรูทีนย่อย **shmget** ตัวอย่างเช่น เพื่อรับเซ็กเมนต์ที่แบ่งใช้ ที่เชื่อมโยงกับคีย์ mykey อยู่แล้วและกำหนด **shmid** ให้กับตัวแปร integer mem_id, ใช้คำสั่งดังนี้:

     ```
     mem_id = shmget( mykey, 4096, IPC_ACCESS );
     ```

3. เชื่อมต่อเซ็กเมนต์ที่แบ่งใช้กับกระบวนการ ด้วยรูทีนย่อย **shmat** ตัวอย่างเช่น เมื่อต้องการเชื่อมต่อ เซ็กเมนต์ที่สร้างก่อนหน้านี้ ให้ใช้คำสั่งดังนี้:

```
ptr = shmat( mem_id );
```

ในตัวอย่างนี้ ตัวแปร ptr เป็นตัวชี้ไปที่โครงสร้างที่กำหนดฟิลด์ในเซ็กเมนต์ ที่แบ่งใช้ ใช้โครงสร้างแม่แบบนี้เพื่อเก็บค่าและเรียกคืนข้อมูลในเซ็กเมนต์ ที่แบ่งใช้ แม่แบบนี้ควรมีชื่อเหมือนกันสำหรับกระบวนการทั้งหมดที่ใช้เซ็กเมนต์

4. การทำงานกับข้อมูลในเซ็กเมนต์โดยใช้โครงสร้างแม่แบบ
5. ปลดการเชื่อมต่อจากเซ็กเมนต์โดยใช้รูทีนย่อย **shmdt**:

```
shmdt( ptr );
```

6. ถ้าเซ็กเมนต์ที่แบ่งใช้ไม่จำเป็นต้องใช้อีก เอาเซ็กเมนต์ออกจากระบบด้วยรูทีนย่อย **shmctl**:

```
shmctl( mem_id, IPC_RMID, ptr );
```

หมายเหตุ: คุณยังสามารถใช้คำสั่ง **ipcs** เพื่อรับข้อมูลเกี่ยวกับเซ็กเมนต์ และ คำสั่ง **ipcrm** เพื่อเอา เซ็กเมนต์ออก

หลักการที่เกี่ยวข้อง:

"การทำความเข้าใจกับการแม็พหน่วยความจำ" ในหน้า 675

ความเร็วที่คำสั่งของแอ็พพลิเคชันจะประมวลผลบนระบบ จะแบ่งส่วนตามจำนวนของการดำเนินการสำหรับการเข้าถึงที่จำเป็น เพื่อขอรับข้อมูลภายนอกหน่วยความจำที่สามารถกำหนดแอดเดรสของโปรแกรมได้

ข้อกำหนดการเขียนโปรแกรมพื้นที่การเพจ

จำนวนของการเพจพื้นที่ว่างที่จำเป็นโดยแอ็พพลิเคชัน ขึ้นอยู่กับชนิดของกิจกรรมที่ดำเนินบนระบบ ถ้าการเพจพื้นที่ว่าง รันต่ำ การประมวลผลอาจสูญหาย

ถ้าการเพจพื้นที่ว่างกระทำไม่ได้ ระบบอาจทำงาน ผิดพลาด เมื่อพบสภาพการเพจพื้นที่ว่างทำได้น้อย ควรมีการกำหนด การเพจพื้นที่ว่างเพิ่มเติม

ระบบมอนิเตอร์จำนวนบล็อกการเพจพื้นที่ว่างที่ไม่ได้ใช้ และตรวจจับเมื่อการเพจพื้นที่ว่างทำไม่ได้ คำสั่ง **vmstat** รับสถิติที่เกี่ยวข้องกับสภาพนี้ เมื่อจำนวนบล็อกการเพจพื้นที่ว่างที่ไม่ได้ใช้ต่ำความขอบเขต ที่เรียกว่าระดับ paging space warning ระบบจะแจ้งกระบวนการทั้งหมด (ยกเว้น **kprocs**) เกี่ยวกับสภาพระดับต่ำนี้ โดยส่งสัญญาณ **SIGDANGER**

หมายเหตุ: ถ้าความไม่เพียงพอยังดำเนินอยู่และมีค่าต่ำกว่าขอบเขตที่สอง ที่เรียกว่าระดับ paging space kill ระบบจะส่งสัญญาณ **SIGKILL** ไปที่กระบวนการที่เป็นผู้ใช้หลักของการเพจพื้นที่ว่าง และไม่มี ตัวจัดการสัญญาณสำหรับสัญญาณ **SIGDANGER** (การดำเนินการเริ่มต้นสำหรับ สัญญาณ **SIGDANGER** คือละเว้นสัญญาณ) ระบบทำการส่ง สัญญาณ **SIGKILL** อย่างต่อเนื่อง จนกระทั่งจำนวนบล็อกการเพจพื้นที่ว่างที่ไม่ได้ใช้ อยู่เหนือระดับ paging space kill ถ้าพารามิเตอร์ **low_ps_handling** ถูกตั้งค่าเป็น 2 (ใต้คำสั่ง **vmo**) และไม่พบกระบวนการที่จะทำการ kill (ไม่มีตัวจัดการ **SIGDANGER**) ระบบจะส่งสัญญาณ **SIGKILL** ไปที่กระบวนการแรกสุดที่มีตัวจัดการสัญญาณสำหรับสัญญาณ **SIGDANGER**

กระบวนการที่จัดสรรหน่วยความจำแบบไดนามิกสามารถประกัน ได้ว่ามีการเพจพื้นที่ว่างที่เพียงพอ โดยการมอนิเตอร์ระดับการเพจพื้นที่ว่าง กับรูทีนย่อย **psdanger** หรือ โดยใช้รูทีนการจัดสรรพิเศษ กระบวนการสามารถหลีกเลี่ยงการถูกยุติการทำงานเมื่อถึงระดับ paging space kill โดยการกำหนดตัวจัดการสัญญาณสำหรับ สัญญาณ **SIGDANGER** และโดยใช้รูทีนย่อย **disclaim** เพื่อทำให้หน่วยความจำว่างและทรัพยากรการเพจพื้นที่ว่าง ที่จัดสรรในข้อมูลและพื้นที่สแต็ก และในเซ็กเมนต์หน่วยความจำที่แบ่งใช้

รูทีนย่อยอื่นที่สามารถช่วยในการเรียกข้อมูลการเพจ แบบไดนามิกจาก VMM รวมถึงรูทีนย่อยดังต่อไปนี้:

รูทีนย่อย	คำอธิบาย
mincore	กำหนดตำแหน่งของเพจหน่วยความจำ
madvise	อนุญาตให้กระบวนการแนะนำระบบเกี่ยวกับการทำงานการเพจ ที่คาดไว้
swapqry	ส่งกลับสถานะอุปกรณ์การเพจ
swapon	เปิดใช้งานการเพจหรือการสลับไปที่อุปกรณ์บล็อกที่กำหนดไว้

หลักการที่เกี่ยวข้อง:

"ภาพรวมพื้นที่แอดเดรสโปรแกรม" ในหน้า 672

Base Operating System จัดเตรียมเซอร์วิส สำหรับโปรแกรมมิงการใช้หน่วยความจำของแอ็พพลิเคชันโปรแกรม

รายการของเซอร์วิสการจัดการกับหน่วยความจำ

ฟังก์ชันหน่วยความจำทำงานบนอาร์เรย์ของอักขระ ในหน่วยความจำที่เรียกพื้นที่หน่วยความจำ

รูทีนย่อยเหล่านี้ช่วยให้คุณสามารถ:

- ค้นหาอักขระภายในพื้นที่หน่วยความจำ
- คัดลอกอักขระระหว่างพื้นที่หน่วยความจำ
- เปรียบเทียบเนื้อหาของพื้นที่หน่วยความจำ
- ตั้งค่าพื้นที่หน่วยความจำเป็นค่า

คุณไม่จำเป็นต้องระบุแฟล็กพิเศษใด ให้กับ คอมไพลเลอร์เพื่อใช้ฟังก์ชันหน่วยความจำ อย่างไรก็ตาม คุณต้องรวม ไฟล์ header สำหรับฟังก์ชันเหล่านี้ในโปรแกรมของคุณ เมื่อต้องการรวมไฟล์ header ให้ใช้คำสั่งดังต่อไปนี้:

```
#include <memory.h>
```

มีการจัดเตรียมเซอร์วิสหน่วยความจำดังต่อไปนี้ให้:

เซอร์วิส	คำอธิบาย
compare_and_swap	เปรียบเทียบและสลับค่าข้อมูล
fetch_and_add	อัพเดตตัวแปร single word แบบ atom
fetch_and_and หรือ **fetch_and_or**	ตั้งค่าหรือล้างบิตในตัวแปร single word แบบ atom
malloc, free, realloc, calloc, mallopt, mallinfo, หรือ **alloca**	จัดสรรหน่วยความจำ
memccpy, memchr, memcmp, memcpy, memset หรือ **memmove**	การดำเนินการกับหน่วยความจำ

เซอร์วิส	คำอธิบาย
moncontrol	เริ่มและหยุดการทำโปรไฟล์กระทำการหลังจากการกำหนดค่าเริ่มต้นโดยรูทีนย่อย **monitor**
monitor	เริ่มและหยุดการทำโปรไฟล์กระทำการโดยใช้พื้นที่ข้อมูลที่กำหนด ในพารามิเตอร์ฟังก์ชัน
monstartup	เริ่มและหยุดการทำโปรไฟล์กระทำการโดยใช้พื้นที่ข้อมูลที่กำหนดขนาดเริ่มต้น
mprotect	แก้ไขการป้องกันการเข้าถึงของช่วงของแอดเดรสที่ระบุ ภายในเซ็กเมนต์หน่วยความจำที่แบ่งใช้
msem_init	กำหนดค่าเริ่มต้น semaphore ในไฟล์ที่แม็พหรือช่วงหนวยความจำที่แบ่งใช้
msem_lock	ล็อก semaphore
msem_remove	เอา semaphore ออก
msem_unlock	ปลดล็อก semaphore
msleep	นำกระบวนการเข้าสู่สถานะ sleep เมื่อ semaphore ไม่ว่าง
mwakeup	ทำให้กระบวนการที่รอ semaphore พร้อมทำงาน
disclaim	ปฏิเสธเนื้อหาของช่วงแอดเดรสหน่วยความจำ
ftok	สร้างคีย์การสื่อสารรหว่างกระบวนการมาตรฐาน
getpagesize	รับข้อมูลขนาดของเพจระบบ
psdanger	กำหนดจำนวนของพื้นที่การเพจที่ว่างซึ่งพร้อมใช้งาน
shmat	เชื่อมต่อเซ็กเมนต์หน่วยความจำที่แบ่งใช้หรือไฟล์ที่แม็พกับกระบวนการ ปัจจุบัน
shmctl	ควบคุมการดำเนินการกับหน่วยความจำที่แบ่งใช้
shmdt	เอาเซ็กเมนต์หน่วยความจำที่แบ่งใช้ออก

เซอร์วิส	คำอธิบาย
shmget	รับเซ็กเมนต์หน่วยความจำที่แบ่งใช้
swapon	เปิดใช้งานการเพจหรือการสลับไปที่อุปกรณ์บล็อกที่กำหนดไว้
swapqry	ส่งกลับสถานะอุปกรณ์

รายการของเซอร์วิสการแม็พหน่วยความจำ

รูทีนย่อยการแม็พหน่วยความจำทำงานบนขอบเขตหน่วยความจำ ที่ถูกแม็พกับรูทีนย่อย **mmap**

รูทีนย่อยเหล่านี้ช่วยให้คุณสามารถ:

- แม็พอ็อบเจ็กต์ไฟล์ลงในหน่วยความจำเสมือน
- ซิงโครไนซ์ไฟล์ที่แม็พ
- กำหนดที่อยู่ของเพจหน่วยความจำ
- กำหนดการปกป้องการเข้าถึงกับ ขอบเขตหน่วยความจำที่แม็พ
- ยกเลิกการแม็พขอบเขตหน่วยความจำที่แม็พ

คุณไม่จำเป็นต้องระบุแฟล็กพิเศษใด ให้กับ คอมไพลเลอร์เพื่อใช้ฟังก์ชันหน่วยความจำ อย่างไรก็ตาม คุณต้องรวมไฟล์ header สำหรับบางรูทีนย่อย ถ้าคำอธิบายรูทีนย่อยระบุ ไฟล์ header คุณสามารถรวมไฟล์ด้วยคำสั่งดังต่อไปนี้:

```
#include <HeaderFile.h>
```

เซอร์วิสการแม็พหน่วยความจำดังต่อไปนี้ถูกจัดเตรียมไว้:

เซอร์วิส	คำอธิบาย
madvise	แนะนำระบบของการทำงานการเพจที่คาดของกระบวนการ
mincore	ระบุที่อยู่ของเพจหน่วยความจำ
mmap	แม็พอ็อบเจ็กต์ไฟล์บนหน่วยความจำเสมือน
mprotect	ปรับเปลี่ยนการปกป้องการเข้าถึงของการแม็พหน่วยความจำ
msync	ซิงโครไนซ์ไฟล์ที่แม็พด้วยอุปกรณ์หน่วยเก็บในระดับต่ำกว่า
munmap	ยกเลิกการแม็พขอบเขตหน่วยความจำที่แม็พ

การเขียนโปรแกรมเวกเตอร์ AIX

ตัวประมวลผลของ PowerPC บางตัวนำส่วนขยายเวกเตอร์ในลักษณะของ Single Instruction Multiple Data (SIMD) ไปใช้งาน

บ่อยครั้งที่อ้างถึง AltiVec หรือ VMX ส่วนขยายเวกเตอร์ในสถาปัตยกรรม PowerPC จะจัดเตรียมชุดคำสั่งเพิ่มเติม สำหรับการดำเนินการเวกเตอร์ และฟังก์ชันแมทริกซ์เชิงคณิตศาสตร์

Vector Arithmetic Logic Unit คือหน่วยการคำนวณในลักษณะของ SIMD ซึ่งคำสั่งเดี่ยวจะดำเนินการกับการดำเนินการแบบเดียวกันสำหรับองค์ประกอบของข้อมูลทั้งหมด ของแต่ละเวกเตอร์ AIX 5.3 ที่มีระดับเทคโนโลยี 5300-30 ที่แนะนำไว้ คือรีลีสแรกของ AIX เพื่อเปิดใช้งานโปรแกรมมิงเวกเตอร์ ตัวประมวลผล IBM PowerPC 970 คือตัวประมวลผลแรกที่สนับสนุนโดย AIX ที่นำส่วนขยายเวกเตอร์ไปใช้งาน ตัวประมวลผลเหล่านี้จะพบได้ในเซอร์เวอร์เบลด JS20 ที่นำเสนอด้วย BladeCenter

ภาพรวมของส่วนขยายเวกเตอร์

ส่วนขยายเวกเตอร์ประกอบด้วยชุดของการลงทะเบียน 32 ตัว 128 บิต ที่สามารถมีเวกเตอร์ต่างๆ ซึ่งรวมถึงเลขจำนวนเต็มที่มีเครื่องหมายหรือไม่มีเครื่องหมายแบบ 8 บิต 16 บิต หรือ 32 บิต หรือทศนิยม 32 บิต IEEE ซึ่งมีสถานะของเวกเตอร์ และควบคุมการลงทะเบียนที่มีสถานะที่เป็นปัญหาเกี่ยวกับการบ่งชี้บิต และควบคุมบิตสำหรับเปิดใช้งาน Java หรือโหมดที่ไม่ใช่จาวาสำหรับการดำเนินการกับเลขทศนิยม

โหมดดีฟอลต์ที่กำหนดค่าเริ่มต้นโดย AIX สำหรับการประมวลผลใหม่คือ โหมดจาวาที่เปิดใช้งาน ซึ่งจัดเตรียมการปฏิบัติตามการดำเนินการกับเลขทศนิยมแบบ IEEE โหมดที่ไม่ใช่จาวาสำรอง ส่งผลให้โหมดมีความชัดเจนต่ำสำหรับการคำนวณเลขทศนิยม ซึ่งอาจนำไปปฏิบัติได้เร็วกว่า และเหมาะสำหรับการดำเนินการที่ระบุเฉพาะ ตัวอย่างเช่น สำหรับตัวประมวลผล PowerPC 970 ที่รันในโหมด จาวา คำสั่งเลขทศนิยมเวกเตอร์บางคำสั่งจะพบกับข้อยกเว้น ถ้าตัวถูกดำเนินการอินพุตหรือผลลัพธ์จะผิดปกติ ส่งผลทำให้มีการเลียนแบบ โดยระบบปฏิบัติการ สำหรับเหตุผลนี้ คุณจะได้รับการสนับสนุน ให้พิจารณาการเปิดใช้งานโหมดที่ไม่ใช่จาวา ถ้าการปัดเศษสามารถยอมรับได้ หรือพยายามหลีกเลี่ยงการคำนวณค่าที่ผิดปกติ

ส่วนขยายเวกเตอร์ยังประกอบด้วยคำสั่งมากกว่า 160 คำสั่งที่มีโหลด และหน่วยเก็บที่เข้าถึงได้ระหว่างการลงทะเบียนเวกเตอร์และหน่วยความจำ สำหรับการจัดการลงทะเบียน เลขทศนิยมและการดำเนินการทางตรรกะ และการดำเนินการเปรียบเทียบเวกเตอร์ คำสั่งคำนวณเลขทศนิยมจะใช้รูปแบบ IEEE 754-1985 แต่ไม่ได้รายงานข้อยกเว้น IEEE ผลลัพธ์ที่เป็นค่าดีฟอลต์จะถูกสร้างไว้สำหรับเงื่อนไขข้อยกเว้นทั้งหมดตามที่ระบุโดย IEEE สำหรับข้อยกเว้นที่ดักไว้ เฉพาะ IEEE ที่เป็นค่าดีฟอลต์ที่ปัดเศษให้ใกล้จำนวนเต็มมากที่สุด จะถูกจัดเตรียมไว้ ไม่มีการแบ่งทศนิยมหรือคำสั่งค่ากำลังสองที่จัดเตรียมไว้ แต่คำสั่งการประเมินการแลกเปลี่ยนจะถูกจัดเตรียมไว้สำหรับการแบ่ง และคำสั่งค่ากำลังสองที่แลกเปลี่ยนจะจัดเตรียมไว้สำหรับค่ากำลังสอง

และยังมีการลงทะเบียนพิเศษแบบ 32 บิตที่จัดการโดยซอฟต์แวร์ ที่แสดงตัวพรางบิตของการลงทะเบียนเวกเตอร์ที่ใช้อยู่ ซึ่งอนุญาตให้ระบบปฏิบัติการออปติไมซ์เวกเตอร์ที่บันทึก และเรียกคืนอัลกอริธึมที่เป็นส่วนหนึ่งของบริบทการจัดการสับเปลี่ยน

การกำหนดรันไทม์ของความสามารถของเวกเตอร์

โปรแกรมสามารถกำหนดระบบที่สนับสนุนการขยายเวกเตอร์ โดยการอ่านฟิลด์ **vmx_version** ของโครงสร้าง _system_configuration ถ้าฟิลด์นี้ไม่ใช่ศูนย์ ตัวประมวลผลของระบบและระบบปฏิบัติการจะมีการสนับสนุน สำหรับการขยายเวกเตอร์ แมโคร __power_vmx() จะถูกจัดเตรียมไว้ใน **/usr/include/sys/systemcfg.h** สำหรับการดำเนินการทดสอบนี้ ซึ่งจะมีประโยชน์สำหรับซอฟต์แวร์ที่ใช้ประโยชน์ส่วนขยายเวกเตอร์ เมื่อแสดง หรือใช้พาธของโค้ดสเกลาร์ที่เทียบเท่า เมื่อไม่ได้แสดงไว้

ส่วนขยาย AIX ABI

AIX Application Binary Interface (ABI) ถูกขยายเพื่อให้สนับสนุน สถานะการลงทะเบียนเวกเตอร์และ conventions เพิ่มเติม โปรดอ้างถึง *Assembler Language Reference* สำหรับคำอธิบายโดยสมบูรณ์ของส่วนขยาย ABI

AIX สนับสนุนข้อกำหนดคุณสมบัติโปรแกรมมิงอินเตอร์เฟส AltiVec. ด้านล่างคือตารางของชนิดข้อมูลเวกเตอร์ ในภาษา C และ C++ ชนิดข้อมูลเวกเตอร์ทั้งหมดมีขนาด 16 ไบต์ และต้องถูกจัดตำแหน่งบนขอบเขตขนาด 16 ไบต์ การรวมที่มีชนิดเวกเตอร์ ต้องปฏิบัติตามระเบียบของการจัดตำแหน่งการรวมกันตามข้อกำหนด ของสมาชิกขนาดใหญ่สุด ถ้าการรวมกันที่มีชนิดเวกเตอร์ที่ถูกแพ็ก จะไม่มีการรับประกันการจัดตำแหน่งของชนิดเวกเตอร์แบบ 16 ไบต์ คอมไพลเลอร์ AIX ที่สนับสนุนข้อกำหนดคุณสมบัติโปรแกรมมิงอินเตอร์เฟส AltiVec จำเป็นต้องมี

ตารางที่ 75. ชนิดข้อมูลเวกเตอร์ C ใหม่ และ C++

ชนิด C และ C++ ใหม่	เนื้อหา
อักขระที่ไม่ใช่เครื่องหมายเวกเตอร์	อักขระที่ไม่ใช่เครื่องหมายเวกเตอร์ 16 ตัว
อักขระที่มีเครื่องหมายเวกเตอร์	อักขระที่มีเครื่องหมาย 16 ตัว
อักขระบูลีนเวกเตอร์	อักขระที่ไม่ใช่เครื่องหมายเวกเตอร์ 16 ตัว
ตัวเลขแบบสั้นที่ไม่ใช่เครื่องหมายเวกเตอร์	ตัวเลขแบบสั้นที่ไม่ใช่เครื่องหมาย 8 ตัว
ตัวเลขแบบสั้นที่มีเครื่องหมายเวกเตอร์	ตัวเลขแบบสั้น 8 ตัว
บูลีนเวกเตอร์แบบสั้น	ตัวเลขแบบสั้นที่ไม่ใช่เครื่องหมาย 8 ตัว
เลขจำนวนเต็มที่ไม่ใช่เครื่องหมายเวกเตอร์	เลขจำนวนเต็มที่ไม่ใช่เครื่องหมาย 4 ตัว
เลขจำนวนเต็มที่มีเครื่องหมายเวกเตอร์	เลขจำนวนเต็มที่มีเครื่องหมาย 4 ตัว
บูลีนเลขจำนวนเต็มเวกเตอร์	เลขจำนวนเต็มที่ไม่ใช่เครื่องหมาย 4 ตัว
ทศนิยมเวกเตอร์	4 ทศนิยม

ตารางต่อไปนี้คือเค้าร่างของระเบียบการใช้การลงทะเบียนเวกเตอร์

ตารางที่ 76. ระบบการลงทะเบียนเวกเตอร์

ชนิดของลงทะเบียน	ลงทะเบียน	สถานะ	การใช้งาน
VRs	VR0	ลบเลือนได้	ถอนการลงทะเบียน
	VR1	ลบเลือนได้	ถอนการลงทะเบียน
	VR2	ลบเลือนได้	อาร์กิวเมนต์เวกเตอร์แรก เวกเตอร์ของฟังก์ชันการส่งคืนค่าตัวแรก
	VR3	ลบเลือนได้	อาร์กิวเมนต์เวกเตอร์ที่สอง ถอนออก
	VR4	ลบเลือนได้	อาร์กิวเมนต์เวกเตอร์ที่สาม ถอนออก
	VR5	ลบเลือนได้	อาร์กิวเมนต์เวกเตอร์ตัวที่สี่ ถอน
	VR6	ลบเลือนได้	อาร์กิวเมนต์เวกเตอร์ตัวที่ห้า ถอน
	VR7	ลบเลือนได้	อาร์กิวเมนต์เวกเตอร์ตัวที่หก ถอนออก
	VR8	ลบเลือนได้	อาร์กิวเมนต์เวกเตอร์ตัวที่เจ็ด ถอนออก
	VR9	ลบเลือนได้	อาร์กิวเมนต์เวกเตอร์ตัวที่แปด ถอนออก
	VR10	ลบเลือนได้	อาร์กิวเมนต์เวกเตอร์ตัวที่เก้า ถอนออก
	VR11	ลบเลือนได้	อาร์กิวเมนต์เวกเตอร์ตัวที่สิบ ถอนออก

ตารางที่ 76. ระบบการลงทะเบียนเวกเตอร์ (ต่อ)

ชนิดของลงทะเบียน	ลงทะเบียน	สถานะ	การใช้งาน
	VR12	ลบเลือนได้	อาร์กิวเมนต์เวกเตอร์ตัวที่สิบเอ็ดถอนออก
	VR13	ลบเลือนได้	อาร์กิวเมนต์เวกเตอร์ตัวที่สิบสองถอนออก
	VR14:19	ลบเลือนได้	ถอนออก
	VR20:31	สงวนไว้ (โหมดปกติ) ไม่ลบเลือนe (โหมด ABI ที่ขยายเพิ่ม)	เมื่อโหมดที่เปิดใช้งานเวกเตอร์ดีฟอลต์ถูกนำมาใช้ การลงทะเบียนเหล่านี้จะถูกสงวนไว้และต้องไม่ถูกนำมาใช้ ในโหมดที่เปิดใช้งานเวกเตอร์ ABI ที่ขยายเพิ่ม การลงทะเบียนเหล่านี้จะไม่ลบเลือนและค่าจะถูกสงวนไว้ระหว่างการเรียกฟังก์ชัน
วัตถุประสงค์พิเศษ	VRSAVE	สงวนไว้	ใน AIX ABI VRSAVE จะไม่ถูกนำมาใช้ โปรแกรมที่ยอมทำตาม ABI ต้องไม่ใช้หรือเปลี่ยน VRSAVE
วัตถุประสงค์พิเศษ	VSCR	ลบเลือนได้	สถานะของเวกเตอร์และการควบคุมการลงทะเบียน จะมีจุดอิ่มตัวและโหมดที่ไม่ใช่จาวาที่ควบคุมบิต

ข้อกำหนดคุณสมบัติโปรแกรมมิงอินเตอร์เฟส AltiVec จะกำหนดการลงทะเบียน VRSAVE ที่ต้องการใช้เป็นตัวพรางบิตของการลงทะเบียนเวกเตอร์ที่ใช้งานอยู่ AIX ต้องการแอ็พพลิเคชันที่ไม่เคยแก้ไขการลงทะเบียน VRSAVE

พารามิเตอร์เวกเตอร์ 12 ตัวแรกในฟังก์ชันจะถูกวางอยู่ใน VR2 ถึง VR13 การลงทะเบียนพารามิเตอร์เวกเตอร์ที่ไม่จำเป็นจะมีค่าที่ไม่ได้กำหนดไว้ ตามรายการในฟังก์ชัน พารามิเตอร์เวกเตอร์ที่แสดงอาร์กิวเมนต์ความยาวที่ผันแปรไม่ได้ ไม่ได้ถูกแรเงาในการลงทะเบียนทั่วไป (GPRs) พารามิเตอร์เวกเตอร์เพิ่มเติมใดๆ จากตัวที่13 และใกล้เคียงจะส่งผ่านหน่วยความจำบนสแต็กของโปรแกรม จัดตำแหน่งขนาด 16 ไบต์ ในตำแหน่งที่แม็พที่เหมาะสมภายในส่วนของพารามิเตอร์ที่สอดคล้องกับตำแหน่งในรายการ พารามิเตอร์

สำหรับรายการอาร์กิวเมนต์ความยาวผันแปรได้ **va_list** ยังคงต้องชี้ไปยังตำแหน่งของหน่วยความจำของพารามิเตอร์ถัดไป เมื่อ **va_arg( )** เข้าถึงชนิดของเว็กเตอร์ **va_list** ต้องถูกจัดตำแหน่ง ลงในขอบเขตแบบ 16 ไบต์ก่อน ตัวรับและผู้ใช้รายการอาร์กิวเมนต์ความยาวผันแปรได้ จะรับผิดชอบต่อการดำเนินการจัดตำแหน่งนี้ก่อนที่จะเรียกข้อมูลพารามิเตอร์ ชนิดเวกเตอร์

โครงสร้างที่ไม่ได้แพ็กหรือ union ที่ส่งผ่านค่าซึ่งสมาชิกเวกเตอร์ในทุกที่ภายในโครงสร้าง จะถูกจัดตำแหน่งไปยังขอบเขตแบบ 16 ไบต์บนสแต็ก

ฟังก์ชันที่ใช้รายการอาร์กิวเมนต์ความยาวที่ผันแปรได้จะมีพารามิเตอร์ทั้งหมด ที่แม็พในพื้นที่อาร์กิวเมนต์ที่เรียงลำดับและจัดตำแหน่งตามชนิด แปดคำแรก (32 บิต) หรือคำคู่ (64 บิต) ของรายการอาร์กิวเมนต์ความยาวที่ผันแปรได้ จะถูกแรเงาใน GPRs r3 – r10 ซึ่งประกอบด้วยพารามิเตอร์เวกเตอร์

ฟังก์ชันที่มีค่าส่งคืนที่ประกาศไว้เป็นชนิดข้อมูลเวกเตอร์ จะวางค่าส่งคืนใน VR2 ฟังก์ชันใดๆ ที่ส่งคืนชนิดเวกเตอร์หรือมีพารามิเตอร์เวกเตอร์ ที่ต้องการฟังก์ชันต้นแบบ ซึ่งจะหลีกเลี่ยงคอมไพลเลอร์การแรเงา VRs ใน GPRs สำหรับกรณีทั่วไป

ความเข้ากันได้และความสามารถในการทำงานร่วมกันของ ABI ระบบเดิม

เนื่องจาก ลักษณะของอินเตอร์เฟส (เช่น setjmp(), longjmp(), sigsetjmp(), siglongjmp(), _setjmp(), _longjmp(), getcontext(), setcontext(), makecontext(), และ swapcontext()) ที่ต้องบันทึกและเรียกคืนสถานะเครื่องที่ไม่ลบเลือน ซึ่งมีความเสี่ยงที่ถูกแนะนำเมื่อพิจารณาถึงการพึ่งพาระหว่างระบบเก่าโมดูล ABI ในระบบเก่า และเวกเตอร์ที่ขยายเพิ่ม หากต้องการแก้ปัญหาที่ซับซ้อน ฟังก์ชันในตระกูล setjmp ใน libc ที่ตั้งอยู่ในสมาชิกแบบสแตติกของ libc ซึ่งหมายความว่า AIX แบบไบนารีที่มีอยู่ทุกตัวจะมีข้อจำกัดเกี่ยวกับสำเนาในตระกูลของ setjmp และอื่นๆ ที่มีอยู่ในเวอร์ชันของ AIX ที่ถูกลิงก์ ยิ่งไปกว่านั้น AIX แบบไบนารีที่มีอยู่จะมีนิยามโครงสร้างข้อมูล jmpbufs และ ucontext ที่มีที่เก็บไม่เพียงพอ ต่อสถานะการลงทะเบียนเวกเตอร์ที่ไม่ลบเลือนเพิ่มเติม

กรณีใดๆ ที่โมดูลระบบเก่าและโมดูลใหม่เรียก interleave หรือเรียกกลับไปยังโมดูลระบบเก่า สามารถดำเนินการกับ longjmp() หรือ setcontext() โดยการส่งผ่านการลิงก์ convention ของโมดูลเวกเตอร์ที่ขยายเพิ่ม ซึ่งมีความเสี่ยงในการยอมรับสถานะการลงทะเบียนเวกเตอร์ที่ไม่ลบเลือน

สำหรับเหตุผลนี้ ขณะที่ AIX ABI นิยามการลงทะเบียนเวกเตอร์ที่ไม่ลบเลือน โหมดการรวมตัวกันที่เป็นค่าดีฟอลต์ขณะใช้เวกเตอร์ (AltiVec) ในคอมไพลเลอร์ AIX จะไม่ถูกนำมาใช้ในการลงทะเบียนเวกเตอร์ที่ไม่ลบเลือนใดๆ ผลลัพธ์นี้ในสภาวะแวดล้อมการรวมตัวกัน ที่เป็นค่าดีฟอลต์จะอนุญาตให้ใช้ประโยชน์ของเวกเตอร์ (AltiVec) อย่างปลอดภัยขณะที่ไม่มีความเสี่ยงที่เกี่ยวข้องกับความสามารถในการทำงานร่วมกันกับไลบรารี ระบบเก่า

สำหรับแอ็พพลิเคชันที่มีความสามารถในการทำงานร่วมกันและการพึ่งพาโมดูล จะถูกทำให้รู้จัก อ็อพชันการรวมตัวกันเพิ่มเติมสามารถเปิดใช้งานได้ ซึ่งจะอนุญาตให้ใช้การลงทะเบียนเวกเตอร์ที่ไม่ลบเลือน โมดูลนี้ควรถูกนำมาใช้ เมื่อโมดูลในระบบเก่าที่ต้องพึ่งพาทั้งหมดและลักษณะการทำงานเป็นที่รู้จัก และเข้าใจว่าไม่มีการพึ่งพาฟังก์ชันเช่น setjmp(), sigsetjmp(), _setjmp(), หรือ getcontext() หรือทำให้มั่นใจว่าการส่งผ่านโมดูลทั้งหมดจะถูกดำเนินการโดยใช้ convention การลิงก์รูทีนย่อยปกติ และไม่มีการเรียกกลับไปยังโมดูลระบบเก่า ที่ถูกใช้

ดีฟอลต์ของสภาวะแวดล้อมการรวมเข้าด้วยกันของ AltiVec จะกำหนด **__VEC__** ไว้ล่วงหน้า ตามคู่มือ *AltiVec Technology Programming Interface Manual*

เมื่ออ็อพชันที่ใช้การลงทะเบียนเวกเตอร์ที่ไม่ลบเลือนถูกเปิดใช้งาน สภาวะแวดล้อมการรวมเข้าด้วยกัน ต้องกำหนด **__EXTABI__** ไว้ล่วงหน้า คุณสามารถคอมไพล์โมดูลที่เปิดใช้งานที่ไม่ใช่เวกเตอร์ที่ต้องการขยายการรับรู้ของ ABI ด้วยการนิยาม **__AIXEXTABI** ซึ่งจะมั่นใจว่า โมดูลเหล่านั้นสามารถโต้ตอบกับโมดูลที่เปิดใช้งานเวกเตอร์ได้อย่างปลอดภัยซึ่งจะถูกเปิดใช้งานเพื่อใช้ การลงทะเบียนเวกเตอร์ที่ไม่ลบเลือน

บริบทส่วนขยาย

เพื่อสนับสนุนสถานะเครื่องเพิ่มเติมที่ต้องการโดยส่วนขยายเวกเตอร์ พร้อมกับส่วนขยายอื่นๆ เช่น คีย์ผู้ใช้ AIX 5.3 ส่วนสนับสนุนที่แนะนำสำหรับโครงสร้างบริบทที่ขยายเพิ่ม ความสามารถในการมองเห็นแอ็พพลิเคชันหลัก ใช้ข้อมูลบริบทของเครื่องที่แสดงอยู่ในโครงสร้าง sigcontext ซึ่งเตรียมไว้ให้กับ handler สัญญาณ และผลของการเรียกใช้งานบริบทของเครื่อง ใน sigcontext จะส่งคืนจาก handler สัญญาณ โครงสร้าง sigcontext คือเซ็ตย่อยของโครงสร้าง ucontext ที่มีขนาดใหญ่กว่า โครงสร้างทั้งสอง มีลักษณะเฉพาะสำหรับ sizeof(struct sigcontext) เมื่อ AIX build บริบทสัญญาณที่ต้องการส่งผ่านไปยัง handler สัญญาณ ซึ่งจะ build โครงสร้าง ucontext บนสแต็กของ handler สัญญาณ บริบทเครื่องในส่วนของบริบทสัญญาณ ต้องมีสถานะของเครื่องที่แอ็คทีฟทั้งหมด ลบเลือนได้และไม่ลบเลือน สำหรับบริบทที่ถูกอินเตอร์รัปต์ หากต้องการบรรลุสิ่งนี้โดยไม่ให้กระทบกับความเข้ากันได้แบบไบนารีด้วย handler สัญญาณที่มีอยู่ พื้นที่ที่จองไว้ก่อนหน้านี้ในโครงสร้าง ucontext จะใช้เป็นการบ่งชี้ข้อมูลบริบทที่ขยายเพิ่ม จะพร้อมใช้งาน

ฟิลด์ที่กำหนดขึ้นใหม่ไว้ใน **ucontext, __extctx** คือแอดเดรสของโครงสร้างบริบทส่วนขยาย **struct __extctx** ตามที่กำหนดไว้ในไฟล์ **sys/context.h** ฟิลด์ใหม่นี้ **__extctx_magic** ภายในโครงสร้าง ucontext จะบ่งชี้ว่า ข้อมูลบริบทที่ขยายเพิ่มจะถูกต้องเมื่อค่าของ **__extctx_magic** มีค่าเท่ากับ **__EXTCTX_MAGIC** สถานะของเครื่อง เวกเตอร์เพิ่มเติมสำหรับเธรดโดยใช้ส่วนขยาย เวกเตอร์ จะถูกบันทึกและเรียกคืนเป็นสมาชิกของส่วนขยายบริบทใหม่นี้ในโครงสร้าง ucontext ซึ่งเป็นส่วนหนึ่งของการส่งสัญญาณและเรียกคืน

โครงสร้าง ucontext จะถูกใช้บน APIs (เช่น getcontext(), setcontext(), swapcontext() และ makecontext()) ในกรณีเหล่านี้ บริบทที่ต้องการบันทึก ไว้ตามการดำเนินการที่ตั้งใจ สำหรับการเรียกการเชื่อมโยง convention ที่ต้องการให้สถานะของเครื่องที่ไม่ลบเลือนสามารถบันทึกได้ เนื่องจากโหมดดีฟอลต์ของการเปิดใช้งานเวกเตอร์บน AIX ตามที่กล่าวไว้ในส่วนของ ABI คือไม่ใช้การลงทะเบียนเวกเตอร์ที่ไม่ลบเลือน ซึ่งไม่มีส่วนขยายของโครงสร้าง ucontext ที่ต้องการสำหรับแอ็พพลิเคชันหลัก ถ้าแอ็พพลิเคชันเลือกที่จะเปิดใช้งาน การใช้การลงทะเบียนเวกเตอร์ที่ไม่ลบเลือน ซึ่งจะเลือกโครงสร้าง ucontext ที่ปรับขนาดตามที่ขยายไว้ ซึ่งมีพื้นที่สำหรับฟิลด์ **__extctx** ที่ถูกสอดแทรกไว้โดยนิยามของ **__EXTABI__** โดยคอมไพล์เลอร์ ucontext ที่ขยายเพิ่มยังถูกเลือกโดยนิยามของ **__AIXEXTABI**

เช่นเดียวกัน jmp_buf สำหรับใช้กับ setjmp() หรือ longjmp() ไม่ต้องการเปลี่ยนแปลงสำหรับ แอ็พพลิเคชันที่เปิดใช้งานเวกเตอร์ในโหมดดีฟอลต์ เนื่องจากการลงทะเบียนเวกเตอร์ที่ไม่ลบเลือน ไม่ได้ถูกนำมาใช้ การเปิดใช้งานของการลงทะเบียนเวกเตอร์ที่ไม่ลบเลือน จะส่งผลทำให้การจัดสรร jmp_buf มีขนาดใหญ่กว่า เนื่องจากนิยามของ **__EXTABI__** โดยคอมไพลเลอร์ บัฟเฟอร์กระโดดที่ขยายเพิ่มยังสามารถเรียกใช้งานโดยนิยามของ __AIXEXTABI

โปรดดูไฟล์ส่วนหัว **sys/context.h** สำหรับโครงสร้างรายละเอียดเพิ่มเติม ของข้อมูลบริบทที่ขยายเพิ่ม

การจัดสรรหน่วยความจำเวกเตอร์และการจัดตำแหน่ง

ชนิดข้อมูลเวกเตอร์จะแนะนำชนิดข้อมูลที่ต้องการจัดตำแหน่งขนาด 16 ไบต์ ตามข้อกำหนดคุณสมบัติโปรแกรมมิงอินเตอร์เฟส AltiVec ชุดของรูทีนย่อย malloc (vec_malloc, vec_free, vec_realloc, vec_calloc) จะถูกจัดเตรียมไว้โดย AIX ที่กำหนดการจัดสรรที่ได้จัดตำแหน่งไว้แล้วขนาด 16 ไบต์

การรวมกันการเปิดใช้งานเวกเตอร์ด้วย **_VEC_** ที่กำหนดโดยคอมไพลเลอร์จะส่งผลกับการเรียก malloc และ calloc แบบเดิมที่เปลี่ยนทิศทางไปยังสำเนาของ vector-safe , vec_malloc และ vec_calloc ตามลำดับ ไม่มีโค้ดเวกเตอร์ที่สามารถคอมไพล์เพื่อเลือกการเปลี่ยนทิศทางแบบ malloc และ calloc โดยกำหนด **__AIXVEC** การจัดตำแหน่งของดีฟอลต์การจัดสรรแบบ malloc(), realloc() และ calloc() ยังสามารถควบคุมได้ในเวลารันไทม์

อันดับแรก ภายนอกโปรแกรมใดๆ ตัวแปรสภาวะแวดล้อมใหม่ MALLOCALIGN สามารถตั้งค่าการจัดตำแหน่งดีฟอลต์ตามต้องการสำหรับการจัดสรร malloc() ในทุกครั้ง ตัวอย่างเช่น

```
MALLOCALIGN=16;  export MALLOCALIGN
```

ตัวแปรสภาวะแวดล้อม MALLOCALIGN สามารถตั้งค่าให้เป็น 2 เท่า ซึ่งมากกว่าหรือเท่ากับขนาดของตัวชี้ในโหมดการประมวลผลที่สอดคล้องกัน (4 ไบต์สำหรับโหมดแบบ 32 บิต 8 ไบต์สำหรับโหมด 64 บิต) ถ้า MALLOCALIGN ถูกตั้งค่าไม่ถูกต้อง ค่าจะถูกปัดขึ้นไปเป็นกำลัง 2 และการจัดสรร malloc() ถัดจะถูกจัดตำแหน่งให้เป็นค่านั้น

และ ภายในโปรแกรม โปรแกรมสามารถใช้อ็อพชันคำสั่งใหม่ กับอินเตอร์เฟส mallopt() เพื่อระบุการจัดตำแหน่งที่ต้องการสำหรับการจัดสรรในอนาคต ตัวอย่างเช่น

```
rc = mallopt(M_MALIGN, 16);
```

โปรดอ้างถึง mallopt และ MALLOCALIGN สำหรับข้อมูลเพิ่มเติม

printf และ scanf ของชนิดข้อมูลเวกเตอร์

ตามข้อกำหนดคุณสมบัติโปรแกรมมิงอินเตอร์เฟส AltiVec ส่วนสนับสนุนจะถูกเพิ่มให้กับ AIX ในเวอร์ชันของ scanf, fscanf, sscanf, wsscanf, printf, fprintf, sprintf, snprintf, wsprintf, vprintf, vfprintf, vsprintf, and vwsprintf สำหรับสตริงรูปแบบการแปลงเวกเตอร์ใหม่ ตัวจัดรูปแบบขนาดใหม่จะเป็นดังนี้ :

- vl หรือ lv จะใช้หนึ่งอาร์กิวเมนต์ และแก้ไขการแปลงเลขจำนวนเต็มที่มีอยู่ ซึ่งส่งผลกับ vector signed int, vector unsigned int หรือ vector bool สำหรับการแปลงเอาต์พุต หรือ vector signed int * หรือ vector unsigned int * สำหรับการแปลงอินพุต ข้อมูลที่ใช้เป็นชุดของคอมโพเนนต์สี่ตัวขนาด 4 ไบต์ พร้อมกับรูปแบบการแปลงที่ใช้ ตามลำดับ
- vh หรือ hv จะใช้หนึ่งอาร์กิวเมนต์ และแก้ไขการแปลงเลขจำนวนเต็มแบบสั้น ซึ่งส่งผลกับ vector signed short หรือ vector unsigned short สำหรับการแปลงเอาต์พุต หรือ vector signed short * หรือ vector unsigned short * สำหรับการแปลงอินพุต ข้อมูลที่ใช้เป็นชุดของคอมโพเนนต์แปดตัวขนาด 2 ไบต์ พร้อมกับรูปแบบการแปลงที่ใช้ตามลำดับ
- v ใช้หนึ่งอาร์กิวเมนต์ และแก้ไขการแปลงเลขจำนวนเต็ม 1 ไบต์ อักขระ 1 ไบต์ หรือเลขทศนิยม 4 ไบต์ ถ้าการแปลงเป็นการแปลงเลขทศนิยาม ผลลัพธ์คือ vector float สำหรับการแปลงเอาต์พุต หรือ vector float * สำหรับการแปลงอินพุต ข้อมูลที่ใช้เป็นชุดของคอมโพเนนต์เลขทศนิยมสี่ตัวขนาด 4 ไบต์ พร้อมกับรูปแบบการแปลงที่ใช้ตามลำดับ ถ้าการแปลงคือการแปลงเลขจำนวนเต็ม หรือการแปลงอักขระ ผลลัพธ์คือ vector signed char, vector unsigned char หรือ vector bool char สำหรับการแปลงเอาต์พุต หรือ vector signed char * หรือ vector unsigned char * สำหรับการแปลงอินพุต ข้อมูลที่ใช้เป็นชุดของคอมโพเนนต์ สิบหกตัวขนาด 1 ไบต์ พร้อมกับรูปแบบการแปลง ตามลำดับ

รูปแบบการแปลงใดๆ ที่สามารถใช้กับรูปแบบเอกพจน์ของชนิดข้อมูลเวกเตอร์ สามารถนำมาใช้กับรูปแบบเวกเตอร์ได้ การแปลงเลขจำนวนเต็ม %d, %x, %X, %u, %i และ %o สามารถใช้กับตัวรับรองความยาวเวกเตอร์ %lv, %vl, %hv, %vh และ %v การแปลงอักขระ %c สามารถนำมาใช้กับตัวรับรองความยาวเวกเตอร์ %v การแปลงทศนิยม %a, %A, %e, %E, %f, %F, %g และ %G สามารถใช้กับตัวรับรองความยาวเวกเตอร์ %v

สำหรับการแปลงอินพุต อักขระตัวคั่นเผื่อเลือกสามารถระบุได้โดยแยกพื้นที่ว่างที่นำหน้าตัวคั่น ถ้าไม่ได้ระบุตัวคั่น ตัวคั่นดีฟอลต์คือช่องว่างที่สอดแทรกอักขระพื้นที่ว่าง ที่นำหน้าตัวคั่น เว้นเสียแต่การแปลงเป็น c จากนั้นการแปลงที่เป็นค่าดีฟอลต์คือ null

สำหรับการแปลงเอาต์พุต อักขระตัวคั่นเผื่อเลือกสามารถระบุได้ทันที โดยนำหน้าการแปลงขนาดเวกเตอร์ ถ้าไม่ได้ระบุตัวคั่นไว้ ตัวคั่นดีฟอลต์คือพื้นที่ เว้นเสียแต่การแปลงเป็น cซ จากนั้นตัวคั่นที่เป็นค่าดีฟอลต์คือ null

แอ็พพลิเคชันเธรด

แอ็พพลิเคชันแบบมัลติเธรดจะหาประโยชน์จากส่วนขยายเวกเตอร์ยังคงได้รับการสนับสนุน แอ็พพลิเคชันเหล่านี้จะสนับสนุนในขอบเขตของระบบ (แบบจำลองเธรด 1:1) และขอบเขตการประมวลผล (แบบจำลองเธรด M:N) ถ้าแอ็พพลิเคชันแบบมัลติเธรด ถูกคอมไพล์ด้วยการลงทะเบียนเวกเตอร์ที่ไม่ลบเลือนที่เปิดใช้งาน pthreads สำหรับแอ็พพลิเคชันจะถูกแฟล็กเป็น ABI pthreads ที่ขยายเพิ่ม ผลลัพธ์จะมีการจัดสรรบัฟเฟอร์ ที่บันทึกบริบทที่มีขนาดใหญ่ภายในไลบรารี pthread สำหรับเธรดเหล่านี้ ดีบักเกอร์ dbx AIX ยังจัดเตรียมส่วนสนับสนุนเต็มรูปแบบสำหรับการดีบักในระดับเครื่องของ โปรแกรมแบบมัลติเธรดที่เปิดใช้งานเวกเตอร์

คอมไพลเลอร์

คอมไพลเลอร์ AIX ที่สนับสนุนส่วนขยายเวกเตอร์ต้องปรับเข้ามาตรฐานกับ AIX ABI Vector Extension ตามที่ได้กล่าวไว้ก่อนหน้านี้ โหมดการรวบรวมที่เปิดใช้งานเวกเตอร์บน AIX ควรเป็นการลงทะเบียนเวกเตอร์ที่ปิดใช้งาน อ็อพชันที่ใช้ในการเปิดใช้การลงทะเบียนเวกเตอร์ สามารถจัดเตรียมไว้ และเปิดใช้งานด้วยการตัดสินใจของคุณ หลังจากที่เข้าใจถึงปัญหาและความเสี่ยงโดยพิจารณาถึงโมดูลใหม่และเก่าที่มีความสามารถในการทำงานร่วมกัน

ขณะที่เปิดใช้งานการลงทะเบียนเวกเตอร์ที่ไม่ลบเลือน คอมไพลเลอร์ C หรือ C++ ต้องกำหนด **__EXTABI__** ไว้ก่อน และเมื่อเปิดใช้งานรูปแบบการรวบรวมเวกเตอร์ คอมไพลเลอร์ C หรือ C++ จะถูกคาดการณ์เพื่อกำหนด **__VEC__** ไว้ล่วงหน้า ถ้าการคอมไพล์โมดูล C หรือ C++ ที่เปิดใช้งานที่ไม่ใช่เวกเตอร์ สำหรับการเชื่อมโยงกับโมดูล Fortran ที่เปิดใช้งานเวกเตอร์ ซึ่งเป็นสิ่งที่ดีที่สุดที่โมดูล C หรือ C++ จะคอมไพล์ด้วย **__AIXVEC** ที่กำหนดไว้ (นิยามที่คล้ายกันกับ **__VEC__**) และยัง **__AIXEXTABI** (นิยามที่คล้ายกันกับ **__EXTABI**) ถ้าการลงทะเบียนเวกเตอร์ที่ไม่ลบเลือนถูกเปิดใช้งานในโมดูล Fortran

นอกจากข้อกำหนดคุณสมบัติโปรแกรมมิงอินเตอร์เฟส AltiVec ซึ่งจัดเตรียมส่วนขยายให้กับภาษา C และ C++ สำหรับโปรแกรมมิงเวกเตอร์ คอมไพลเลอร์จะอนุญาตให้หาประโยชน์ของส่วนขยายเวกเตอร์ในค่าติดตั้งของการ optimization เมื่อกำหนดเป้าหมายตัวประมวลผลที่สนับสนุนส่วนขยายเวกเตอร์

โปรดอ้างถึงเอกสารคู่มือคอมไพลเลอร์ของคุณสำหรับรายละเอียดเพิ่มเติม

แอสเซมเบลอร์

แอสเซมเบลอร์ AIX ในไดเร็กทอรี **/usr/ccs/bin/as** สนับสนุนชุดคำสั่งเพิ่มเติมที่กำหนดโดยส่วนขยายเวกเตอร์ และปฏิบัติโดยตัวประมวลผล PowerPC 970 คุณสามารถใช้โหมด -m970 ใหม่หรือ .machine 970 pseudo op ภายในซอร์สไฟล์ เพื่อเปิดใช้งานกลุ่มของคำสั่งเวกเตอร์ใหม่ โปรดอ้างถึง *Assembler Language Reference* สำหรับข้อมูลเพิ่มเติม

ดีบักเกอร์

ดีบักเกอร์ dbx AIX ใน /usr/ccs/bin/dbx จะสนับสนุนการดีบักในระดับเครื่อง ของโปรแกรมที่เปิดใช้งานเวกเตอร์ ส่วนสนับสนุนนี้ประกอบด้วยความสามารถในการถอดแยกคำสั่งเวกเตอร์ใหม่ และเพื่อแสดงและตั้งค่าลงทะเบียนเวกเตอร์ ค่าใหม่ของ $instructionset ที่มีค่า 970 ได้กำหนดไว้สำหรับการเปิดใช้งานการถอดแยกภาษาแอสเซมบลี ของ PowerPC 970 ระบุคำสั่ง ซึ่งจะสอดแทรกคำสั่งเวกเตอร์ เมื่อไม่ได้รัน dbx บนระบบ PowerPC 970 หมายเหตุ ถ้าการรัน dbx บน PowerPC 970 $instructionset ดีฟอลต์จะมีค่า 970

หากต้องการดูเวกเตอร์ที่ลงทะเบียนแล้ว คำสั่งย่อย unset $novregs ต้องถูกนำมาใช้ ซึ่งเวกเตอร์ที่ลงทะเบียนแล้วจะไม่ถูกแสดงตามค่าดีฟอลต์ และ ถ้าตัวประมวลผลไม่สนับสนุนส่วนขยายเวกเตอร์ หรือการประมวลผลหรือเธรดที่ต้องการตรวจสอบไม่ได้ใช้ส่วนขยายเวกเตอร์ ดังนั้น ไม่มีสถานะการลงทะเบียนเวกเตอร์ที่จะถูกแสดง หรือ คำสั่งย่อยการลงทะเบียนจะพิมพ์เวกเตอร์ที่ลงทะเบียนไว้ทั้งหมด และเนื้อหาของเลขฐานสิบหก

คุณยังสามารถแสดงการลงทะเบียนเวกเตอร์แต่ละตัวได้ จัดรูปแบบตามชนิดพื้นฐาน ตัวอย่างเช่น print $vr0 จะแสดงเนื้อหาของการลงทะเบียน VR0 เป็นอาร์เรย์ของเลขจำนวนเต็ม 4 หมายเลข print $vr0c จะแสดงเนื้อหาของการลงทะเบียน VR0 ซึ่งเป็นอาร์เรย์ของอักขระ 16 ตัวอักษร print $vr0s จะแสดงเนื้อหาของการลงทะเบียน VR0 ซึ่งเป็นอาร์เรย์แบบ 8 shorts และ print $vr0f จะแสดงเนื้อหาของการลงทะเบียน VR0 ซึ่งเป็นอาร์เรย์ของ 4 floats

คุณสามารถกำหนดการลงทะเบียนเวกเตอร์ทั้งหมด ตัวอย่างเช่น การกำหนด $vr0 = $vr1 หรือกำหนดองค์ประกอบเวกเตอร์แต่ละตัวของการลงทะเบียนเวกเตอร์ หากกำลังกำหนดองค์ประกอบของอาร์เรย์ ตัวอย่างเช่น กำหนด $vr0[3] = 0x11223344 โดยตั้งค่าสมาชิกเลขจำนวนเต็ม ตัวที่ 4 ของ VR0 กำหนดให้ $vr0f[0] = 1.123 ซึ่งมีผลลัพธ์เป็นสมาชิกของ VR0 อันดับแรกที่ตั้งค่าเป็น 1.123

คุณสามารถติดตามการลงทะเบียนเวกเตอร์โดยตลอดการประมวลผลของฟังก์ชัน หรือโปรแกรม ตัวอย่างเช่น tracei $vr0 ใน main จะแสดงเนื้อหาของ VR0 แต่ละครั้งที่ถูกแก้ไขใน main() เช่นเดียวกัน ด้วยการระบุหนึ่งในรูปแบบการลงทะเบียน ($vr0f, $vr0c, $vr0s) ใน tracei การแสดงผลเนื้อหาแต่ละครั้ง จะถูกจัดรูปแบบตามลำดับ

ตราบเท่าที่คอมไพลเลอร์ยังคงแสดงชนิดข้อมูลเวกเตอร์ เป็นอาร์เรย์ของชนิดพื้นฐาน dbx ควรที่จะสามารถแสดงชนิดข้อมูลเวกเตอร์ ที่จัดรูปแบบเป็นอาร์เรย์

โปรดอ้างถึงเอกสารคู่มือคำสั่ง **dbx** สำหรับข้อมูลเพิ่มเติม

การเปิดใช้งานสำหรับดีบักเกอร์ในกลุ่มที่สามยังจัดเตรียมไว้ในรูปของ PTT_READ_VEC และ PTT_WRITE_VEC ของการดำเนินการ ptrace ใหม่สำหรับการอ่านหรือการเขียนสถานะการลงทะเบียนเวกเตอร์ สำหรับเธรด โปรดอ้างถึงเอกสารคู่มือ ptrace สำหรับรายละเอียด

ระบบไฟล์ **/proc** ยังคงได้รับการพัฒนาเพื่อสนับสนุน ดีบักเกอร์แบบอิง **/proc** สถานะและไฟล์ lwpstatus สำหรับการประมวลผลและเธรดที่เปิดใช้งานเวกเตอร์ตามลำดับจะถูกขยายเพื่อสอดแทรก สถานะการลงทะเบียนเวกเตอร์ ข้อความการควบคุมข้อความใหม่ นั่นคือ PCSVREG จะได้รับการสนับสนุนสำหรับ การเขียนไฟล์การควบคุมการประมวลผลและเธรดสำหรับการตั้งค่าสถานะการลงทะเบียนเวกเตอร์ โปรดอ้างถึงการอ้างอิงไฟล์ **/proc** สำหรับรายละเอียดเพิ่มเติม

ไฟล์แกน

AIX ยังสนับสนุนการรวมกันของเครื่องเวกเตอร์ที่มีสถานะเป็นส่วนหนึ่งของไฟล์แกน สำหรับกระบวนหรือเธรดที่เปิดใช้งานเวกเตอร์ หากตัวประมวลผล หรือเธรดที่กำลังใช้ส่วนขยายเวกเตอร์จะเป็นสถานะเครื่องเวกเตอร์ที่สอดแทรก อยู่ในอิมเมจหลักสำหรับเธรดนั้น หมายเหตุ ถ้าคุณเลือกรูปแบบไฟล์หลัก AIX 4.3 ก่อนล่วงหน้า สถานะของเว็กเตอร์จะไม่ถูกสอดแทรกไว้ สถานะของเวกเตอร์จะสนับสนุนรูปแบบไฟล์หลัก ปัจจุบันเท่านั้น คุณสามารถใช้คำสั่ง dbx เพื่ออ่านและแสดงสถานะของเครื่องเวกเตอร์ของไฟล์หลักที่เปิดใช้งานเวกเตอร์

การจัดสรรหน่วยความจำระบบโดยใช้ระบบย่อย malloc

หน่วยความจำจะถูกจัดสรรให้กับแอ็พพลิเคชัน โดยใช้ระบบย่อย **malloc**

ระบบย่อย malloc คือ API สำหรับจัดการหน่วยความจำ ซึ่งประกอบด้วยรูทีนย่อยต่อไปนี้:

- **malloc**
- **calloc**
- **realloc**
- **free**
- **mallopt**
- **mallinfo**
- **alloca**
- **valloc**
- **posix_memalign**

รูทีนย่อย **malloc** จะจัดการกับอ็อบเจ็กต์หน่วยความจำโลจิคัลที่เรียกว่า *ฮีป* ฮีปคือส่วนของหน่วยความจำ ที่อยู่ในพื้นที่แอดเดรสของแอ็พพลิเคชันระหว่างไบต์สุดท้ายของข้อมูลที่จัดสรรโดยคอมไพลเลอร์ และจุดสิ้นสุดของส่วนของข้อมูล ฮีบคืออ็อบเจ็กต์หน่วยความจำที่จัดสรรและหน่วยความจำที่ส่งคืนโดยระบบย่อย **malloc** ของ API

ระบบย่อย **malloc** จะดำเนินการกับการดำเนินการสำหรับหน่วยความจำพื้นฐานต่อไปนี้:

- การจัดสรร:

  ดำเนินการโดย **malloc, calloc valloc, alloca** และรูทีนย่อย **posix_memalign**
- การจัดสรรคืน:

  ดำเนินการโดยรูทีนย่อย **free**
- การจัดสรรใหม่:

  ดำเนินการโดยรูทีนย่อย **realloc**

รูทีนย่อย **mallopt** และ **mallinfo** จะสนับสนุนความเข้ากันได้กับ System V รูทีนย่อย **mallinfo** สามารถนำมาใช้ในระหว่างการพัฒนาโปรแกรมเพื่อขอรับข้อมูลเกี่ยวกับฮีปที่ถูกจัดการโดยรูทีนย่อย **malloc** รูทีนย่อย **mallopt** สามารถนำมาใช้เพื่อไม่ยอมรับการจัดตำแหน่งหน้า หน่วยความจำอิสระที่มีขนาดเท่าหน้า และเปิดใช้งานและปิดใช้งานตัวจัดสรรดีฟอลต์ เช่นเดียวกับรูทีนย่อย **malloc** รูนทีนย่อย **valloc** จัดเตรียมความเข้ากันได้กับ Berkeley Compatibility Library

สำหรับข้อมูลเพิ่มเติม โปรดดูส่วนต่อไปนี้:

การทำงานกับฮีพกระบวนการ

_edata คือสัญลักษณ์ที่มีแอดเดรสอยู่ในไบต์แรก แล้วตามด้วยไบต์สุดท้ายของข้อมูลโปรแกมที่กำหนดค่าเริ่มต้นแล้ว สัญลักษณ์ **_edata** จะอ้างถึงจุดเริ่มต้นของฮีปการประมวลผล ซึ่งจะถูกขยายโดยระบบย่อย **malloc** เมื่อจัดสรรบล็อกแรกของข้อมูล ระบบย่อย **malloc** จะขยายฮีปการประมวลผลโดยเพิ่มค่าการประมวลผลให้กับ brk ซึ่งจะใช้แทนจุดสิ้นสุดของฮีปการประมวลผล และจะกระทำได้โดยการเรียกรูทีนย่อย **sbrk** ระบบย่อย **malloc** จะขยายฮีปการประมวลผลตามความต้องการของแอ็พพลิเคชันที่กำหนดไว้

ฮีปการประมวลผลจะถูกแบ่งออกเป็นบล็อกหน่วยความจำ *allocated* และ *freed* พูลว่างจะประกอบด้วยหน่วยความจำที่พร้อมใช้งาน สำหรับการจัดสรรในลำดับต่อมา การจัดสรรจะเสร็จสิ้น โดยการลบบล็อกหน่วยความจำเป็นอันดับแรกออกจากพูลว่าง จากนั้นส่งคืนการเรียกฟังก์ชันที่ตัวชี้ไปยังบล็อกนี้ การจัดสรรใหม่จะเสร็จสิ้นโดยจัดสรรบล็อกหน่วยความจำของขนาดใหม่ ย้ายข้อมูลในบล็อกต้นทางไปยังบล็อกใหม่ และทำให้บล็อกต้นฉบับว่าง บล็อกหน่วยความจำที่ถูกจัดสรรแล้วประกอบด้วยชิ้นส่วนของฮีปการประมวลผล ที่ถูกใช้โดยแอ็พพลิเคชัน เนื่องจากบล็อกหน่วยความจำไม่ได้ถูกลบออกจากฮีป (บล็อกเหล่านั้นจะเปลี่ยนสถานะจากบล็อกว่างไปเป็นจัดสรรแล้ว) ขนาดของฮีปการประมวลผลไม่ได้ลดลง เมื่อหน่วยความจำเป็นอิสระจากแอ็พพลิเคชัน

พื้นที่แอดเดรสกระบวนการในแอ็พพลิเคชันแบบ 32 บิต

แอ็พพลิเคชันโปรแกรมแบบ 32 บิตที่รันอยู่บนระบบจะมีพื้นที่แอดเดรส ที่แบ่งออกเป็นเซ็กเมนต์ต่อไปนี้:

เซ็กเมนต์	คำอธิบาย
0x00000000 to 0x0fffffff	มีเคอร์เนล
0x10000000 to 0x1fffffff	มีข้อความแอ็พพลิเคชันโปรแกรม
0x20000000 to 0x2fffffff	มีข้อมูลแอ็พพลิเคชันโปรแกรม ฮีปการประมวลผล และสแต็กของแอ็พพลิเคชัน
0x30000000 to 0xcfffffff	พร้อมใช้งานโดยหน่วยความจำแบบแบ่งใช้ หรือเซอร์วิส **mmap**
0xd0000000 to 0xdfffffff	มีข้อความไลบรารีแบบแบ่งใช้
0xe0000000 to 0xefffffff	พร้อมใช้งานโดยหน่วยความจำแบบแบ่งใช้ หรือเซอร์วิส **mmap**
0xf0000000 to 0xffffffff	มีแอ็พพลิเคชันที่แบ่งใช้ข้อมูลไลบรารี

พื้นที่แอดเดรสกระบวนการในแอ็พพลิเคชันแบบ 64 บิต

แอ็พพลิเคชันแบบโปรแกรมแบบ 64 บิตที่รันอยู่บนระบบที่มีพื้นที่แอดเดรส ที่แบ่งออกเป็นเซ็กเมนต์ต่อไปนี้:

เซ็กเมนต์	คำอธิบาย
0x0000 0000 0000 0000 to 0x0000 0000 0fff ffff	มีเคอร์เนล
0x0000 0000 f000 0000 to 0x0000 0000 ffff ffff	สงวนไว้
0x0000 0001 0000 0000 to 0x07ff ffff ffff ffff	มีข้อความแอ็พพลิเคชันโปรแกรม ข้อมูลแอ็พพลิเคชันโปรแกรม ฮีปการประมวลผล และหน่วยความจำแบบแบ่งใช้ หรือเซอร์วิส **mmap**
0x0800 0000 0000 0000 to 0x08ff ffff ffff ffff	โหลดอ็อบเจ็กต์ส่วนตัว
0x0900 0000 0000 0000 to 0x09ff ffff ffff ffff	ข้อความและข้อมูลไลบรารีที่แบ่งใช้
0x0f00 0000 0000 0000 to 0x0fff ffff ffff ffff	สแต็กของแอ็พพลิเคชัน

หมายเหตุ: AIX ใช้เทคนิคการจัดสรรเพจที่หน่วงเวลา สำหรับหน่วยเก็บที่จัดสรรให้กับแอ็พพลิเคชัน เมื่อหน่วยเก็บได้ถูกจัดสรรให้กับแอ็พพลิเคชันด้วยรูทีนย่อยแล้ว เช่น **malloc** ไม่มีพื้นที่การเพจที่ถูกจัดสรรให้กับหน่วยเก็บนั้น จนกว่าหน่วยเก็บจะถูกอ้างอิงถึง เทคนิคนี้มีประโยชน์สำหรับแอ็พพลิเคชัน ที่จัดสรรเซ็กเมนหน่วยความจำขนาดใหญ่ที่กระจัดกระจาย อย่างไรก็ตาม เทคนินี้สามารถส่งผลกระทบกับความสามารถในการเคลื่อนย้ายของแอ็พพลิเคชันที่จัดสรรจำนวนหน่วยความจำ ที่มีขนาดใหญ่มาก ถ้าแอ็พพลิเคชันคาดการณ์ว่า การเรียก malloc จะเกิดความล้มเหลว เมื่อไม่มีหน่วยเก็บเพียงพอที่จะสนับสนุนคำร้องขอหน่วยความจำ แอ็พพลิเคชันอาจจัดสรรหน่วยความจำมากเกินไป เมื่อหน่วยความจำนี้ถูกอ้างอิงในภายหลัง เครื่องจะมีพื้นที่ในการเพจที่ไม่เพียงพอ และระบบปฏิบัติการ จะหยุดการทำงานของการประมวลผล ดังนั้น ระบบจะไม่ใช้หน่วยความจำเสมือนทั้งหมด แอ็พพลิเคชันที่จัดสรรหน่วยความจำต้องมั่นใจว่า หน่วยเก็บด้านหลังมีอยู่สำหรับหน่วยเก็บที่กำลังจัดสรร การตั้งค่าตัวแปรสภาวะแวดล้อม **PSALLOC** ให้มีค่า `PSALLOC=early` จะเปลี่ยนเทคนิคการจัดสรรพื้นที่การเพจ ไปเป็นอัลกอริธึมการจัดสรรในตอนต้น การจัดสรรในตอนต้น พื้นที่การเพจจะถูกกำหนดไว้ หากหน่วยความจำถูกร้องขอ สำหรับข้อมูลเพิ่มเติม โปรดดู พื้นที่การเพจและหน่วยความจำเสมือน ใน *Operating system and device management*

การทำความเข้าใจกับนโยบายการจัดสรรระบบ

นโยบายการจัดสรรระบบ จะอ้างถึงชุดของโครงสร้างข้อมูล และอัลกอริธึมที่ใช้เพื่อแสดงฮีป และเพื่อนำการจัดสรร จัดสรรคืน และจัดสรรใหม่ไปใช้งาน ระบบย่อย **malloc** จะสนับสนุนนโยบายการจัดสรรที่ต่างกัน ซึ่งรวมถึงนโยบายการจัดสรรที่เป็นค่าดีฟอลต์ นโยบายการจัดสรรแบบ watson นโยบายการจัดสรร malloc 3.1 และนโยบายการจัดสรรที่ผู้ใช้กำหนดเอง API สำหรับการเข้าถึงระบบย่อย **malloc** จะเป็นลักษณะเฉพาะสำหรับนโยบายการจัดสรรทั้งหมด เฉพาะการนำไปปฏิบัติเท่านั้นที่ต่างกัน

คุณสามารถใช้ตัวแปรสภาวะแวดล้อมต่อไปนี้เพื่อระบุนโยบายการจัดสรร และอ็อพชันปกติหรืออ็อพชันการดีบักสำหรับนโยบาย:

- **MALLOCTYPE** ระบุนโยบายการจัดสรร
- **MALLOCOPTIONS** ระบุอ็อพชันปกติให้กับนโยบายการจัดสรรที่เลือกไว้
- **MALLOCDEBUG** ระบุอ็อพชันการดีบักให้กับนโยบายการจัดสรรที่เลือกไว้

- **MALLOCALIGN** ระบุการจัดตำแหน่ง **malloc** ภายนอกที่เป็นค่าดีฟอลต์ให้กับโปรแกรม

นโยบายการจัดสรรที่เป็นค่าดีฟอลต์จะมีประสิทธิภาพมากขึ้น และจะมีตัวเลือกสำหรับแอ็พพลิเคชันหลัก นโยบายการจัดสรรอื่น จะมีคุณสมบัติที่มีลักษณะการทำงานเฉพาะบางอย่างที่สามารถใช้เป็นประโยชน์ในสถานการณ์ที่ระบุเฉพาะ ตามที่ได้กล่าวไว้ใน การเปรียบเทียบนโยบายการจัดสรรที่ต่างกัน

อ็อพชันบางตัวในนโยบายการจัดสรรต่างๆ จะทำงานร่วมกันได้กับอ็อพชันอื่นๆ และสามารถใช้เรียงตามกันได้ เมื่อคุณกำลังใช้อ็อพชันเรียงตามกัน ให้ใช้เครื่องหมายจุลภาค (,) คั่นระหว่างอ็อพชันที่ระบุโดยตัวแปรสภาวะแวดล้อม **MALLOCOPTIONS** และ **MALLOCDEBUG**

ตัวแปรสภาวะแวดล้อม **MALLOCALIGN** สามารถตั้งค่าให้เป็นการจัดตำแหน่งที่เป็นค่าดีฟอลต์ ที่ต้องการสำหรับการจัดสรร **malloc( )** ในทุกๆ ครั้ง ตัวอย่างเช่น

```
MALLOCALIGN=16;  export MALLOCALIGN
```

ตัวแปรสภาวะแวดล้อม **MALLOCALIGN** สามารถตั้งค่าโดยให้มีค่าเป็น 2 เท่าซึ่งจะมากกว่า หรือเท่ากับตัวชี้ในโหมดการทำงานที่สอดคล้องกัน (4 ไบต์ สำหรับโหมด 32 บิต 8 ไบต์สำหรับโหมด 64 บิต) สำหรับโปรแกรมที่เปิดใช้งานเวกเตอร์แบบ 32 บิต ตัวแปรสภาวะแวดล้อมนี้สามารถตั้งค่าเป็น 16 ดังนั้น **malloc( )** ทั้งหมดจะถูกจัดตำแหน่งสำหรับชนิดข้อมูลเวกเตอร์ ถ้าจำเป็น หมายเหตุ โปรแกรมเวกเตอร์แบบ 64 บิตจะได้รับการจัดสรรที่จัดแหน่ง 16 ไบต์

และ ภายในโปรแกรม โปรแกรมสามารถใช้รูทีน **mallopt(M_MALIGN, 16)** เพื่อเปลี่ยนดีฟอลต์ของ **malloc( )** เพื่อเตรียมการจัดสรรที่จัดตำแหน่ง 16 บิต รูทีน **mallopt(M_MALIGN)** อนุญาตให้โปรแกรมควบคุมการจัดตำแหน่ง malloc ที่เป็นค่าดีฟอลต์แบบไดนามิกในเวลารันไทม์

การทำความเข้าใจกับนโยบายการจัดสรรดีฟอลต์

นโยบายการจัดสรรค่าดีฟอลต์ยังคงไว้ซึ่งพื้นที่ว่างในฮีป ที่เป็นโหนดอยู่ในทรีการค้นหาแบบไบนารี *cartesian* ซึ่งโหนดจะถูกเรียงลำดับจากซ้ายไปขวาตามแอดเดรส (แอดเดรสที่เพิ่มขึ้นจะอยู่ทางขวา) และจากบนลงล่างตามขนาด (ซึ่งจะไม่มีchild ที่มีขนาดใหญ่กว่า parent) โครงสร้างข้อมูลนี้จะไม่กำหนดข้อจำกัดเกี่ยวกับจำนวนของขนาดบล็อก ที่สนับสนุนโดยทรี การอนุญาตให้ใช้ช่วงกว้างๆ ของขนาดบล็อกที่อาจเกิดขึ้นได้ เทคนิคการจัดโครงสร้างใหม่จะช่วยออปติไมซ์เวลาในการเข้าถึงตำแหน่งโหนด การแทรก และการลบ และยังช่วยป้องกันการแตกแฟรกเมนต์ด้วย

นโยบายการจัดสรรค่าดีฟอลต์จะสนับสนุนความสามารถต่อไปนี้ :

การจัดสรร

จำนวนการใช้งานที่ขนาดเล็กซึ่งจำเป็น ในการให้เซอร์วิสคำร้องขอการจัดสรร เนื่องจากความต้องการคำนำหน้าเมตาดาต้า และความต้องการจัดตำแหน่งของบล็อกหน่วยความจำแต่ละบล็อกตามความเหมาะสม ขนาดของคำนำหน้าเมตาดาต้าสำหรับการจัดสรรคือ 8 และ 16 ไบต์สำหรับโปรแกรมแบบ 32 บิต และ 64 บิต ตามลำดับ แต่ละบล็อกจะมีการจัดตำแหน่งบนขอบเขตขนาด 16 หรือ 32 ไบต์ ดังนั้นจำนวนของหน่วยความจำทั้งหมดที่ต้องการสำหรับการจัดสรรขนาด n คือ:

```
size = roundup(n + prefix_size, alignment requirement)
```

ตัวอย่างเช่น การจัดสรรขนาด 37 ในการประมวลผลแบบ 32 บิตต้องการ `roundup(37 + 8, 16)` ซึ่งเท่ากับ 48 ไบต์

โหนดของทรีที่มีแอดเดรสต่ำที่สุดซึ่ง จะมีขนาดใหญ่กว่าหรือเท่ากับขนาดที่ต้องการจะถูกลบออกจากทรี ถ้าพบบล็อก ที่มีขนาดใหญ่กว่าขนาดที่ต้องการ บล็อกจะถูกแบ่งออกเป็นสองบล็อก : บล็อกหนึ่งสำหรับขนาดที่ต้องการ และบล็อกที่สอง สำหรับส่วนที่เหลืออยู่ บล็อกที่สองจะเรียกว่า *runt* ซึ่งจะส่งคืนไปยังทรีว่าง สำหรับการจัดสรรในอนาคต บล็อกแรกจะส่งคืนตัวเรียก

ถ้าไม่พบบล็อกที่มีขนาดที่เพียงพอในทรีว่าง อีปจะถูกขยาย บล็อกที่มีขนาดของส่วนขยายที่ได้รับ จะถูกเพิ่มไปยังทรีว่าง และการจัดสรรจะดำเนินการต่อตามที่ระบไว้ก่อนหน้า

การจัดสรรคืน

บล็อกหน่วยความจำที่ถูกจัดสรรคืนด้วยรูทีนย่อย **free** จะถูกส่งคืนไปยังทรี ที่ root แต่ละโหนดในพาธไปยังจุดแทรก สำหรับโหนดใหม่จะถูกตรวจสอบเพื่อดูว่าโหนดนั้นอยู่ติดกันกับ โหนดที่แทรกหรือไม่ ถ้าอยู่ติดกัน ทั้งสองโหนดจะถูกผสานเข้าด้วยกัน และโหนดที่ผสานใหม่นี้จะถูกจัดวางใหม่ในทรี ถ้าไม่พบบล็อกที่อยู่ติดกัน โหนดจะถูกแทรกที่ตำแหน่งที่เหมาะสมในทรี การผสานบล็อกให้อยู่ติดกัน สามารถลดการแตกแฟรกเมนต์ลงได้

การจัดสรรใหม่

ถ้าขนาดของบล็อกที่จัดสรรใหม่ใหญ่กว่าบล็อกเดิม บล็อกเดิมจะถูกส่งคืนไปยังทรีที่ว่างด้วยรูทีนย่อย **free** เพื่อให้การรวมตัวกัน สามารถเกิดขึ้นได้ บล็อกใหม่ของขนาดที่ร้องขอจะถูกจัดสรร ขอมูลจะถูกย้ายจากบล็อกเดิมไปยังบล็อกใหม่ และบล็อกใหม่จะถูกส่งคืนไปยังตัวเรียก

ถ้าขนาดของบล็อกที่ถูกจัดสรรใหม่มีขนาดเล็กกว่าบล็อกเดิม บล็อกจะถูกแยกออก และบล็อกที่เล็กที่สุดจะถูกส่งคืนไปยังทรีที่ว่าง

ข้อจำกัด

นโยบายการจัดสรรดีฟอลต์จะสนับสนุนอ็อพชันต่อไปนี้:

- Malloc Multiheap
- Malloc Buckets
- Malloc Disclaim
- Malloc Thread Cache
- การทำความเข้าใจกับอ็อพชัน no_overwrite

การทำความเข้าใจกับนโยบายการจัดสรร watson

นโยบายการจัดสรรแบบ Watson ยังคงไว้ซึ่งพื้นที่ว่างในฮีปซึ่งเป็นโหนดอยู่ในทรีสีแดง–สีดำ ที่แยกจากกันสองทรี: ทรีแรกจะเรียงตามแอดเดรส ทรีอื่นจะเรียงตามขนาด ทรีสีแดง–สีดำนี้จะจัดเตรียมตัวทำการดำเนินการให้ง่ายขึ้น และมีประสิทธิภาพดีกว่าทรีแบบ cartesian ของตัวจัดสรรดีฟอลต์ ดังนั้น นโยบายการจัดสรรแบบ watson จะเร็วกว่าแบบดีฟอลต์

การจัดสรร

นโยบายการจัดสรรแบบ Watson มีข้อกำหนดเกี่ยวกับการใช้แบบเดียวกับ นโยบายการจัดสรรค่าดีฟอลต์

ขนาดของทรีจะถูกค้นหาบล็อกที่เล็กที่สุดที่เป็นไปได้ ซึ่งจะมีขนาดใหญ่กว่าหรือเท่ากับขนาดที่ต้องการ บล็อกนี้จะถูกลบออกจากทรีขนาด ถ้าพบบล็อกที่มีขนาดใหญ่กว่าขนาดที่ต้องการ บล็อกนั้นจะถูกแบ่งออกเป็นสองบล็อก : บล็อกแรกสำหรับขนาดที่เหลืออยู่ และบล็อกที่สองสำหรับขนาดที่ต้องการ บล็อกแรกจะเรียกว่า *runt* และจะส่งคืนไปยังทรีขนาดสำหรับการจัดสรรในอนาคต บล็อกที่สองจะส่งคืนไปยังตัวเรียก ถ้าบล็อกที่พบในทรีขนาดมีขนาดตรงกับขนาดที่ต้องการ บล็อกจะถูกลบออกจากทรีขนาดและทรีแอดเดรส จากนั้น จะส่งคืนไปยังตัวเรียก

ถ้าไม่พบบล็อกที่มีขนาดที่เพียงพอในทรีว่าง ฮีปการประมวลผลจะถูกขยายออก บล็อกขนาดของส่วนขยายนี้จะถูกเพิ่มไปยังทรีขนาดและทรีแอดเดรส และการจัดสรรยังคงดำเนินการต่อตามที่กล่าวไว้ก่อนหน้า

การจัดสรรคืน

บล็อกหน่วยความจำที่จัดสรรคืนด้วยรูทีนย่อย **free** จะถูกส่งคืนไปยังทรีแอดเดรสที่ root แต่ละโหนดในพาธไปยังจุดแทรกสำหรับโหนดใหม่จะถูกตรวจสอบเพื่อดูว่าโหนดนั้นอยู่ติดกันกับ โหนดที่แทรกหรือไม่ ถ้าอยู่ติดกัน ทั้งสองโหนดจะถูกผสานเข้าด้วยกัน และโหนดที่ผสานใหม่นี้จะถูกจัดวางใหม่ในทรี ถ้าไม่พบบล็อกที่อยู่ติดกัน โหนดจะถูกแทรกที่ตำแหน่งที่เหมาะสมทั้งในทรีแอดเดรส และทรีขนาด

หลังจากการแทรก ทั้งทรีสีแดง-สีดำเต้องถูกตรวจสอบสำหรับการแก้ไขการสร้างความสมดุล

การจัดสรรใหม่

ถ้าขนาดของบล็อกที่จัดสรรใหม่ใหญ่กว่าบล็อกเดิม บล็อกเดิมจะถูกส่งคืนไปยังทรีว่างด้วยรูทีนย่อย **free** เพื่อให้การรวมตัวกันสามารถเกิดขึ้นได้ บล็อกใหม่ของขนาดที่ร้องขอจะถูกจัดสรร ข้อมูลจะถูกย้ายจากบล็อกเดิมไปยังบล็อกใหม่ และบล็อกใหม่จะถูกส่งคืนไปยังตัวเรียก

ถ้าขนาดของบล็อกที่ถูกจัดสรรใหม่มีขนาดเล็กกว่าบล็อกเดิม บล็อกจะถูกแยกออก และส่วนที่เหลือจะถูกส่งคืนไปยังทรีว่าง

ข้อจำกัด

นโยบายการจัดสรรแบบ Watson จะสนับสนุนอ็อพชันต่อไปนี้:

- Malloc Multiheap
- Malloc Disclaim
- Malloc Thread Cache
- การทำความเข้าใจกับอ็อพชัน no_overwrite

การทำความเข้าใจกับนโยบายการจัดสรร malloc 3.1

นโยบายการจัดสรรแบบ malloc 3.1 สามารถเลือกได้โดยตั้งค่า `MALLOCTYPE=3.1` ก่อนที่จะเริ่มต้นทำงานกับการประมวลผล หลังจากนั้น โปรแกรมแบบ 32 บิตทั้งหมดที่รันโดย shell จะใช้นโยบายการจัดสรรแบบ malloc 3.1 (โปรแกรมแบบ 64 บิตจะยังคงใช้นโยบายการจัดสรรค่าดีฟอลต์)

นโยบายการจัดสรรแบบ malloc 3.1 ยังคงไว้ซึ่งฮีปซึ่งเป็นชุดของที่ฝากข้อมูลแฮช 28 แต่ละที่จะชี้ไปยังรายการที่ลิงก์ แต่ละรายการที่ลิงก์ จะมีบล็อกของขนาดโดยเฉพาะ ดัชนีที่อยู่ในที่ฝากข้อมูลแฮชบ่งชี้ถึง ขนาดของบล็อกในรายการที่ลิงก์ ขนาดของบล็อกจะถูกคำนวณโดยใช้สูตรต่อไปนี้ :

$$size = 2^{i+4}$$

โดยที่ *i* จะระบุที่ฝากข้อมูล นั่นหมายความว่า บล็อกที่อยู่ในรายการที่ยึดด้วยที่ฝากข้อมูลที่มีขนาดเป็นศูนย์คือ 20+4 = 16 ไบต์ long ดังนั้น จึงกำหนดคำนำหน้าขนาด 8 ไบต์ บล็อกเหล่านี้สามารถตอบสนองความต้องการของคำร้องขอสำหรับบล็อกระหว่าง 0 และ 8 ไบต์ long ตารางต่อไปนี้แสดงถึงวิธีการกระจายขนาดที่ร้องขอ ระหว่างที่ฝากข้อมูล

หมายเหตุ: อัลกอริธึมสามารถใช้ได้มากเป็นสองเท่าของจำนวนของหน่วยความจำที่เกิดขึ้นจริง ซึ่งร้องขอโดยแอ็พพลิเคชันเพจพิเศษจะต้องการที่ฝากข้อมูลที่มีขนาด 4096 ไบต์ เนื่องจากอ็อบเจ็กต์ที่มีขนาดเท่าเพจหรือใหญ่กว่าจะถูกจัดตำแหน่งตามเพจ เนื่องจากคำนำหน้าจะนำหน้าอยู่หน้าบล็อกในทันที เพจทั้งหมดจะต้องการเพียงแค่คำนำหน้า เท่านั้น

ที่ฝากข้อมูล	ขนาดบล็อก	ขนาดที่แม็พ	เพจที่ใช้
0	16	0... 8	
1	32	9.. 24	
2	64	25... 56	
3	128	57 ... 120	
4	256	121 ... 248	
5	512	249 ... 504	
6	1K	505 ... 1K-8	
7	2K	1K-7 ... 2K-8	
8	4K	2K-7 ... 4K-8	2
9	8K	4K-7 ... 8K-8	3
10	16K	8K-7 ... 16K-8	5
11	32K	16K-7 ... 32K-8	9
12	64K	32K-7 ... 64K-8	17
13	128K	64K-7 ... 128K-8	33
14	256K	128K-7 ... 256K-8	65
15	512K	256K-7 ... 512K-8	129
16	1M	256K-7 ... 1M-8	257
17	2M	1M-7 ... 2M-8	513
18	4M	2M-7 ... 4M-8	1K + 1
19	8M	4M-7 ... 8M-8	2K + 1
20	16M	8M-7 ... 16M-8	4K + 1
21	32M	16M-7 ... 32M-8	8K + 1
22	64M	32M-7 ... 64M-8	16K + 1
23	128M	64M-7 ... 128M-8	32K + 1
24	256M	128M-7 ... 256M-8	64K + 1
25	512M	256M-7 ... 512M-8	128K + 1

ที่ฝากข้อมูล	ขนาดบล็อก	ขนาดที่แม็พ	เพจที่ใช้
26	1024M	512M-7 ... 1024M-8	256K + 1
27	2048M	1024M-7 ... 2048M-8	512K + 1

การจัดสรร

บล็อกที่ถูกจัดสรรจากพูลว่างด้วยการแปลงไบต์ที่ร้องขอในครั้งแรก ไปเป็นดัชนีในอาร์เรย์ที่ฝากข้อมูลโดยใช้สมการต่อไปนี้:

```
ต้องการ = ร้องข้อ + 8

ถ้า ต้องการ <= 16
then
ที่ฝากข้อมูล = 0

ถ้า ต้องการ > 16
then
bucket = (log(needed)/log(2) ปัดเศษลงเป็นจำนวนเต็ม) - 3
```

ขนาดของแต่ละบล็อกในรายการที่ยึดโดย bucket คือ block size = $2^{bucket + 4}$ ถ้ารายการใน bucket เป็น null หน่วยความจำจะถูกจัดสรรโดยใช้รูทีนย่อย **sbrk** เพื่อเพิ่มบล็อกเข้ากับรายการ ถ้าขนาดบล็อกน้อยกว่าเพจ เพจจะถูกจัดสรรโดยใช้รูทีนย่อย **sbrk** และจำนวนของบล็อกที่ได้รับ โดยหารด้วยขนาดบล็อกในขนาดเพจที่ถูกเพิ่ม ให้กับรายการ ถ้าขนาดบล็อกเท่ากับหรือมากกว่าเพจ หน่วยความจำที่ต้องการจะถูกจัดสรรโดยใช้รูทีนย่อย **sbrk** และบล็อกเดี่ยวที่เพิ่มไปยังรายการว่างสำหรับที่ฝากข้อมูล ถ้ารายการว่างไม่ว่าง บล็อกที่อยู่ที่ส่วนหัวของรายการจะถูกส่งคืนไปยังตัวเรียก บล็อกถัดไปในรายการ จะกลายเป็นหัวใหม่

การจัดสรรคืน

เมื่อบล็อกของหน่วยความจำถูกส่งคืนพูลว่าง ดัชนีที่ฝากข้อมูลจะถูกคำนวณด้วยการจัดสรร บล็อกที่ถูกป้อนจะถูกเพิ่มให้กับส่วนหัวของรายการว่าง สำหรับที่ฝากข้อมูล

การจัดสรรใหม่

เมื่อบล็อกของหน่วยความจำถูกจัดสรรใหม่ ขนาดที่ต้องการจะถูกเปรียบเทียบกับขนาดของบล็อกที่มีอยู่ เนื่องจากความต่างของขนาดแบบคร่าวๆ ที่ถูกจัดการโดยที่ฝากข้อมูลเดี่ยว ขนาดบล็อกใหม่จะแม็พกับที่ฝากข้อมูลเดียวกับ ขนาดบล็อกต้นฉบับ ในกรณีนี้ ความยาวของคำนำหน้าจะถูกอัพเดตเพื่อให้มีผลกับขนาดใหม่และบล็อกที่เหมือนกันที่ถูกส่งคืน ถ้าขนาดที่ต้องการมากกว่าบล็อกที่มีอยู่ บล็อกจะถูกป้อน บล็อกใหม่จะถูกจัดสรรจากที่ฝากข้อมูลใหม่ และข้อมูลจะถูกย้ายจากบล็อกเก่าไปยังบล็อกใหม่

ข้อจำกัด

การตั้งค่า MALLOCTYPE=3.1 จะเปิดใช้งานเฉพาะกับนโยบาย malloc 3.1 สำหรับโปรแกรมแบบ 32 บิตเท่านั้น สำหรับโปรแกรมแบบ 64 บิตที่ใช้นโยบาย malloc 3.1 ตัวแปรสภาวะแวดล้อม **MALLOCTYPE** ต้องตั้งค่าเป็น MALLOCTYPE=3.1_64BIT นโยบายการจัดสรรนี้จะมีประสิทธิภาพน้อยกว่าค่าดีฟอลต์ และไม่ขอแนะนำให้ใช้ในกรณีส่วนใหญ่

นโยบายการจัดสรร malloc 3.1 จะสนับสนุนอ็อพชันต่อไปนี้:

- Malloc Disclaim

- การทำความเข้าใจกับอ็อพชัน no_overwrite

การทำความเข้าใจกับนโยบายการจัดสรรพูล

พูล **Malloc** คือผลการทำงานระดับสูงด้านหน้าฟังก์ชัน libc **malloc, calloc, free, posix_memalign** และ **realloc** เพื่อจัดการหน่วยเก็บอ็อบเจ็กต์ที่เล็กกว่า 513 ไบต์ ผลการทำงานที่มีข้อได้เปรียบจะได้รับจาก ความยาวของพาธที่สั้นกว่าและการใช้ประโยชน์จาก ข้อมูลแคชได้ดีกว่า สำหรับแอ็พพลิเคชันแบบมัลติเธรด ยังมีประโยชน์ เพิ่มเติมที่จุดยึดพูลโลคัลของเธรดถูกใช้เพื่อหลีกเลี่ยงการดำเนินการ atomic ด้านหน้านี้ สามารถนำมาใช้ในการเชื่อมกับ schema ของการจัดการหน่วยเก็บใดๆ ที่จัดเตรียมไว้ใน libc ในปัจจุบัน (*yorktown* และ *watson*)

หากต้องการใช้พูล **malloc** ให้รันคำสั่งต่อไปนี้:

```
export MALLOCOPTIONS=pool<:max_size>
```

เมื่อระบุอ็อพชันนี้แล้ว การเก็บรวบรวมพูลที่สร้างในระหว่างการกำหนดค่าเริ่มต้น **malloc** ซึ่งแต่ละพูลจะลิงก์รายการของอ็อบเจ็กต์ขนาดที่กำหนดค่าไว้ พูลที่เล็กที่สุดสามารถเก็บอ็อบเจ็กต์ ของขนาดตัวชี้ (เช่น 8 ไบต์สำหรับแอ็พพลิเคชัน 32 บิต หรือ 16 ไบต์ สำหรับแอ็พพลิเคชัน 64 บิต) แต่ละพูลที่ต่อเนื่องกัน สามารถบรรจุอ็อบเจ็กต์ที่มีขนาดตัวชี้ใหญ่กว่า พูลก่อนหน้านี้ ซึ่งหมายความว่า มี 128 พูลสำหรับแอ็พพลิเคชันแบบ 32 บิต และ 64 พูลสำหรับแอ็พพลิเคชันแบบ 64 บิต การเก็บรวบรวมพูลที่แสดงเป็นอาร์เรย์ของตัวชี้ที่ "ตัวยึด" ที่ลิงก์กับรายการ

พูล **Malloc** จะใช้หน่วยความจำของตนเอง พูลฮีปไม่ได้แบ่งใช้ด้วย **malloc** มาตรฐาน เมื่อระบุไว้ อ็อพชัน **max_size** จะถูกปัดขึ้นตามค่าที่สูงกว่า 2 MB และใช้เพื่อควบคุมขนาดของฮีปพูล อ็อพชัน **max_size** สามารถระบุเป็นเลขทศนิยมหรือเลขฐานสิบหาก นำหน้าด้วย 0x หรือ 0X (ตัวอย่างเช่น `export MALLOCOPTIONS=pool:0x1700000` จะตั้งค่า **max_size** ให้เป็น 24 MB หลังจากที่ปัดเศษขึ้นเป็นจำนวนเต็ม

สำหรับแอ็พพลิเคชันแบบ 32 บิต ขนาดของพูลฮีปจะเริ่มต้นที่ 2 MB ถ้าพื้นที่เพิ่มเติมต้องการ และหน่วยเก็บพูลฮีปทั้งหมดน้อยกว่า **max_size** ซึ่งได้มาเพิ่ม 2 MB พื้นที่ 2 MB แต่ละส่วน จะอยู่บน 2 MB แต่ต้องการไม่ให้ต่อเนื่องพื้นที่ 2 MB อื่นใด สำหรับแอ็พพลิเคชันแบบ 64 บิต พูลฮีปที่ต่อเนื่องกันแบบเดี่ยวของ **max_size** จะถูกจัดสรรในระหว่างการกำหนดค่าเริ่มต้น **malloc** และจะไม่ถูกขยาย ถ้าไม่ได้ระบุ **max_size** ไว้ ระบบจะดีฟอลต์เป็น 512 MB สำหรับแอ็พพลิเคชันแบบ 32 บิต และ 32 MB สำหรับแอ็พพลิเคชันแบบ 64 บิต สำหรับโหมด 32 บิตและ 64 บิต **max_size** จะถูกตั้งค่าเป็น 512 MB ถ้าระบุขนาดที่ใหญ่กว่า สำหรับโหมด 32 บิต **max_size** ถูก ตั้งค่าเป็น 512MB และสำหรับโหมด 64 บิต **max_size** ถูกตั้งค่าเป็น 3.7 GB ถ้าระบุขนาดที่ใหญ่กว่า

การใช้ประโยชน์จากหน่วยเก็บ

ตัวยึดพูลทั้งหมดจะถูกตั้งค่าเป็น NULL หรือว่างในตอนต้น เมื่อพูล **malloc** ให้เซอร์วิส คำร้องขอและพูลที่สอดคล้องกันจะว่างเปล่า รูทีนจะถูกเรียกซึ่งจะจัดสรรหน่วยเก็บจากอีปของพูลในด้วยขนาด 1024 ไบต์แบบต่อเนื่องในเขตของ 1024 ไบต์ อ็อบเจ็กต์จำนวนมากที่มีขนาดที่ร้องขอ จะถูก "สร้าง" แอดเดรสแรกจะส่งคืนเพื่อตอบสนองความต้องการของคำร้องขอ ขณะที่อ็อบเจ็กต์ที่เหลืออยู่จะถูกลิงก์เข้าด้วยกัน และวางอยู่บนตัวยึดพูล สำหรับส่วนที่มีขนาด 1024 ไบต์ จะมี 2 ไบต์สำหรับรายการในตารางสำรอง ที่ใช้โดยอย่างอิสระเพื่อกำหนดขนาดของอ็อบเจ็กต์ที่ส่งคืน

เมื่ออ็อบเจ็กต์เป็นอิสระจากพูล **malloc** อ็อบเจ็กต์นั้นก็จะถูก "ส่ง" ไปยังตัวยึดพูลที่เหมาะสม ไม่มีความพยายามในการรวมบล็อก เพื่อสร้างอ็อบเจ็กต์ที่มีขนาดใหญ่กว่า

เนื่องจากลักษณะการทำงานนี้ พูล **malloc** อาจใช้หน่วยเก็บที่มากกว่ารูปแบบอื่นของ **malloc**

การจัดตำแหน่ง

การจัดตำแหน่งดีฟอลต์สำหรับรูทีนย่อย **malloc( ), calloc( )** และ **realloc( )** ต้องถูกระบุโดยตั้งค่าตัวแปรสภาวะแวดล้อม **MALLOCALIGN** อย่างเหมาะสม รูทีนย่อย **posix_memalign( )** ยังคงทำงาน แม้ว่าจะไม่ได้ตั้งค่าตัวแปรสภาวะแวดล้อม **MALLOCALIGN** ไว้ ถ้า **MALLOCALIGN** มีขนาดใหญ่กว่า 512 พูล **malloc** จะไม่ถูกใช้

ประสิทธิภาพของแคช

อ็อบเจ็กต์หน่วยความจำที่จัดสรรด้วย **พูล malloc** จะไม่มีคำนำหน้าหรือคำต่อท้าย บรรทัดข้อมูลแคชจะถูกแพ็กไว้อย่างแน่นหนา ด้วยแอ็พพลิเคชันที่สามารถใช้ข้อมูล อ็อบเจ็กต์หน่วยความจำทั้งหมดที่มีขนาด 2 เท่าจะถูกจัดวางตำแหน่งบนขอบเขตที่เท่ากับขนาดนั้น แต่ละอ็อบเจ็กต์จะมี จำนวนของบรรทัดแคชเพียงเล็กน้อยอยู่ภายใน รูทีนย่อย **malloc** และ **free** ไม่ได้สแกนทรีหรือรายการที่ลิงก์ และไม่ได้ทำให้แคช“เสียหาย”

การสนับสนุนแบบมัลติเธรด

พูล **Malloc** ช่วยเพิ่มประสิทธิภาพการทำงานได้อย่างมากใน สถานการณ์แบบมัลติเธรด เนื่องจากช่วยลด contention การล็อก และ ความต้องการการดำเนินการ atomic

การสนับสนุน Load balancing

ในสถานการณ์แบบมัลติเธรดบางสถานการณ์ ฟรีพูลของหนึ่งเธรดอาจเพิ่มขึ้นอย่างมากเนื่องจากการฟรีซ้ำๆ กันของหน่วยความจำที่จัดสรรแบบไดนามิก อย่างไรก็ตาม เธรดอื่นอาจไม่สามารถใช้หน่วยความจำนี้ได้

การสนับสนุน Load-balancing ทำให้เธรดรีลีสครึ่งหนึ่งของหน่วยความจำ ในแต่ละพูลหลังจากที่พูลถึงค่าขีดจำกัด เพื่อให้เธรดอื่นสามารถใช้ได้ คุณสามารถปรับค่าขีดจำกัด ที่พูลของเธรดจะถูกปรับใหม่ได้

เมื่อต้องการเปิดใช้งานการสนับสนุน load-balancing คุณต้องเอ็กซ์พอร์ตอ็อพชัน ต่อไปนี้:

```
export MALLOCOPTIONS=pool:0x80000000,pool_balanced
```

```
export MALLOCFREEPOOL=min_size<-max_size>:threshold_value<,min_size<-max_size>:
threshold_value, ... >,default:threshold
```

ตัวอย่าง ต่อไปนี้จะตั้งค่าขีดจำกัดสำหรับพูลที่จัดเตรียม หน่วยความจำ 0 –16 ไบต์และ 256 chunk และค่าขีดจำกัดของพูลที่ให้บริการ chunk 32 ไบต์ถึง chunk 512 ไบต์ สำหรับพูลที่เหลือ chunk 128 ๆบต์เป็นค่าขีดจำกัด

```
export MALLOCFREEPOOL=0-16:256,32:512,default:128
```

การสนับสนุนการดีบัก

ไม่มีดีบักในเวอร์ชันของผลการทำงานที่สูงในส่วนหน้า ถ้าตัวแปรสภาวะแวดล้อม **MALLOCDEBUG** ถูกตั้งค่าไว้ อ็อพชันพูลจะถูกละเว้น ซึ่งจะคาดการณ์ว่า แอ็พพลิเคชันจะถูกดีบักโดยใช้ **malloc** แบบ "ปกติ" ก่อนที่จะเรียกใช้การสร้างพูล

การทำความเข้าใจกับนโยบายการจัดสรรที่ผู้ใช้กำหนด

ระบบย่อย malloc จะจัดเตรียมกลไกโดยตลอดซึ่งผู้ใช้อาจพัฒนาอัลกอริธึมของตนเอง เพื่อแม็พฮีปของระบบกับการจัดสรรหน่วยความจำ

การทำความเข้าใจกับอ็อพชัน no_overwrite

อ็อพชันเพิ่มเติมที่พร้อมใช้งานกับการจัดสรรนโยบายทั้งหมดคือ no_overwrite หากต้องการลดการใช้โค้ด glink ภายในระบบย่อย malloc description ฟังก์ชันสำหรับ API ระบบย่อย **malloc** จะถูกเขียนทับด้วย descriptor ฟังก์ชันสำหรับการนำไปปฏิบัติจริง เนื่องจากโปรแกรมบางตัว เช่น ตัวดีบักในกลุ่มที่สามอาจไม่ทำงาน เมื่อตัวชี้ฟังก์ชันถูกแก้ไขอยู่ในวิธีการนี้ อ็อพชัน no_overwrite สามารถใช้เพื่อปิดใช้งานการ optimization นี้ได้

หากต้องการปิดใช้งานการ optimization นี้ ให้ตั้งค่า MALLOCOPTIONS=no_overwrite ก่อนที่เริ่มต้นการประมวลผล

การเปรียบเทียบนโยบายการจัดสรรต่างๆ

นโยบายการจัดสรร malloc ต่างๆ ที่อธิบายเพิ่มเติมไว้ข้างต้นจะให้ความยืดหยุ่นกับ ผู้พัฒนาแอ็พพลิเคชัน เมื่อใช้หรือรวมวิธีการสนับสนุนเข้าด้วยกันหรือแยกจากกัน ซึ่งเป็นความรับผิดชอบของผู้พัฒนาในการจดจำความต้องการเฉพาะของแอ็พพลิเคชัน และปรับพารามิเตอร์นโยบายการจัดสรรต่างๆ ด้วยวิธีที่ได้รับประโยชน์

การเปรียบเทียบนโยบายการจัดสรรดีฟอลต์และ malloc 3.1

เนื่องจากนโยบายการจัดการ malloc 3.1 จะปัดเศษขึ้นให้เต็มจำนวนของขนาดของการจัดสรรแต่ละครั้ง ที่ร้องขอกำลังถัดไปของ 2 ซึ่งสามารถสร้างความสามารถในการพิจารณาการแตกแฟรกเมนต์หน่วยความจำเสมือน และหน่วยความจำที่ใช้จริง และตำแหน่งของการอ้างอิงที่ไม่มีคุณภาพ นโยบายการจัดสรรที่เป็นค่าดีฟอลต์ เป็นตัวเลือกที่ดีกว่า เนื่องจากนโยบายนี้จะจัดสรรจำนวนของพื้นที่ที่แน่นอนที่ร้องขอ และมีประสิทธิภาพมากกว่าในเรื่องของนำบล็อกของหน่วยความจำที่ใช้แล้วก่อนหน้านี้ กลับคืนมา

แต่น่าเสียดาย แอ็พพลิเคชันโปรแกรมบางตัวอาจได้รับผลกระทบจากนโยบายการจัดสรรแบบ malloc 3.1 สำหรับผลการทำงานที่สามารถยอมรับได้ หรือสำหรับการแก้ไขหน้าที่การทำงาน ตัวอย่างเช่น โปรแกรมที่เป็นส่วนเกินที่จุดสิ้นสุดของอาร์เรย์อาจทำงานไม่ถูกต้อง เมื่อใช้ตัวจัดสรร malloc 3.1 เท่านั้น เนื่องจากพื้นที่เพิ่มเติมถูกจัดเตรียมไว้โดยกระบวนการปัดเศษขึ้นเป็นจำนวนเต็ม โปรแกรมเดียวกันนี้ อาจพบกับลักษณะการทำงานที่ไม่แน่นอน หรือเกิดความล้มเหลว เมื่อใช้กับตัวจัดสรรที่เป็นค่าดีฟอลต์ เนื่องจากตัวจัดสรรที่เป็นค่าดีฟอลต์จะจัดสรรเฉพาะจำนวนไบต์ที่ร้องขอเท่านั้น

สำหรับตัวอย่างอื่น เนื่องจากพื้นที่ที่ไม่เพียงพอสำหรับการเรียกคืนของอัลกอริธึมการจัดสรร malloc 3.1 แอ็พพลิเคชันโปรแกรมจะได้รับพื้นที่ที่ได้ตั้งค่าให้เป็นศูนย์เสมอ (เมื่อการประมวลผลสัมผัสกับเพจที่กำหนดในเซ็กเมนต์การทำงานในครั้งแรก เพจนั้นจะถูกตั้งค่าให้เป็นศูนย์) แอ็พพลิเคชันอาจขึ้นอยู่กับ ผลกระทบนี้สำหรับการแก้ไขการประมวลผล ตามความเป็นจริงแล้ว พื้นที่ที่ถูกจัดสรรที่มีขนาดเป็นศูนย์ จะไม่ถูกระบุฟังก์ชันของรูทีนย่อย **malloc** และจะส่งผลทำให้ผลการทำงานที่ไม่จำเป็นไม่มีประสิทธิภาพสำหรับโปรแกรม ซึ่งจะกำหนดค่าเริ่มต้นตามที่ต้องการและไม่ต้องไม่เป็นศูนย์ เนื่องจาก ตัวจัดสรรดีฟอลต์จะเริ่มใช้พื้นที่ที่นำกลับมาใช้ใหม่ได้มากขึ้น โปรแกรมที่ต้องพึ่งพาการรับหน่วยเก็บที่มีขนาดเป็นศูนย์จาก **malloc** จะเกิดความล้มเหลว เมื่อตัวจัดสรรดีฟอลต์ถูกใช้

เช่นเดียวกัน ถ้าโปรแกรมยังคงจัดสรรโครงสร้างใหม่เพื่อให้มีขนาดใหญ่ขึ้น ตัวจัดสรร malloc 3.1 อาจไม่ต้องการย้ายโครงสร้าง ให้บ่อยขึ้น ในหลายๆ กรณี รูทีนย่อย **realloc** สามารถสร้างการใช้พื้นที่พิเศษที่ได้จัดเตรียมโดยการปัดเศษในอัลกอริธึมของการจัดสรร malloc 3.1 ตัวจัดสรรดีฟอลต์จะย้ายโครงสร้างไปยังพื้นที่ที่มีขนาดใหญ่กว่า เนื่องจากพื้นที่ใกล้เคียงที่มีอยู่ถูกเรียกโดยรูทีนย่อย **malloc** ที่อยู่เหนือโครงสร้างนี้ ซึ่งอาจแสดงถึงการปรากฏให้เห็นของการลดระดับผลการทำงานในรูทีนย่อย **realloc** เมื่อตัวจัดสรรดีฟอลต์ถูกใช้แทนตัวจัดสรร malloc 3.1 ซึ่งในความเป็นจริง จะเป็นการเผชิญหน้ากับต้นทุนที่อยู่ในโครงสร้างของ แอ็พพลิเคชันโปรแกรม

การดีบักดีบักที่มีการจัดการฮีประบบที่ผิด

ระบบย่อย **malloc** นำเสนอการเก็บรวบรวมเครื่องมือในการดีบัก ที่ช่วยให้ผู้พัฒนาแอ็พพลิเคชันดีบัก และแก้ไขข้อผิดพลาดในการจัดการฮีปของโปรแกรม เครื่องมือการดีบักเหล่านี้จะถูกควบคุมผ่านตัวแปรสภาวะแวดล้อม **MALLOCDEBUG**

การสรุปของตัวแปรสภาวะแวดล้อมและอ็อพชัน malloc

ตารางต่อไปนี้จะแสดงความเข้ากันได้ระหว่างตัวแปรสภาวะแวดล้อม **MALLOCTYPE** และ **MALLOCOPTIONS**

ตารางที่ 77. ความเข้ากันได้ระหว่างตัวแปรสภาวะแวดล้อม **MALLOCTYPE** *และ* **MALLOCOPTIONS**

	มัลติฮีป (และอ็อพชันย่อย)	buckets (and suboptions)	แคชของเธรด	ไม่ยอมรับ	no_overwrite
Default Allocator	ใช่	ใช่	ใช่	ใช่	ใช่
3.1	ไม่	ไม่	ใช่	ใช่	ใช่
Watson	ไม่	ไม่	ไม่ใช่	ไม่	ไม่
Watson2	ไม่	ไม่	ไม่	ไม่	ไม่
ผู้ใช้:	ไม่	ไม่	ไม่	ไม่	ใช่

ตารางที่ 78. Compatibility between **MALLOCDEBUG** *and* **MALLOCTYPE** *Environment Variables*

	York Town<Default Allocator>	3.1	Watson	Watson2	ผู้ใช้:
catch_overflow (and suboptions)	ใช่	ไม่ใช่	ใช่	ใช่	ไม่
report_allocations	ใช่	ไม่ใช่	ใช่	ใช่	ไม่ใช่
postfree_checking	ใช่	ไม่	ใช่	ใช่	ไม่
validate_ptrs	ใช่	ไม่	ใช่	ใช่	ไม่ใช่
การติดตาม	ใช่	ไม่ใช่	ใช่	ใช่	ไม่ใช่
บันทึกการทำงาน	ใช่	ไม่ใช่	ใช่	ใช่	ไม่ใช่
ถ้อยคำ	ไม่ใช่	ไม่ใช่	ไม่	ไม่ใช่	ไม่

อ็อพชัน **MALLOCDEBUG** ทั้งหมดทำงานร่วมกันได้และสนับสนุน **MALLOCOPTIONS**

การทำความเข้าใจถึงนโยบายการจัดสรร Watson2

ระบบย่อย Watson2 malloc ปรับใช้กับลักษณะการทำงานของแอ็พพลิเคชัน เมื่อมีการเปลี่ยนแปลงจากเธรดเดี่ยวไปยังเป็นเธรดจำนวนมาก และจากเธรดจำนวนมากไปเป็นเธรดเดี่ยว และใช้กลไกที่ระบุเฉพาะเธรด ซึ่งใช้โครงสร้างฮีปด้วยจำนวนที่แตกต่างกันโดยขึ้นอยู่กับ ลักษณะการทำงานของโปรแกรม ดังนั้นจึงไม่มีอ็อพชันคอนฟิกูเรชัน ที่จำเป็นต้องมี ระบบย่อย Watson2 malloc มี O (logN) ที่หักลบ ต้นทุนต่อการดำเนินการสำหรับเวิร์กโหลดจำนวนมาก เนื่องจากการดำเนินการจำนวนมากสามารถ รันที่เวลาคงที่โดยไม่มีการซิงโครไนซ์

การจัดสรร

การจัดสรรถูกจัดการผ่านชุดของกลไกต่างๆ กลไกเหล่านี้ขึ้นอยู่กับพารามิเตอร์ เช่น จำนวนของเธรดที่แอ็คทีฟ ขนาดของคำร้องขอ และการจัดสรรคืนประวัติและกระบวนการ ชุดของกลไกเข้าถึงได้จากการแคชที่ระบุเฉพาะเธรด และใช้จำนวนฮีปที่เปลี่ยนแปลงได้ ซึ่งมีความใกล้เคียงกันของเธรดกับแผนผัง Double-red-black กับ Page-based coalescing

การจัดสรรคืน

การจัดสรรคืนขึ้นอยู่กับพารามิเตอร์ที่เหมือนกับ ลักษณะการทำงานของการจัดสรร โดยปกติแล้ว บล็อกการส่งคืนจะถูกดักจับใน แคชที่ระบุเฉพาะเธรด ซึ่งอ้างอิงตามความใกล้เคียงกันของฮีปและการใช้ประโยชน์จากความสามารถ หน่วยความจำอาจถูกส่งคืนไปยังหนึ่งในโครงสร้างฮีปจำนวนมาก เมื่อเวลาผ่านไป เนื้อหาจากโครงสร้างฮีปจำนวนมากจะถูกรวบรวมเป็น โครงสร้างฮีปทั่วไปเพื่อปรับปรุง coalescing และลดการแตกแฟรกเมนต์ ฮีป เมื่อต้องการปรับปรุงความทนทานต่อข้อผิดพลาดของแอ็พพลิเคชัน ตัวจัดสรรจะระบุการจัดสรรคืนของตัวชี้ที่ไม่ถูกต้องหรือบล็อกที่ล้มเหลว ไปยังระดับและตัวกรงการดำเนินการเหล่านี้

การจัดสรรใหม่

บล็อกขนาดใหญ่ของหน่วยความจำที่มีเพียงพอจะถูกนำกลับมาใช้ ถ้าบล็อกปัจจุบันไม่สามารถเป็นไปตามคำร้องขอ บล็อกนั้นจะถูกแทนที่ด้วย การจัดสรรคืนปกติและการจัดสรร

ข้อจำกัด

ระบบย่อย Watson2 malloc ปรับใช้กับแอ็พพลิเคชัน และไม่ได้ต้องการอ็อพชันใดๆ เพิ่มเติม แต่ระบบย่อย Watson2 malloc สนับสนุนคุณสมบัติการดีบักต่อไปนี้ ซึ่งถูกควบคุมโดยตัวแปร `MALLOCDEBUG`: `validate_ptrs`, `report_allocations` และ `trace` รายงานที่เกี่ยวข้องกับการจัดสรร สามารถเปลี่ยนทิศทางไปเป็นไฟล์ได้โดยใช้อ็อพชัน `output:<filename>` โปรดดู "เครื่องมือดีบัก malloc" ในหน้า 713 สำหรับข้อมูลโดยละเอียด เกี่ยวกับตัวแปร `MALLOCDEBUG`

หลักการที่เกี่ยวข้อง:

"ภาพรวมพื้นที่แอดเดรสโปรแกรม" ในหน้า 672
Base Operating System จัดเตรียมเซอร์วิส สำหรับโปรแกรมมิงการใช้หน่วยความจำของแอ็พพลิเคชันโปรแกรม

"เครื่องมือดีบัก malloc" ในหน้า 713
การดีบักแอ็พพลิเคชันที่ที่จัดการจัดสรรหน่วยความจำไม่ถูกต้องโดยระบบย่อย **malloc** เป็นเรื่องยากและน่าเบื่อหน่ายได้ นี้เนื่องจาก โดยทั่วไปแล้วไม่มีการทำให้ซิงโครไนซ์กันระหว่างการแทรกของข้อผิดพลาดและ การแสดงออกของอาการผลลัพธ์

"การแทนที่ malloc ที่ผู้ใช้กำหนด" ในหน้า 709
ผู้ใช้สามารถแทนที่ระบบย่อยหน่วยความจำ (รูทีนย่อย **malloc**, **calloc**, **realloc**, **free**, **mallopt** และ **mallinfo**) ด้วยหนึ่งการออกแบบของผู้ใช้เอง

"Malloc multiheap" ในหน้า 721
โดยดีฟอลต์ ระบบย่อย malloc ใช้ฮีปเดียว หรือพูลหน่วยความจำว่าง

"Malloc buckets" ในหน้า 722
Malloc buckets มีส่วนขยายของตัวจัดสรรดีฟอลต์ที่อิงตาม bucket ที่เป็นทางเลือก

"Malloc trace" ในหน้า 726
Malloc Trace จัดให้มีส่วนขยายที่เป็นทางเลือกสำหรับระบบย่อย malloc เพื่อใช้ กับสิ่งอำนวยความสะดวกการติดตาม

"Malloc log" ในหน้า 727
Malloc Log เป็นส่วนขยายทางเลือกของระบบย่อย **malloc** ที่อนุญาตให้ผู้ใช้รับข้อมูลที่เกี่ยวกับการจัดสรรที่แอ็คทีฟขณะนี้ที่จัดโดยกระบวนการการเรียกใช้ ข้อมูลนี้จะถูกนำไปใช้ในการพิจาณาปัญหา และการวิเคราะห์ผลการทำงาน

“Malloc disclaim” ในหน้า 729
Malloc Disclaim เป็นส่วนขยายทางเลือกของระบบย่อย malloc ที่จัดให้ ผู้ใช้มีวิธีเปิดใช้งานการปฏิเสธหน่วยความจำที่ส่งคืนโดยรูทีนย่อย **free** อัตโนมัติ

การแทนที่ malloc ที่ผู้ใช้กำหนด

ผู้ใช้สามารถแทนที่ระบบย่อยหน่วยความจำ (รูทีนย่อย **malloc, calloc, realloc, free, mallopt** และ **mallinfo**) ด้วยหนึ่งการออกแบบของผู้ใช้เอง

หมายเหตุ: การแทนที่ระบบย่อยหน่วยความจำที่เขียนด้วย C++ ไม่ได้รับการสนับสนุน เนื่องจากการใช้ระบบย่อยหน่วยความจำ **libc.a** ใน ไลบรารี C++ **libC.a**

ระบบย่อยหน่วยความจำที่มีอยู่แล้วได้ใช้กับแอ็พพลิเคชันทั้งแบบเธรด และแบบไม่ใช่เธรด ระบบย่อยหน่วยความจำที่ผู้ใช้กำหนดต้องเป็น threadsafe เพื่อให้ทำงานทั้งในกระบวนการที่เป็นเธรด และไม่เป็นเธรด เนื่องจาก ไม่มีการตรวจสอบเพื่อยืนยันว่าเป็น ถ้าโมดูลหน่วยความจำไม่เป็น threadsafe ถูกโหลดในแอ็พพลิเคชันแบบเธรด หน่วยความจำและข้อมูลอาจ เสียหาย

อ็อบเจ็กต์ 32 และ 64 บิตระบบย่อยหน่วยความจำที่ผู้ใช้กำหนดต้องถูกวาง อยู่ในไฟล์เก็บถาวรที่มีอ็อบเจ็กต์ที่แบ่งใช้ 32 บิตชื่อ `mem32.o` และอ็อบเจ็กต์ที่แบ่งใช้ 64 ชื่อ `mem64.o`

อ็อบเจ็กต์ที่แบ่งใช้โดยผู้ใช้ต้องเอ็กซ์พอร์ตสัญลักษณ์ต่อไปนี้:

- __malloc__
- __free__
- __realloc__
- __calloc__
- __mallinfo__
- __mallopt__
- __malloc_init__
- __malloc_prefork_lock__
- __malloc_postfork_unlock__

อ็อบเจ็กต์ที่แบ่งใช้โดยผู้ใช้สามารถเลือกเอ็กซ์พอร์ตสัญลักษณ์ต่อไปนี้:

- __malloc_start__
- __posix_memalign__

การเรียกทำงานจะไม่หยุดถ้าสัญลักษณ์เหล่านี้ไม่มีอยู่

ฟังก์ชันถูกกำหนดดังนี้:

`void *__malloc__(size_t):`
ฟังก์ชันนี้เทียบเท่ากับการใช้รูทีนย่อย **malloc**

`void __free__(void *):`
ฟังก์ชันนี้เทียบเท่ากับการใช้รูทีนย่อย **free**

`void *__realloc__(void *, size_t):`
ฟังก์ชันนี้เทียบเท่ากับการใช้รูทีนย่อย **realloc**

`void *__calloc__(size_t, size_t):`
ฟังก์ชันนี้เทียบเท่ากับการใช้รูทีนย่อย **calloc**

`int __mallopt__(int, int):`
ฟังก์ชันนี้เทียบเท่ากับการใช้รูทีนย่อย **mallopt**

`struct mallinfo __mallinfo__():`
ฟังก์ชันนี้เทียบเท่ากับการใช้รูทีนย่อย **mallinfo**

`void __malloc_start__()`
ฟังก์ชันนี้จะถูกเรียกใช้ทันทีก่อนเรียกใช้ malloc entry point ที่ผู้ใช้กำหนดอื่น

`void __posix_memalign__()`
ฟังก์ชันนี้เทียบเท่ากับการใช้รูทีนย่อย **posix_memalign** ถ้าสัญลักษณ์นี้ไม่มีอยู่ การทำงานจะไม่หยุด แต่การเรียกใช้ไปยังรูทีนย่อย **posix_memalign** จะทำให้เกิดผลลัพธ์ ที่ไม่คาดคิด

ฟังก์ชันต่อไปนี้ถูกใช้โดยระบบย่อยเธรดเพื่อจัดการ ระบบย่อยหน่วยความจำที่ผู้ใช้กำหนดในสภาวะแวดล้อมแบบมัลติเธรด ซึ่งถูกเรียกใช้ต่อเมื่อแอ็พพลิเคชัน และ/หรือโมดูลที่ผู้ใช้กำหนด ถูกโยงกับ **libpthreads.a** แม้ว่าระบบย่อยที่ ผู้ใช้กำหนดไม่ใช่ threadsafe และไม่โยงกับ **libpthreads.a** ฟังก์ชันเหล่านี้ต้องถูกกำหนดและเอ็กซ์พอร์ต มิฉะนั้น อ็อบเจ็กต์จะไม่ถูก โหลด

`void __malloc_init__(void)`
เรียกใช้โดยรูทีนการกำหนดค่าเริ่มต้น pthread ฟังก์ชันนี้ถูกใช้เพื่อ เตรียมข้อมูลเบื้องต้นให้แก่ระบบย่อยหน่วยความจำผู้ใช้แบบเธรด ในกรณีส่วนใหญ่ นี้จะรวม การสร้างและการเตรียมข้อมูลเบื้องต้นรูปแบบของการล็อกข้อมูลบางส่วน แม้ว่าโมดูลระบบย่อยหน่วยความจำ ที่ผู้ใช้กำหนดเองจะถูกโมดูลยงกับ **libpthreads.a** ระบบย่อยหน่วยความจำที่ผู้ใช้กำหนดเอง *ต้อง* ทำงานก่อน `__malloc_init__()` ถูกเรียกใช้

`void __malloc_prefork_lock__(void)`
เรียกใช้โดย pthreads เมื่อรูทีนย่อย `fork` ถูกเรียกใช้ ฟังก์ชันนี้ถูกใช้เพื่อให้แน่ใจว่าระบบย่อยหน่วยความจำอยู่ในสถานะที่ทราบก่อน `fork()` และยังอยู่เช่นนั้น จนกระทั่ง `fork()` ส่งค่ากลับ ในกรณีส่วนใหญ่ นี้รวมถึงการจัดเตรียมการล็อกระบบย่อยหน่วยความจำ

`void __malloc_postfork_unlock__(void)`
เรียกใช้โดย pthreads เมื่อรูทีนย่อย `fork` ถูกเรียกใช้ ฟังก์ชันนี้ถูกใช้เพื่อทำให้ระบบย่อยหน่วยความจำพร้อมใช้งานใน พาเรนต์และชายน์หลังการ `fork` นี้ควร เลิกทำงานที่ทำเสร็จโดย `__malloc_prefork_lock__` ในกรณีส่วนใหญ่ นี้รวมถึงการรีลีสการล็อกระบบย่อยหน่วยความจำ

ฟังก์ชันทั้งหมดต้องถูกเอ็กซ์พอร์ตจากโมดูลที่แบ่งใช้โมดูลแยก ต้องมีอยู่สำหรับการใช้ 32 และ 64 บิตโดยเก็บในไฟล์เก็บถาวร ตัวอย่างเช่น:

- โมดูล **mem.exp**:

```
__malloc__
__free__
__realloc__
__calloc__
__mallopt__
__mallinfo__
```

```
__malloc_init__
__malloc_prefork_lock__
__malloc_postfork_unlock__
__malloc_start__
```

- โมดูล **mem_functions32.o** :
  มีฟังก์ชัน 32 บิต ที่จำเป็นทั้งหมด
- โมดูล **mem_functions64.o** :
  มีฟังก์ชัน 64 บิต ที่จำเป็นทั้งหมด

ตัวอย่างต่อไปนี้ใช้สำหรับการสร้างอ็อบเจ็กต์ที่แบ่งใช้ พารามิเตอร์ `-lpthreads` จำเป็นต่อเมื่ออ็อบเจ็กต์ใช้ฟังก์ชัน pthread เท่านั้น

- การสร้างอ็อบเจ็กต์ที่แบ่งใช้ 32 บิต:

```
ld -b32 -m -o mem32.o mem_functions32.o \
-bE:mem.exp \
-bM:SRE -lpthreads -lc
```

- การสร้างอ็อบเจ็กต์ที่แบ่งใช้ 64 บิต:

```
ld -b64 -m -o mem64.o mem_functions64.o \
-bE:mem.exp \
-bM:SRE -lpthreads -lc
```

- การสร้างไฟล์เก็บถาวร(ชื่ออ็อบเจ็กต์ที่แบ่งใช้ต้องเป็น `mem32.o` สำหรับอ็อบเจ็กต์ 32 บิต และ `mem64.o` สำหรับ อ็อบเจ็กต์ 64 บิต):

```
ar -X32_64 -r archive_name mem32.o mem64.o
```

การเปิดใช้งานระบบย่อยหน่วยความจำที่ผู้ใช้กำหนด

ระบบย่อยหน่วยความจำที่ผู้ใช้กำหนดเองสามารถถูกเปิดใช้งานโดยใช้หนึ่งในค่าต่อไปนี้:

- ตัวแปรสภาวะแวดล้อม **MALLOCTYPE**
- ตัวแปรโกลบอล `_malloc_user_defined_name` ในแอ็พพลิเคชันของผู้ใช้

ในการใช้ตัวแปรสภาวะแวดล้อม **MALLOCTYPE** ไฟล์เก็บถาวร ที่มีระบบย่อยหน่วยความจำที่ผู้ใช้กำหนดถูกระบุโดยการตั้งค่า **MALLOCTYPE** เป็น `user:`*archive_name* โดยที่ *archive_name* อยู่ใน `libpath` ของแอ็พพลิเคชัน หรือพาธถูกระบุในตัวแปรสภาวะแวดล้อม **LIBPATH**

ในการใช้ตัวแปรโกลบอล `_malloc_user_defined_name` แอ็พพลิเคชันของผู้ใช้ต้องประกาศตัวแปรโกลบอลดังนี้:

```
char *_malloc_user_defined_name="archive_name"
```

โดยที่ *archive_name* ต้องอยู่ใน libpath ของแอ็พพลิเคชันหรือพาธที่ระบุในตัวแปรสภาวะแวดล้อม **LIBPATH**

หมายเหตุ:

1. เมื่อแอ็พพลิเคชัน setuid รัน ตัวแปรสภาวะแวดล้อม **LIBPATH** จะถูกละเว้นดังนั้นไฟล์เก็บถาวรต้องอยู่ใน libpath ของแอ็พพลิเคชัน
2. *archive_name* ไม่สามารถมีข้อมูลพาธ

3. เมื่อทั้งตัวแปรสภาวะแวดล้อม **MALLOCTYPE** และ ตัวแปรโกลบอล `_malloc_user_defined_name` ถูกใช้เพื่อระบุ *archive_name* ไฟล์เก็บถาวร ที่ระบุโดย **MALLOCTYPE** จะแทนที่ค่าที่ระบุ โดย `_malloc_user_defined_name`

ข้อควรพิจารณาสำหรับ 32 บิตและ 64 บิต

ถ้าไฟล์เก็บถาวรไม่มีทั้งอ็อบเจ็กต์ที่แบ่งใช้ 32 บิตและ 64 บิต และระบบย่อยหน่วยความจำที่ผู้ใช้กำหนดเองถูกเปิดใช้งานโดยใช้ตัวแปรสภาวะแวดล้อม **MALLOCTYPE** จะเกิดปัญหาในการทำงานการประมวลผล 64 บิต จากแอ็พพลิเคัน 32 บิตและการประมวลผล 32 บิตจากแอ็พพลิเคชัน 64 บิต เมื่อการประมวลผลใหม่ถูกสร้างโดยใช้รูทีนย่อย **exec** การประมวลผลจะสืบทอดสภาพแวดล้อมของแอ็พพลิเคชันที่เรียกใช้ นี้หมายความ ว่าตัวแปรสภาวะแวดล้อม **MALLOCTYPE** จะถูกสืบทอด และการประมวลผลใหม่จะพยายามโหลดระบบย่อยหน่วยความจำที่ผู้ใช้กำหนดเอง ถ้าสมาชิกไฟล์เก็บถาวรไม่มีอยู่สำหรับโปรแกรมประเภทนี้ การโหลดจะ ล้มเหลวและการกระบวนการใหม่จะออกจากการทำงาน

ข้อความพิจารณาสำหรับเธรด

ฟังก์ชันที่ระบุทั้งหมดต้องทำงานในสภาวะแวดล้อมแบบมัลติเธรด แม้ว่าโมดูลจะถูกลิงก์กับ libpthreads.a อย่างน้อย `__malloc__()` *ต้อง* ทำงานก่อน `__malloc_init__()` ถูกเรียกใช้และ pthreads ถูกเตรียมข้อมูลเบื้องต้น นี้จำเป็น เนื่องจากการกำหนดค่าเริ่มต้น pthread ต้องการ `malloc()` ก่อน `__malloc_init__()` ถูกเรียกใช้

ฟังก์ชันหน่วยความจำที่มีทั้งหมดต้องทำงานทั้งในสภาวะแวดล้อมแบบเธรด และแบบไม่มีเธรด ฟังก์ชัน `__malloc__()` ควรสามารถรันได้เสร็จสมบูรณ์โดยไม่มีการขึ้นต่อกันใดๆ บน `__malloc_init__()` (นั่นคือ `__malloc__()` ควร ถือว่า `__malloc_init__()` ยัง *ไม่*ได้รันอยู่) หลังจาก `__malloc_init__()` ดำเนินการเสร็จสมบูรณ์ `__malloc__()` สามารถขึ้นกับ การทำงานใดๆ ที่สำเร็จโดย `__malloc_init__()` นี้จำเป็น เนื่องจากการกำหนดค่าเริ่มต้น pthread ใช้ `malloc()` ก่อน `__malloc_init__()` ถูกเรียกใช้

ตัวแปรต่อไปนี้ถูกจัดให้มีเพื่อป้องกันมิให้เรียกใช้รูทีนที่ เกี่ยวข้องกับเธรด:

- ตัวแปร `__multi_threaded` เป็นศูนย์จนกว่า เธรดถูกสร้างขึ้นเมื่อเปลี่ยนเป็นค่าไม่ใช่ศูนย์ และจะไม่ตั้งค่าใหม่เป็นศูนย์สำหรับการประมวลผลนั้น
- ตัวแปร `__n_pthreads` เป็น `-1` จนกว่า pthreads ถูกเตรียมข้อมูลเบื้องต้นเมื่อถูกตั้งค่าเป็น `1` จากณ จุดนั้นค่าจะเป็นจำนวนนับของจำนวนเธรดที่แอ็คทีฟ

ตัวอย่าง:

ถ้า `__malloc__()` ใช้ `pthread_mutex_lock()` โค้ดอาจมีหน้าตาคล้ายโค้ดต่อไปนี้:

```
if (__multi_threaded)
pthread_mutex_lock(mutexptr);

/* ..... work ....... */

if (__multi_threaded)
pthread_mutex_unlock(mutexptr);
```

ในตัวอย่างนี้ `__malloc__()` ถูกห้าม มิให้เรียกใช้งานฟังก์ชัน pthread ก่อนที่ pthreads จะถูกเตรียมข้อมูลเบื้องต้นครบถ้วน แอ็พพลิเคชัน แบบเธรดเดียวยังเพิ่มขึ้นเนื่องจากการล็อกไม่ถูกทำจนกว่าเธรดที่สอง เริ่มทำงาน

ข้อจำกัด

ระบบย่อยหน่วยความจำที่เขียนด้วย C++ ไม่ได้รับการสนับสนุนเนื่องจากการกำหนดค่าเริ่มต้น และการขึ้นต่อกันของระบบย่อยหน่วยความจำ **libC.a** และ **libc.a**

ข้อความแสดงความผิดพลาดไม่ถูกแปลเนื่องจากรูทีนย่อย **setlocale** ใช้ `malloc()` เพื่อเตรียมข้อมูลเบื้องต้น locales ถ้า `malloc()` ล้มเหลว ดังนั้นรูทีนย่อย **setlocale** จะไม่สามารถเสร็จได้ และแอ็พพลิเคชันยังคงอยู่ใน POSIX locale ดังนั้น จะแสดงเฉพาะข้อความภาษาอังกฤษ ค่าดีฟอลต์เท่านั้น

โปรแกรมที่สร้างข้อมูลสถิติที่มีอยู่ไม่สามารถใช้ระบบย่อยหน่วยความจำที่ผู้ใช้กำหนดเอง โดยไม่ทำการคอมไพล์ใหม่

การรายงานข้อผิดพลาด

ครั้งแรกที่รูทีนย่อย **malloc** ถูกเรียกใช้ อ็อบเจ็กต์ 32 หรือ 64 บิตในไฟล์เก็บถาวรที่ระบุโดยตัวแปรสภาวะแวดล้อม **MALLOCTYPE** จะถูกโหลด ถ้าการโหลดล้มเหลว จะมีข้อความแสดง และแอ็พพลิเคชันออกจากการทำงาน ถ้าการโหลดสำเร็จ จะพยายามทำการตรวจสอบ ว่ามีสัญลักษณ์ที่ต้องการทั้งหมดแสดงอยู่ ถ้ามีสัญลักษณ์ใดขาดหาย แอ็พพลิเคชันจะถูกยกเลิกการทำงานและรายการสัญลักษณ์ที่หายจะแสดงให้ทราบ

หลักการที่เกี่ยวข้อง:

"การจัดสรรหน่วยความจำระบบโดยใช้ระบบย่อย malloc" ในหน้า 696
หน่วยความจำจะถูกจัดสรรให้กับแอ็พพลิเคชัน โดยใช้ระบบย่อย **malloc**

เครื่องมือดีบัก malloc

การดีบักแอ็พพลิเคชันที่ที่จัดการจัดสรรหน่วยความจำไม่ถูกต้องโดยระบบย่อย **malloc** เป็นเรื่องยากและน่าเบื่อหน่ายได้ นี้เนื่องจาก โดยทั่วไปแล้วไม่มีการทำให้ซิงโครไนซ์กันระหว่างการแทรกของข้อผิดพลาดและ การแสดงออกของอาการผลลัพธ์

การเพิ่มขึ้นของความยาก คือความซับซ้อนในการสืบทอดของการจัดสรรหน่วยความจำ ที่กำลังมีการจัดการนับพันที่ถูกทำเลิกทำ และเข้าถึง (บางครั้ง) แบบอะซิงโครนัส และแบบซิงโครนัส ทั้งหมดภายในบริบทมัลติเธรด ที่จำเป็นต้องมีการซิงโครไนซ์ที่ทนทาน และมีประสิทธิภาพ

ไม่ว่าด้วยเหตุใด โฟกัสของเครื่องมือการดีบักของเราจะเน้นที่การเลื่อนเวลา ตรวจพบอาการให้เข้าใกล้เวลาที่เกิดการแทรกข้อผิดพลาดเข้ามามากขึ้นเป็นหลัก นี้จะช่วยผู้พัฒนาแอ็พพลิเคชันสามารถระบุได้แม่นยำขึ้นว่าส่วนใดของโค้ดที่เกี่ยวข้องกับการเกิดข้อผิดพลาดนี้

เครื่องมือการดีบักที่แตกต่างกันหลายๆ ตัวได้รับการพัฒนาให้ใช้กับ malloc บางตัวสามารถใช้ร่วมกับเครื่องมือการดีบักอื่นๆ และกับนโยบายการจัดสรรทั้งหมด ขณะที่เครื่องมืออื่นๆ มีข้อจำกัดการใช้งานมากกว่า เครื่องมือดีบักหลายๆ ตัว ใช้รีซอร์สเพิ่มจากที่จำเป็นต้องใช้โดยการประมวลผล ซึ่งจะขึ้นอยู่กับผู้พัฒนาแอ็พพลิเคชันที่จะต้องจัดให้มีรีซอร์สที่เพียงพอเมื่อจำเป็น

ข้อควรพิจารณาด้านผลการทำงาน

เครื่องมือ debug malloc ไม่เหมาะสำหรับการใช้งานเต็มเวลา เป็นประจำ หรือ ใช้ในทั้งระบบ แม้ว่าเครื่องมือจะถูกออกแบบให้กระทบต่อผลการทำงานน้อยที่สุด เมื่อแอ็พพลิเคชันถูกดีบัก การส่งผลเชิงลบอย่างยิ่งต่อปริมาณงานของระบบโดยรวม อาจส่งผลได้ถ้าใช้ในระบบเป็นวงกว้าง โดยเฉพาะ การตั้งค่า`MALLOCDEBUG=catch_overflow` ในไฟล์ **/etc/environment** ไม่แนะนำให้ทำ และจะทำให้เกิดปัญหาต่อระบบ อย่างยิ่ง เช่นการใช้พื้นที่การสลับหน้ามากเกิน เครื่องมือ debug malloc ควรใช้เฉพาะเพื่อดีบักแอ็พพลิเคชันเดี่ยว หรือกลุ่มแอ็พพลิเคชันขนาดเล็ก ในเวลาเดียวกัน

เนื่องจากการทำงานพิเศษเกี่ยวข้องกับกาตรวจสอบตอนรันไทม์แตกต่างกัน ระบบย่อย **malloc** จะมีผลการทำงานลดลงอย่างเห็นได้เมื่อเปิดใช้เครื่องมือ debug malloc (ขึ้นอยู่กับว่าใช้เครื่องมือใด) แต่ไม่ใช่ประเด็น ที่แอ็พพลิเคชันจะไม่สามารถใช้งานได้ หลังจากปัญหา ได้รับการแก้ไข เครื่องมือ debug malloc ควรถูกปิดทำงานเพื่อเรียกคืนผลการทำงานระบบย่อย **malloc**

ข้อควรพิจารณาเกี่ยวกับดิสก์และหน่วยความจำ

โดยมีเครื่องมือ `catch_overflow` หรือ Malloc Log ถูกเปิดใช้งาน ระบบย่อย **malloc** จะใช้หน่วยความจำเพิ่มขึ้นอย่างเห็นได้ชัด

สำหรับ `catch_overflow` แต่ละครั้งของการร้องขอ malloc จะถูกเพิ่มขนาด 4096 + 2 เท่าของค่า unsigned long จากนั้นปัดเศษเป็นผลคูณค่าถัดไป ของแมโคร PAGESIZE `catch_overflow` จะพิสูจน์ว่ามีความอ่อนไหวต่อหน่วยความจำ มากเกินไปในการใช้สำหรับแอ็พพลิเคชันขนาดใหญ่มาก แต่สำหรับแอ็พพลิเคชันส่วนใหญ่ที่ จำเป็นต้องมีการดีบักหน่วยความจำ การใช้หน่วยความจำเพิ่มพิเศษไม่ควรส่งผลให้เกิดปัญหา สำหรับแอ็พพลิเคชันสำหรับ การใช้อ็อพชัน **debug_range** และ **functionset** ใน `catch_overflow` สามารถลดการใช้งาน หน่วยความจำลงอย่างมาก ทำให้สามารถดีบักโปรแกรมทีละส่วนได้

สำหรับ Malloc Logเร็กคอร์ดการจัดสรรถูกเก็บสำหรับทุกการจัดสรรที่แอ็คทีฟ ในการประมวลผล โอเวอร์เฮดหน่วยความจำนี้สามารถลดลงได้โดยการระบุ จำนวนของตัวชี้สแต็กที่บันทึกให้น้อยลง

ถ้าแอ็พพลิเคชันที่กำลังดีบักมีการเรียกใช้รูทีนการจัดสรรระบบย่อย **malloc** บ่อย อาจทำให้เกิดปัญหาการใช้งานหน่วยความจำ ที่มีการเปิดใช้งานเครื่องมือ debug malloc ที่อาจทำให้แอ็พพลิเคชันไม่สามารถทำงาน ได้อย่างเหมาะสมในเซ็กเมนต์เดี่ยว ถ้าเกิดปัญหานี้ อาจเป็นประโยชน์ที่จะเปิดให้ แอ็พพลิเคชันเข้าถึงหน่วยความจำเพิ่มเติมโดยการใช้คำสั่ง **ulimit** และอ็อพชัน **-bmaxdata** ของคำสั่ง **ld**

สำหรับวัตถุประสงค์ของการรันโดยเปิดใช้เครื่องมือ debug malloc ให้ตั้งค่า ulimit สำหรับตัวแปรทั้งข้อมูล (-d) และสแต็ก (-s) ดังนี้:

```
ulimit -d unlimited
ulimit -s unlimited
```

ในการสำรองจำนวนสูงสุด 8 เซ็กเมนต์สำหรับการกระบวนการ 32 บิต อ็อพชัน **-bmaxdata** ควรถูกระบุเป็น `-bmaxdata:0x80000000`

เมื่อปิดใช้เครื่องมือ debug malloc ค่าดีฟอลต์สำหรับ ulimit และ -bmaxdata สามารถเรียกคืนได้

สำหรับข้อมูลเพิ่มเติมเกี่ยวกับคำสั่ง **ulimit** และ อ็อพชัน **-bmaxdata** โปรดดูที่ Large Program Support

เครื่องมือ debug malloc ไม่เหมาะสำหรับการใช้งานในสถานการณ์การดีบักบางอย่าง เนื่องจากเครื่องมือ debug malloc บางตัวต้องใช้โอเวอร์เฮดของหน้าอย่างน้อยหนึ่งหน้า ต่อการจัดสรร โปรแกรมที่ออกการจัดสรรการจัดสรรหลายๆ ครั้งจะพบว่า การใช้งานหน่วยความจำของตนเพิ่มขึ้นอย่างรวดเร็ว โปรแกรมเหล่านี้อาจพบปัญหา ความล้มเหลวใหม่ขณะการร้องขอการจัดสรรหน่วยความจำถูกปฏิเสธเนื่องจากไม่มี หน่วยความจำ หรือพื้นที่การสลับหน้า ความล้มเหลวเหล่านี้เป็นข้อผิดพลาดในโปรแกรมที่ ไม่จำเป็นต้องทำการดีบัก และไม่ใช่ข้อผิดพลาดในเครื่องมือ debug malloc

ตัวอย่างเฉพาะตัวอย่างหนึ่งของปัญหานี้คือ X server ซึ่งออกการร้องขอการจัดสรร ขนาดเล็กจำนวนมากระหว่างการกำหนดค่าเริ่มต้นและการดำเนินการ ความพยายามใดๆ ที่จะรัน X server โดยใช้คำสั่ง X หรือ xinit ด้วยการเปิดใช้ catch_overflow จะทำให้เกิดความล้มเหลวที่ X server เนื่องจากไม่มีหน่วยความจำเพียงพอ อย่างไรก็ตาม สามารถดีบัก X ในแบบส่วนย่อยได้

โดยใช้อ็อพชัน debug_range หรือ functionset ไคลเอ็นต์ X โดยทั่วไปจะไม่พบปัญหาการทำงาน ในการรันที่เปิดใช้งาน catch_overflow ในการใช้ catch_overflow บน โปรแกรมไคลเอ็นต์ X ใช้ขั้นตอนต่อไปนี้:

1. เริ่มทำงาน X server โดยปิดทำงาน catch_overflow
2. เริ่มทำงานหน้าต่างเทอร์มินัล (ตัวอย่างเช่น dtterm, xterm, aixterm)
3. ตั้งค่าตัวแปรสภาวะแวดล้อมที่เหมาะสมภายในเซสชันหน้าต่างเทอร์มินัล เพื่อเปิดใช้ catch_overflow
4. เรียกใช้โปรแกรมไคลเอ็นต์ X ที่จะถูกดีบักจากในหน้าต่างเดียวกัน

การเปิดใช้งานการดีบัก malloc

Debug Malloc ไม่ถูกเปิดใช้งานเป็นค่าดีฟอลต์ แต่ถูกเปิดใช้งานและตั้งค่าโดย การตั้งค่าตัวแปรสภาวะแวดล้อม **MALLOCDEBUG** เป็น อ็อพชันที่เหมาะสม ถ้าจำเป็นต้องใช้มากกว่าหนึ่งอ็อพชัน สามารถคั่นอ็อพชันด้วย เครื่องหมายจุลภาค (,) อ็อพชันที่ร้องขอใน tandem ต้องเข้ากันได้ซึ่งกันและกัน

หมายเหตุ: เมื่อต้องการปิดใช้งานการดีบัก malloc ให้ยกเลิกการตั้งค่าตัวแปรสภาวะแวดล้อม MALLOCDEBUG โดยใช้คำสั่ง **unset MALLOCDEBUG**

เครื่องมือการดีบัก Malloc

เครื่องมือการดีบัก malloc ต่อไปนี้มีใช้ได้:

- การตรวจหาบัฟเฟอร์โอเวอร์โฟลว์
  - align
  - override_signal_handling
  - debug_range
  - functionset
  - allow_overreading
  - postfree_checking
- Malloc Trace
- Malloc Log
  - report_allocations
  - validate_ptrs
- Malloc Detect
  - ถ้อยคำ
  - checkarena
  - output
  - continue
- Malloc debug Fill

การตรวจหาบัฟเฟอร์โอเวอร์โฟลว์

บางครั้งข้อผิดพลาดการจัดการหน่วยความจำมีสาเหตุจากแอ็พพลิเคชันโปรแกรม เขียนลงในพื้นที่ที่เลยจุดสิ้นสุดของบัฟเฟอร์ที่จัดสรร เนื่องจากความผิดพลาดนี้ยังไม่ส่งผลตามมา ในทันที จึงไม่เกิดอาการของปัญหาจนกว่าอีกนานในภายหลังเมื่อหน่วยความจำที่ถูกเขียนทับ (โดยปกติเป็นของการจัดสรรอื่น) ถูกอ้างถึง และไม่มีข้อมูล ที่ถูกเก็บไว้ตอนแรกให้เรียกใช้ได้อีกต่อไป

อ็อพชันการดีบัก catch_overflow มีอยู่เพื่อให้ผู้ใช้ระบุ การเขียนทับหน่วยความจำ การอ่านทับ การปล่อยฟรีซ้ำ และการใช้หน่วยความจำที่ว่าง ที่จัดสรรโดยรูทีนย่อย **malloc** ปัญหาหน่วยความจำที่ตรวจพบโดย เครื่องมือ catch_overflow ส่งผลให้ยกเลิกการเรียกใช้ หรือการละเมิดการแบ่งเป็นเซ็กเมนต์ (**SIGSEGV**) ในกรณีส่วนใหญ่ เมื่อตรวจพบข้อผิดพลาด แอ็พพลิเคชันจะหยุดทำงานทันทีและสร้างไฟล์ core

อ็อพชัน catch_overflow ส่งผลต่อการจัดสรรของนโยบายและอ็อพชัน การจัดสรรต่อไปนี้:

- Default Allocation Policy
- Watson Allocation Policy
- Malloc Multiheap Option
- Malloc Threadcache Option
- Malloc Disclaim Option

อ็อพชันการดีบัก catch_overflow ถูกเปิดใช้งานโดยการตั้งค่า MALLOCDEBUG=catch_overflow นี้จะเปิดทำงาน identification ของการเขียนทับ และการอ่านทับหน่วยความจำ

align

โดยดีฟอลต์ รูทีนย่อย **malloc** ส่งกลับตัวชี้ ที่จัดตำแหน่งบนขอบเขต 2 word นี้จำเป็นสำหรับความสอดคล้องกันมาตรฐาน และสำหรับโปรแกรมซึ่งไม่สามารถยอมรับการเข้าถึงหน่วยความจำที่ไม่กำหนดตำแหน่ง (เช่น โปรแกรม ที่ใช้คอมโพเนนต์ DCE) อย่างไรก็ตามเนื่องจากการเปลี่ยนแปลงในการนำอ็อพชัน catch_overflow ไปใช้ ทำให้โปรแกรมอาจเขียนทับลงในบัฟเฟอร์ด้วยจำนวนที่น้อยกว่าค่าการจัดตำแหน่งโดยไม่ถูกตรวจพบโดย catch_overflow อ็อพชัน align สามารถใช้เพื่อบอกระบบย่อย **malloc** ให้เพิกเฉยการจัดตำแหน่งดีฟอลต์นี้ เพื่อลดหรือกำจัดจำนวนไบต์ที่บัฟเฟอร์สามารถถูก เขียนทับโดยไม่มีการตรวจพบ โดยสามารถระบุการจัดตำแหน่งแบบกำหนดเองสำหรับ ผลคูณของสองระหว่าง 0 ถึง 4096 (เช่น 0,1,2,4,...). ค่า 0 และ 1 ถูกถือว่าเหมือนกัน คือไม่มีการจัดตำแหน่งหน่วยความจำ ดังนั้น การเข้าถึงหน่วยความจำใดๆ ที่เลยพื้นที่ที่จัดสรรจะก่อให้เกิด SEGFAULT

อ็อพชัน align เป็นส่วนหนึ่งของอ็อพชัน catch_overflow และจะมีความหมายเมื่อเปิดใช้งาน catch_overflow เท่านั้น To enable a non-default alignment, set the **MALLOCDEBUG** environment variable as follows:

```
MALLOCDEBUG=catch_overflow,align:n
```

โดยที่ *n* คือการจัดตำแหน่งที่ต้องการ

ในการคำนวณจำนวนไบต์ที่อ่านทับหรือเขียนทับ อ็อพชัน catch_overflow จะอนุญาตให้การร้องขอการจัดสรรที่กำหนดเมื่อ n คือการจัดตำแหน่ง ที่ร้องขอ และ size คือจำนวนไบต์ที่จะถูกจัดสรร ใช้สูตร ต่อไปนี้:

```
((((size / n) + 1) * n) - size) % n
```

ตัวอย่างต่อไปนี้แสดงผลของอ็อพชัน align ต่อ ความสามารถของแอ็พพลิเคชันที่จะทำการอ่านทับ หรือเขียนทับที่อ็อพชัน catch_overflow ถูกเปิดใช้งาน ในตัวอย่างนี้ อ็อพชัน align ถูกระบุด้วยค่า 2:

```
MALLOCDEBUG=align:2,catch_overflow
```

อ็อพชัน `catch_overflow` จัดการการอ่านทับและการเขียนทับ ดังนี้:

- เมื่อจำนวนไบต์คู่ถูกจัดสรร malloc จะจัดสรรตามจำนวนไบต์ที่ร้องขอ จริงๆ ซึ่งจะมีการอ่านทับหรือเขียนทับ 0 ไบต์
- เมื่อจำนวนไบต์คี่ถูกจัดสรร malloc จะจัดสรรตามจำนวนไบต์ ที่ร้องขอ บวกเพิ่มหนึ่งไบต์เพื่อให้เป็นไปตามการจัดตำแหน่งที่บังคับ ซึ่งอนุญาตให้อ่านทับหรือเขียนทับได้ 1 ไบต์

override_signal_handling

อ็อพชัน `catch_overflow` รายงานรายงานด้วยวิธีใดวิธีหนึ่ง ต่อไปนี้:

- ข้อผิดพลาดการเข้าถึงหน่วยความจำ (เช่นพยายามอ่านหรือเขียนเลยจากจุดสิ้นสุด ของหน่วยความจำที่จัดสรร) ทำให้เกิดการละเมิดการแบ่งเป็นเซ็กเมนต์(SIGSEGV) ส่งผลให้เกิด core dump
- สำหรับข้อผิดพลาดประเภทอื่นๆ (เช่นพยายามปล่อยฟรีพื้นที่ที่ฟรี อยู่แล้ว) อ็อพชัน `catch_overflow` จะส่งเอาต์พุตเป็นข้อความแสดงความผิดพลาด ที่เรียกใช้ฟังก์ชันการเลิกทำงาน ซึ่งจะส่งสัญญาณ SIGIOT เพื่อสิ้นสุด การประมวลผลปัจจุบัน

ถ้าโปรแกรมที่เรียกใช้กำลังบล็อกหรือได้รับสัญญาณ SIGSEGV และ SIGIOT อ็อพชัน `catch_overflow` จะถูกกันมิให้รายงาน ข้อผิดพลาด อ็อพชัน `override_signal_handling` จัดให้มีวิธี ข้ามสถานการณ์นี้โดยไม่ต้องมีการบันทึกและการสร้างแอ็พพลิเคชันใหม่

ถ้าอ็อพชัน `override_signal_handling` ถูกระบุ อ็อพชัน `catch_overflow` จะดำเนินการต่อไปนี้เมื่อมีการเรียกใช้แต่ละครั้ง ไปที่รูทีนระบบย่อย **malloc** :

1. ปิดใช้งานตัวจัดการสัญญาณที่มีอยู่ ที่ตั้งค่าโดยแอ็พพลิเคชันสำหรับ SIGSEGV หรือ SIGIOT
2. ตั้งค่าการดำเนินการสำหรับทั้ง SIGIOT และ SIGSEGV เป็นค่าดีฟอลต์ (SIG_DFL)
3. เลิกการบล็อกทั้ง SIGIOT และ SIGSEGV

ถ้าตัวจัดการสัญญาณแอ็พพลิเคชันแก้ไขการดำเนินการสำหรับ SIGSEGV ระหว่าง การเรียกใช้รูทีนการจัดสรรหน่วยความจำ และจากนั้นพยายามเข้าถึงหน่วยความจำที่ใช้ไม่ได้ อ็อพชัน `catch_overflow` จะไม่สามารถรายงาน ข้อผิดพลาดได้ (แอ็พพลิเคชันจะไม่ออกจากการทำงาน และจะไม่มีการสร้าง core file)

หมายเหตุ:

1. อ็อพชัน `override_signal_handling` อาจไม่สามารถใช้ได้ใน สภาพแวดล้อมแอ็พพลิเคชันแบบเธรด เนื่องจากอ็อพชัน `catch_overflow` ใช้รูทีนย่อย **sigprocmask** และการประมวลผลแบบเธรดจำนวนมาก ใช้รูทีนย่อย **pthread_sigmask**
2. ถ้าเธรดเรียกใช้รูทีนย่อย **sigwait** โดยไม่รวม SIGSEGV และ SIGIOT ในสัญญาณ และในภายหลังอ็อพชัน `catch_overflow` ตรวจพบข้อผิดพลาด เธรดจะหยุดทำงาน เนื่องจาก อ็อพชัน `catch_overflow` สามารถสร้าง SIGSEGV หรือ SIGIOT เท่านั้น
3. ถ้าตัวชี้ไปยังหน่วยความจำที่ไม่ถูกต้องถูกส่งไปที่รูทีนในเคอร์เนล รูทีน ในเคอร์เนลจะล้มเหลว และส่วนใหญ่จะส่งกลับค่าที่มี errno ตั้งค่าเป็น EFAULT ถ้าแอ็พพลิเคชัน ไม่ได้ตรวจสอบค่าที่ส่งกลับบจากการเรียกใช้ระบบ ข้อผิดพลาดนี้อาจไม่ถูกตรวจพบ

debug_range

โดยดีฟอลต์ ถ้าอ็อพชัน catch_overflow ถูกเปิดเปิดใช้งาน การตรวจหา บัฟเฟอร์โอเวอร์โฟลว์จะถูกดำเนินการสำหรับทุกการจัดสรรในโปรแกรม ถ้าอ็อพชัน debug_range ถูกระบุ การร้องขอการจัดสรรที่อยู่ในช่วงระหว่างขนาดสูงสุดและ ตำแหน่งที่ผู้ใช้กำหนดเท่านั้นที่จะถูกตรวจพบบัฟเฟอร์โอเวอร์โฟลว์โดยอ็อพชัน catch_overflow มิฉะนั้น จะไม่มีการตรวจหาบัฟเฟอร์โอเวอร์โฟลว์ อ็อพชันนี้อนุญาตให้ผู้ใช้ควบคุมจำนวน รีซอร์สหน่วยความจำพิเศษที่ถูกใช้โดยอ็อพชัน catch_overflow โดยการใช้เครื่องมือเฉพาะในกรณีที่เจาะจงเท่านั้น

อ็อพชัน debug_range มีความสำคัญในบริบทของ อ็อพชัน catch_overflow เท่านั้น โดยถูกเปิดใช้งานดังนี้:

```
MALLOCDEBUG=catch_overflow,debug_range:min:max
```

โดยที่ *min* คือขีดจำกัดล่างและ *max* คือขีดจำกัดบนของช่วงซึ่งการตรวจหาบัฟเฟอร์โอเวอร์โฟลว์ จะถูกดำเนินการ ถ้า 0 ถูกระบุเป็นค่าต่ำสุด ดังนั้นค่าใดๆ ที่ น้อยกว่าค่าสูงสุดจะถูกทำการตรวจหาบัฟเฟอร์โอเวอร์โฟลว์ ถ้า 0 ถูกระบุเป็นค่าสูงสุด ดังนั้นค่าใดๆ ที่ มากกว่าค่าต่ำสุดจะถูกทำการตรวจหาบัฟเฟอร์โอเวอร์โฟลว์

ข้อจำกัด

เนื่องจากข้อกำหนดของการนำไปใช้ภายใน แต่ละการจัดสรรจะยังคง จำเป็นต้องใช้ขนาดหน้าที่มีความยาวอย่างน้อยหนึ่งหน้า ดังนั้นอ็อพชัน debug_range แทบจะลดการใช้โอเวอร์เฮดของอ็อพชัน catch_overflow เท่านั้น มากกว่าการลบออก

ถ้ารูทีนย่อย **realloc** ถูกเรียกใช้ด้วยการร้องขอการจัดสรร ที่อยู่ภายในช่วงที่ผู้ใช้ระบุ จะทำการตรวจหาบัฟเฟอร์โอเวอร์โฟลว์ แม้ว่า การจัดสรรเริ่มแรกจะไม่อยู่ภายในช่วง ที่ระบุ ในทางกลับกันของเงื่อนไขนี้เป็นจริงเช่นกัน

หมายเหตุ: ถ้าอ็อพชัน override_signal ถูกตั้งค่าร่วมกับ อ็อพชัน debug_range การยกเลิกลักษณะการทำงานของสัญญาณ SIGIOT และ SIGSEGV ถูกดำเนินการในทุกการจัดสรร

functionset

เนื่องจากข้อกำหนดของการนำไปใช้ภายใน แต่ละการจัดสรรจะยังคง จำเป็นต้องใช้ขนาดหน้าที่มีความยาวอย่างน้อยหนึ่งหน้า ดังนั้นอ็อพชัน functionset แทบจะลดการใช้โอเวอร์เฮดของอ็อพชัน catch_overflow เท่านั้น มากกว่าการลบออก

ถ้ารูทีนย่อย **realloc** ถูกเรียกใช้จากฟังก์ชัน ที่เป็นสมาชิกของรายการฟังก์ชันที่ผู้ใช้ระบุ จะทำการตรวจหาบัฟเฟอร์โอเวอร์โฟลว์ แม้ว่าการจัดสรรเริ่มแรกจะไม่ได้ทำจากฟังก์ชัน ที่ระบุ ในทางกลับกันของเงื่อนไขนี้เป็นจริงเช่นกัน

หมายเหตุ: ถ้าอ็อพชัน override_signal ถูกตั้งค่าร่วมกับ อ็อพชัน functionset การยกเลิกลักษณะการทำงานของสัญญาณ SIGIOT และ SIGSEGV ถูกดำเนินการในทุกการจัดสรร

อ็อพชัน functionset ไม่ได้ตรวจสอบความถูกต้องของฟังก์ชัน ที่ระบุในรายการ

allow_overreading

โดยดีฟอลต์ เมื่ออ็อพชันการดีบัก catch_overflow ถูกเปิดใช้งาน และโปรแกรมที่เรียกใช้พยายามอ่านจากตำแหน่งที่เลยจุดสิ้นสุดของหน่วยความจำที่จัดสรรไว้ จะเกิดการละเมิดการแบ่งเป็นเซ็กเมนต์ และการประมวลผลจะสร้าง core dump อย่างไรก็ตาม ผู้ใช้อาจไม่สนใจในการตรวจจับข้อผิดพลาดประเภทนี้ และอาจ เปิดใช้งาน catch_overflow ไว้เพื่อตรวจนับการเขียนทับที่อันตรายกว่านั้น การระบุอ็อพชัน allow_overreading จะทำให้อ็อพชัน catch_overflow ไม่สนใจการอ่านทับ ทำให้ข้อผิดพลาดประเภทอื่นซึ่งอาจถูกพิจารณาว่ามีความร้ายแรง มากกว่าสามารถถูกตรวจพบได้ก่อน

อ็อพชัน allow_overreading มีความสำคัญในบริบทของ อ็อพชัน catch_overflow เท่านั้น โดยถูกเปิดใช้งานดังนี้:

```
MALLOCDEBUG=catch_overflow,allow_overreading,
```

postfree_checking

ข้อจำกัด

อ็อพชัน postfree_checking ใช้หน่วยความจำพิเศษ เป็นจำนวนมาก โปรแกรมที่มีความต้องการใช้หน่วยความจำขนาดใหญ่มากอาจไม่สามารถ ใช้อ็อพชัน postfree_checking ได้

Malloc trace

Malloc Trace คืออ็อพชันการดีบักที่ออกแบบเพื่อให้ทำการติดตามการเรียกใช้ทั้งหมด ที่ไปยัง API ระบบย่อย **malloc** ผ่านสิ่งอำนวยความสะดวก การติดตามระบบ

Malloc log

Malloc Log คืออ็อพชันการดีบักที่ออกแบบมาให้ผู้ใช้ที่มีฐานข้อมูล รันไทม์ของการจัดสรรที่แอ็คทีฟในระบบย่อย **malloc**

report_allocations

อ็อพชัน report_allocations เป็นเครื่องมือสำหรับการตรวจหา หน่วยความจำรั่วไหลในแอ็พพลิเคชันโปรแกรม อ็อพชัน report_allocations ใช้ฐานข้อมูลที่สร้างขึ้นโดย Malloc Log เพื่อรายงานรายการของการจัดสรร ที่ผู้ใช้ครอบครองอยู่ในขณะนี้ เร็กคอร์ดของแต่ละการจัดสรรที่สำเร็จจะถูกทำ ในตอนที่มีการร้องขอโดย Malloc Log เมื่อการจัดสรรถูกยกเลิก Malloc Log จะลบเร็กคอร์ดของตนออกจากฐานข้อมูล เมื่อออกจากการประมวลผล รายการ ของการจัดสรรที่ยังแอ็คทีฟจะถูกพิมพ์ไปยัง stderr เพื่อแสดงรายการการจัดสรร ที่ยังไม่เคยถูกปล่อยฟรีโดยผู้เรียกใช้

อ็อพชัน report_allocations ต้องการใช้ฟังก์ชันการทำงานของ Malloc Log เพื่อทำงาน ดังนั้น Malloc Log จึงถูกเปิดใช้งาน เมื่อ report_allocations ถูกเปิดใช้งาน อ็อพชัน report_allocations ถูกเปิดใช้งาน ดังนี้:

```
MALLOCDEBUG=report_allocations
```

validate_ptrs

โดยดีฟอลต์ API ระบบย่อย **malloc** ไม่ได้ตรวจสอบความถูกต้อง ตัวชี้อินพุตของตนเพื่อให้แน่ใจว่าอ้างถึงหน่วยความจำที่ถูกจัดสรรก่อนหน้า อย่างแท้จริง ถ้าหนึ่งในตัวชี้เหล่านี้ไม่ถูกต้อง อาจเกิดความล้มเหลวฮีป ที่ร้ายแรง การระบุอ็อพชัน validate_ptrs จะทำให้ APIs ของระบบย่อย **malloc** ดำเนินการตรวจสอบความถูกต้องเพิ่มในตัวชี้อินพุต ถ้าพบว่าตัวชี้ไม่ถูกต้อง (คือ ไม่ได้อ้างถึงหน่วยความจำที่ถูกจัดสรรก่อนหน้านี้โดยการเรียกใช้ API ระบบย่อย **malloc**) ข้อความแสดงข้อผิดพลาดจะระบุเหตุที่ไม่ถูกต้องจะถูกพิมพ์ ฟังก์ชัน abort จะถูกเรียกใช้และสร้างไฟล์ core อ็อพชัน validate_ptrs จะเหมือนกับอ็อพชันย่อย verbose อ็อพชัน validate_ptrs จะไม่มีผลถ้าอ็อพชัน postfree_checking ถูกเปิดใช้งาน

อ็อพชัน validate_ptrs ถูกเปิดใช้งานดังนี้:

```
MALLOCDEBUG=validate_ptrs
```

Malloc detect

Malloc Detect เป็นอ็อพชันการดีบักที่ออกแบบเพื่อตรวจหาและรายงานความล้มเหลว ของโครงสร้างข้อมูลระบบย่อย **malloc** ภายในของ ทุกการเรียกใช้ไปยัง API ระบบย่อย **malloc**

ถ้อยคำ

อ็อพชันย่อยของ Malloc Detect

checkarena

อ็อพชันย่อยของ Malloc Detect

output

โดยดีฟอลต์ อ็อพชันการดีบัก malloc ส่งเอาต์พุตไปยัง stderr นี้ อาจไม่เป็นที่ต้องการสำหรับทุกโปรแกรม อ็อพชัน `output` มีอยู่เพื่อ จัดให้มีปลายทางอื่นสำหรับข้อมูลที่จะพิมพ์ เอาต์พุตสามารถถูกส่งไปยัง stderr, stdout หรือไฟล์ใดๆ บนระบบ

อ็อพชัน `output` ถูกเปิดใช้งานดังนี้:

```
MALLOCDEBUG=output:<filename>
```

continue

อ็อพชันการดีบัก malloc มากมายเรียกใช้ **abort( )** เมื่อ ตรวจพบข้อผิดพลาด นี้ไม่ใช่ลักษณะการทำงานที่ต้องการเสมอไป สำหรับโปรแกรมทั้งหมด อ็อพชัน `continue` มีอยู่เพื่อแนะนำระบบย่อย **malloc** ให้ดำเนินต่อหลังตรวจพบข้อผิดพลาดที่ซิงโครนัส แทนที่จะยกเลิกการประมวลผล ข้อความแสดงความผิดพลาดจะยังคงถูกบันทึกไปยังแชนเนล ที่เหมาะสม

อ็อพชัน `continue` ถูกเปิดใช้งานดังนี้:

```
MALLOCDEBUG=continue
```

Malloc debug fill

Malloc debug fill เป็นอ็อพชันการดีบักที่ถูกออกแบบให้เติมหน่วยความจำที่ถูกจัดสรรโดยใช้การเรียกใช้ malloc() calls โดยผู้ใช้ระบุแพ็ตเทิร์นสำหรับจุดประสงค์ของการดีบัก

แพ็ตเทิร์นควรระบุเป็นสตริง (เช่น export MALLOCDEBUG=fill:"abc" จะตั้งค่าหน่วยความจำที่ถูกจัดสรรโดยใช้ malloc ที่มีแพ็ตเทิร์น "abc") และอนุญาตให้มจำนวนอักขระได้มากถึง 28 อักขระ ถ้าไม่ได้ระบุแพ็ตเทิร์น อ็อพชัน fill จะถูกข้าม

อ็อพชัน malloc debug fill สามารถเปิดใช้งานได้ดังนี้:

```
MALLOCDEBUG=fill:pattern
```

แพ็ตเทิร์นอาจเป็นตัวเลขฐานแปดหรือฐานสิบหกที่ระบุในรูปแบบสตริง เช่นแพ็ตเทิร์น "\101" จะถูกใช้แทนเลขฐานแปด สำหรับอักขระ 'A' และแพ็ตเทิร์น "\x41" จะใช้แทนเลขฐานสิบหกสำหรับอักขระ 'A'

ถ้าระบุตัวเลขฐานแปดไม่ถูกต้อง เช่น\777 ซึ่งไม่สามารถอยู่ภายใน 1 ไบต์ จะถูกเก็บเป็น\377 ซึ่งเป็นเลขฐานแปดสูงสุดที่สามารถเก็บใน 1 ไบต์

หลักการที่เกี่ยวข้อง:

“การจัดสรรหน่วยความจำระบบโดยใช้ระบบย่อย malloc” ในหน้า 696
หน่วยความจำจะถูกจัดสรรให้กับแอ็พพลิเคชัน โดยใช้ระบบย่อย **malloc**

Malloc multiheap

โดยดีฟอลต์ ระบบย่อย malloc ใช้ฮีปเดียว หรือพูลหน่วยความจำว่าง

อย่างไรก็ตาม ยังมีความสามารถมัลติฮีปที่เป็นทางเลือกเพื่อให้ใช้หลายฮีปของหน่วยความจำว่าง มากกว่าใช้เพียงฮีปเดียว

วัตถุประสงค์ของการจัดให้มีความสามารถแบบหลายฮีปในระบบย่อย **malloc** คือเพื่อปรับปรุงผลการทำงานของแอ็พพลิเคชันแบบเธรด ที่กำลังทำงานบนระบบมัลติโพรเซสเซอร์ เมื่อระบบย่อย **malloc** ถูกจำกัดให้ใช้ฮีปเดียว การร้องขอการจัดสรรหน่วยความจำที่เกิดขึ้นพร้อมกัน ที่ได้รับจากเธอที่ทำงานบนโพรเซสเซอร์แยกจะถูกจัดเรียงลำดับ ดังนั้นระบบย่อย **malloc** จะสามารถให้บริการได้ครั้งละหนึ่งเธรดเท่านั้น ส่งผลกระทบอย่างยิ่งต่อผลการทำงานของระบบมัลติโพรเซสเซอร์

ด้วยการเปิดใช้งานความสามารถ malloc multiheap ระบบย่อย **malloc** จะสร้างจำนวนฮีปคงที่สำหรับใช้งาน โดยจะเริ่มใช้หลายฮีปหลังเริ่มทำงานเธรดที่สอง (การประมวลผลจะเปลี่ยนเป็นแบบมัลติเธรด) การร้องขอการจัดสรรหน่วยความจำแต่ละการร้องขอจะถูกให้บริการโดยใช้หนึ่งในฮีป ที่มีอยู่ จากนั้นระบบย่อย **malloc** สามารถประมวลผลการร้องขอ การจัดสรรหน่วยความจำในแบบขนาน ตราบเท่าที่จำนวนเธรดที่ให้บริการการร้องขอ ที่เกิดขึ้นพร้อมกันมีค่าน้อยกว่าหรือเท่ากับจำนวนฮีป

ถ้าจำนวนของเธรดที่ให้บริการการร้องขอที่เกิดขึ้นพร้อมกันเกินจำนวน ฮีป การร้องขอที่เกิดขึ้นพร้อมกันที่เพิ่มมาจะถูกเรียงลำดับ ยกเว้น ว่าเหตุการณ์นี้เกิดขึ้นในลักษณะดำเนินต่อเนื่อง ผลการทำงานโดยรวมของระบบย่อย **malloc** ควรได้รับการปรับปรุงเป็นสำคัญเมื่อหลายเธรด กำลังทำการเรียกใช้รูทีนย่อย **malloc** ในสภาวะแวดล้อม มัลติโพรเซสเซอร์

การเปิดใช้งาน malloc multiheap

Malloc multiheap ไม่ถูกเปิดใช้งานเป็นค่าดีฟอลต์ โดยถูกเปิดใช้งานและตั้งค่า โดยการตั้งค่าตัวแปรสภาวะแวดล้อม **MALLOCOPTIONS** ในการ เปิดใช้งาน malloc multiheap ด้วยค่ากำหนดดีฟอลต์ ให้ตั้งค่า `MALLOCOPTIONS=multiheap` ก่อนประมวลผลเริ่มทำงาน การตั้งค่า **MALLOCOPTIONS** ใน ลักษณะนี้จะเปิดใช้งาน malloc multiheap ด้วยการตั้งค่าดีฟอลต์ โดยมี 32 ฮีปและอัลกอริธึมการเลือกฮีปแบบด่วน

อ็อพชัน Malloc multiheap

อ็อพชัน Malloc Multiheap มีดังนี้:

- multiheap:n
- considersize

แต่ละอ็อพชันเหล่านี้ถูกอธิบายรายละเอียดในเอกสารนี้

ในการตั้งค่าใดๆ ของอ็อพชันเหล่านี้ ใช้ไวยากรณ์ต่อไปนี้:

```
MALLOCOPTIONS=[multiheap:n] | [considersize]
```

โดยสามารถระบุหนึ่งหรือทั้งสองอ็อพชันในลำดับใดๆ ตราบใดที่อ็อพชันถูกคั่น ด้วยจุลภาค ดังในตัวอย่างต่อไปนี้:

```
MALLOCOPTIONS=multiheap:3,considersize
```

ในตัวอย่างก่อนหน้า malloc multiheap จะถูกเปิดใช้งานโดยมีสาม ฮีป และอัลกอริธึมการเลือกฮีปที่ค่อนข้างช้ากว่าที่พยายามลดขนาดการประมวลผล ให้มีขนาดเล็กที่สุด

แต่ละอ็อพชันการตั้งค่าควรถูกระบุครั้งเดียวเมื่อตั้งค่า **MALLOCOPTIONS** ถ้าอ็อพชันการกำหนดคอนฟิก ถูกระบุมากกว่าหนึ่งครั้งต่อการตั้งค่า มีเพียงอินสแตนซ์สุดท้ายเท่านั้นที่จะมีผลใช้

อ็อพชัน Malloc Multiheap ถูกอธิบายดังนี้:

multiheap:n

โดยดีฟอลต์ จำนวนฮีปสูงสุดที่มีสำหรับ malloc multiheap คือ 32 อ็อพชัน *multiheap:n* สามารถใช้เพื่อเปลี่ยนจำนวนสูงสุดของฮีปเป็นค่าใดๆ ตั้งแต่ 1 ถึง 32 โดยที่ *n* คือจำนวนฮีป ถ้า *n* ถูกตั้งค่า นอกขอบเขตที่กำหนด จะใช้ค่าดีฟอลต์ 32 แทน การเปิดใช้งานฮีป ควรเปิดเท่าที่จำเป็นสำหรับข้อกำหนดการประมวลผลเท่านั้น ฮีปที่เปิดใช้งานโดยไม่จำเป็นอาจเพิ่มจำนวนการแตกแฟรกเมนต์และการเสียประโยชน์

considersize

โดยดีฟอลต์ malloc multiheap เลือกฮีปที่มีอยู่ถัดไป ถ้าอ็อพชัน *considersize* ถูกระบุ malloc multiheap จะใช้อัลกอริธึมการเลือกฮีปแบบอื่นที่พยายามเลือกฮีปที่มีอยู่ ที่มีพื้นที่ว่างเพียงพอสำหรับการจัดการตามการร้องขอ นี้อาจลดขนาด ชุดการทำงานของการประมวลผลให้เล็กสุดโดยการลดจำนวนของการเรียกใช้รูทีนย่อย **sbrk** อย่างไรก็ตาม เนื่องจากจำเป็นต้องมีการประมวลผลเพิ่มเติม อัลกอริธึมการเลือกฮีป *considersize* จะค่อนข้างช้ากว่าอัลกอริธึมการเลือกฮีปดีฟอลต์

ถ้าฮีปไม่สามารถจัดสรรพื้นที่ได้ รูทีนย่อย **malloc** จะส่งกลับค่า NULL และตั้งค่า errno เป็น ENOMEM ถ้าไม่มีหน่วยความจำพร้อมใช้ในฮีปปัจจุบัน ระบบย่อย **malloc** จะตรวจสอบฮีปอื่นเพื่อหาพื้นที่ที่พร้อมใช้ได้

หลักการที่เกี่ยวข้อง:

"การจัดสรรหน่วยความจำระบบโดยใช้ระบบย่อย malloc" ในหน้า 696
หน่วยความจำจะถูกจัดสรรให้กับแอ็พพลิเคชัน โดยใช้ระบบย่อย **malloc**

Malloc buckets

Malloc buckets มีส่วนขยายของตัวจัดสรรดีฟอลต์ที่อิงตาม bucket ที่เป็นทางเลือก

โดยมุ่งช่วยปรับปรุงผลการทำงานสำหรับแอ็พพลิเคชันที่ ออกการร้องขอการจัดสรรพื้นที่ขนาดเล็กจำนวนมาก เมื่อเปิดใช้งาน malloc buckets การร้องขอให้มีการจัดสรรที่อยู่ภายในช่องที่กำหนดไว้แล้วของขนาดบล็อกจะถูก ประมวลผลโดย malloc buckets การร้องขออื่นทั้งหมดถูกประมวลผลในลักษณะปกติ โดยตัวจัดสรรดีฟอลต์

Malloc buckets ไม่ถูกเปิดใช้งานเป็นค่าดีฟอลต์ โดยถูกเปิดใช้งานและตั้งค่า ก่อนที่การประมวลผลจะเริ่มทำงานโดยการตั้งค่าตัวแปรสภาวะแวดล้อม **MALLOCOPTIONS**

ส่วนประกอบ Bucket และการกำหนดขนาด

bucket ประกอบด้วยบล็อกของหน่วยความจำที่แบ่งออกเป็นจำนวนบล็อก ขนาดเล็กที่กำหนดไว้ล่วงหน้าของขนาดเท่ากัน แต่ละบล็อกเป็นหน่วยของหน่วยความจำ ที่สามารถจัดสรรได้ แต่ละ bucket ถูกกำหนดโดยใช้หมายเลข bucket bucket แรกคือ bucket 0 bucket ที่สองคือ bucket 1 และ bucket ที่สามคือ bucket 2 ตามลำดับ bucket แรกคือ bucket ขนาดเล็กสุด และแต่ละ bucket ที่ตามมา จะมีขนาดใหญ่กว่า bucket ที่อยู่ข้างหน้า โดยใช้สูตรที่อธิบายภายหลัง ในส่วนนี้ โดยมีอยู่ 128 buckets สูงสุดต่อหนึ่งฮีป

ขนาดบล็อกสำหรับแต่ละ bucket คือผลคูณของแฟคเตอร์ขนาด bucket แฟคเตอร์ขนาด bucket เท่ากับขนาดบล็อกของ bucket แรก แต่ละบล็อก ใน bucket ที่สองจะมีขนาดสองเท่าของขนาดนี้ แต่ละบล็อกใน bucket ที่สาม มีขนาดสามเท่าของขนาดนี้ ตามลำดับ ดังนั้นขนาดบล็อกของ bucket ที่กำหนด ถูกพิจารณาดังนี้:

```
block size = (bucket number + 1) * bucket sizing factor
```

ตัวอย่าง แฟคเตอร์ขนาด bucket คือ 16 ทำให้มีขนาดบล็อก 16 ไบต์สำหรับ bucket แรก (bucket 0), 32 ไบต์สำหรับ bucket ที่สอง (bucket 1), 48 ไบต์สำหรับ bucket ที่สาม (bucket 2) ตามลำดับ

แฟคเตอร์ขนาด bucket ต้องเป็นผลคูณของ 8 สำหรับการใช้ 32 บิต และผลคูณของ 16 สำหรับการใช้ 64 บิตเพื่อรับประกันว่าแอดเดรสที่ส่งกลับจากฟังก์ชันระบบย่อย malloc มีการจัดเรียงอย่างเหมาะสมสำหรับ ชนิดข้อมูลทั้งหมด

ขนาด bucket สำหรับ bucket ที่กำหนดถูกพิจารณาดังนี้:

```
ขนาด bucket = จำนวนบล็อกต่อ bucket * (malloc overhead +
              ((bucket number + 1) * bucket sizing factor))
```

สูตรก่อนหน้าสามารถนำไปใช้เพื่อพิจารณาจำนวนไบต์ที่ต้องการ แท้จริงสำหรับแต่ละ bucket ในสูตรนี้ malloc overhead อ้างถึงขนาด ของ malloc construct ภายในที่จำเป็นสำหรับแต่ละบล็อกใน bucket construct ภายในนี้ยาว 8 ไบต์สำหรับแอ็พพลิเคชัน 32 บิต และ 16 ไบต์ สำหรับแอ็พพลิเคชัน 64 บิต ไม่ได้เป็นส่วนหนึ่งของพื้นที่ที่จัดสรรได้ที่มี สำหรับผู้ใช้ แต่เป็นส่วนหนึ่งของขนาดทั้งหมดของแต่ละ bucket

จำนวนบล็อกต่อ bucket จำนวน buckets และแฟคเตอร์ขนาด bucket ทั้งหมดถูกตั้งค่าด้วยตัวแปรสภาวะแวดล้อม **MALLOCOPTIONS**

การประมวลผลการจัดสรรจาก buckets

บล็อกจะถูกกจัดสรรจากหนึ่งใน buckets เมื่อใดก็ตามที่ malloc buckets ถูกเปิดใช้งาน และการร้องขอการจัดสรรตกอยู่ในช่วงของขนาดบล็อก ที่กำหนดโดย buckets แต่ละการร้องขอการจัดสรรได้รับการจัดการจาก bucket ที่เล็กที่สุดที่เป็นไปได้ เพื่อสงวนพื้นที่

ถ้าได้รับการร้องขอการจัดสรรสำหรับ bucket และบล็อกทั้งหมด พร้อมให้จัดสรรได้ malloc buckets จะขนาด bucket โดยอัตโนมัติ เพื่อให้บริการการร้องขอ จำนวนของบล็อกใหม่ถูกเพิ่มเพื่อขยาย bucket นั้นเท่ากับจำนวนของบล็อกที่มีอยู่ภายใน bucket เสมอ ซึ่งตั้งค่าโดยการตั้งค่าตัวแปรสภาวะแวดล้อม **MALLOCOPTIONS**

การสนับสนุนสำหรับการประมวลผลแบบหลายฮีพ

ความสามารถ malloc multiheap จัดให้มีวิธีเปิดใช้การหลายฮีป malloc เพื่อเพิ่มผลการทำงานของแอ็พพลิเคชันที่เป็นเธรดที่ทำงานในระบบ มัลติโพรเซสเซอร์ Malloc buckets สนับสนุนจำนวนสูงสุด 128 buckets ต่อฮีป นี้อนุญาตให้ระบบย่อย **malloc** สนับสนุนการเปิดใช้งานที่เกิดขึ้นพร้อมกันของ malloc buckets และ malloc multiheap เพื่อให้การประมวลผลแบบเธรดที่กำลังทำงานบนระบบ มัลติโพรเซสเซอร์สามารถใช้ประโยชน์จากอัลกอริธึม buckets

การเปิดใช้งาน malloc buckets

Malloc buckets ไม่ถูกเปิดใช้งานเป็นค่าดีฟอลต์ แต่ถูกเปิดใช้งานและตั้งค่า โดยการตั้งค่าตัวแปรสภาวะแวดล้อมต่อไปนี้:

- **MALLOCTYPE**
- **MALLOCOPTIONS**

ตัวแปรสภาวะแวดล้อม **MALLOCTYPE** ต้องถูกตั้งค่าเป็น ตัวจัดสรรดีฟอลต์เมื่อใช้ Malloc Buckets ในการเปิดใช้งาน malloc buckets ด้วยค่ากำหนดดีฟอลต์ ให้ตั้งค่า `MALLOCOPTIONS=buckets` ก่อนที่การประมวลผล เริ่มทำงาน ในการเปิดใช้งาน malloc buckets ด้วยอ็อพชันการตั้งค่าที่ผู้ใช้ระบุ ให้ตั้งค่า `MALLOCOPTIONS=buckets,options` ก่อนการประมวลผลเริ่มทำงาน โดยที่ *options* เป็นรายการของอ็อพชันการตั้งค่าที่กำหนดไว้แล้วอย่างน้อยหนึ่งค่า ที่คั่นด้วยจุลภาค

อ็อพชันการกำหนดคอนฟิก Malloc buckets

ตัวแปรสภาวะแวดล้อม **MALLOCOPTIONS** สามารถใช้ เพื่อจัดให้มี malloc buckets ที่มีอ็อพชันการตั้งค่าที่กำหนดไว้แล้วอย่างน้อยหนึ่งค่า ต่อไปนี้:

```
number_of_buckets:n
bucket_sizing_factor:n
blocks_per_bucket:n
bucket_statistics:[stdout|stderr|pathname]
no_mallinfo
```

อ็อพชันเหล่านี้แต่ละอ็อพชันจะอธิบายในรายละเอียดใน **MALLOCOPTIONS**

ในการตั้งค่าตัวแปรสภาวะแวดล้อม **MALLOCOPTIONS** ให้ใช้ไวยากรณ์ต่อไปนี้:

```
MALLOCOPTIONS=[buckets,[ number_of_buckets:n | bucket_sizing_factor:n | blocks_per_bucket:n |
bucket_statistics:[stdout|stderr|pathname] | no_mallinfo],...]
```

โดยสามารถระบุได้มากกว่าหนึ่งอ็อพชัน (และในลำดับใดก็ได้) ตราบใดที่อ็อพชัน ยังถูกคั่นด้วยจุลภาค ตัวอย่าง:

```
MALLOCOPTIONS=buckets,number_of_buckets:128,bucket_sizing_factor:8,bucket_statistics:stderr
MALLOCOPTIONS=buckets,bucket_statistics:stdout,blocks_per_bucket:512
```

เครื่องหมายจุลภาคเป็นตัวคั่นเดียวเท่านั้นที่ใช้ได้สำหรับการแบ่งอ็อพชันการตั้งค่า ในไวยากรณ์นี้ การใช้ตัวคั่นอื่น (เช่น ช่องว่าง) ระหว่างอ็อพชัน จะทำให้อ็อพชันการตั้งค่าถูกแยกวิเคราะห์ไม่ถูกต้อง

แต่ละอ็อพชันการตั้งค่าควรถูกระบุครั้งเดียวเท่านั้นเมื่อตั้งค่าตัวแปรสภาวะแวดล้อม **MALLOCOPTIONS** ถ้าอ็อพชันคอนฟิกูเรชัน ถูกระบุมากกว่าหนึ่งครั้งต่อการตั้งค่า มีเพียงอินสแตนซ์สุดท้ายเท่านั้นที่จะมีผลใช้

ถ้าอ็อพชันการตั้งค่าถูกระบุด้วยค่าที่ไม่ถูกต้อง malloc buckets จะเขียนข้อความเตือนไปยังข้อผิดพลาดมาตรฐาน จากนั้นดำเนินงานต่อโดยใช้ ค่าดีฟอลต์ที่ระบุไว้

อ็อพชันการตั้งค่า Malloc Buckets จะทราบโดยระบบย่อย **malloc** ต่อเมื่ออ็อพชัน `buckets` ถูก ตั้งค่า ดังในตัวอย่างต่อไปนี้:

```
MALLOCOPTIONS=number_of_buckets:8,buckets,bucket_statistics:stderr
```

อ็อพชัน Malloc buckets

number_of_buckets:*n*

อ็อพชัน number_of_buckets:*n* สามารถใช้เพื่อ ระบุจำนวนของ buckets ที่พร้อมใช้ต่อฮีป โดยที่ *n* คือจำนวนของ buckets ค่าที่ระบุสำหรับ *n* จะใช้กับฮีปที่พร้อมใช้ทั้งหมด

ค่าดีฟอลต์สำหรับ number_of_buckets คือ 16 ค่าต่ำสุดที่อนุญาตคือ 1 ค่าสูงสุดที่อนุญาตคือ 128

bucket_sizing_factor:*n*

อ็อพชัน bucket_sizing_factor:*n* สามารถใช้เพื่อ ระบุแฟคเตอร์ขนาด bucket โดยที่ *n* คือ ขนาด bucket เป็นไบต์

ค่าที่ระบุสำหรับ bucket_sizing_factor ต้องเป็นผลคูณของ 8 สำหรับการใช้ 32 บิต และผลคูณของ 16 สำหรับ การใช้ 64 บิต ค่าดีฟอลต์สำหรับ bucket_sizing_factor คือ 32 สำหรับ การใช้ 32 บิตและ 64 สำหรับการใช้ 64 บิต

blocks_per_bucket:*n*

อ็อพชัน blocks_per_bucket:*n* สามารถใช้เพื่อ ระบุจำนวนของบล็อกที่มีอยู่ภายในแต่ละ bucket โดยที่ *n* คือจำนวน บล็อก ค่านี้ใช้กับ buckets ทั้งหมด ค่าของ *n* ยังใช้เพื่อพิจารณา จำนวนบล็อกที่จะเพิ่มเมื่อ bucket ถูกขยายโดย อัตโนมัติเนื่องจาก บล็อกทั้งหมดได้ถูกจัดสรร

ค่าดีฟอลต์สำหรับ blocks_per_bucket คือ 1024

bucket_statistics:[stdout|stderr|*pathname***]**

อ็อพชัน bucket_statistics จะทำให้ระบบย่อย **malloc** แสดงเอาต์พุตสรุปข้อมูลสถิติสำหรับ malloc buckets เมื่อสิ้นสุด แต่ละการประมวลผลปกติที่เรียกใช้ระบบย่อย malloc ขณะที่ malloc buckets ถูกเปิดใช้งาน ข้อมูลสรุปนี้แสดงข้อมูล การตั้งค่า buckets และจำนวน การร้องขอการจัดสรรที่ถูกประมวลผลสำหรับแต่ละ bucket ถ้าหลายฮีปถูก เปิดใช้งาน ด้วยวิธี malloc multiheap จำนวนการร้องขอการจัดสรรที่แสดง สำหรับแต่ละ bucket จะเป็นผลรวมของการร้องขอ การจัดสรรทั้งหมดที่ประมวลผลสำหรับ bucket นั้นสำหรับฮีปทั้งหมด

ข้อมูลสรุปสถิติ buckets จะถูกเขียนไปยัง ปลายทางเอาต์พุตหนึ่งในปลายทางต่อไปนี้ ตามที่ระบุด้วยอ็อพชัน bucket_statistics

- **stdout** – เอาต์พุตมาตรฐาน
- **stderr** – ข้อผิดพลาดมาตรฐาน
- *pathname* – ชื่อพาธที่ผู้ใช้ระบุ

ถ้าชื่อพาธที่ผู้ใช้ระบุถูกกำหนดให้ เอาต์พุตด้านสถิติจะถูกต่อท้าย เนื้อหาที่มีอยู่แล้วของไฟล์ (ถ้ามี)

เอาต์พุตมาตรฐานไม่ควร ถูกใช้เป็นปลายทางเอาต์พุตสำหรับการประมวลผลที่มีเอาต์พุตถูกไพพ์ เป็นอินพุตไปยัง การประมวลผลอื่น

อ็อพชัน bucket_statistics ถูกปิดใช้งาน เป็นค่าดีฟอลต์

หมายเหตุ: การร้องขอการจัดสรรเพิ่มเติมหนึ่งการร้องขอจะถูกแสดง ใน bucket แรกสำหรับรูทีนย่อย **atexit** ที่พิมพ์ ข้อมูลสรุปสถิติเสมอ สำหรับการประมวลผลแบบเธรด การร้องขอการจัดสรร เพิ่มจะถูกแสดงสำหรับบาง buckets เนื่องจากการเรียกใช้ระบบย่อย malloc ที่ออก โดยไลบรารีแบบเธรด

no_mallinfo

ถ้าคุณระบุ `MALLOCOPTIONS=no_mallinfo` การตั้งค่า **mallinfo** จะถูกปิดใช้งานและข้อมูลเกี่ยวกับฮีปที่ จัดการโดย ระบบย่อย **malloc** จะไม่ถูกบันทึกการทำงาน

การกำหนดคอนฟิกดีฟอลต์สำหรับ Malloc buckets

ตารางต่อไปนี้สรุปการตั้งค่าดีฟอลต์ malloc buckets

อ็อพชันการตั้งค่า	ค่าดีฟอลต์ (32 บิต)	ค่าดีฟอลต์ (64 บิต)
จำนวน buckets ต่อฮีป	16	16
แฟคเตอร์ขนาด bucket	32 ไบต์	64 ไบต์
ขอบเขตการจัดสรร	1 ถึง 512 ไบต์ (รวม)	1 ถึง 1024 ไบต์ (รวม)
จำวนบล็อกที่มีภายในแต่ละ bucket	1024	1024
ข้อมูลสรุปสถิติ bucket	ปิดใช้งาน	ปิดใช้งาน

การตั้งค่าดีฟอลต์สำหรับ malloc buckets ควรเพียงพอเพื่อให้ สามารถปรับปรุงผลการทำงานสำหรับหลายแอ็พพลิเคชันที่ออกการร้องขอ การจัดสรรพื้นที่ขนาดเล็กจำนวนมาก อย่างไรก็ตาม อาจเป็นไปได้ที่จะทำได้เพิ่ม โดยการตั้งค่าตัวแปรสภาวะแวดล้อม **MALLOCOPTIONS** เพื่อแก้ไขการตั้งค่าลักษณะดีฟอลต์ ก่อนการแก้ไขการตั้งค่าดีฟอลต์ ทำความคุ้นเคยกับข้อกำหนดและการใช้งานหน่วยความจำของแอ็พพลิเคชัน Malloc buckets สามารถถูกเปิดใช้งานด้วยอ็อพชัน bucket_statistics เพื่อปรับ การตั้งค่า buckets ให้ดีขึ้น

ข้อจำกัด

เนื่องจากการเปลี่ยนแปลงในข้อกำหนดและการใช้งานหน่วยความจำ บางแอ็พพลิเคชัน อาจไม่ได้รับประโยชน์จาก scheme การจัดสรรหน่วยความจำที่ใช้โดย malloc buckets ดังนั้น จึงไม่แนะนำให้เปิดใช้งาน malloc buckets สำหรับการใช้ทั้งระบบ เพื่อให้เกิดผลการทำงานที่ดีที่สุด malloc buckets ควรถูกเปิดใช้งานและตั้งค่าตาม ความต้องการของแต่ละแอ็พพลิเคชัน

หลักการที่เกี่ยวข้อง:

“การจัดสรรหน่วยความจำระบบโดยใช้ระบบย่อย malloc” ในหน้า 696
หน่วยความจำจะถูกจัดสรรให้กับแอ็พพลิเคชัน โดยใช้ระบบย่อย **malloc**

Malloc trace

Malloc Trace จัดให้มีส่วนขยายที่เป็นทางเลือกสำหรับระบบย่อย malloc เพื่อใช้ กับสิ่งอำนวยความสะดวกการติดตาม

การติดตามรูทีนย่อย **malloc, realloc** และ **free** จะถูก บันทึกเพื่อใช้ในการพิจารณาปัญหาและการวิเคราะห์ผลการทำงาน

Malloc Trace ไม่ถูกเปิดใช้งานเป็นค่าดีฟอลต์ แต่สามารถเปิดใช้งานและตั้งค่าก่อนหน้าที่ การประมวลผลจะเริ่มทำงานผ่านตัวแปรสภาวะแวดล้อม **MALLOCDEBUG**

เหตุการณ์ที่บันทึกโดย malloc trace

tracehook IDs ที่ใช้สำหรับ Malloc Trace มีดังนี้:

- HKWD_LIBC_MALL_SUBSYSTEM
- HKWD_LIBC_MALL_INTERNAL

เมื่อเปิดใช้การติดตามสำหรับ HKWD_LIBC_MALL_SUBSYSTEM อินพุตพารามิเตอร์ รวมถึงค่าส่งกลับสำหรับแต่ละการเรียกใช้ของรูทีนย่อย **malloc, realloc** และ **free** ถูกบันทึก ในระบบย่อยการติดตาม

เมื่อเปิดใช้การติดตามสำหรับ HKWD_LIBC_MALL_INTERNAL และเปิดใช้เครื่องมือ การดีบัก Malloc Detect แล้ว Malloc Trace จะบันทึกเหตุการณ์การติดตามทุกครั้งที่ Malloc Detect ตรวจพบข้อผิดพลาดในโครงสร้างข้อมูลภายในของระบบย่อย malloc

การเปิดใช้งาน malloc trace

Malloc Trace ไม่ถูกเปิดใช้งานเป็นค่าดีฟอลต์ โดยถูกเปิดใช้งานและตั้งค่าโดย การตั้งค่าตัวแปรสภาวะแวดล้อม **MALLOCDEBUG** ในการเปิดใช้งาน Malloc Trace ในตั้งค่าตัวแปรสภาวะแวดล้อม **MALLOCDEBUG** โดยพิมพ์ต่อไปนี้บนบรรทัดคำสั่ง:

```
MALLOCDEBUG=trace
```

ในการเปิดใช้งานคุณลักษณะ Malloc Debug อื่นๆ ให้ตั้งค่าตัวแปรสภาวะแวดล้อม **MALLOCDEBUG** ดังนี้:

```
MALLOCDEBUG=[trace, other_option]
```

ข้อจำกัด

คุณลักษณะการดีบัก Malloc Trace เข้ากันได้กับนโยบายและอ็อพชัน malloc ต่อไปนี้:

- Default Allocation Policy
- 3.1 Allocation Policy
- Watson Allocation Policy
- Watson2 Allocation Policy
- Malloc Buckets
- Malloc Multiheap
- Malloc Thread Cache
- Malloc Disclaim Option

หลักการที่เกี่ยวข้อง:

"การจัดสรรหน่วยความจำระบบโดยใช้ระบบย่อย malloc" ในหน้า 696
หน่วยความจำจะถูกจัดสรรให้กับแอ็พพลิเคชันโดยใช้ระบบย่อย **malloc**

Malloc log

Malloc Log เป็นส่วนขยายทางเลือกของระบบย่อย **malloc** ที่อนุญาตให้ผู้ใช้รับข้อมูลที่เกี่ยวกับการจัดสรรที่แอ็คทีฟขณะนี้ที่จัดโดยกระบวนการการเรียกใช้ ข้อมูลนี้จะถูกนำไปใช้ในการพิจาณาปัญหา และการวิเคราะห์ผลการทำงาน

ข้อมูลที่บันทึกไว้ใน malloc log

Malloc Log จะบันทึกข้อมูลต่อไปนี้สำหรับแต่ละการจัดสรรที่แอ็คทีฟ:

- แอดเดรสที่ส่งกลับไปยังผู้เรียกใช้
- ขนาดของการจัดสรร
- ฮีปที่การจัดสรรได้รับบริการ
- การติดตามย้อนกลับของสแต็กของฟังก์ชันการเรียกใช้ ความลึกของการติดตามย้อนกลับ ที่ถูกบันทึกเป็นอ็อพชันที่ตั้งค่าได้

นอกจากนั้น ยังสามารถบันทึกข้อมูลต่อไปนี้เป็นทางเลือก:

- ID การประมวลผลของกระบวนการการเรียกใช้
- ID เธรดของเธรดการthread
- หมายเลขลำดับของการจัดสรรด้วยการร้องขอไปยังการเริ่มทำงานการประมวลผล
- เวลาจริงที่ทำการจัดสรร

ด้วยการเปิดใช้งาน Malloc Log แต่ละการจัดสรรที่ทำสำเร็จต้องการจำนวนโอเวอร์เฮด เพิ่มเติมที่ใช้เพื่อเก็บ metadata โอเวอร์เฮดนี้ประมาณ 50–100 ไบต์สำหรับแอ็พพลิเคชัน 32 บิต และเป็นสองเท่าสำหรับแอ็พพลิเคชัน 64 บิต จำนวน การเปลี่ยนแปลงโอเวอร์เฮดขึ้นอยู่กับอ็อพชันที่ถูกตั้งค่า

ข้อมูล Malloc Log สามาถเข้าถึงได้ด้วยวิธีต่อไปนี้:

- การใช้คำสั่งย่อย DBX malloc
- การใช้อ็อพชันดีบัก `report_allocations` malloc

การเปิดใช้งาน malloc log

Malloc Log ไม่ถูกเปิดใช้งานเป็นค่าดีฟอลต์ ในการเปิดใช้งาน Malloc Log ด้วย ค่ากำหนดดีฟอลต์ ให้ตั้งค่าตัวแปรสภาวะแวดล้อม **MALLOCDEBUG** ดังนี้:

```
MALLOCDEBUG=log
```

ในการเปิดใช้งาน Malloc Log ด้วยอ็อพชันการตั้งค่าที่ผู้ใช้ระบุ ให้ตั้งค่าตัวแปรสภาวะแวดล้อม **MALLOCDEBUG** ดังนี้:

```
MALLOCDEBUG=log:extended,stack_depth:6
```

หมายเหตุ: ค่า Malloc Log ดีฟอลต์เป็นดังนี้:

extended
ดีฟอลต์เป็น off การระบุอ็อพชันนี้จะเปิดใช้งานการบันทึกการทำงานของ metadata การจัดสรรที่เป็นทางเลือกที่ระบุด้านบน พารามิเตอร์นี้จะมีผล ต่อจำนวนโอเวอร์เฮดที่ต้องการสำหรับแต่ละการจัดสรร This option does not have any effect if `MALLOCTYPE` is set to `watson2`.

stack_depth
ใช้เพื่อระบุความลึกของสแต็กการเรียกใช้ฟังก์ชันที่ถูกบันทึก สำหรับแต่ละการจัดสรร พารามิเตอร์นี้มีผลต่อจำนวนโอเวอร์เฮดที่ต้องการ สำหรับแต่ละการจัดสรร ค่าดีฟอลต์คือ 4 ค่าสูงสุดคือ 64

ข้อจำกัด

ผลการทำงาน-v'โปรแกรมทั้งหมดอาจลดลงได้เมื่อเปิดใช้งาน Malloc Log เนื่องจากต้องเสียที่เก็บข้อมูลในหน่วยความจำ การใช้งานหน่วยความจำจะเพิ่มมากขึ้นเช่นกัน

คุณลักษณะการดีบัก Malloc Log เข้ากันได้กับนโยบายและอ็อพชัน malloc ต่อไปนี้:

- Default Allocation Policy
- Watson Allocation Policy
- Watson2 Allocation Policy

- Malloc Buckets
- Malloc Multiheap
- Malloc Thread Cache
- Malloc Disclaim

หลักการที่เกี่ยวข้อง:

"การจัดสรรหน่วยความจำระบบโดยใช้ระบบย่อย malloc" ในหน้า 696
หน่วยความจำจะถูกจัดสรรให้กับแอ็พพลิเคชันโดยใช้ระบบย่อย **malloc**

Malloc disclaim

Malloc Disclaim เป็นส่วนขยายทางเลือกของระบบย่อย malloc ที่จัดให้ ผู้ใช้มีวิธีเปิดใช้งานการปฏิเสธหน่วยความจำที่ส่งคืนโดยรูทีนย่อย **free** อัตโนมัติ

นี้เป็นประโยชน์สำหรับอินสแตนซ์ ที่การประมวลผลมีการใช้งานพื้นที่การแบ่งหน้าสูง แต่ไม่ได้ใช้หน่วยความจำ อย่างแท้จริง

Malloc Disclaim ไม่ถูกเปิดใช้งานเป็นค่าดีฟอลต์ โดยถูกเปิดใช้งานและตั้งค่า ก่อนการประมวลผลเริ่มทำงานโดยตัวแปรสภาวะแวดล้อม **MALLOCOPTIONS** ดังนี้:

```
MALLOCOPTIONS=disclaim
```

หลักการที่เกี่ยวข้อง:

"การจัดสรรหน่วยความจำระบบโดยใช้ระบบย่อย malloc" ในหน้า 696
หน่วยความจำจะถูกจัดสรรให้กับแอ็พพลิเคชันโดยใช้ระบบย่อย **malloc**

Malloc detect

Malloc Detect จัดให้มีการรายงานความผิดพลาดทางเลือกและส่วนขยายการตรวจหา สำหรับระบบย่อย **malloc** ข้อมูลเกี่ยวกับข้อผิดพลาดที่ เกิดขึ้นในสภาพแวดล้อม malloc จะถูกรายงานและมีการดำเนินการ ถ้าระบุไว้

Malloc Detect สามารถแบ่งออกเป็นความสามารถที่ต่างกันสามส่วน:

- การตรวจหาข้อผิดพลาด
- การรายงานข้อผิดพลาดโดยใช้ stderr
- การรายงานข้อผิดพลาดโดยใช้ฟังก์ชันที่แอ็พพลิเคชันกำหนด

การตรวจหาข้อผิดพลาด

บางข้อผิดพลาดในระบบย่อย **malloc** จะตรวจพบได้ง่าย ข้อผิดพลาดเช่นการปล่อยตัวชี้ซึ่งไม่ใช่แอดเดรสฮีปที่ถูกต้องถูกตรวจพบ แบบซิงโครนัสในโค้ดพาธของการปล่อย อย่างไรก็ตาม ข้อผิดพลาดที่เกิดโดยเหตุการณ์ อะซิงโครนัส เช่นความล้มเหลวฮีป จะตรวจหาได้ยากมากกว่า อ็อพชัน `check_arena` ของ Malloc Detect ถูกออกแบบเพื่อตรวจสอบความล้มเหลวประเภทนี้ในลักษณะซิงโครนัส เมื่อมีการเรียกใช้ทุกครั้งไปยัง API ระบบย่อย **malloc** การตรวจสอบโครงสร้างข้อมูลภายในจะถูกดำเนินการ ถ้าพบความล้มเหลว จะรายงานไปยังแอ็พพลิเคชัน นี้ยังจัดให้มี เวลาอีกจุดหนึ่งซึ่งทำการดีบักปัญหาที่ยุ่งยากเหล่านี้ อ็อพชัน `check_arena` สามารถถูกเปิดใช้งานโดยการตั้งค่าตัวแปรสภาวะแวดล้อม **MALLOCDEBUG** ดังนี้:

```
MALLOCDEBUG=checkarena
```

การรายงานข้อผิดพลาดโดยใช้ stderr

วิธีทั่วไปในการรายงานข้อผิดพลาดในระบบย่อย **malloc** คือการใช้ค่าส่งกลับและตัวแปรสภาวะแวดล้อม errno อ็อพชัน verbose ของ Malloc Detect อนุญาตให้ข้อผิดพลาดเหล่านี้ถูกพิมพ์ออก สตริงข้อผิดพลาดมาตรฐานของแอ็พพลิเคชันโปรแกรม นี้จัดให้มีวิธีที่เห็นได้ชัดเจนยิ่งขึ้น มีรายละเอียดมากขึ้นสำหรับการรายงานข้อผิดพลาดใน malloc อ็อพชันรายละเอียด สามารถเปิดใช้งานได้โดยการตั้งค่า **MALLOCDEBUG** ดังนี้:

```
MALLOCDEBUG=verbose
```

การรายงานข้อผิดพลาดโดยใช้ฟังก์ชันที่แอ็พพลิเคชันกำหนด

Malloc Detect ยังอนุญาตให้ผู้ใช้มีฟังก์ชันที่ระบบย่อย **malloc** จะเรียกใช้เมื่อใดก็ตามที่พบข้อผิดพลาด Malloc Report จะเรียกใช้ฟังก์ชันที่จัดให้มีโดยแอ็พพลิเคชัน รอการส่งกลับ จากนั้นส่งกลับ ตามปกติ สิ่งอำนวยความสะดวกนี้อนุญาตให้แอ็พพลิเคชันดำเนินการงานดีบัก สิ่งที่จำเป็นซึ่งต้องมีก่อนอนุญาตให้โปรแกรมดำเนินการต่อ ในการเปิดใช้งาน สิ่งอำนวยความสะดวกนี้ แอ็พพลิเคชันต้องตั้งค่าของตัวชี้ฟังก์ชันโกลบอล **malloc_err_function** เท่ากับค่าของรูทีนข้อผิดพลาดของ แอ็พพลิเคชัน ตัวอย่างเช่น:

```
extern void (*malloc_err_function)(int, ...)
malloc_err_function = &application_malloc_err_hdl
```

ข้อจำกัด

คุณลักษณะการดีบัก Malloc Detect เข้ากันได้กับนโยบายและอ็อพชัน malloc ต่อไปนี้:

- Default Allocation Policy
- Malloc Buckets Option
- Watson Allocation Policy
- Malloc Multiheap
- Malloc Threadcache
- Malloc Disclaim Option

การกำหนดค่าและการใช้ malloc thread cache

Malloc Thread Cache ดูแลพูลหน่วยความจำที่ไม่ได้จัดสรรของแต่ละเธรด เพื่อลดความขัดแย้งสำหรับโครงสร้างฮีปโกลบอล

แคช นี้พยายามเตรียมจัดสรรส่วนหน่วยความจำสำหรับไว้ใช้ในอนาคตตาม รูปแบบของการจัดสรรที่ดำเนินไว้แล้วโดยเธรด ถ้าสามารถให้บริการ การร้องขอการจัดสรรโดยใช้ส่วนหนึ่งในส่วนที่ยังไม่ได้จัดสรรใน Thread Cache จะ ถูกลบออกจากแคช และส่งกลับไปยังผู้เรียกใช้ ถ้าไม่สามารถให้บริการการร้องขอ การจัดสรรโดยใช้ส่วนที่ยังไม่ได้จัดสรรในแคช การร้องขอ จะถูกส่งไปยังโครงสร้างฮีปโกลบอล

วิธีการจัดสรร Thread cache

ครั้งแรกที่เธรดร้องขอหน่วยความจำขนาดน้อยกว่า 4096 ไบต์ แคชเธรด เตรียมจัดสรรไว้หลาย chunks ของหน่วยความจำที่มีขนาดเดียวกันจากโครงสร้าง ฮีปโกลบอล รวมทั้งสำรองพื้นที่ขนาดใหญ่ของหน่วยความจำเพื่อใช้ในการให้บริการ สำหรับการร้องขอในอนาคต ถ้าเธรดปล่อยฟรีส่วนของหน่วยความจำ ส่วนนั้นจะ ถูกเก็บในแคชเธรดเพื่อการการจัดสรรในอนาคต อย่างไรก็ตาม ถ้าระหว่างการฟรี ขนาดของแคชเธรดเกินค่าเส้นแบ่งที่กำหนด ครึ่งหนึ่งของอิลิเมนต์ ในแคชจะถูกส่งกลับไปยังตัวจัด

สรรสนับสนุน โดยพื้นฐาน ลักษณะการทำงาน ของแคชเธรดสามารถอธิบายได้เป็น "ตัวประมวลผลแบ็ตช์" ซึ่งจัดกลุ่มการเรียกใช้ การจัดสรร/การยกเลิกการจัดสรรแต่ละครั้งจะถูกรันด้วยกันที่เวลาหนึ่ง นี้ส่งผลให้ ช่วยลดความขัดแย้งของฮีปโกลบอล และในหลายๆ กรณีจะช่วยให้มีประสิทธิภาพมากขึ้น

การเปิดใช้งาน malloc thread cache

Malloc Thread Cache ถูกเปิดใช้เป็นค่าดีฟอลต์สำหรับตัวจัดสรร Watson โดย สามารถถูกปิดใช้งานก่อนที่การประมวลผลจะเริ่มทำงานได้โดยการตั้งค่าตัวแปรสภาวะแวดล้อม **MALLOCOPTIONS** ดังนี้:

```
$ MALLOCOPTIONS=threadcache:off
```

Malloc Thread Cache ไม่ถูกเปิดใช้งานเป็นค่าดีฟอลต์สำหรับตัวจัดสรรดีฟอลต์โดยสามารถถูกเปิดใช้งานก่อนหน้าที่การประมวลผลจะเริ่มทำงานได้โดยการตั้งค่าตัวแปรสภาวะแวดล้อม **MALLOCOPTIONS** ดังนี้

```
$ MALLOCOPTIONS=threadcache
```

การเขียนโค้ด reentrant และ threadsafe

ในกระบวนการแบบเธรดเดี่ยว มีได้เพียงหนึ่งโฟลว์ของการควบคุม โค้ดที่เรียกใช้งานโดยกระบวนการเหล่านี้ไม่จำเป็นต้องเป็น reentrant หรือ threadsafe ในโปรแกรมแบบมัลติเธรด ฟังก์ชันเดียวกัน และ รีซอร์สเดียวกันอาจถูกเข้าถึงพร้อมกันโดยโฟลว์ควบคุมหลายโฟลว์

เพื่อปกป้อง integrity ของรีซอร์ส โค้ดที่เขียนขึ้นสำหรับโปรแกรมแบบมัลติเธรด ต้องเป็น reentrant และ threadsafe

reentrance และ thread safety จะเกี่ยวกับวิธีการที่ฟังก์ชันจัดการกับรีซอร์ส การเข้าใหม่ และความปลอดภัยของเธรดเป็นแนวคิดแยกกัน: ฟังก์ชันสามารถเป็น reentrant, threadsafe, ทั้งสอง หรือไม่ใช่ทั้งหมด

ส่วนนี้ให้ข้อมูลเกี่ยวกับการเขียนโปรแกรม reentrant และ threadsafe ซึ่งจะไม่ครอบคลุมหัวข้อของการเขียนโปรแกรมแบบเธรดที่มีประสิทธิภาพ โปรแกรมแบบเธรดที่มีประสิทธิภาพจะเป็นโปรแกรมที่ทำงานแบบขนาน คุณต้องพิจารณาถึงประสิทธิภาพของเธรดในระหว่างที่ออกแบบโปรแกรม โปรแกรมแบบเธรดเดี่ยวที่มีอยู่ สามารถทำเป็นโปรแกรมแบบเธรดที่มีประสิทธิภาพได้ แต่การทำเช่นนี้จำเป็นต้องออกแบบและเขียนโปรแกรมขึ้นใหม่

reentrance

การทำงานแบบ reentrant ไม่ได้พักข้อมูลสแตติกผ่านการเรียกแบบต่อเนื่อง หรือไม่ได้ส่งคืนตัวชี้ไปยังข้อมูลสถิติ ข้อมูลทั้งหมด จะถูกจัดเตรียมไว้โดยตัวเรียกของการทำงาน การทำงานแบบ reentrant ต้องไม่เรียกการทำงานแบบไม่ใช่ reentrant

บ่อยครั้งที่การทำงานที่ไม่ใช่ reentrant สามารถระบุได้โดยอินเตอร์เฟส และการใช้งานที่อยู่ภายนอก แต่ไม่เสมอไป ตัวอย่างเช่น รูทีนย่อย **strtok** ไม่ใช่ reentrant เนื่องจากรูทีนย่อยจะพักสตริงที่แตกอยู่ในโทเค็น รูทีนย่อย **ctime** ยังคงไม่ใช่ reentrant แต่ส่งคืนตัวชี้ไปยังข้อมูลสถิติที่ถูกเขียนทับโดยการเรียกแต่ละครั้ง

Thread safety

ฟังก์ชัน threadsafe ปกป้องรีซอร์สที่แบ่งใช้มิให้เข้าถึง พร้อมกันโดยการล็อก Thread safety จะเกี่ยวข้องกับการนำฟังก์ชันไปใช้งาน และไม่ส่งผลถึงอินเตอร์เฟสภายนอก

ในภาษา C ตัวแปรโลคัลจะถูกจัดสรรไว้ บนสแต็กแบบไดนามิก ดังนั้น ฟังก์ชันใดๆ ที่ไม่ใช่ข้อมูลสแตติก หรือรีซอร์ส ที่แบ่งใช้อื่นๆ จะเป็น threadsafe ดังในตัวอย่างต่อไปนี้:

```
/* threadsafe function */
int diff(int x, int y)
{
        int delta;

        delta = y - x;
        if (delta < 0)
                delta = -delta;

        return delta;
}
}
```

การใช้ข้อมูลโกลบอลคือ thread-unsafe ข้อมูลโกลบอลควรถูกรักษาไว้ต่อเธรด หรือต่อการครอบคลุม ดังนั้น การเข้าถึงสามารถ serialize ได้ เธรดอาจอ่านโค้ดระบุความผิดพลาดที่สอดคล้องกับข้อผิดพลาด ที่มีต้นเหตุมาจากเธรดอื่น ใน AIX เธรดแต่ละตัวมีค่า **errno** เป็นของตนเอง

การสร้างฟังก์ชัน reentrant

ในกรณีส่วนใหญ่ การทำงานแบบไม่ใช่ reentrant ต้องแทนที่ด้วยการทำงานที่มีอินเตอร์เฟสที่ถูกแก้ไขให้เป็นแบบ reentrant การทำงานแบบไม่ใช่ reentrant ไม่สามารถนำมาใช้ได้โดยเธรดที่มีจำนวนมาก นอกนั้นจาก อาจเป็นไปไม่ได้ ที่จะทำให้ฟังกชันที่ไม่ใช่ reentrant เป็น threadsafe

การส่งคืนข้อมูล

การทำงานที่ไม่มี reentrant จำนวนมากจะส่งคืนตัวชี้ไปยังข้อมูลสถิติ ซึ่งสามารถหลีกเลี่ยงได้ด้วยวิธีต่อไปนี้:

- การส่งข้อมูลที่จัดสรรแบบไดนามิก ในกรณีนี้ เป็นความรับผิดชอบต่อตัวเรียก ที่ต้องจัดหาหน่วยเก็บที่ว่าง ประโยชน์คืออินเตอร์เฟสที่ทำไม่ต้องการการแก้ไข อย่างไรก็ตาม ไม่มีความแน่นอนสำหรับความเข้ากันได้แบบย้อนกลับ โปรแกรมสำหรับเธรดเดี่ยวที่มีอยู่โดยใช้ฟังก์ชันที่ถูกแก้ไขโดยไม่มีการแก้ไข จะไม่เพิ่มหน่วยเก็บที่ว่าง และนำไปสู่การขาดแคลนหน่วยความจำ
- การใช้หน่วยเก็บที่จัดเตรียมโดยตัวเรียก เมธอดนี้ขอแนะนำให้ใช้ แม้ว่าอินเตอร์เฟสต้องถูกแก้ไขก็ตาม

ตัวอย่างเช่น ฟังก์ชัน **strtoupper** คือการแปลงสตริงให้เป็นตัวพิมพ์ใหญ่สามารถนำมาใช้กับชิ้นส่วนของโค้ดต่อไปนี้ :

```
/* non-reentrant function */
char *strtoupper(char *string)
{
        static char buffer[MAX_STRING_SIZE];
        int index;

        for (index = 0; string[index]; index++)
                buffer[index] = toupper(string[index]);
        buffer[index] = 0

        return buffer;
}
```

ฟังก์ชันนี้ไม่ใช่ reentrant (หรือ threadsafe) หากต้องการสร้างฟังก์ชันแบบ reentrant โดยส่งคืนข้อมูลที่จัดสรรแล้วแบบไดนามิก ฟังก์ชันนั้นต้องคล้ายกับชิ้นส่วนของโค้ดที่แสดงดังต่อไปนี้:

```
/* reentrant function (a poor solution) */
char *strtoupper(char *string)
{
        char *buffer;
        int index;

        /* error-checking should be performed! */
        buffer = malloc(MAX_STRING_SIZE);

        for (index = 0; string[index]; index++)
                buffer[index] = toupper(string[index]);
        buffer[index] = 0

        return buffer;
}
```

ฟังก์ชันที่ดีกว่าประกอบด้วยการแก้ไขอินเตอร์เฟส ตัวเรียกต้องจัดเตรียมหน่วยเก็บสำหรับสตริงทั้งแบบอินพุต และเอาต์พุต ในชิ้นส่วนของโค้ดต่อไปนี้:

```
/* reentrant function (a better solution) */
char *strtoupper_r(char *in_str, char *out_str)
{
        int index;

        for (index = 0; in_str[index]; index++)
        out_str[index] = toupper(in_str[index]);
        out_str[index] = 0

        return out_str;
}
```

รูทีนย่อยไลบรารีภาษา C มาตรฐานที่ไม่ใช่ reentrant ถูกสร้างให้เป็นแบบ reentrant โดยใช้หน่วยเก็บข้อมูลที่ตัวเรียกจัดเตรียมไว้

การเก็บข้อมูลผ่านการเรียกแบบต่อเนื่อง

ไม่มีข้อมูลที่เก็บไว้ผ่านการเรียกแบบต่อเนื่อง เนื่องจากความแตกต่างของเธรดอาจเรียกฟังก์ชันในแบบต่อเนื่อง ถ้าฟังก์ชันต้องรักษาข้อมูลบางส่วน ผ่านการเรียกในแบบต่อเนื่อง เช่น บัฟเฟอร์การทำงานหรือตัวชี้ ตัวเรียกควรจัดเตรียมข้อมูลนี้

โปรดพิจารณาตัวอย่างต่อไปนี้ ฟังก์ชันจะส่งคืนอักขระตัวพิมพ์เล็กแบบต่อเนื่อง ของสตริง สตริงจะถูกจัดเตรียมเฉพาะกับการเรียกในครั้งแรก ด้วยรูทีนย่อย **strtok** ฟังก์ชันจะส่งคืนค่า 0 เมื่อเข้าถึงจุดสิ้นสุดของสตริง ฟังก์ชันสามารถนำไปใช้ได้ ดังที่แสดงอยู่ในชิ้นส่วนของโค้ดต่อไปนี้:

```
/* non-reentrant function */
char lowercase_c(char *string)
{
        static char *buffer;
        static int index;
        char c = 0;
```

```
        /* stores the string on first call */
        if (string != NULL) {
                buffer = string;
                index = 0;
        }

        /* searches a lowercase character */
        for (; c = buffer[index]; index++) {
                if (islower(c)) {
                        index++;
                        break;
                }
        }
        return c;
}
```

h

ฟังก์ชันนี้ไม่ใช่แบบ reentrant หากต้องการทำเป็นแบบ reentrant ข้อมูลสถิติ นั่นคือตัวแปร **index** ต้องถูกรักษาไว้ด้วยตัวเรียก reentrant ในเวอร์ชันของฟังก์ชันที่สามารถนำมาใช้ได้ ดังที่แสดงอยู่ในชิ้นส่วนของโค้ดต่อไปนี้:

```
/* reentrant function */
char reentrant_lowercase_c(char *string, int *p_index)
{
        char c = 0;

        /* no initialization - the caller should have done it */

        /* searches a lowercase character */
        for (; c = string[*p_index]; (*p_index)++) {
                if (islower(c)) {
                        (*p_index)++;
                        break;
                  }
        }
        return c;
}
```

อินเตอร์เฟสของฟังก์ชันจะเปลี่ยนไป และทำให้การใช้เปลี่ยนไปด้วย ตัวเรียกต้องจัดเตรียมสตริงสำหรับการเรียกแต่ละครั้ง และต้องกำหนดค่าเริ่มต้นของดัชนีให้มีค่า 0 ก่อนการเรียกในครั้งแรก ดังที่แสดงไว้ในชิ้นส่วนของโค้ดต่อไปนี้:

```
char *my_string;
char my_char;
int my_index;
...
my_index = 0;
while (my_char = reentrant_lowercase_c(my_string, &my_index)) {
        ...
}
```

การทำให้ฟังก์ชันเป็น threadsafe

ในโปรแกรมแบบมัลติเธรด ฟังก์ชันทั้งหมดที่เรียกใช้โดยหลายเธรด ต้องเป็น threadsafe อย่างไรก็ตาม วิธีแก้ปัญหาจะมีสำหรับการใช้รูทีนย่อยที่ thread-unsafe ในโปรแกรมแบบมัลติเธรด ฟังก์ชันที่ไม่ใช่ reentrant โดยปกติ เป็น thread-unsafe แต่การทำให้เป็น reentrant มักทำให้เป็น threadsafe เช่นกัน

การล็อกรีซอร์สแบบแบ่งใช้

ฟังก์ชันที่ใช้ข้อมูลสแตติก หรือรีซอร์สที่แบ่งใช้ใดๆ เช่น ไฟล์หรือเทอร์มินัล ต้อง serialize การเข้าถึงรีซอร์สเหล่านี้โดยการล็อกเพื่อให้เป็น threadsafe ตัวอย่างเช่น ฟังก์ชันต่อไปนี้เป็นแบบ thread-unsafe:

```
/* thread-unsafe function */
int increment_counter()
{
        static int counter = 0;

        counter++;
        return counter;
}
```

เมื่อต้องการให้เป็น threadsafe ตัวแปรสแตติก **counter** ต้องถูกป้องกัน โดยสแตติกล็อก ดังในตัวอย่างต่อไปนี้:

```
/* pseudo-code threadsafe function */
int increment_counter();
{
        static int counter = 0;
        static lock_type counter_lock = LOCK_INITIALIZER;

        pthread_mutex_lock(counter_lock);
        counter++;
        pthread_mutex_unlock(counter_lock);
        return counter;
}
```

ในแอ็พพลิเคชันโปรแกรมแบบมัลติเธรดที่ใช้ไลบรารีเธรด mutexes ควรถูกใช้สำหรับการ serialize รีซอร์สที่แบ่งใช้ไลบรารีที่เป็นอิสระ อาจจำเป็นต้องทำงานอยู่ภายนอกบริบทของเธรด และใช้ล็อกชนิดอื่น

การแก้ไขปัญหาสำหรับฟังก์ชัน thread-unsafe

มีความเป็นไปได้ที่จะใช้วิธีแก้เพื่อใช้ฟังก์ชันแบบ thread-unsafe ที่ถูกเรียกโดยเธรดจำนวนมาก ซึ่งจะเป็นประโยชน์โดยเฉพาะเมื่อใช้ไลบรารี thread-unsafe ในโปรแกรมแบบมัลติเธรด สำหรับการทดสอบหรือ ขณะรอเวอร์ชัน threadsafe ของไลบรารีให้พร้อมใช้งาน วิธีแก้ นำไปสู่การนำไปใช้ในบางส่วน เนื่องจากมีการ serialize ฟังก์ชันทั้งหมดหรือกลุ่มของฟังก์ชัน ต่อไปนี้คือวิธีแก้ที่เป็นไปได้:

- ใช้ล็อกแบบโกลบอลสำหรับไลบรารี และล็อกไว้ในแต่ละครั้งที่คุณใช้ไลบรารี (การเรียกไลบรารีรูทีน หรือการใช้ตัวแปรไลบรารีโกลบอล) โซลูชันนี้ สามารถสร้างผลการทำงานที่เป็นคอขวดได้ เนื่องจากมีเพียงเธรดเดียวเท่านั้นที่สามารถเข้าถึงส่วนใดๆ ของไลบรารี ณ เวลาที่กำหนดไว้ใดๆ โซลูชันในโค้ดจำลองต่อไปนี้ สามารถยอมรับได้ หากไลบรารีถูกเข้าถึงวิธีแก้ที่นำมาใช้ นานๆ ครั้ง หรือขณะที่เริ่มต้น

  ```
  /* this is pseudo code! */

  lock(library_lock);
  library_call();
  unlock(library_lock);
  ```

```
lock(library_lock);
x = library_var;
unlock(library_lock);
```

- ใช้ล็อกสำหรับคอมโพเนนต์ไลบรารีแต่ละตัว (ตัวแปรรูทีนหรือตัวแปรโกลบอล) หรือกลุ่มของคอมโพเนนต์ โซลูชันนี้คือสิ่งที่ซับซ้อนในการนำไปปฏิบัติ มากกว่าตัวอย่างก่อนหน้านี้ แต่สามารถปรับปรุงผลการทำงานได้ เนื่องจากวิธีแก้นี้ ควรนำมาใช้ในแอ็พพลิเคชันโปรแกรมและไม่ได้ใช้ในไลบรารี mutexes สามารถนำมาใช้สำหรับการล็อกไลบรารีได้

```
/* this is pseudo-code! */

lock(library_moduleA_lock);
library_moduleA_call();
unlock(library_moduleA_lock);

lock(library_moduleB_lock);
x = library_moduleB_var;
unlock(library_moduleB_lock);
```

ไลบรารี Reentrant และ threadsafe

ไลบรารี Reentrant และ threadsafe เป็นประโยชน์ในขอบเขตที่กว้างของ สภาวะแวดล้อมการโปรแกรมแบบขนาน (และอะซิงโครนัส) ไม่เพียงภายใน เธรด โดยเป็นแนวทางโปรแกรมมิงที่ดีที่จะใช้และเขียนฟังก์ชัน reentrant และ threadsafe เสมอ

การใช้ไลบรารี

หลายๆ ไลบรารีที่มากับ AIX Base Operating System เป็น threadsafe ในเวอร์ชันปัจจุบันของ AIX ไลบรารีต่อไปนี้ เป็น threadsafe:

- ไลบรารี C มาตรฐาน (**libc.a**)
- ไลบรารีที่เข้ากันได้แบบ berkeley (**libbsd.a**)

รูทีนย่อยใน C มาตรฐานบางตัวไม่ใช่ reentrant เช่น รูทีนย่อย **ctime** and **strtok** reentrant ในเวอร์ชันของรูทีนย่อยมีชื่อของรูทีนย่อยเดิมที่มีคำต่อท้าย **_r** (ขีดเส้นใต้แล้วตามด้วยตัวอักษร *r*)

เมื่อเขียนโปรแกรมแบบมัลติเธรด ให้ใช้เวอร์ชัน reentrant ของรูทีนย่อยแทนเวอร์ชันต้นฉบับ ตัวอย่างเช่น ชิ้นส่วนของโค้ดต่อไปนี้:

```
token[0] = strtok(string, separators);
i = 0;
do {
        i++;
        token[i] = strtok(NULL, separators);
} while (token[i] != NULL);
```

ควรแทนด้วยโปรแกรมแบบมัลติเธรดโดยส่วนของ โค้ดต่อไปนี้:

```
char *pointer;
...
token[0] = strtok_r(string, separators, &pointer);
i = 0;
```

```
do {
        i++;
        token[i] = strtok_r(NULL, separators, &pointer);
} while (token[i] != NULL);
```

ไลบรารีแบบ thread-unsafe อาจไม่ถูกนำมาใช้โดยเธรดเดียวในโปรแกรม โปรดมั่นใจว่า ความไม่เป็นลักษณะเฉพาะของเธรดที่ใช้ไลบรารี มิฉะนั้นโปรแกรมจะมีลักษณะการทำงานที่ไม่คาดการณ์ไว้ หรืออาจหยุดทำงาน

การแปลงไลบรารี

พิจารณาสิ่งต่อไปนี้เมื่อแปลงไลบรารีที่มีอยู่ไปเป็น ไลบรารี reentrant และ threadsafe ข้อมูลนี้จะใช้กับไลบรารีในภาษา C เท่านั้น

- ระบุตัวแปรโกลบอลที่ถูกเอ็กซ์พอร์ต ตัวแปรเหล่านั้นจะถูกกำหนดในไฟล์ส่วนหัวด้วยคีย์เวิร์ด **export** ตัวแปรโกลบอลที่ถูกเอ็กซ์พอร์ต ควรถูกครอบคลุมไว้ ตัวแปรควรถูกทำให้เป็นส่วนตัว (กำหนดโดยคีย์เวิร์ด **static** ในซอร์สโค้ดไลบรารี) และเข้าถึงรูทีนย่อย (อ่านและเขียน) ควรถูกสร้างขึ้น
- ระบุตัวแปรสแตติก และรีซอร์สแบ่งใช้อื่นๆ ตัวแปรสแตติก จะถูกกำหนดด้วยคีย์เวิร์ด **static** ล็อกควรเชื่อมโยงกับ รีซอร์สแบ่งใช้ใดๆ เศษของการล็อก คือการเลือกจำนวนของล็อก จะได้รับผลกระทบกับผลการทำงานของไลบรารี หากต้องการกำหนดค่าเริ่มต้นให้กับล็อก ตัวช่วยกำหนดค่าเริ่มต้นเพียงครั้งเดียวอาจถูกนำมาใช้
- ระบุการทำงานที่ไม่ใช่ reentrant และทำเป็นแบบ reentrant สำหรับ ข้อมูลเพิ่มเติม โปรดดูที่ การสร้าง ฟังก์ชัน Reentrant
- ระบุฟังก์ชัน thread-unsafe และทำให้เป็น threadsafe สำหรับ ข้อมูลเพิ่มเติม โปรดดูที่ การทำให้ฟังก์ชันเป็น threadsafe

หลักการที่เกี่ยวข้อง:

"การกำหนดค่าเริ่มต้นครั้งเดียว" ในหน้า 510

บางไลบรารี C ถูกออกแบบสำหรับการกำหนดค่าเริ่มต้น แบบไดนามิก ซึ่งการกำหนดค่าเริ่มต้นโกลบอลสำหรับไลบรารีถูกดำเนินการ เมื่อโพรซีเดอร์แรกในไลบรารีถูกเรียก

ข้อมูลที่เกี่ยวข้อง:

admin

cdc

เดลต้า

get

prs

sccsdiff

sccsfile

การสร้างแพ็กเกจซอฟต์แวร์สำหรับการติดตั้ง

หัวข้อนี้ให้ข้อมูลเกี่ยวกับการจัดเตรียมแอ็พพลิเคชัน ที่จะติดตั้งด้วยคำสั่ง **installp**

ส่วนนี้อธิบายรูปแบบและเนื้อหาของแพ็กเกจการติดตั้งผลิตภัณฑ์ซอฟต์แวร์ ที่ต้องถูกจัดเตรียมโดยผู้พัฒนาผลิตภัณฑ์ ซึ่งให้คำอธิบายเกี่ยวกับไฟล์ที่จำเป็น และที่เป็นทางเลือก ที่เป็นส่วนหนึ่งของการติดตั้งซอฟต์แวร์ หรือแพ็กเกจการอัพเดต

แพ็กเกจการติดตั้งผลิตภัณฑ์ซอฟต์แวร์เป็นไฟล์รูปแบบการสำรองข้อมูล ซึ่งมีไฟล์ของผลิตภัณฑ์ซอฟต์แวร์ ไฟล์ควบคุม การติดตั้งที่จำเป็น และไฟล์กำหนดการติดตั้งเองที่เป็นทางเลือก คำสั่ง **installp** ถูกใช้เพื่อติดตั้งและปรับปรุงผลิตภัณฑ์ ซอฟต์แวร์

แพ็กเกจการติดตั้ง มียูนิตที่จัดกลุ่มแบบโลจิคัล ติดตั้งได้ และแยกต่างหาก เรียกว่า *filesets* แต่ละ fileset ในแพ็กเกจต้องเป็นของผลิตภัณฑ์เดียวกัน

การอัพเดต fileset หรือ *แพ็กเกจการอัพเดต* คือแพ็กเกจซึ่ง มีโมดิฟิเคชันใน fileset ที่มีอยู่

ตลอดทั้งหัวข้อนี้ คำว่า *ระบบมาตรฐาน* จะถูกใช้ เพื่ออ้างถึงระบบที่ไม่ได้ถูกกำหนดค่าเป็นระบบแบบไม่ใช้ดิสก์

หมายเหตุ: ถ้าเอกสารออนไลน์ของคุณถูกเขียนด้วย HTML คุณควร ลงทะเบียนเอกสารของคุณกับ AIX Information Center The AIX Information Center provides navigation and search capabilities for system documentation. For instructions on how to use the information center, select **Information Center** Help from the navigation bar. For instructions on installing and configuring the information center on AIX, see the *AIX 5L™ Version 5.3 Installation Guide and Reference*.

ข้อกำหนดโพรซีเดอร์การติดตั้ง

- การติดตั้งต้อง *ไม่* จำเป็น ต้องมีการตอบสนองจากผู้ใช้ การกำหนดคอนฟิกผลิตภัณฑ์ที่ต้องการการโต้ตอบของผู้ใช้ ต้องเกิดขึ้นก่อน หรือหลังการติดตั้ง
- การติดตั้งทั้งหมดของ filesets อิสระหรือการปรับปรุง กับ filesets อิสระต้องสามารถกระทำได้ระหว่างการติดตั้งเดียว
- การติดตั้งไม่ควรจะต้องมีการเริ่มระบบใหม่ การติดตั้งจะหยุดทำงานส่วนของระบบที่เกี่ยวข้องกับการติดตั้ง และระบบจำเป็นต้องรีสตาร์ทหลังการติดตั้งเพื่อให้ การติดตั้งมีผลเต็มที่

ข้อกำหนดข้อมูลการควบคุมแพ็กเกจ

ข้อมูลการควบคุมแพ็กเกจจะต้อง:

- ระบุข้อกำหนดการติดตั้งทั้งหมดที่ filesets มีบน filesets อื่น
- ระบุข้อกำหนดขนาดระบบไฟล์สำหรับ การติดตั้ง fileset

รูปแบบของแพ็กเกจซอฟต์แวร์

แพ็กเกจการติดตั้งหรือการปรับปรุง ต้องเป็นไฟล์เดี่ยว ในรูปแบบการสำรองข้อมูลที่สามารถถูกคืนค่าได้โดยคำสั่ง **installp** ระหว่างการติดตั้ง ไฟล์นี้สามารถถูก แจกจ่ายผ่านเทป ดิสเก็ต หรือซีดีรอม

ข้อกำหนดการแบ่งพาร์ติชันแพ็กเกจ

เพื่อสนับสนุนไคลเอ็นต์เวิร์กสเตชันแบบไร้ดิสก์หรือไร้ข้อมูล ส่วนที่เจาะเพาะเครื่อง ของแพ็กเกจ (*ส่วน root*) ต้องถูกแยกออกจากส่วนที่ใช้ร่วมกันได้ระหว่างเครื่องของแพ็กเกจ (*ส่วน usr*) ส่วน usr ของแพ็กเกจมีไฟล์ซึ่งอยู่ใน ระบบไฟล์ **/usr** หรือ **/opt**

การติดตั้งของส่วน root ของแพ็กเกจต้องไม่เปลี่ยนแปลงไฟล์ใดๆ ในระบบไฟล์ **/usr** ระบบไฟล์ **/usr** ไม่สามารถเขียนได้ระหว่างการติดตั้งส่วน root ของระบบไคลเอ็นต์ แบบไร้ดิสก์หรือไร้ข้อมูล ส่วนจำเพาะ–เครื่อง (root) ควรรวม ข้อมูลทั้งหมดที่ไม่ได้อยู่ในระบบไฟล์ **/usr** หรือ **/opt**

การทำแพ็กเกจสำหรับเวิร์กโหลดพาร์ติชัน

บางผลิตภัณฑ์ซอฟต์แวร์ต้องการการพิจารณาเป็นพิเศษ เมื่อเป็นแพ็กเกจสำหรับ workload partitioning (WPAR) เพื่อให้ผลิตภัณฑ์ ซอฟต์แวร์ปรับใช้ใน WPAR ได้สำเร็จ ต้องถูกแพ็กเกจโดยที่ ต้องไม่พยายามเขียนลงในระบบไฟล์ **/usr** หรือ **/opt** ระหว่างส่วน root ของการประมวลผล เนื่องจาก WPAR เมาท์ ระบบไฟล์เหล่านั้นในโหมดอ่านอย่างเดียว คล้ายๆ กัน การกำหนดคอนฟิกใดๆ ที่จะถูกดำเนินการในแต่ละระบบที่มีผลิตภัณฑ์ถูกติดตั้ง ต้องถูกดำเนินการจากส่วน root ของแพ็กเกจ

ถ้าชุดไฟล์ไม่ได้มีจุดประสงค์เพื่อติดตั้ง ลงในพาร์ติชันวิร์กโหลด ชุดไฟล์ต้องถูกกำหนดด้วยแอ็ตทริบิวต์ PRIVATE ในไฟล์ **lpp_name** ของแพ็กเกจ

ถ้า fileset ต้องถูกกำหนดคอนฟิกแตกต่างกัน เมื่อติดตั้งภายใน WPAR สคริปต์การแพ็กเกจจะตรวจสอบตัวแปรสภาวะแวดล้อม `INUWPAR` เพื่อพิจารณาว่า fileset กำลังถูกติดตั้งภายใน WPAR หรือไม่

ถ้า fileset ถูกกำหนดคอนฟิกแตกต่างกันเมื่อติดตั้ง ลงใน WPAR จะถูกกำหนดคอนฟิกใหม่เมื่อสร้าง WPAR จาก สำเนาระบบ เนื่องจาก fileset ไม่ถูกติดตั้งลงใน WPAR ในตอนต้น เจ้าของ Fileset สามารถสร้างโปรแกรมในไดเร็กทอรี `/usr/lib/wpars/wparconvert.d/usr` และ `/usr/lib/wpars/wparconvert.d/root` ซึ่งจะถูกรันเมื่อมีการแปลงส่วน *usr* และ root จาก fileset ที่จะรันภายใน WPAR สำเนาระบบ ไฟล์ที่สามารถเรียกทำงานได้ทั้งหมด ภายในไดเร็กทอรีเหล่านั้นจะถูกเรียกทำงานเรียงตามลำดับตัวอักษร (ภาษา C) เมื่อสำเนาระบบ WPAR เริ่มทำงานครั้งแรก

ข้อมูลผลิตภัณฑ์สำคัญของซอฟต์แวร์

ข้อมูลเกี่ยวกับผลิตภัณฑ์ซอฟต์แวร์และตัวเลือกที่ติดตั้งได้ถูกรักษาไว้ใน ฐานข้อมูล Software Vital Product Data (SWVPD) SWVPD ประกอบด้วย ชุดของคำสั่งและคลาสอ็อบเจ็กต์ Object Data Manager (ODM) สำหรับ ดูแลรักษาข้อมูลผลิตภัณฑ์ซอฟต์แวร์ คำสั่ง SWVPD ถูกจัดเตรียมสำหรับ ผู้ใช้เพื่อเคียวรี (**lslpp**) และตรวจสอบ (**lppchk**) ผลิตภัณฑ์ ซอฟต์แวร์ที่ติดตั้ง คลาสอ็อบเจ็กต์ ODM กำหนดขอบเขตและรูปแบบของ ข้อมูลผลิตภัณฑ์ซอฟต์แวร์ที่ถูกดูแล

คำสั่ง **installp** ใช้ Object Data Manager เพื่อรักษาข้อมูลดังต่อไปนี้ในฐานข้อมูล SWVPD:

- ชื่อของผลิตภัณฑ์ซอฟต์แวร์ (ตัวอย่างเช่น bos.adt)
- รุ่นของผลิตภัณฑ์ซอฟต์แวร์
- ระดับรีลีสของผลิตภัณฑ์ซอฟต์แวร์ ซึ่งระบุการเปลี่ยนแปลงกับ อินเตอร์เฟสโปรแกรมมิงภายนอกของผลิตภัณฑ์ซอฟต์แวร์
- ระดับการเปลี่ยนแปลงของผลิตภัณฑ์ซอฟต์แวร์ ซึ่งระบุการเปลี่ยนแปลง ที่ไม่มีผลกับอินเตอร์เฟสภายนอกของผลิตภัณฑ์ซอฟต์แวร์
- ระดับคงที่ของผลิตภัณฑ์ซอฟต์แวร์ ซึ่งระบุการปรับปรุงเล็กน้อย ที่ถูก สร้างลงในระดับการเปลี่ยนแปลงปกติในภายหลัง
- ชื่อ checksum และขนาดของไฟล์ที่สร้างผลิตภัณฑ์ซอฟต์แวร์ หรือตัวเลือก
- สถานะของผลิตภัณฑ์ซอฟต์แวร์: available, applying, applied, committing, committed, rejecting หรือ broken
- ระดับเทคโนโลยีและข้อมูล APAR
- ไดเร็กทอรีปลายทางและโปรแกรมติดตั้งสำหรับซอฟต์แวร์ที่เป็นแพ็กเกจแบบไม่มี installp สามารถใช้ได้

ส่วนการสร้างแพ็กเกจผลิตภัณฑ์ซอฟต์แวร์

เพื่อสนับสนุนการติดตั้งในสภาพแวดล้อมแบบ ไคลเอ็นต์/เซิร์ฟเวอร์ การแพ็กเกจการติดตั้งถูกแบ่งออกเป็นส่วนต่างๆ ดังนี้:

usr
มีส่วนของผลิตภัณฑ์ซึ่งสามารถใช้ร่วมกันระหว่างหลายเครื่อง ที่มีสถาปัตยกรรมฮาร์ดแวร์เข้ากันได้ สำหรับระบบมาตรฐาน ไฟล์เหล่านี้ ถูกเก็บในแผนผังไฟล์ **/usr** or **/opt**

root
มีส่วนของผลิตภัณฑ์ซึ่งไม่สามารถแบ่งใช้ระหว่างเครื่องได้ แต่ละไคลเอ็นต์ต้องมีสำเนาของตัวเอง ส่วนใหญ่ซอฟต์แวร์นี้ต้องการสำเนา แยกสำหรับแต่ละเครื่องถูกเชื่อมโยงกับการกำหนดค่าของเครื่อง หรือผลิตภัณฑ์ สำหรับระบบมาตรฐาน ไฟล์ในส่วน root ถูกเก็บใน แผนผังไฟล์ root (/) ส่วน root ของ fileset ต้อง อยู่ในแพ็กเกจเดียวกันกับส่วน usr ของ fileset ถ้า fileset มี ส่วน root จะต้องมีส่วน usr

คู่มือระบบไฟล์ตัวอย่างสำหระบการแบ่งพาร์ติชันแพ็กเกจ

ต่อไปนี้เป็นคำอธิบายสั้นๆ ของระบบไฟล์และ ไดเร็กทอรี คุณสามารถใช้ข้อมูลนี้เป็นแนวทางสำหรับการแยกแพ็กเกจผลิตภัณฑ์ออกเป็นส่วน root, usr และ share

บางส่วนของไดเร็กทอรี root-part และเนื้อหา:

/dev
ไฟล์อุปกรณ์เครื่องที่ใช้อยู่

/etc
ไฟล์การกำหนดค่าเครื่องเช่น **hosts** และ **passwd**

/sbin
ยูทิลิตีระบบที่จำเป็นในการบูตเครื่อง

/var
ไฟล์ข้อมูลและไฟล์บันทึกจำเพาะระบบ

บางส่วนของไดเร็กทอรี usr-part และเนื้อหาประกอบด้วย:

/usr/bin
คำสั่งและสคริปต์ (ordinary executables)

/usr/sbin
คำสั่งการดูแลระบบ

/usr/include
Include files

/usr/lib
ไลบรารี คำสั่งที่ไม่ใช่สำหรับผู้ใช้ และข้อมูลตามระบบสถาปัตยกรรม

/opt
ไลบรารี คำสั่งที่ไม่ใช่สำหรับผู้ใช้ และสคริปต์ปกติที่เกี่ยวข้อง กับผลิตภัณฑ์ที่ไม่ใช่ระบบปฏิบัติการ

หลักการตั้งชื่อแพ็กเกจและชุดไฟล์

ใช้ระเบียบดังต่อไปนี้เมื่อตั้งชื่อซอฟต์แวร์แพ็กเกจ และ filesets:

- ชื่อแพ็กเกจ (*PackageName*) ควรเริ่มด้วย ชื่อระบบ ถ้าแพ็กเกจมีเพียงหนึ่ง fileset ที่ติดตั้งได้ ชื่อ fileset สามารถเหมือนกับ *PackageName* ชื่อแพ็กเกจทั้งหมด ต้องไม่ซ้ำ
- ชื่อ fileset มีฟอร์ม:

  ```
  ProductName.PackageName.FilesetName.extension
  ```

  โดยที่:
  - `ProductName` ระบุผลิตภัณฑ์หรือกลุ่มวิธีแก้ปัญหา
  - `PackageName` ระบุกลุ่มการทำงานภายในผลิตภัณฑ์
  - `FilesetName` (เป็นทางเลือก) ระบุชุดการทำงานจำเพาะ ของไฟล์และไลบรารีที่จะถูกติดตั้ง
  - ส่วนขยาย (เป็นทางเลือก) คำอธิบายเนื้อหาเพิ่มเติม
- ชื่อ fileset มีมากกว่าหนึ่งอักขระและเริ่มต้นด้วยตัวอักษร
- อักขระทั้งหมดในชื่อ fileset ต้องเป็นอักขระ ASCII อักขระ ที่ถูกต้องคือตัวอักษรตัวพิมพ์เล็กและตัวพิมพ์เล็ก ตัวเลข เครื่องหมายขีดเส้นใต้ (_), เครื่องหมายบวก (+) และเครื่องหมายลบ จุด (.) ใช้เป็นตัวคั่น ในชื่อ fileset
- ชื่อ fileset ไม่สามารถจบด้วยมหัพภาคหรือจุด
- ความยาวสูงสุดสำหรับชื่อ fileset คือ 144 ไบต์
- ชื่อ fileset ทั้งหมดต้องไม่ซ้ำภายในแพ็กเกจ

หลักการตั้งชื่อส่วนขยายของชุดไฟล์

ส่วนขยาย	คำอธิบาย Fileset
.adt	ชุดเครื่องมือพัฒนาแอ็พพลิเคชัน
.com	โค้ดทั่วไปที่จำเป็นต่อ filesets ที่เหมือนกัน
.compat	โค้ดความเข้ากันได้ที่อาจถูกเอาออกในรีลีสในอนาคต
.diag	การสนับสนุนการวินิจฉัย
.fnt	แบบอักษร
.help. *Language*	ไฟล์วิธีใช้ Common Desktop Environment (CDE) สำหรับภาษาเฉพาะ
.loc.language	Locale
.msg. *Language*	ไฟล์ข้อความสำหรับภาษาเฉพาะ
.rte	สภาวะแวดล้อมรันไทม์หรือ ชุดขั้นต่ำสำหรับผลิตภัณฑ์
.ucode	Microcode

สิ่งที่ต้องพิจารณาเกี่ยวกับการตั้งชื่อพิเศษสำหรับการสร้างแพ็กเกจไดรเวอร์อุปกรณ์

คำสั่งตัวจัดการการกำหนดค่า (**cfgmgr**) ติดตั้งการสนับสนุนซอฟต์แวร์โดยอัตโนมัติ สำหรับอุปกรณ์ที่สามารถตรวจพบได้ ซึ่งมีอยู่บนสื่อบันทึกการติดตั้งและได้ถูกแพ็กเกจพร้อมกับระเบียบ การตั้งชื่อดังต่อไปนี้:

`devices.`*`BusTypeID.CardID.Etension`*

โดยที่:

- *`BusTypeID`* ระบุชนิดของบัส ซึ่งการ์ดเชื่อมต่อ (ตัวอย่างเช่น **pci** for PCI
- *`CardID`* ระบุตัวระบุเลขฐานสิบหก เฉพาะที่สัมพันธ์กับชนิดการ์ด
- *`Extension`* ระบุส่วนของไดรเวอร์ ที่ถูกรวมมาด้วย (ตัวอย่างเช่น rte อาจเป็นส่วนย่อสำหรับ runtime และ diag เป็นส่วนย่อสำหรับ diagnostics)

ตัวอย่างเช่น สมมติว่าอุปกรณ์อีเทอร์เน็ตเชื่อมต่อกับบัส PCI และถูกระบุโดยตัวจัดการการกำหนดคอนฟิกให้มีตัวบ่งชี้ การ์ดเฉพาะเป็น 1410bb02 แพ็กเกจของ filesets ที่สัมพันธ์กับอุปกรณ์อีเทอร์เน็ตนี้จะมีชื่อเป็น **devices.pci.1410bb02** fileset สภาพแวดล้อมรันไทม์ภายในแพ็กเกจนี้จะมีชื่อเป็น **devices.pci.1410bb02.rte**

สิ่งที่ต้องพิจารณาเกี่ยวกับการตั้งชื่อพิเศษสำหรับการสร้างแพ็กเกจแค็ตตาล็อกข้อความ

ผู้ใช้ที่ติดตั้งแพ็กเกจสามารถร้องขอให้ติดตั้งแค็ตตาล็อก ข้อความโดยอัตโนมัติ เมื่อมีการร้องขอนี้ ระบบจะติดตั้ง filesets ข้อความโดยอัตโนมัติสำหรับภาษาหลัก ถ้า filesets ข้อความ มีอยู่บนสื่อบันทึกการติดตั้งและถูกแพ็กเกจมาด้วย ระเบียบการตั้งชื่อดังต่อไปนี้:

Product.msg.*Language*.*SubProduct*

ส่วนเติมท้าย *SubProduct* ที่เป็นทางเลือกถูกใช้เมื่อผลิตภัณฑ์มี หลาย filesets แค็ตตาล็อกข้อความสำหรับภาษาเดียวกัน แต่ละ fileset แค็ตตาล็อกข้อความใช้กับ *SubProduct* ต่างกัน คุณสามารถเลือก เพื่อใช้หนึ่ง fileset ข้อความกับ ผลิตภัณฑ์ทั้งหมด

ตัวอย่างเช่นผลิตภัณฑ์ Super_Widget มีตัวเลือกชุดของ fileset plastic และ metal แค็ตตาล็อกข้อความ Super_Widget ภาษาอังกฤษ สหรัฐทั้งหมดสามารถแพ็กเกจใน fileset เดียวชื่อ Super_Widget.msg.en_US ถ้าจำเป็นต้องใช้ filesets แค็ตตาล็อกข้อความแยกกันสำหรับอ็อพชัน plastic และ metal filesets แค็ตตาล็อกข้อความภาษาอังกฤษสหรัฐจะชื่อ Super_Widget.msg.en_US.plastic และ Super_Widget.msg.en_US.metal

หมายเหตุ: fileset ข้อความที่ปรับให้เข้ากับระเบียบการตั้งชื่อนี้ *ต้อง* มี installed-requisite (**instreq**) บน fileset อื่นผลิตภัณฑ์ เพื่อประกันการติดตั้งอัตโนมัติ ของ fileset ข้อความ

File names

ไฟล์ที่ส่งมอบด้วยซอฟต์แวร์แพ็กเกจต้องไม่มี เครื่องหมายจุลภาคหรือโคลอน เครื่องหมายจุลภาคหรือโคลอนถูกใช้เป็นตัวแยก ในไฟล์ควบคุมที่ใช้ระหว่างกระบวนการติดตั้งซอฟต์แวร์ชื่อไฟล์ มีอักขระที่ไม่ใช่ ASCII ได้ พาธเต็มสำหรับชื่อไฟล์ต้องไม่มาก เกินกว่า 128 อักขระ

การระบุระดับการปรับรุ่นชุดไฟล์

ระดับ fileset ถูกอ้างอิงเป็น *ระดับ* หรืออีกแบบเป็น *v.r.m.f* หรือ *VRMF* และมีรูปแบบ:

Version.Release.Modification.FixLevel

โดยที่:

- *Version* คือเขตข้อมูลตัวเลข 1 ถึง 2 หลักที่ ระบุหมายเลขเวอร์ชัน
- *Release* คือเขตข้อมูลตัวเลข 1 ถึง 2 หลักที่ ระบุหมายเลขรีลีส
- *Modification* เป็นเขตข้อมูลตัวเลข 1 ถึง 4 หลัก ที่ระบุระดับการปรับเปลี่ยน
- *FixLevel* คือเขตข้อมูลตัวเลข 1 ถึง 4 หลัก ที่ระบุระดับการแก้ไข

ระดับการติดตั้ง fileset ฐาน คือระดับการติดตั้งเริ่มต้นแบบเต็มของ fileset ระดับนี้มี ไฟล์ทั้งหมดใน fileset ตรงกันข้ามกับการปรับปรุง fileset ซึ่งอาจมี เซ็ตย่อยของไฟล์จาก fileset สมบูรณ์

filesets ทั้งหมดในซอฟต์แวร์แพ็กเกจควรมี ระดับ fileset เดียวกัน แม้ว่าจะไม่จำเป็นสำหรับแพ็กเกจ AIX 4.1-formatted

สำหรับระดับใหม่ทั้งหมดของ fileset ระดับของ fileset ต้องเพิ่มขึ้น คำสั่ง **installp** ใช้ระดับ fileset เพื่อตรวจสอบระดับต่อมาของผลิตภัณฑ์บนการติดตั้งที่ตามมา

การมาก่อนของระดับ fileset อ่านจากซ้ายไปขวา (ตัวอย่างเช่น `5.2.0.0` ใหม่กว่า `4.3.0.0`)

เนื้อหาของแพ็กเกจซอฟต์แวร์

ส่วนนี้อธิบายไฟล์ที่มีในแพ็กเกจ การติดตั้งหรืออัพเดต ชื่อพาธของไฟล์ถูกกำหนดสำหรับชนิดแพ็กเกจการติดตั้ง สำหรับแพ็กเกจปรับปรุง ซึ่ง *PackageName* เป็นส่วนหนึ่ง ของชื่อพาธ ถูกแทนที่โดย *PackageName*/*FilesetName*/*FilesetLevel*

พาร์ท usr ของแพ็กเกจการติดตั้งหรือปรับปรุงมี ไฟล์ควบคุมการติดตั้งดังต่อไปนี้:

- **./lpp_name**: ไฟล์นี้ให้ข้อมูลเกี่ยวกับ ซอฟต์แวร์แพ็กเกจที่จะถูกติดตั้งหรือปรับปรุง สำหรับเหตุผลด้านประสิทธิภาพ ไฟล์ **lpp_name** ควรเป็นไฟล์แรกในไฟล์ backup-format ที่สร้างแพ็กเกจการติดตั้งซอฟต์แวร์
- **./usr/lpp/*PackageName*/liblpp.a**: ไฟล์เก็บถาวรนี้มีไฟล์ควบคุมที่ใช้โดยกระบวนการติดตั้ง สำหรับการติดตั้งหรือการปรับปรุงส่วน usr ของซอฟต์แวร์แพ็กเกจ
- ไฟล์ทั้งหมดที่สำรองข้อมูลสัมพันธ์กับ root ซึ่งจะถูกเรียกคืนค่าเพื่อ การติดตั้งหรือปรับปรุงของส่วน usr ของผลิตภัณฑ์ซอฟต์แวร์

ถ้าแพ็กเกจการติดตั้งหรือปรับปรุง มีส่วน root ส่วน root จะมีไฟล์ดังต่อไปนี้:

- **./usr/lpp/*PackageName*/inst_root/liblpp.a**: ไลบรารีนี้มีไฟล์ควบคุมที่ใช้โดยกระบวนการติดตั้งสำหรับ process การติดตั้งหรือการปรับปรุงส่วน root ของซอฟต์แวร์แพ็กเกจ
- ไฟล์ทั้งหมดที่จะถูกเรียกคืนสำหรับการติดตั้งหรือ ปรับปรุงของส่วน root ของซอฟต์แวร์แพ็กเกจ สำหรับ ระดับการติดตั้ง fileset ฐานไฟล์เหล่านี้ต้องถูกสำรองข้อมูลโดยสัมพันธ์ กับ **./usr/lpp/*PackageName*/inst_root**

ตัวอย่างเนื้อหาของแพ็กเกจซอฟต์แวร์

แพ็กเกจ `farm.apps` มี `farm.apps.hog 4.1.0.0` fileset `farm.apps.hog 4.1.0.0` fileset ส่งมอบไฟล์ดังต่อไปนี้:

```
/usr/bin/raisehog (ในส่วน usr ของแพ็กเกจ)
/usr/sbin/sellhog
 (ในส่วน usr ของแพ็กเกจ)

/etc/hog
 (ในส่วน root ของแพ็กเกจ)
```

แพ็กเกจ `farm.apps` มี ไฟล์อย่างน้อยดังต่อไปนี้:

```
./lpp_name
./usr/lpp/farm.apps/liblpp.a
./usr/lpp/farm.apps/inst_root/liblpp.a
./usr/bin/raisehog
./usr/sbin/sellhog
./usr/lpp/farm.apps/inst_root/etc/hog
```

Fileset ปรับปรุง `farm.apps.hog 4.1.0.3` ส่งมอบอัพเดตให้กับไฟล์ดังต่อไปนี้:

```
/usr/sbin/sellhog
/etc/hog
```

แพคเกจอัพเดต fileset มีไฟล์ดังต่อไปนี้:

```
./lpp_name
./usr/lpp/farm.apps/farm.apps.hog/4.1.0.3/liblpp.a
./usr/lpp/farm.apps/farm.apps.hog/4.1.0.3/inst_root/liblpp.a
./usr/sbin/sellhog
./usr/lpp/farm.apps/farm.apps.hog/4.1.0.3/inst_root/etc/hog
```

หมายเหตุ: ไฟล์จากส่วน root ของแพ็กเกจถูกเรียกคืนภายใต้ไดเร็กทอรี **inst_root** ไฟล์ที่ติดตั้งสำหรับส่วน machine-dependent root ของแพ็กเกจถูกเรียกคืนสัมพันธ์กับไดเร็กทอรี **inst_root** ซึ่งช่วยในการติดตั้งไฟล์จำเพาะเครื่องในระบบ ไฟล์ root ของหลายระบบ ไฟล์ส่วน root ถูกติดตั้ง ลงในส่วน root ของระบบโดยการคัดลอกไฟล์จากไดเร็กทอรี **inst_root** ซึ่งทำให้เครื่องหลายเครื่องสามารถใช้ส่วน machine-independent usr ทั่วไปร่วมกันได้

ไฟล์ข้อมูลแพ็กเกจ lpp_name

แต่ละซอฟต์แวร์แพ็กเกจต้องมีไฟล์ข้อมูลแพ็กเกจ **lpp_name** ไฟล์ **lpp_name** ให้ข้อมูลคำสั่ง **installp** เกี่ยวกับแพ็กเกจ และแต่ละ fileset ในแพ็กเกจ อ้างอิงรูปสำหรับตัวอย่างไฟล์ **lpp_name** สำหรับแพ็กเกจอัพเดต fileset ตัวเลขและลูกศรใน รูปอ้างอิงถึงเขตข้อมูลที่ถูกอธิบายในตาราง

ชื่อฟิลด์	รูปแบบ	ตัวคั่น	คำอธิบาย
1. รูปแบบ	จำนวนเต็ม	ช่องว่าง	ระบุระดับรีลีสของ **installp** ซึ่ง แพ็กเกจนี้ถูกสร้าง ค่ามีดังนี้: • 1 - AIX 3.1 • 3 - AIX 3.2 • 4 - AIX 4.1 และสูงกว่า
2. แพล็ตฟอร์ม	อักขระ	ช่องว่าง	ระบุแพล็ตฟอร์มซึ่งแพ็กเกจนี้ถูกสร้าง ค่ามีดังนี้: • R - RISC • I - Intel • N - Neutral
3. ชนิดแพ็กเกจ	อักขระ	ช่องว่าง	ระบุว่านี่เป็นแพ็กเกจการติดตั้งหรืออัพเดตและมีชนิด ใด ค่ามีดังนี้: • I - Installation • S - Single update • SR - Single update required • ML - Technology level update
4. PackageName	อักขระ	ช่องว่าง	ชื่อของซอฟต์แวร์แพ็กเกจ (*PackageName*)
	{	บรรทัดใหม่	ระบุจุดเริ่มต้นของส่วนที่ทำซ้ำได้ของข้อมูลจำเพาะ fileset
5.ชื่อ Fileset	อักขระ	ช่องว่าง	ชื่อเต็มของ fileset ฟิลด์นี้เริ่มด้วยข้อมูลส่วนหัว สำหรับ fileset หรืออัพเดต fileset
6. ระดับ	แสดงในคอลัมน์คำอธิบาย	ช่องว่าง	ระดับของ fileset ที่จะถูกติดตั้ง รูปแบบ คือ: *Version.Release.Modification.FixLevel* **หมายเหตุ:** ระดับ สามารถกำหนดไว้โดยอัตโนมัติโดยใช้ไวยากรณ์การผสม <, > และ = ตัวอย่างเช่น `*prereq bos.rte v<5` หรือ `*prereq bos.rte v=5 r=3`
7. วอลุ่ม	จำนวนเต็ม	ช่องว่าง	ระบุหมายเลขวอลุ่มที่ fileset ตั้งอยู่ถ้าจัดส่ง ในสื่อบันทึก multivolume

ชื่อฟิลด์	รูปแบบ	ตัวคั่น	คำอธิบาย
8. Bosboot	อักขระ	ช่องว่าง	ระบุว่า bosboot จำเป็นหรือไม่ในการติดตั้งดังต่อไปนี้ ค่ามีดังนี้: • N – ห้ามเริ่มทำงาน bosboot • b – เริ่มทำงาน bosboot
9. เนื้อหา	อักขระ	ช่องว่าง	ระบุว่าส่วนรวมใน fileset หรือการอัพเดต fileset ค่ามีดังนี้: • B –usr and root part • U –usr part only
10. ภาษา	อักขระ	ช่องว่าง	ควรถูกตั้งค่าเป็นภาษาที่แสดง ถ้าเลือก C locale ปกติจะตั้งค่าเป็น en_US
11. คำอธิบาย	อักขระ	# หรือบรรทัดใหม่	คำอธิบาย Fileset คำอธิบายถูกจำกัดที่ 60 อักขระ
12. หมายเหตุ	อักขระ	บรรทัดใหม่	(เป็นทางเลือก) ข้อคิดเห็นเพิ่มเติม
	[	บรรทัดใหม่	ระบุจุดเริ่มต้นของส่วนเนื้อความของข้อมูล fileset
13. ข้อมูล Requisite	ที่อธิบายตารางดังต่อไปนี้	บรรทัดใหม่	(เป็นทางเลือก) ข้อมูลที่เกี่ยวข้องกันของการติดตั้งที่ fileset มีใน filesets และอัพเดต fileset อื่น ดูในส่วนของตารางต่อไปนี้สำหรับคำอธิบายละเอียด
	%	บรรทัดใหม่	ระบุการแยกระหว่างข้อมูล requisite และขนาด
14. ข้อมูลขนาดและข้อตกลงการอนุญาตใช้สิทธิ์	อธิบายภายหลังในหัวข้อนี้	บรรทัดใหม่	ข้อกำหนดขนาดโดยข้อมูลไดเร็กทอรีและข้อตกลงการอนุญาตใช้สิทธิ์ ดูส่วนขนาดและข้อมูลข้อตกลงการอนุญาตใช้สิทธิ์ภายหลัง ในหัวข้อนี้สำหรับคำอธิบายโดยละเอียด
	%	บรรทัดใหม่	ระบุการแยกระหว่างข้อมูลขนาดและการอนุญาตใช้สิทธิ์
	%	บรรทัดใหม่	ระบุการแยกระหว่างข้อมูลการอนุญาตใช้สิทธิ์และ การเข้าแทนที่
15. ข้อมูลการเข้าแทนที่	อธิบายภายหลังในหัวข้อ	บรรทัดใหม่	ข้อมูลเกี่ยวกับ fileset ก่อนหน้าที่แทนที่ด้วย fileset นี้
	%	บรรทัดใหม่	ระบุการแยกระหว่างข้อมูลการอนุญาตใช้สิทธิ์และการแก้ไข
16. ข้อมูลการแก้ไข	อธิบายภายหลังในหัวข้อ	บรรทัดใหม่	ข้อมูลเกี่ยวกับการแก้ไขที่มีในอัพเดต fileset โปรดดูที่ส่วน ข้อมูลโปรแกรมฟิกซ์ภายหลัง ในหัวข้อนี้ สำหรับคำอธิบายโดยละเอียด
	]	บรรทัดใหม่	ระบุจุดสิ้นสุดของส่วนเนื้อความของข้อมูล fileset
	}	บรรทัดใหม่	ระบุจุดสิ้นสุดของส่วนที่ทำซ้ำได้ของข้อมูลจำเพาะ fileset

```
1 23    4
| ||    |               6         7 8 9  10       11
4 RSfarm.apps {  |      |         | | |   |        |
 5--> farm.apps.hog04.01.0000.0003 1 N U en_US Hog Utilities
12-->#  ...
[
13--> *ifreq bos.farming.rte (4.2.0.0) 4.2.0.15
%
14--> /usr/sbin 48
14--> /usr/lpp/farm.apps/farm.apps.hog/4.1.0.3 280
14--> /usr/lpp/farm.apps/farm.apps.hog/inst_root/4.1.0.3.96
14--> /usr/lpp/SAVESPACE 48
14--> /lpp/SAVESPACE 32
14--> /usr/lpp/bos.hos/farm.apps.hog/inst_root/4.1.0.3/ etc 32
%
%
15--> ranch.hog 4.1.0.0
%
16--> IX51366 Hogs producing eggs.
16--> IX81360 Piglets have too many ears.
]
}
```

ส่วนข้อมูล Requisite

ส่วนข้อมูล requisite มีข้อมูลเกี่ยวกับ ความเกี่ยวข้องกันกับ filesets หรืออัพเดต fileset อื่น แต่ละ requisite ที่แสดงในส่วน requisite ต้องตรงตามกฎ requisite เพื่อสามารถใช้กับ fileset หรืออัพเดต fileset

ก่อนการติดตั้งหรือการอัพเดตเกิดขึ้น คำสั่ง **installp** เปรียบเทียบสถานะของ filesets ที่จะถูกติดตั้ง กับข้อกำหนดที่แสดงในไฟล์ **lpp_name** แฟล็ก **-g** ถูกระบุกับคำสั่ง **installp** ข้อมูลที่จำเป็นที่หายไปจะถูกเพิ่มให้กับรายการ filesets ที่จะถูกติดตั้ง filesets ถูกจัดลำดับสำหรับการติดตั้งตาม สิ่งที่จำเป็นต้องมี ก่อนที่ fileset ถูกติดตั้งคำสั่ง **installp** ได้รับการตรวจสอบข้อมูลที่จำเป็นสำหรับ fileset นั้นในทันที การตรวจสอบนี้ เพื่อพิสูจน์ว่าข้อมูลจำเป็นทั้งหมดที่ติดตั้งก่อนหน้าในกระบวนการติดตั้งได้ติดตั้งสำเร็จ และข้อมูลจำเป็นครบถ้วน

ในคำอธิบายดังต่อไปนี้ของชนิดต่างกันของข้อมูลจำเป็น *RequiredFilesetLevel* แสดง ระดับ fileset ต่ำสุดที่ตรงตามข้อกำหนด ยกเว้น เมื่อถูกบล็อกอย่างชัดเจนสำหรับสาเหตุที่อธิบายไว้ใน Supersede Information Section ระดับที่ใหม่กว่าของชุดไฟล์ที่ตรงกับข้อกำหนดบนระดับก่อนหน้านี้ ตัวอย่างเช่น ข้อมูลจำเป็น ใน `plum.tree 2.2.0.0` fileset ครบถ้วน ตาม `plum.tree 3.1.0.0` fileset

เอกสารสิ่งที่จำเป็นต้องมี

สิ่งที่จำเป็นต้องมีแสดงว่า fileset ที่ระบุต้องถูกติดตั้งที่ ระดับ fileset ที่กำหนดหรือที่ระดับสูงกว่าก่อนที่ fileset ปัจจุบัน สามารถติดตั้งได้สำเร็จ ถ้า fileset ที่จำเป็นต้องมีถูกกำหนดตารางเวลาเพื่อถูกติดตั้ง คำสั่ง **installp** จัดลำดับรายการของ filesets เพื่อติดตั้ง เพื่อให้แน่ใจว่าตรงกับสิ่งที่จำเป็นต้องมี สิ่งที่จำเป็นต้องมีบน fileset ภายในแพ็กเกจเดียวกันไม่ถูกรับประกัน

ไวยากรณ์

***prereq** *Fileset RequiredFilesetLevel*

ไวยากรณ์อื่น

Fileset RequiredFilesetLevel

อัพเดต fileset มีสิ่งที่จำเป็นต้องมีที่ไม่ระบุชัดเจนกับ base-level fileset ถ้าสิ่งที่จำเป็นต้องมีที่ระบุไม่ชัดเจนไม่เพียงพอ คุณต้องระบุสิ่งที่จำเป็นต้องมีอื่นอย่างชัดเจน *Version* และ *Release* ของอัพเดตและสิ่งที่จำเป็นต้องมี ที่ไม่ระบุชัดเจนเหมือนกัน ถ้า *FixLevel* ของ อัพเดตคือ 0, *ModificationLevel* และ *FixLevel* ของสิ่งที่จำเป็นต้องมีที่ไม่ระบุชัดเจน เป็น 0 ทั้งคู่ หรือมิฉะนั้นสิ่งที่จำเป็นต้องมีที่ไม่ระบุชัดเจน มี *ModificationLevel* ที่เหมือนกับ *ModificationLevel* ของอัพเดตและ *FixLevel* เป็น 0 ตัวอย่างเช่นอัพเดตระดับ 4.1.3.2 ต้องการให้ระดับ 4.1.3.0 ถูกติดตั้งก่อนการติดตั้งอัพเดต อัพเดตระดับ 4.1.3.0 ต้องการให้ระดับ 4.1.0.0 ของตัวเองถูกติดตั้งก่อน การติดตั้งอัพเดต

Corequisite

corequisite แสดงว่า fileset ที่ระบุต้องถูกติดตั้งสำหรับ fileset ปัจจุบันเพื่อให้ทำงานสำเร็จ ที่จุดสุดท้ายของกระบวนการ ติดตั้ง คำสั่ง **installp** ส่งข้อความคำเตือน สำหรับ corequisites ที่ไม่พบ corequisite สามารถถูกใช้เพื่อระบุ สิ่งที่จำเป็นต้องมีระหว่าง filesets ภายในแพ็กเกจเดียวกัน

ไวยากรณ์

***coreq** *Fileset RequiredFilesetLevel*

If requisite

if requisite แสดงว่า fileset ที่ระบุจำเป็นต้องเป็นที่ *RequiredFilesetLevel* เฉพาะถ้า fileset ถูกติดตั้ง ที่ *InstalledFilesetLevel* นี่เป็นเรื่องปกติที่ใช้ เพื่อเชื่อมโยงการทำงานข้อมูลที่เกี่ยวข้องระหว่างอัพเดต fileset ตัวอย่างดังต่อไปนี้ แสดง if requisite:

```
*ifreq plum.tree (1.1.0.0) 1.1.2.3
```

ไวยากรณ์

***ifreq** *Fileset* [(*InstalledFilesetLevel*)] *RequiredFilesetLevel*

ถ้า `plum.tree` fileset ไม่ได้ถูกติดตั้ง ตัวอย่างนี้จะไม่ทำการติดตั้ง ถ้า `plum.tree` fileset ถูกติดตั้งอยู่แล้ว ที่ระดับดังต่อไปนี้ ตัวอย่างนี้จะไม่ทำให้เกิดการติดตั้งระดับ `1.1.2.3`:

`1.1.2.3`
ระดับนี้ตรงกับ *RequiredFilesetLevel*

`1.2.0.0`
ระดับนี้เป็นระดับ fileset ฐานต่างกัน

`1.1.3.0`
ระดับนี้มาก่อนหน้า *RequiredFilesetLevel*

ถ้า `plum.tree` fileset ถูกติดตั้งอยู่แล้ว ที่ระดับดังต่อไปนี้ ตัวอย่างนี้ทำให้เกิดการติดตั้งระดับ `1.1.2.3`:

`1.1.0.0`
ระดับนี้ตรงกับ *InstalledFilesetLevel*

`1.1.2.0`
ระดับนี้เป็นระดับฐานเดียวกับ *InstalledFilesetLevel* และระดับต่ำกว่า *RequiredFilesetLevel*

พารามิเตอร์ (*InstalledFilesetLevel*) เป็นทางเลือก ถ้าเว้นไว้ *Version* และ *Release* ของ*InstalledFilesetLevel* และ *RequiredFilesetLevel* จะถือว่า เหมือนกัน ถ้า *FixLevel* ของ *RequiredFilesetLevel* เป็น 0 *ModificationLevel* และ *FixLevel*

ของ *InstalledFilesetLevel* จะเป็น 0 ทั้งคู่ ไม่เช่นนั้น *InstalledFilesetLevel* จะมี *ModificationLevel* ที่ เหมือนกับ *ModificationLevel* ของ *RequiredFilesetLevel* และ *FixLevel* เป็น 0 ตัวอย่างเช่น ถ้า *RequiredFilesetLevel* เป็น `4.1.1.1` และไม่ได้ระบุ *InstalledFilesetLevel* *InstalledFilesetLevel* จะเป็น `4.1.1.0` ถ้า *RequiredFilesetLevel* เป็น `4.1.1.0` และไม่มีการกำหนดพารามิเตอร์ *InstalledFilesetLevel*, *InstalledFilesetLevel* คือ `4.1.0.0`.

Installed requisite

installed requisite แสดงว่า fileset ที่ระบุควรถูกติดตั้งโดยอัตโนมัติ เฉพาะถ้า fileset ที่ตรงกันถูกติดตั้งอยู่แล้ว หรืออยู่ในรายการของ filesets ที่จะติดตั้ง requisite ที่ติดตั้งแล้วถูกติดตั้งด้วย ถ้าผู้ใช้ร้องให้ติดตั้ง อัพเดต fileset ไม่ มี requisite ที่ติดตั้ง เนื่องจาก fileset มีไฟล์ข้อความ สำหรับแพ็กเกจเฉพาะไม่ควรถูกติดตั้งโดยอัตโนมัติโดยไม่มี ส่วนอื่นของแพ็กเกจถูกติดตั้ง fileset ข้อความควรมี requisite ที่ติดตั้งสำหรับ fileset อื่นในผลิตภัณฑ์อยู่เสมอ

ไวยากรณ์

```
*instreq Fileset RequiredFilesetLevel
```

Group requisite

group requisite บ่งชี้ว่าเงื่อนไข requisite ต่างกันเป็นไปตาม ที่ requisite ต้องการได้ group requisite สามารถรับ prerequisites, corequisites, if-requisites และ nested group requisites *หมายเลข* นำหน้า { *RequisiteExpressionList* } ระบุ จำนวนรายการใน *RequisiteExpressionList* เป็น สิ่งจำเป็น ตัวอย่างเช่น `>2` ระบุว่า ต้องการอย่างน้อยสามไอเท็มใน *RequisiteExpressionList*

ไวยากรณ์

```
>Number { RequisiteExpressionList }
```

ตัวอย่างส่วนข้อมูล Requisite

1. ตัวอย่างดังต่อไปนี้แสดงการใช้ corequisites `book.create 12.30.0.0` fileset ไม่ทำงาน ถ้าไม่มี `layout.text 1.1.0.0` และ `index.generate 2.3.0.0` filesets ติดตั้งอยู่ ดังนี้ส่ง requisite สำหรับ `book.create 12.30.0.0` จะมี:

   ```
   *coreq layout.text 1.1.0.0
   *coreq index.generate 2.3.0.0
   ```

   `index.generate 3.1.0.0` fileset ตรงตาม `index.generate` requisite ต้องการ, เนื่องจาก `3.1.0.0` เป็นระดับใหม่กว่าระดับ `2.3.0.0` ที่ต้องการ
2. ตัวอย่างดังต่อไปนี้แสดงการใช้ของ ชนิด requisite ทั่วไปเพิ่มเติม Fileset `new.fileset.rte 1.1.0.0` มี requisites ดังต่อไปนี้:

   ```
   *prereq database.rte 1.2.0.0
   *coreq spreadsheet.rte 1.3.1.0
   *ifreq wordprocessorA.rte (4.1.0.0) 4.1.1.1
   *ifreq wordprocessorB.rte 4.1.1.1
   ```

   `database.rte` fileset ต้องถูกติดตั้งที่ระดับ `1.2.0.0` หรือ สูงกว่าก่อน `new.fileset.rte` สามารถถูก ติดตั้งได้ ถ้า `database.rte` และ `new.fileset.rte` ถูกติดตั้งในเซสชันการติดตั้งเดียวกัน โปรแกรมติดตั้ง ติดตั้ง `database` fileset ก่อน `new.fileset.rte` fileset

   `spreadsheet.rte` fileset ต้องถูกติดตั้ง ที่ระดับ `1.3.1.0` หรือสูงกว่าเพื่อให้ `new.fileset.rte` fileset ทำงานอย่างถูกต้อง `spreadsheet.rte` fileset ไม่จำเป็นต้องถูกติดตั้งก่อน `new.fileset.rte` fileset ถูกติดตั้ง ได้จัดเตรียมให้ ติดตั้งทั้ง

สอง fileset ในเซสชันการติดตั้งเดียวกัน ถ้าระดับ adequate ของ spreadsheet.rte fileset ไม่ได้ถูกติดตั้งโดยจุดสิ้นสุดของเซสชันการติดตั้ง ข้อความคำเตือนจะถูกส่งแจ้งว่า corequisite ไม่ตรงตามที่ต้องการ
ถ้า wordprocessorA.rte fileset ถูกติดตั้ง (หรือถูกติดตั้งด้วย new.fileset.rte) ที่ระดับ 4.1.0.0, แล้วอัพเดต wordprocessorA.rte fileset update ต้องถูกติดตั้งที่ระดับ 4.1.1.1 หรือสูงกว่า
ถ้า wordprocessorB.rte fileset ถูกติดตั้ง (หรือถูกติดตั้งด้วย new.fileset.rte) ที่ระดับ 4.1.1.0, แล้วอัพเดต wordprocessorB.rte fileset update ต้องถูกติดตั้งที่ระดับ 4.1.1.1 หรือสูงกว่า

3. ตัวอย่างดังต่อไปนี้แสดง requisite ที่ติดตั้ง Fileset Super.msg.fr_FR.Widget ที่ระดับ 2.1.0.0 มี install requisite ดังต่อไปนี้:

   ```
   *instreq Super.Widget 2.1.0.0
   ```

   Super.msg.fr_FR.Widget fileset ไม่สามารถถูกติดตั้งโดยอัตโนมัติ เมื่อ Super.Widget fileset ไม่ได้ถูกติดตั้ง Super.msg.fr_FR.Widget fileset สามารถถูกติดตั้งได้เมื่อ Super.Widget fileset ไม่ได้ถูกติดตั้ง ถ้า fileset ถูกแสดงบน รายการของ filesets ที่จะถูกติดตั้ง
4. ตัวอย่างดังต่อไปนี้แสดง group requisite อย่างน้อยหนึ่งใน prerequisite filesets ที่แสดงต้องถูกติดตั้ง (ทั้งคู่สามารถ ถูกติดตั้ง) ถ้ามีการติดตั้ง spreadsheet_1.rte fileset ต้องอยู่ที่ระดับ 1.2.0.0 หรือสูงกว่า หรือ spreadsheet_2.rte fileset ต้องอยู่ที่ระดับ 1.3.0.0 หรือสูงกว่า

   ```
   >0 {
   *prereq spreadsheet_1.rte 1.2.0.0
   *prereq spreadsheet_2.rte 1.3.0.0
   }
   ```

ข้อมูลขนาดและข้อตกลงการใช้ไลเซนส์

ส่วนข้อมูลขนาดและข้อตกลงการอนุญาตใช้สิทธิ์ มีข้อมูลเกี่ยวกับข้อกำหนดพื้นที่ว่างดิสก์และข้อตกลงการอนุญาตใช้สิทธิ์สำหรับ fileset

ข้อมูลขนาด

ข้อมูลนี้ถูกใช้โดยกระบวนการติดตั้งเพื่อประกันมี พื้นที่ว่างดิสก์เพียงพอสำหรับการติดตั้งหรืออัพเดตสำเร็จ ข้อมูลขนาด มีรูปแบบ:

```
Directory PermanentSpace [TemporarySpace]
```

Additionally, the product developer can specify **PAGESPACE** or **INSTWORK** in the full-path name field to indicate disk space requirements for paging space and workspace needed in the package directory during the installation process.

ไดเร็กทอรี
ชื่อพาธเต็มของไดเร็กทอรีที่มีข้อกำหนดขนาด

PermanentSpace

ขนาด (บล็อก 512 ไบต์) ของพื้นที่ถาวรที่จำเป็นสำหรับ การติดตั้งหรืออัพเดต พื้นที่ถาวรเป็นพื้นที่ว่างที่จำเป็นหลังจากการติดตั้งสมบูรณ์ เขตข้อมูลนี้มีความหมายต่างกันในกรณี ดังต่อไปนี้:

ถ้า *Directory* คือ **PAGESPACE**, *PermanentSpace* แสดง ขนาดของพื้นที่เพจที่จำเป็น (บล็อก 512 ไบต์) เพื่อทำการติดตั้ง

ถ้า *Directory* คือ **INSTWORK**, *PermanentSpace* แสดงจำนวน บล็อก 512 ไบต์ที่จำเป็นสำหรับการแยกไฟล์ควบคุมที่ใช้ระหว่างการติดตั้ง ไฟล์ควบคุมเหล่านี้เป็นไฟล์ที่ถูกเก็บถาวรเป็นไฟล์ **liblpp.a**

`TemporarySpace`

ขนาด (บล็อก 512 ไบต์) ของพื้นที่ชั่วคราวที่จำเป็นสำหรับ การติดตั้งเท่านั้น พื้นที่ชั่วคราวจะถูกส่งคืนหลังจากการติดตั้งสมบูรณ์ ค่า *TemporarySpace* เป็นทางเลือก ตัวอย่าง ของพื้นที่ชั่วคราวคือพื้นที่ที่จำเป็นในการเชื่อมโยงไฟล์อ็อบเจ็กต์ที่รันได้ใหม่ อีกตัวอย่างคือพื้นที่ที่จำเป็นในการเก็บถาวรอ็อบเจ็กต์ไฟล์ลงใน ไลบรารี เมื่อต้องการเก็บถาวรลงในไลบรารี คำสั่ง **installp** จะสร้าง สำเนาของไลบรารี, เก็บถาวรอ็อบเจ็กต์ไฟล์ลงในไลบรารีที่ทำสำเนา และย้ายไลบรารีที่คัดลอกไปที่ไลบรารีดั้งเดิม พื้นที่สำหรับสำเนาของ ไลบรารีถือว่าเป็นพื้นที่ชั่วคราว

เมื่อ *Directory* คือ **INSTWORK**, *TemporarySpace* แสดงจำนวนบล็อก 512 ไบต์ ที่จำเป็นสำหรับไฟล์ **liblpp.a** ที่ยังไม่ได้แยก

ตัวอย่างดังต่อไปนี้แสดงส่วนข้อมูลขนาด:

```
/usr/bin          30
/lib              40  20
PAGESPACE         10
INSTWORK          10    6
```

เนื่องจากเป็นเรื่องยากที่จะคาดถึงการที่ระบบไฟล์ดิสก์ จะถูกประกอบเข้าเข้ากับแผนผังไฟล์ รายการชื่อพาธไดเร็กทอรีในส่วนข้อมูลขนาด ควรมีความเจาะจงเท่าที่เป็นไปได้ ตัวอย่างเช่น จะดีกว่า ถ้ามีหนึ่งรายการสำหรับ **/usr/bin** และหนึ่งรายการสำหรับ **/usr/lib** แทนการมีรายการเดียวที่รวมกันสำหรับ **/usr** เนื่องจาก **/usr/bin** และ **/usr/lib** สามารถมีอยู่บนระบบไฟล์ต่างกันซึ่งถูกเชื่อมโยงเข้ามาภายใต้ **/usr** โดยทั่วไป จะเป็นการดีที่สุดที่จะรวมรายการสำหรับแต่ละ ไดเร็กทอรีลงในไฟล์ที่ถูกติดตั้ง

สำหรับอัพเดตแพ็กเกจเท่านั้น ข้อมูลขนาดต้อง รวมไฟล์เก่า (ที่จะถูกแทนที่) ซึ่งจะย้ายไปไว้ในไดเร็กทอรี **save** ไฟล์เก่าเหล่านี้จะถูกคืนค่า ถ้าอัพเดตถูก ปฏิเสธในภายหลัง เพื่อระบุข้อกำหนดขนาดเหล่านี้ อัพเดตแพ็กเกจ ต้องระบุไดเร็กทอรีพิเศษดังต่อไปนี้:

`/usr/lpp/SAVESPACE`
ไดเร็กทอรี **save** สำหรับไฟล์ส่วน usr โดยค่าเริ่มต้น ไฟล์ส่วน usr ถูกบันทึกในไดเร็กทอรี **/usr/lpp/***PackageName*/*FilesetName*/*FilesetLevel***.save**

`/lpp/SAVESPACE`
ไดเร็กทอรี **save** สำหรับไฟล์ส่วน root โดยค่าเริ่มต้น ไฟล์ส่วน root ถูกบันทึกในไดเร็กทอรี **/lpp/***PackageName*/*FilesetName*/*FilesetLevel***.save**

ข้อมูลข้อตกลงการใช้ไลเซนส์

ส่วนเพิ่มเติมล่าสุดต่อกระบวนการติดตั้ง AIX อนุญาตให้เจ้าของผลิตภัณฑ์ ขอให้ลูกค้า *เซ็น* ข้อตกลงการอนุญาตใช้สิทธิ์ ก่อนการติดตั้งผลิตภัณฑ์ คำสั่ง **installp** อ่านไฟล์ *lpp_name* ของ fileset ตามปกติสำหรับอิมเมจ ถ้ามีรายการ `LAF` หรือ `LAR` ในส่วนขนาดของไฟล์ *lpp_name* คำสั่ง **installp** เรียกคำสั่ง **inulag** ซึ่งแสดงการอนุญาตใช้สิทธิ์ ในหน้าต่างและบันทึกการยอมรับการอนุญาตใช้สิทธิ์ของลูกค้า If the customer refuses to accept the license, the installation is halted for that product.

ตำแหน่งที่เก็บไฟล์การอนุญาตใช้สิทธิ์

การกำหนดตำแหน่งไฟล์การอนุญาตใช้สิทธิ์ขึ้นอยู่กับเจ้าของผลิตภัณฑ์ อย่างไรก็ตาม ของ แนะนำให้นำการอนุญาตใช้สิทธิ์ไว้ใน /usr/swlag/LANG ชื่อที่แนะนำสำหรับไฟล์ License Agreement คือกำหนดเป็น ProductName_VersionRelease.la ไม่มีข้อกำหนดในการใช้ชื่อหรือตำแหน่งนี้ คำสั่ง **installp** และการแพ็กเกจนี้เพียงให้เมธอดเพื่อส่งข้อมูล ไปที่ลูกค้า เครื่องมือ และเนื้อหาไฟล์ทั้งหมดต้องถูกจัดเตรียมโดย ผลิตภัณฑ์

ข้อกำหนดการแปลสำหรับไฟล์การอนุญาตใช้สิทธิ์

ถ้ามีข้อกำหนดซึ่งไฟล์ License Agreement files ถูกแปล เป็นภาษาที่สนับสนุน ขอแนะนำให้แยกไฟล์ สำหรับแต่ละภาษาที่ต่างกัน การแปลนี้อาจจำเป็นต้องมีการ สร้างไฟล์หลายไฟล์

วิธีจัดส่งไฟล์การอนุญาตใช้สิทธิ์

ผลิตภัณฑ์สามารถจัดส่งไฟล์การอนุญาตใช้สิทธิ์เป็นส่วนหนึ่งของผลิตภัณฑ์หลักหรือใน fileset แยกสำหรับผลิตภัณฑ์ ผลิตภัณฑ์จำนวนมากเลือกที่จะสร้าง fileset แยกสำหรับการจัดส่งเพียงไฟล์การอนุญาตใช้สิทธิ์ ซึ่งช่วยให้ผลิตภัณฑ์มีคุณลักษณะต่างกันในการสร้างไฟล์การอนุญาตใช้สิทธิ์หนึ่งไฟล์ และ fileset ที่สามารถจัดส่งบน สื่อบันทึกทั้งหมดที่จำเป็นสำหรับแต่ละคุณลักษณะ แทนการที่จะต้องรวมไฟล์ลงใน คุณลักษณะต่างกัน ข้อเสนอแนะปัจจุบันสำหรับชื่อ fileset คือ lpp.license หรือ lpp.loc.license ในปัจจุบันผลิตภัณฑ์ส่วนใหญ่ใช้ข้อเสนอแนะแรก ถ้าคุณต้องการให้ fileset การอนุญาตใช้สิทธิ์ถูกซ่อนจากลูกค้าในการติดตั้งมาตรฐาน ให้ใช้ lpp.loc.license เนื่องจาก fileset การอนุญาตใช้สิทธิ์ไม่จำเป็นต้องถูกเลือกสำหรับการติดตั้ง

วิธีแพ็กเกจไฟล์การอนุญาตใช้สิทธิ์

ตัวไฟล์ไม่เคยถูกแสดงในไฟล์ *fileset.al* หรือ *fileset.inventory* สำหรับ fileset คำสั่ง **installp** ค้นหา การอนุญาตใช้สิทธิ์เนื่องจากรายการในส่วนขนาดของไฟล์ *ProductName* ชนิดของรายการคือ:

LAF

License **A**greement **F**ile แจ้งกับคำสั่ง **installp** ว่าไฟล์การอนุญาตใช้สิทธิ์ นี้ถูกจัดส่งใน fileset นี้

ไฟล์ข้อตกลงการอนุญาตใช้สิทธิ์ถูกบ่งชี้ โดยรายการส่วนขนาดด้วย:

LAF<lang>license_file size

คำศัพท์ ความหมาย

LAF หมายถึง **L**icense **A**greement **F**ile

<lang> ภาษาซึ่งไฟล์ถูกแปล ซึ่งปกติจะเป็นรายการ ดังนี้ en_US, fr_FR, Ja_JP และ zh_TW ถ้าไม่ได้ระบุ <lang> ดังนั้นจะถือว่าไม่มีการแปลไฟล์ข้อตกลงและ เข้ารหัสเป็น ASCII ถ้าข้อตกลงการใช้ไลเซนส์ถูกแปล <lang> ต้องเป็นส่วนหนึ่งของพาธเพื่อให้รายการข้อกำหนดสามารถเชื่อมโยง กับไฟล์ได้

license_file
เป็น fullpath ไปที่ไฟล์การอนุญาตใช้สิทธิ์ ตามที่จะถูกพบในอิมเมจ และในระบบ พาธที่แนะนำมีรูปแบบ /usr/swlag/en_US/ProductName_VersionRelease.la

ขนาด นี่เป็นขนาดจริง ในบล็อก 512 ไบต์ ของไฟล์การอนุญาตใช้สิทธิ์เพื่อประกัน ว่า **installp** ได้รับเนื้อที่ว่างเพียงพอในการนำไฟล์การอนุญาตใช้สิทธิ์ ไปไว้ในระบบ

LAR

License **A**greement **R**equisite แจ้งคำสั่ง **installp** ว่า fileset นี้ต้องการการติดตั้งไฟล์ข้อตกลงการอนุญาตใช้สิทธิ์ที่แสดงรายการ ซึ่ง ไม่เหมือนกับ prerequisite เนื่องจากเป็นไฟล์และไม่ได้อยู่ใน fileset ไฟล์และ filesets มีรูปแบบและวัตถุประสงค์ต่างกัน ซึ่งไม่ควร สับสน

คำศัพท์ ความหมาย

LAR หมายถึง **L**icense **A**greement **R**equisite

req_license_file
พาธเต็มไปที่ไฟล์การอนุญาตใช้สิทธิ์ที่จำเป็นในการติดตั้ง fileset นี้ โดยปกติรายการเหล่านี้ใช้ %L ในพาธแทนชื่อภาษาจริง เพื่อให้ดูไฟล์ที่เหมาะสมได้โดยไม่ต้องบังคับ ให้ลูกค้าต้องดูภาษาทั้งหมด

โดยปกติหนึ่ง fileset จัดส่งไฟล์ และมีรายการ **LAF** ทั้งหมด filesets อื่นในผลิตภัณฑ์ซึ่งต้องการการอนุญาตใช้สิทธิ์นี้ จัดส่งเฉพาะรายการ **LAR** fileset ซึ่งจัดส่งรายการ **LAF** ยังมีไฟล์ที่แสดงที่ตำแหน่งพาธเต็มของไฟล์ในอิมเมจ BFF แต่ไฟล์ไม่ถูกแสดงในไฟล์ *fileset.al* หรือ *fileset.inventory* สำหรับ fileset การออกแบบที่ใช้สำหรับการอนุญาตใช้สิทธิ์อิเล็กทรอนิกส์ *ไม่* ต้องการให้ไฟล์ถูกลงทะเบียนกับ SWVPD คำสั่ง **installp**:

1. ค้นหาข้อกำหนดในไฟล์
2. ตรวจสอบระบบเพื่อดูว่ามีการยอมรับหรือไม่
3. ถ้าไฟล์ไม่ได้รับการยอมรับ
   a. ค้นหา fileset ที่จัดส่งไฟล์
   b. แยก (คืนค่า) เฉพาะไฟล์นั้นจากอิมเมจ BFF
   c. แสดงไฟล์แก่ลูกค้า

ตัวอย่างชุดไฟล์ LAF

ต่อไปนี้เป็นตัวอย่างของ fileset ที่จัดส่งไฟล์การอนุญาตใช้สิทธิ์:

```
iced.tea.loc.license 03.01.0000.0000 1 N U en_US IcedTea Recipe License Information
[
%
INSTWORK 16 160
LAF/usr/swlag/de_DE/iced.tea.la 24
LAF/usr/swlag/DE_DE/iced.tea.la 24
LAF/usr/swlag/en_US/iced.tea.la 24
LAF/usr/swlag/EN_US/iced.tea.la 24
LAF/usr/swlag/es_ES/iced.tea.la 24
LAF/usr/swlag/ES_ES/iced.tea.la 24
LAF/usr/swlag/fr_FR/iced.tea.la 24
LAF/usr/swlag/FR_FR/iced.tea.la 24
LAF/usr/swlag/it_IT/iced.tea.la 24
LAF/usr/swlag/IT_IT/iced.tea.la 24
LAF/usr/swlag/ja_JP/iced.tea.la 24
LAF/usr/swlag/JA_JP/iced.tea.la 32
LAF/usr/swlag/Ja_JP/iced.tea.la 24
LAF/usr/swlag/ko_KR/iced.tea.la 24
LAF/usr/swlag/KO_KR/iced.tea.la 24
LAF/usr/swlag/pt_BR/iced.tea.la 24
LAF/usr/swlag/PT_BR/iced.tea.la 24
LAF/usr/swlag/ru_RU/iced.tea.la 24
LAF/usr/swlag/RU_RU/iced.tea.la 48
```

```
LAF/usr/swlag/zh_CN/iced.tea.la 16
LAF/usr/swlag/zh_TW/iced.tea.la 16
LAF/usr/swlag/Zh_TW/iced.tea.la 16
LAF/usr/swlag/ZH_TW/iced.tea.la 24
%
%
%
]
```

ตัวอย่าง LAR Fileset

ต่อไปนี้เป็นตัวอย่างของ fileset ที่จัดส่งข้อกำหนดการอนุญาตใช้สิทธิ์ ไปที่ไฟล์การอนุญาตใช้สิทธิ์:

```
iced.tea.server 03.01.0000.0010 1 N B en_US Iced Tea Recipe Group
[
*prereq bos.net.tcp.client 5.1.0.10
*coreq iced.tea.tools 5.1.0.10
*coreq Java14.sdk 1.4.0.1
%
/usr/bin 624
/usr/lib/objrepos 24
/usr/include 16
/usr/include/sys 56
/usr/lpp/iced.tea 22
/usr/samples/iced.tea 8
/usr/samples/iced.tea/server 504
/usr/lpp/iced.tea/inst_root/etc/tea 8
/usr/iced.tea 8
/usr/lpp/iced.tea/inst_root/etc/tea/Top 8
INSTWORK 208 96
/lpp/iced.tea 104
/etc/tea 8
/etc/objrepos 8
/etc/tea/Top 8
/tmp 0 6
LAR/usr/swlag/%L/iced.tea.la 0
%
%
%
]
```

ส่วนข้อมูล Supersede

ส่วนข้อมูล supersede แสดงระดับของ fileset หรืออัพเดต fileset ซึ่ง fileset นี้หรืออัพเดต fileset อาจ (หรืออาจจะไม่) ถูกใช้แทน ข้อมูล Supersede เป็นทางเลือกและใช้ได้ เฉพาะกับแพ็กเกจการติดตั้งแบบ AIX 4.1 -formatted fileset และแพ็กเกจอัพเดต AIX 3.2 -formatted fileset

fileset ที่ใหม่กว่าแทนที่เวอร์ชันเก่าของ fileset นั้นนอกจากว่าส่วน supersedes ของไฟล์ **lpp_name** ระบุระดับล่าสุดของ fileset ที่แทนที่ ในกรณีที่เกิดได้ยาก ซึ่ง fileset ไม่แทนที่ระดับก่อนหน้าทั้งหมดของ fileset นั้นคำสั่ง **installp** ไม่ใช้ fileset เพื่อให้ตรงตาม requisites ในระดับที่เก่ากว่าระดับของ fileset ที่แสดงในส่วนการแทนที่

อัพเดต fileset แทนที่อัพเดตที่เก่ากว่าเฉพาะ fileset นั้นเท่านั้น ถ้ามี ไฟล์ กระบวนการการกำหนดค่า และข้อมูล requisite ทั้งหมดในอัพเดต fileset ที่เก่ากว่า คำสั่ง **installp** กำหนดว่าอัพเดต fileset แทนที่อัพเดตอื่น สำหรับ fileset นั้นในเงื่อนไขดังต่อไปนี้:

- เวอร์ชัน รีลีส และระดับการปรับเปลี่ยนสำหรับ อัพเดตเหมือนกัน ระดับการแก้ไขทั้งสองมีค่าไม่เป็นศูนย์ และอัพเดต ที่มีระดับการแก้ไขสูงกว่าไม่อยู่ในสิ่งที่จำเป็นต้องมีบนระดับของ fileset ที่มากกว่าหรือเท่ากับระดับของอัพเดตที่มีระดับการแก้ไขต่ำกว่า
- ระดับเวอร์ชันและรีลีสสำหรับอัพเดตเหมือนกัน และอัพเดตที่มีระดับการปรับเปลี่ยนสูงกว่าไม่อยู่ใน สิ่งที่จำเป็นต้องมีในระดับของ fileset ที่มากกว่าหรือเท่ากับระดับของ อัพเดตที่มีระดับการปรับเปลี่ยนต่ำกว่า

ตัวอย่างเช่น อัพเดต fileset `farm.apps.hog 4.1.0.1` จัดส่งอัพเดตของ `/usr/sbin/sellhog` อัพเดต Fileset `farm.apps.hog 4.1.0.3` ส่งมอบ อัพเดตให้กับไฟล์ `/usr/sbin/sellhog` และไฟล์ `/etc/hog` อัพเดต Fileset `farm.apps.hog 4.1.1.2` ส่งมอบอัพเดตให้กับไฟล์ `/usr/bin/raisehog`

อัพเดต `farm.apps.hog 4.1.0.3` แทนที่ `farm.apps.hog 4.1.0.1` เนื่องจาก อัพเดตส่งมอบไฟล์เหมือนกันและใช้กับระดับเดียวกัน, `farm.apps.hog 4.1.0.0`

อัพเดต `farm.apps.hog 4.1.1.2` ไม่แทนที่ทั้ง `farm.apps.hog 4.1.0.3` หรือ `farm.apps.hog 4.1.0.1` เนื่องจากอัพเดตไม่มี ไฟล์เหมือนกันและใช้กับระดับต่างกัน, `farm.apps.hog 4.1.1.0` อัพเดต `farm.apps.hog 4.1.1.0` แทนที่ `farm.apps.hog 4.1.0.1` และ `farm.apps.hog 4.1.0.3`

ส่วน Supersede สำหรับระดับการติดตั้งชุดไฟล์ (ระดับฐาน)

แพ็กเกจการติดตั้ง AIX 4.1-formatted fileset มี รายการแทนที่ดังต่อไปนี้ได้:

`Barrier Entry`
: ระบุระดับ fileset ซึ่งความเข้ากันไม่ได้หลักถูกแนะนำ ความเข้ากันไม่ได้นี้เก็บ fileset ปัจจุบันจากการทำให้ requisites เป็นไปตามความต้องการกับระดับของ fileset ก่อนหน้าระดับที่ระบุ

`Compatibility Entry`
: ระบุว่า fileset สามารถถูกใช้เพื่อให้ตรงกับความต้องการ requisites ของ fileset อื่น รายการความเข้ากันได้ถูกใช้เมื่อ fileset ถูกเปลี่ยนชื่อหรือ ทำให้หมดอายุการใช้งาน หนึ่ง fileset เท่านั้นที่สามารถแทน fileset ที่ระบุ คุณอาจระบุเพียง หนึ่งรายการความเข้ากันได้สำหรับแต่ละ fileset

ไฟล์ **lpp_name** มี barrier ได้มากที่สุดหนึ่ง barrier และหนึ่งรายการความเข้ากันได้สำหรับ fileset

รายการ barrier ประกอบด้วยชื่อ fileset และระดับ fileset เมื่อมีการแจ้งความเข้ากันไม่ได้ รายการ barrier จำเป็น สำหรับ fileset เท่านั้นในกรณีที่พบน้อยที่ระดับของ fileset ถูกแจ้งความเข้ากันไม่ได้เช่นการทำงานที่ต้องการโดย filesets ที่เชื่อมโยงได้ถูกปรับเปลี่ยนหรือเอาออกให้เป็นไปตามความต้องการใน ระดับก่อนหน้าของ fileset ที่ไม่ตรง รายการ barrier ต้องมีอยู่ในเวอร์ชัน ที่ตามมาทั้งหมดของ fileset ที่บ่งชี้ระดับล่าสุดของ fileset ที่ตรงความต้องการ filesets ที่เชื่อมโยงกัน

ตัวอย่างเช่น ถ้าความเข้ากันไม่ได้หลักถูกแจ้งใน fileset `Bad.Idea 6.5.6.0`, ส่วนข้อมูลแทนที่ สำหรับแต่ละแพ็กเกจการติดตั้ง `Bad.Idea` fileset จากระดับ fileset `6.5.6.0` ขึ้นไปจะ มีรายการ `Bad.Idea 6.5.6.0` barrier รายการ barrier นี้จะป้องกัน requisite ของ `Bad.Idea 6.5.4.0` ไม่ให้ตรงตามความต้องการของระดับใดๆ ของ `Bad.Idea` ที่มากกว่าหรือเท่ากับ `6.5.6.0`

รายการความเข้ากันได้ประกอบด้วยชื่อ fileset (ต่างจาก fileset ในแพ็กเกจ) และระดับ fileset ระดับ fileset ระบุ ระดับซึ่งจำเป็นกับ fileset (และระดับก่อนหน้าของ fileset นั้น) ที่ระบุตรงกันตาม fileset ในแพ็กเกจการติดตั้ง ความเข้ากันได้ มีประโยชน์เมื่อ fileset ที่ระบุเก่าหรือถูกเปลี่ยนชื่อ และ ฟังก์ชันจาก fileset ที่ระบุมีอยู่ใน fileset ปัจจุบัน ระดับ fileset ในรายการความเข้ากันได้ควรสูงกว่าระดับ ที่คาดว่าจะมีอยู่สำหรับ fileset ที่ระบุ

ตามตัวอย่าง สมมุติว่า `Year.Full 19.91.0.0` fileset ไม่ถูกจัดส่งอีกต่อไป ในรูปยูนิต แต่ถูกแยกออกเป็น ส่วนเล็กๆ เป็นแต่ละ filesets เฉพาะหนึ่งในส่วนที่เล็กให้ผลลัพธ์ filesets บางที `Winter 19.94.0.0` ควรมีรายการความเข้ากันได้ของ `Year.Full 19.94.0.0` รายการความเข้ากันได้นี้ทำให้ `Winter 19.94.0.0` fileset ตรงตามความต้องการของการเชื่อมโยง filesets ใน `Year.Full` ที่ระดับ `19.94.0.0` และก่อนหน้า

การประมวลผล Supersede

คำสั่ง **installp** มี คุณลักษณะพิเศษต่อไปนี้สำหรับการติดตั้ง filesets และการอัพเดต fileset ซึ่งแทนที่ filesets หรือการอัพเดต fileset:

- ถ้าสื่อบันทึกการติดตั้งไม่มี fileset หรือการอัพเดต fileset ที่ผู้ใช้ร้องขอเพื่อติดตั้ง fileset หรือการอัพเดต fileset บนสื่อบันทึกการติดตั้งสามารถถูกติดตั้ง

  ตัวอย่าง สมมุติผู้ใช้เรียกใช้คำสั่ง **installp** ที่มีแฟล็ก **-g** (การติดตั้งสิ่งที่จำเป็นโดยอัตโนมัติ) เพื่อติดตั้ง `farm.apps.hog 4.1.0.2` fileset ถ้าสื่อบันทึกการติดตั้งมี `farm.apps.hog 4.1.0.4` fileset เท่านั้น คำสั่ง **installp** จะติดตั้ง `farm.apps.hog 4.1.0.4` fileset เนื่องจากจะแทนที่ระดับที่ร้องของ
- ถ้าระบบและสื่อบันทึกการติดตั้งไม่มี fileset หรือการอัพเดต fileset ที่จำเป็น สิ่งที่จำเป็นสามารถมีได้โดยการแทนที่ fileset หรือการอัพเดต fileset
- ถ้าการอัพเดตได้รับการร้องขอให้ทำการติดตั้งและแฟล็ก **-g** ถูกระบุ การร้องขอจะสำเร็จได้โดยการแทนที่ ที่ใหม่สุดบนสื่อบันทึกการติดตั้ง

  เมื่อแฟล็ก **-g** ถูกระบุด้วยคำสั่ง **installp** การอัพเดตใดๆ ที่ได้รับการร้องขอเพื่อทำการติดตั้ง (ไม่ว่าโดยชัดแจ้งหรือ โดยนัย) จะทำได้โดยการอัพเดตแทนที่ใหม่ล่าสุดบนสื่อบันทึก การติดตั้ง ถ้าผู้ใช้ต้องการติดตั้งระดับการอัพเดตเป็นการเฉพาะ ไม่จำเป็นต้องเป็น ระดับล่าสุด ผู้ใช้สามารถเรียกใช้คำสั่ง **installp** โดยไม่ต้องใช้แฟล็ก **-g**
- ถ้าการอัพเดตและการอัพเดตแทนที่ (ทั้งคู่อยู่บน สื่อบันทึกการติดตั้ง) ได้รับการร้องขอให้ติดตั้ง คำสั่ง **installp** จะติดตั้งการอัพเดตที่ใหม่กว่าเท่านั้น

  ในกรณีนี้ ถ้าผู้ใช้ต้องการนำใช้การอัพเดตที่เจาะจง รวมถึงการอัพเดตแทนที่ จากสื่อบันทึกการติดตั้ง ผู้ใช้ต้องดูการดำเนินการ **installp** แยกกันสำหรับแต่ละระดับการอัพเดต โปรดทราบว่าการดำเนินในลักษณะนี้ จะไม่มีความหมายถ้าการอัพเดตทั้งสองถูกนำใช้และยอมรับ (**-ac**) การยอมรับการอัพเดตครั้งที่สองจะเป็นการลบการอัพเดตครั้งแรกออก จากระบบ

ส่วนข้อมูลการแก้ไข

ส่วนข้อมูลการแก้ไขเป็นทางเลือก รายการส่วนข้อมูล การแก้ไขมีคีย์เวิร์ดการแก้ไข และรายละเอียดยาวไม่เกิน 60 อักขระอธิบาย ปัญหาที่ถูกปัญหา คีย์เวิร์ดการแก้ไขเป็น identifier 16 อักขระหรือน้อยกว่า ที่สอดคล้องกับการแก้ไข คีย์เวิร์ดการแก้ไขเริ่มต้นด้วย **ix, iy, IY** และ **IX** ถูกสำรองเพื่อใช้งานโดยผู้จัดทำ ระบบปฏิบัติการ

ระดับเทคโนโลยีคือโปรแกรมแก้ไขที่เป็นระดับการอัพเดต หลัก แพ็คเกจการบำรุงรักษาเชิงป้องกันที่จัดทำเป็นระยะเป็นระดับเทคโนโลยี ตัวระบุ ระดับเทคโนโลยีขึ้นต้นด้วยชื่อของผลิตภัณฑ์ซอฟต์แวร์ (ไม่ใช่ แพ็กเกจ) แล้วตามด้วยจุดหนึ่งจุด (.) และระบุระดับ เช่น `farm.4.1.1.0`

ไฟล์ไลบรารีควบคุมการติดตั้ง liblpp.a

ไฟล์ **liblpp.a** เป็นไฟล์เก็บถาวร ที่มีไฟล์ที่จำเป็นสำหรับการควบคุมการติดตั้งแพ็กเกจ คุณสามารถสร้างไฟล์ **liblpp.a** สำหรับแพ็คเกจของคุณโดยใช้ คำสั่ง **ar** ที่มีแฟล็ก *-g* บนระบบใหม่กว่า AIX 4.3 เพื่อให้แน่ใจว่าไฟล์เก็บถาวร 32 บิต ถูกสร้าง ส่วนนี้อธิบายไฟล์ต่างๆ ที่คุณสามารถเก็บในไฟล์เก็บถาวร **liblpp.a**

ตลอดทั้งส่วนนี้ *Fileset* จะปรากฏในชื่อของไฟล์ควบคุม *Fileset* แทนชื่อของ fileset แยกที่จะถูกติดตั้งภายใน ซอฟต์แวร์แพ็กเกจ ตัวอย่าง สมมุติไฟล์รายการที่นำใช้ถูกแสดงเป็น *Fileset*.**al** ไฟล์รายการที่นำใช้ สำหรับอ็อพชัน `bos.net.tcp.client` ของผลิตภัณฑ์ซอฟต์แวร์ `bos.net` จะเป็น `bos.net.tcp.client.al`

สำหรับไฟล์ใดๆ ที่คุณรวมในไฟล์เก็บถาวร **liblpp.a** นอกเหนือจากไฟล์ที่แสดงรายการในส่วนนี้ คุณ ควรใช้ข้อกำหนดการตั้งชื่อต่อไปนี้:

- ถ้าไฟล์ถูกใช้ในการติดตั้งของ fileset ที่เจาะจง ชื่อไฟล์ควรขึ้นต้นด้วยคำนำหน้า *Fileset* .
- ถ้าใช้ไฟล์เป็นไฟล์ทั่วไปสำหรับหลายๆ fileset ในแพ็กเกจเดียวกัน ชื่อไฟล์ควรขึ้นต้นด้วย **lpp.**

หลายๆ ไฟล์ที่อธิบายในส่วนนี้เป็นทางเลือก ไฟล์ทางเลือกจำเป็นต่อเมื่อฟังก์ชันที่ไฟล์จัดให้มีต้องการใช้ สำหรับ fileset หรือการอัพเดต fileset เว้นแต่จะกำหนดไว้ ไฟล์อยู่ในทั้ง แพ็กเกจการติดตั้งสมบูรณ์ และแพ็กเกจการอัพเดต fileset

ไฟล์ข้อมูลที่อยู่ในไฟล์ liblpp.a

Fileset`.al`
: รายการนำใช้ ไฟล์นี้แสดงรายการไฟล์ทั้งหมดที่จะถูกเรียกคืนสำหรับ fileset นี้ ไฟล์ถูกแสดงหนึ่งรายการต่อหนึ่งบรรทัด พร้อมด้วยพาธที่สัมพันธ์กับ root เช่นใน `./usr/bin/pickle` ไฟล์รายการนำใช้จำเป็นต้องใช้ ถ้ามีไฟล์ใดๆ ถูกส่งมาพร้อมกับ fileset และการอัพเดต fileset

Fileset`.cfginfo`
: ไฟล์คำสั่งพิเศษ ไฟล์นี้แสดงหนึ่งคีย์เวิร์ดต่อหนึ่งบรรทัด แต่ละคีย์เวิร์ดระบุคุณสมบัติพิเศษของ fileset หรือ การอัพเดต fileset คีย์เวิร์ดที่รู้จักในขณะนี้เท่านั้นคือ **BOOT** ซึ่ง ทำให้ข้อความถูกสร้างขึ้นหลังการติดตั้งเสร็จสมบูรณ์เพื่อแจ้งว่าระบบจำเป็นต้องรีสตาร์ท

Fileset`.cfgfiles`
: รายการไฟล์ที่ผู้ใช้กำหนดคอนฟิกได้ และคำสั่งการประมวลผลสำหรับใช้เมื่อ นำใช้ระดับการติดตั้งที่ใหม่กว่าหรือเท่ากับของ fileset ที่ติดตั้ง ไว้แล้ว ก่อนการเรียกคืนไฟล์ที่แสดงรายการในไฟล์ *Fileset*.**al** ระบบจะบันทึกไฟล์ที่แสดงไว้ในไฟล์ *Fileset*.**cfgfiles** ในภายหลัง ไฟล์ที่บันทึกไว้เหล่านี้ถูกประมวลผลตามวิธีการจัดการที่ระบุ ในไฟล์ *Fileset*.**cfgfiles**

Fileset`.copyright`
: ข้อมูลลิขสิทธิ์ที่จำเป็นสำหรับ fileset ไฟล์นี้ประกอบด้วย ชื่อเต็มของผลิตภัณฑ์ซอฟต์แวร์ตามด้วยหมายเหตุลิขสิทธิ์

Fileset`.err`
: ไฟล์เท็มเพลตข้อผิดพลาดที่ใช้เป็นอินพุตไปยังคำสั่ง **errupdate** เพื่อเพิ่มหรือลบรายการใน Error Record Template Repository ไฟล์นี้โดยปกติใช้โดยใช้ซอฟต์แวร์สนับสนุนอุปกรณ์ คำสั่ง **errupdate** สร้างไฟล์ *Fileset*.**undo.err** สำหรับใช้ในการคลีนอัพ โปรดดูที่ คำสั่ง **errupdate** สำหรับ ข้อมูลเกี่ยวกับรูปแบบของไฟล์ *Fileset*.**err** ไฟล์นี้สามารถรวมในส่วนรากของ fileset เท่านั้น

Fileset`.fixdata`
: ไฟล์รูปแบบ Stanza ไฟล์นี้มีข้อมูลเกี่ยวกับการแก้ไข ที่อยู่ใน fileset หรือการอัพเดต fileset

Fileset.inventory
ไฟล์ inventory ไฟล์นี้มีข้อมูลผลิตภัณฑ์ที่สำคัญต่อซอฟต์แวร์ ที่จำเป็นสำหรับไฟล์ใน fileset หรือการอัพเดต fileset ไฟล์ inventory เป็นไฟล์รูปแบบ stanza ที่มีรายการสำหรับแต่ละไฟล์ที่จะถูกติดตั้ง หรืออัพเดต

Fileset.namelist
รายการ filesets ที่เลิกใช้แล้วและ fileset ปัจจุบัน (ถ้ามี) ที่ มีไฟล์ที่มีอยู่ใน fileset ที่จะถูกติดตั้ง ไฟล์นี้ ใช้สำหรับการติดตั้งของผลิตภัณฑ์ซอฟต์แวร์ที่ทำแพ็คเกจใหม่เท่านั้น

Fileset.odmadd or *Fileset*.*.odmadd
Stanzas ที่จะเพิ่มไปยังฐานข้อมูล ODM (Object Data Manager)

Fileset.rm_inv
ลบไฟล์ inventory ออก ไฟล์นี้สำหรับการติดตั้งของผลิตภัณฑ์ซอฟต์แวร์ ที่ทำแพ็คเกจใหม่และต้องมีอยู่แล้วถ้า fileset ไม่ได้ทำการแทนโดยตรง สำหรับ fileset ที่เลิกใช้แล้ว ไฟล์รูปแบบ stanza นี้มีชื่อของไฟล์ที่ จำเป็นต้องถูกลบออกจาก filesets ที่เลิกใช้แล้ว

Fileset.size
ไฟล์นี้มีข้อกำหนดพื้นที่สำหรับไฟล์ ที่อยู่ใน fileset นี้ดังอธิบายก่อนหน้านี้ในส่วนนี้

Fileset.trc
ไฟล์เท็มเพลตรายงานกการติดตาม คำสั่ง **trcupdate** ใช้ไฟล์นี้เพื่อเพิ่ม, แทนที่ หรือลบรายการ รายงานการการติดตามใน ไฟล์ **/etc/trcfmt** คำสั่ง **trcupdate** จะสร้างไฟล์ *Fileset*.**undo.trc** สำหรับใช้ในการคลีนอัพ เฉพาะส่วน root ของแพ็คเกจ เท่านั้นที่สามารถมีไฟล์ *Fileset*.**trc**

lpp.acf
ไฟล์ควบคุมการเก็บถาวรสำหรับทั้งแพ็คเกจ ไฟล์นี้จำเป็นต้องใช้เฉพาะ เมื่อเพิ่มหรือแทนที่ไฟล์สมาชิกที่เก็บถาวรไปยังไฟล์เก็บถาวรที่มีอยู่แล้ว บนระบบ ไฟล์ควบคุมการเก็บถาวรประกอบด้วยบรรทัดที่มี คู่ของไฟล์สมาชิกในไดเร็กทอรีชั่วคราวดังแสดงในไฟล์ *Fileset*.**al** และไฟล์เก็บถาวรที่มีสมาชิก อยู่ ทั้งสองถูกแสดงรายการโดยสัมพันธ์กับ root ดังใน:

```
./usr/ccs/lib/libc/member.o ./usr/ccs/lib/libc.a
```

lpp.README
ไฟล์ Readme ไฟล์นี้มีข้อมูลที่ผู้ใช้ควรอ่านก่อน ใช้ซอฟต์แวร์ ไฟล์นี้เป็นทางเลือกและยังสามารถมีชื่อเป็น **README**, **lpp.doc**, **lpp.instr** หรือ **lpp.lps**

productid
ไฟล์ identification ผลิตภัณฑ์ ไฟล์นี้ประกอบด้วยบรรทัดเดียวที่ระบุ ชื่อผลิตภัณฑ์, identifier ผลิตภัณฑ์ (จำกัด 20 อักขระ) และชื่อ คุณลักษณะเผื่อเลือก (จำกัด 10 อักขระ)

ไฟล์ที่สามารถเรียกทำงานที่เป็นทางเลือกที่อยู่ในไฟล์ liblpp.a

ไฟล์เรียกทำงานที่เฉพาะสำหรับผลิตภัณฑ์ที่อธิบายใน ส่วนนี้ถูกเรียกใช้ระหว่างกระบวนการติดตั้ง ยกเว้นระบุ เป็นอย่างอื่น ชื่อไฟล์ที่ลงท้ายด้วย **_i** ถูกใช้ระหว่างการประมวลผล การติดตั้งเท่านั้น และชื่อไฟล์ที่ลงท้ายด้วย **_u** ถูกใช้ ในการประมวลผล การอัพเดต fileset เท่านั้น ไฟล์ทั้งหมดที่อธิบายในส่วนนี้เป็นทางเลือก และสามารถเป็นๆเชลล์สคริปต์ หรือโมดูลอ็อบเจ็กต์ที่รันได้ แต่ละ โปรแกรมควรมีค่าส่งกลับเป็น 0 (ศูนย์) ยกเว้นโปรแกรมต้องการให้ การติดตั้งหรือการอัพเดตล้มเหลว

Fileset.config or *Fileset*.config_u
แก้ไขการตั้งค่าเมื่อใกล้สิ้นสุดกระบวนการการติดตั้งดีฟอลต์ หรือการอัพเดต *Fileset*.**config** ถูกใช้ ระหว่างกระบวนการการติดตั้งเท่านั้น

Fileset.odmdel or*Fileset*.*.odmdel
อัพเดตข้อมูลฐานข้อมูล ODM สำหรับ fileset ก่อนเพิ่มรายการ ODM ใหม่สำหรับ fileset ระเบียบวิธีการตั้งชื่อไฟล์ **odmdel** เปิดให้ fileset มีไฟล์ **odmdel** หลายไฟล์

Fileset.pre_d
บ่งชี้ว่า fileset อาจถูกลบออกระหว่างการถอนการติดตั้ง โปรแกรม ต้องส่งกลับค่า 0 (ศูนย์) ถ้าสามารถลบ fileset ออก Filesets สามารถ ลบออกได้โดยค่าดีฟอลต์ โปรแกรมควรสร้างข้อความแสดงความผิดพลาดที่ระบุ เหตุผลที่ fileset ไม่สามารถลบออก

Fileset.pre_i or *Fileset*.pre_u
รันก่อนที่จะเรียกคืน หรือบันทึกไฟล์จากรายการนำใช้ใน แพ็กเกจ แต่หลังการลบไฟล์ออกจากเวอร์ชันของ fileset ที่ติดตั้งก่อนหน้า

Fileset.pre_rej
รันก่อนการดำเนินการปฏิเสธ หรือก่อนการแสดงตัวอย่าง ของการดำเนินการปฏิเสธของ fileset ใช้สคริปต์เพื่อพิจารณาว่า fileset สามารถถูกปฏิเสธได้หรือไม่ อย่าใช้สคริปต์นี้เพื่อรันคำสั่งใดๆ ซึ่ง เปลี่ยนแปลงค่าใดๆ บนระบบ ถ้าสคริปต์ออกจากการทำงานโดยมีโค้ดส่งกลับไม่ใช่ศูนย์ การดำเนินการปฏิเสธจะไม่ได้รับอนุญาตให้ทำ

Fileset.pre_rm
รันระหว่างการติดตั้ง fileset ก่อนที่จะลบไฟล์ออกจาก เวอร์ชันของ fileset ที่ติดตั้งก่อนหน้านี้

Fileset.post_i or *Fileset*.post_u
รันหลังการเรียกคืนไฟล์จากรายการนำใช้ของการติดตั้งหรืออัพเดต fileset

Fileset.unconfig *Fileset*.unconfig_u
เลิกดำเนินการประมวลผลการตั้งค่าที่ดำเนินในการติดตั้งหรือการอัพเดต ระหว่างการถอนการติดตั้งหรือการปฏิเสธของ fileset *Fileset*.**unconfig** ถูกใช้ระหว่างกระบวนการถอนการติดตั้งเท่านั้น

Fileset.unodmadd
ลบรายการที่ถูกเพิ่มในฐานข้อมูล ODM ระหว่างการติดตั้ง หรือการอัพเดต

Fileset.unpost_i or*Fileset*.unpost_u
เลิกทำการประมวลผลที่ดำเนินการต่อจากการเรียกคืนไฟล์จาก รายการนำใช้การติดตั้งหรือการอัพเดตระหว่างการถอนการติดตั้งหรือการปฏิเสธของ fileset

Fileset.unpre_i or *Fileset*.unpre_u
เลิกทำการประมวลผลที่ดำเนินการก่อนหน้าการเรียกคืนไฟล์จาก รายการนำใช้การติดตั้งหรือการอัพเดตระหว่างการถอนการติดตั้งหรือการปฏิเสธของ fileset

ถ้าไฟล์ใดๆ ของไฟล์เรียกทำงานเหล่านี้รันคำสั่งที่ อาจเปลี่ยนแปลงการตั้งค่าอุปกรณ์บนเครื่อง ไฟล์เรียกทำงานนั้นควร ตรวจสอบตัวแปรสภาวะแวดล้อม INUCLIENTS ก่อนรันคำสั่ง ถ้า ตัวแปรสภาวะแวดล้อม INUCLIENTS ถูกตั้งค่า ไม่ควรรันคำสั่งสภาพแวดล้อม Network Installation Management (NIM) ใช้คำสั่ง **installp** สำหรับวัตถุประสงค์หลายอย่าง บางข้ออาจต้องการคำสั่ง **installp** เพื่อตัดข้ามการประมวลผลปกติบางขั้นตอน NIM ตั้งค่า ตัวแปรสภาวะแวดล้อม INUCLIENTS เมื่อการประมวลผลปกติต้องถูกข้าม

ถ้ากระบวนการติดตั้งดีฟอลต์ไม่เพียงพอ สำหรับแพ็คเกจของคุณ คุณสามารถใช้ไฟล์เรียกทำงานดังต่อไปนี้ในไฟล์ **liblpp.a** ถ้าไฟล์เหล่านี้จัดให้มีในแพ็กเกจของคุณ คำสั่ง **installp** จะใช้ไฟล์ที่จัดให้โดยแพ็กเกจ แทนไฟล์ค่าดีฟอลต์ของระบบ ไฟล์ที่จัด

ให้โดยแพ็คเกจของคุณต้องมี การทำงานเหมือนกับไฟล์ดีฟอลต์มิฉะนั้นจะเกิดผลลัพธ์ที่ไม่คาดคิด คุณสามารถใช้ไฟล์ดีฟอลต์เป็นโมเดลสำหรับการสร้างไฟล์ของคุณเอง ขอแนะนำ อย่างยิ่งให้ใช้ไฟล์ดีฟอลต์แทนไฟล์ที่แพ็คเกจจัดให้มี

`instal`
ใช้แทนสคริปต์การติดตั้งดีฟอลต์ **/usr/lib/instl/instal** คำสั่ง **installp** เรียกใช้ไฟล์เรียกทำงานนี้ถ้า fileset ในแพ็คเกจการติดตั้งถูกนำใช้

`lpp.cleanup`
ใช้แทนสคริปต์คลีนอัพการติดตั้งดีฟอลต์ **/usr/lib/instl/cleanup** คำสั่ง **installp** เรียกใช้ไฟล์เรียกทำงานนี้ถ้า fileset ในแพ็คเกจการติดตั้งหรือการอัพเดตถูกนำใช้ บางส่วน และต้องถูกลบค่าเพื่อใส่ fileset กลับเข้าไปใน สถานะที่สอดคล้องกัน

`lpp.deinstal`
ใช้แทนสคริปต์การลบ fileset ดีฟอลต์ **/usr/lib/instl/deinstal** ไฟล์เรียกทำงานต้องอยู่ในไดเร็กทอรี **/usr/lpp/***PackageName* คำสั่ง **installp** เรียกใช้ไฟล์เรียกทำงานนี้ถ้า fileset ในแพ็คเกจการติดตั้งถูกลบออก

`lpp.reject`
ใช้แทนสคริปต์การปฏิเสธการติดตั้งดีฟอลต์ **/usr/lib/instl/reject** คำสั่ง **installp** เรียกใช้ไฟล์เรียกทำงานนี้ถ้าการอัพเดต fileset ในแพ็คเกจการอัพเดตถูกปฏิเสธ (สคริปต์ ดีฟอลต์ **/usr/lib/instl/reject** คือลิงก์ไปยังสคริปต์ **/usr/lib/instl/cleanup**)

`update`
ใช้แทนสคริปต์กาสคริปต์อัพเดต fileset ดีฟอลต์ **/usr/lib/instl/update** คำสั่ง **installp** เรียกใช้ไฟล์เรียกทำงานนี้ถ้า fileset ในแพ็คเกจการอัพเดตถูกนำใช้ (สคริปต์ **/usr/lib/instl/update** ดีฟอลต์คือลิงก์ไปยังสคริปต์ **/usr/lib/instl/instal**)

เพื่อให้แน่ใจว่าเข้ากันได้กับคำสั่ง **installp** ไฟล์เรียกทำงาน **instal** หรือ **update** ที่มีมากับซอฟต์แวร์แพ็กเกจต้อง:

- ประมวลผล filesets ทั้งหมดในซอฟต์แวร์แพ็กเกจ โดยสามารถประมวลผลการติดตั้งสำหรับ filesets ทั้งหมดหรือเรียกใช้ไฟล์เรียกทำงานอื่นสำหรับแต่ละ fileset
- ใช้คำสั่ง **inusave** เพื่อบันทึกระดับปัจจุบันของไฟล์ใดๆ ที่จะติดตั้ง
- ใช้คำสั่ง **inurest** เพื่อเรียกคืนไฟล์ที่จำเป็นทั้งหมดสำหรับส่วน usr จาก สื่อบันทึกการแจกจ่าย
- ใช้คำสั่ง **inucp** เพื่อทำสำเนาไฟล์ที่จำเป็นทั้งหมดสำหรับส่วน root จากไดเร็กทอรี **/usr/lpp/***Package_Name***/inst_root**
- สร้างไฟล์ **$INUTEMPDIR/status** ที่ระบุว่าสำเร็จหรือล้มเหลวสำหรับแต่ละ fileset ที่กำลังถูกถูกติดตั้งหรืออัพเดต
- ส่งกลับโค้ดออกจากการรทำงานที่ระบุสถานะของการติดตั้ง ถ้าไฟล์เรียกทำงาน **instal** หรือ **update** ส่งโค้ดส่งกลับที่ไม่เป็นศูนย์ และไม่พบไฟล์ **status** กระบวนการติดตั้งจะถูกว่า filesets ทั้งหมดล้มเหลว

ไฟล์ที่สามารถเรียกทำงานได้ที่เป็นทางเลือกที่อยู่ในไฟล์ Fileset.al

Fileset`.unconfig_d`
เลิกทำการดำเนินการตั้งค่า fileset เฉพาะที่ทำระหว่าง การติดตั้งและการอัพเดตของ fileset ไฟล์ *Fileset***.unconfig_d** ถูกใช้เมื่อแฟล็ก **-u** ถูกระบุกับคำสั่ง **installp** ถ้าไฟล์นี้ไม่ได้จัดให้มีและแฟล็ก **-u** ถูกระบุ การดำเนินการ *Fileset***.unconfig,** *Fileset***.unpost_i** และ *Fileset***.unpre_i** จะถูกดำเนินการ

คำอธิบายเพิ่มเติมสำหรับการติดตั้งไฟล์ควบคุม eไฟล์ Fileset.cfgfiles

ชุดไฟล์ *Fileset*.**cfgfiles** แสดงรายการไฟล์การตั้งค่าที่จำเป็นต้องบันทึกเพื่อใช้โอนย้ายไปยัง fileset เวอร์ชันใหม่โดยไม่ให้สูญเสียข้อมูลที่ผู้ใช้ ตั้งค่าไว้ ในการรักษาข้อมูลการตั้งค่าของผู้ใช้ ไฟล์ *Fileset*.**cfgfiles** ต้องถูกจัดให้มีในไฟล์ **liblpp.a** ที่เหมาะสม (usr หรือ root)

Fileset.**cfgfiles** มีรายการหนึ่งบรรทัดสำหรับแต่ละไฟล์ที่จะถูกบันทึก แต่ละรายการ มีชื่อไฟล์ (ชื่อพาธที่สัมพันธ์กับ root), ตัวคั่น white-space และคีย์เวิร์ดอธิบายวิธีการจัดการสำหรับการโอนย้ายไฟล์ คีย์เวิร์ดวิธีการจัดการได้แก่:

`preserve`
แทนที่ไฟล์เวอร์ชันใหม่ที่ติดตั้งด้วยเวอร์ชันที่บันทึกไว้ จากไดเร็กทอรีที่บันทึก หลังการแทนที่ไฟล์ใหม่ด้วยเวอร์ชันที่บันทึก ไฟล์ที่บันทึกไว้จากไดเร็กทอรีบันทึกการตั้งค่าจะถูกลบออก

`auto_merge`
ระหว่างการประมวลผล *Fileset*.**post_i** ไฟล์เรียกทำงานที่จัดโดยผลิตภัณฑ์จะรวมข้อมูลที่จำเป็นจาก ไฟล์เวอร์ชันใหม่ที่ติดตั้งเข้ากับเวอร์ชันก่อนหน้าของไฟล์ ที่บันทึกในไดเร็กทอรีบันทึกการตั้งค่า หลังการประมวลผล *Fileset*.**post_i** คำสั่ง **installp** จะแทนที่ไฟล์เวอร์ชันใหม่ที่ติดตั้งกับเวอร์ชันที่รวม ในไดเร็กทอรีบันทึกการตั้งค่า (ถ้ามี) จากนั้นลบ ไฟล์ที่บันทึก

`hold_new`
แทนที่ไฟล์เวอร์ชันใหม่ที่ติดตั้งด้วยเวอร์ชันที่บันทึกไว้ จากไดเร็กทอรีที่บันทึก ไฟล์เวอร์ชันใหม่ถูกนำไปวางในไดเร็กทอรีบันทึก การตั้งค่าแทนเวอร์ชันเก่า ผู้ใช้จะสามารถอ้างถึง เวอร์ชันใหม่

`user_merge`
เก็บไฟล์เวอร์ชันใหม่ที่ติดตั้งบนระบบ และรักษา ไฟล์เวอร์ชันเก่าในไดเร็กทอรีบันทึกการตั้งค่า ผู้ใช้ จะสามารถอ้างอิงเวอร์ชันเก่าเพื่อดำเนินการผสานใดๆ ที่อาจ จำเป็น คีย์เวิร์ดนี้ควรเลี่ยงถ้าทำได้

`other`
ใช้ในกรณีที่ไม่มีวิธีการจัดการที่กำหนดวิธีอื่น ที่เหมาะสม คำสั่ง **installp** จะบันทึกไฟล์ใน ไดเร็กทอรีบันทึกไฟล์และไม่ให้การสนับสนุนอื่นๆ การดำเนินการอื่นใด และการจัดการไฟล์การตั้งค่าต้องกระทำโดยไฟล์เรียกทำงาน ที่ผลิตภัณฑ์จัดให้ผู้พัฒนาผลิตภัณฑ์มีหน้าที่จัดทำเอกสาร การจัดการไฟล์

ไฟล์เรียกทำงาน *Fileset*.**post_i** สามารถใช้เพื่อทำการจัดการที่เจาะจงหรือการผสาน ข้อมูลการตั้งค่าที่ไม่สามารถกระทำได้ผ่านทางกระบวนการติดตั้ง ดีฟอลต์

ไฟล์คอนฟิกูเรชันจะถูกแสดงรายการในไฟล์*Fileset*.**cfgfiles** จะถูกบันทึกในไดเร็กทอรีบันทึกคอนฟิกูเรชันที่มีชื่อพาธแบบสัมพันธ์เดียวกับที่ระบุในไฟล์ *Fileset*.**cfgfiles** ชื่อของไดเร็กทอรีบันทึก การตั้งค่าถูกเก็บในตัวแปรสภาวะแวดล้อม **MIGSAVE** ไดเร็กทอรีบันทึกที่สอดคล้องกับส่วนของแพ็กเกจที่กำลังถูกติดตั้ง ไดเร็กทอรีต่อไปนี้เป็นไดเร็กทอรีบันทึกการตั้งค่า:

`/usr/lpp/save.config`
สำหรับส่วน usr

`/lpp/save.config`
สำหรับส่วน root

ถ้ารายการไฟล์ที่คุณจำเป็นต้องบันทึกต่างกัน ขึ้นกับระดับที่ติดตั้งขณะนี้ของ fileset ไฟล์ *Fileset*.**cfgfiles** ต้องมีรายการ ไฟล์การตั้งค่าทั้งหมดที่อาจพบ ถ้าจำเป็น ไฟล์เรียกทำงาน *Fileset*.**post_i** (หรือไฟล์เรียกทำงานที่ จัดให้มีโดยผลิตภัณฑ์อื่น) ต้องจัดการส่วนที่ต่างกัน

ตัวอย่าง สมมุติคุณมี fileset (**change.rte**) ที่มีหนึ่งไฟล์สามารถตั้งค่าได้ ดังนั้น ใน root **change.rte.cfgfiles** จะมีไฟล์หนึ่งถูกแสดงรายการ:

```
/etc/change_user   user_merge
```

เมื่อโอนย้ายจาก fileset เก่าของคุณ (**change.obj**) ไปยัง **change.rte** คุณไม่สามารถรักษา านี้ได้เนื่องจากรูปแบบเปลี่ยนแปลง อย่างไรก็ตาม เมื่อโอนย้ายจาก ระดับเก่า **change.rte** ไปเป็นระดับใหม่ **change.rte** ไฟล์สามารถคงรักษาไว้ได้ ในกรณีนี้ คุณอาจต้องการ สร้างสคริปต์ **change.rte.post_i** ที่ตรวจสอบเพื่อดูว่า fileset ใดที่คุณกำลังโอนย้าย และเพื่อดำเนินการได้เหมาะสม วิธีนี้ ถ้าผู้ใช้ ทำการเปลี่ยนแปลงในไฟล์ **/etc/change_user** ไฟล์จะถูกบันทึก

สคริปต์ root **change.bar.post_i** อาจเป็นดังนี้:

```
#! /bin/ksh
rc=0
grep -q change.rte $INSTALLED_LIST
if [$? = 0]
ดังนั้น
mv $MIGSAVE/etc/change_user/ /etc/change_user
rc=1
fi
exit $rc
```

$INSTALLED_LIST ถูกสร้างและเอ็กซ์พอร์ตโดย **installp** โปรดดูที่ การติดตั้งสำหรับ Control Files Specifically for Repackaged Products สำหรับข้อมูลเพิ่มเติมเกี่ยวกับไฟล์คอนฟิกูเรชัน *Fileset*.**installed_list** ตัวแปร *$MIGSAVE* มีชื่อของไดเร็กทอรีที่ไฟล์การตั้งค่าส่วน root ถูกบันทึก

คำสั่ง **installp** ไม่สร้าง คำเตือนหรือข้อความแสดงความผิดพลาดถ้าไม่พบไฟล์ที่แสดงรายการในไฟล์ *Fileset*.**cfgfiles** คำสั่ง **installp** ยังไม่สามารถข้อความสำหรับไฟล์ที่ไม่พบ ระหว่างเฟสการประมวลผล *Fileset*.**post_i** ต่อไปนี้เมื่อไฟล์การตั้งค่าที่บันทึกถูกประมวลผลตาม วิธีการจัดการ ถ้าจำเป็นต้องมีคำเตือนหรือข้อความแสดงความผิดพลาด ไฟล์เรียกทำงานที่ผลิตภัณฑ์จัดให้มีต้องสร้างข้อความนั้น

ดังตัวอย่างของไฟล์ *Fileset*.**cfgfiles** ชุดไฟล์ `Product_X.option1` ต้องกู้คืนข้อมูลการกำหนดค่าจากไฟล์คอนฟิกูเรชันสามไฟล์ ที่อยู่ในส่วน root ของชุดไฟล์ `Product_X.option1.cfgfiles` ถูกรวมในส่วน root ของไฟล์ `liblpp.a` และมีรายการต่อไปนี้:

```
./etc/cfg_leafpreserve
./etc/cfg_pudding hold_new
./etc/cfg_newtonpreserve
```

ไฟล์ Fileset.fixdata

Fileset.`fixdata`
ไฟล์รูปแบบ stanza ที่อธิบายการแก้ไขที่มีอยู่ในการอัพเดต fileset (หรือในการติดตั้ง fileset ถ้าใช้แทนอัพเดต)

ข้อมูลในไฟล์นี้ถูกเพิ่มในฐานข้อมูลการแก้ไข คำสั่ง **instfix** ใช้ฐานข้อมูลนี้เพื่อระบุการแก้ไข ที่ติดตั้งบนระบบ ถ้า *Fileset*.**fixdata** มีอยู่ในแพ็คเกจ ข้อมูลการแก้ไขในฐานข้อมูลการแก้ไข จะถูกอัพเดตเมื่อแพ็กเกจถูกนำใช้

แต่ละการแก้ไขใน fileset ควรมี stanza ของตนในไฟล์ *Fileset*.**fixdata** *Fileset*.**fixdata** stanza มี รูปแบบต่อไปนี้:

```
fix:
name = FixKeyword
abstract = Abstract
type = {f | p}
filesets = FilesetName FilesetLevel
[FilesetName FilesetLevel ...]
[symptom = [Symptom]]
```

- *FixKeyword* ต้องไม่เกิน 16 อักขระ
- *Abstract* อธิบายการแก้ไขและต้องไม่เกิน 60 อักขระ
- ในฟิลด์ **type f** แทน การแก้ไข และ **p** แทนการอัพเดตการบำรุงรักษาเชิงป้องกัน
- ฟิลด์ **filesets** มีรายการของ filesets และระดับ filesets ที่คั่นด้วยอักขระขึ้นบรรทัดใหม่
- *FilesetLevel* เป็นระดับเริ่มต้นที่ fileset นำส่งการแก้ไขทั้งหมดหรือบางส่วน
- *Symptom* เป็นรายละเอียดเผื่อเลือกของปัญหา ที่แก้ไขโดยการแก้ไข *Symptom* ไม่จำกัดจำนวน อักขระ

ตัวอย่างต่อไปนี้แสดง *Fileset*.**fixdata** stanza สำหรับปัญหา MS21235 การแก้ไขสำหรับปัญหานี้มีอยู่ในสอง filesets

```
fix:
name = MS21235
abstract = 82 gigabyte diskette drive unusable on Mars
type = f
filesets = devices.mca.8d77.rte 12.3.6.13
devices.mca.8efc.rte 12.1.0.2
symptom = The 82 gigabyte subatomic diskettes fail to operate in a Martian environment.
```

ไฟล์ Fileset.inventory

Fileset.inventory
: ไฟล์ที่มีข้อมูลที่ระบุเกี่ยวกับแต่ละไฟล์ที่จะ ติดตั้งหรืออัพเดตสำหรับ fileset

sysck
: คำสั่งที่ใช้ไฟล์ *Fileset*.**inventory** เพื่อป้อนข้อมูลชื่อไฟล์ ชื่อผลิตภัณฑ์ ประเภท เช็คซัม ขนาด ลิงก์ และ symlink ในฐานข้อมูลซอฟต์แวร์

ไฟล์ *Fileset*.**inventory** จำเป็นต้องใช้ในแต่ละส่วนของ fileset ที่ติดตั้ง และอัพเดตไฟล์ ถ้าแพ็กเกจมีส่วน root ที่ไม่มีไฟล์ที่จะติดตั้ง (ทำการตั้งค่าเท่านั้น) ส่วน root ไม่จำเป็นต้องใช้ ไฟล์ *Fileset*.**inventory**

หมายเหตุ: ไฟล์ *Fileset*.**inventory** ไม่สนับสนุนไฟล์ที่ มีขนาดใหญ่กว่า 2 GB ถ้าคุณจัดส่งไฟล์ที่มีขนาดใหญ่กว่า 2 GB ให้รวมในไฟล์ *fileset*.**al** ของคุณ แต่อย่ารวมในไฟล์ *Fileset*.**inventory** ของคุณ **sysck** ไม่ได้รับการอัพเดตเพื่อจัดการไฟล์ ที่มีขนาดใหญ่กว่า 2GB และระบบไฟล์ **/usr** บนเครื่องส่วนใหญ่ จะไม่ถูกสร้างให้มีความสามารถสำหรับไฟล์ที่มีขนาดใหญ่กว่า 2GB (โดยดีฟอลต์)

ไฟล์ inventory ประกอบด้วยข้อความ ASCII ในรูปแบบ stanza ชื่อของ stanza คือชื่อพาธเต็มของไฟล์ที่จะถูกติดตั้ง ชื่อ stanza ลงท้ายด้วยโคลอน (:) และตามด้วยอักขระขึ้นบรรทัดใหม่ ไฟล์แอ็ตทริบิวต์ตามหลังชื่อ stanza และมีรูปแบบ *Attribute=Value* แต่ละแอ็ตทริบิวต์ ถูกแสดงบนบรรทัดแยก

Fileset.**inventory** stanza มีรูปแบบต่อไปนี้:

```
inventory:
type     = type
class    = inventory,apply,C2_exclude,fileset
owner    = owner_name
group    = group_name
mode     = TCB | SUID | SGID,permissions
target   = fullpath_filename
link     = fullpath_to_hardlink [additional_hardlinks]
size     =<blank> | VOLATILE |size
checksum =<blank> | VOLATILE |"checksum"
```

แอ็ตทริบิวต์	คำอธิบาย
file_name	พาธเต็มไปยังไฟล์หรือลิงก์โดยสัมพันธ์กับ root (./) ต่อด้วยโคลอน (:) จากนั้นบรรทัดใหม่ ความยาวสูงสุดสำหรับชื่อพาธแบบเต็ม นี้คือ 255 อักขระ
type	ประเภทของรายการ file_name โดยที่ประเภทที่ใช้ได้เป็นประเภทหนึ่งในประเภทต่อไปนี้: **คีย์เวิร์ด ความหมาย** **FILE** ไฟล์มาตรฐาน **DIRECTORY** ไดเร็กทอรี **SYMLINK** พาธเต็มไปยังลิงก์สัญลักษณ์
class	ระบุวิธีที่ **file_name** จะถูกอ้างอิง ระหว่างการติดตั้ง ฟิลด์นี้ต้องมีอย่างน้อยสองคีย์เวิร์ดของคีย์เวิร์ดต่อไปนี้: **รายการประเภท ความหมาย** **inventory** ระบุว่าไฟล์ยังคงอยู่หลังการติดตั้งเสร็จสมบูรณ์ ไฟล์ถูกเพิ่มในฐานข้อมูล inventory SWVPD ค่านี้ไม่ควรใช้กับ ไฟล์ประเภท **A** ในไฟล์ *fileset.il* **apply** ระบุว่าไฟล์จะถูกเรียกคืนจากสื่อบันทึกการสื่อบันทึก ฟิลด์ *file_name* แสดงรายการอยู่ในรายการนำใช้ (*fileset.al*) ค่านี้ ไม่ควรใช้สำหรับไฟล์ประเภท **I** ในไฟล์ *fileset.il* **C2_exclude** ระบุว่าไฟล์ควรถูกแยกออกจากการทำงานบนระบบ C2 Secure ถ้าใช้แฟล็กนี้ ไฟล์ควรถูกแสดงรายการในไฟล์ *fileset.tcb* ด้วย
owner	ระบุเจ้าของไฟล์หลังการติดตั้ง อย่าใช้ uid สำหรับ ฟิลด์นี้ ค่าแอ็ตทริบิวต์ต้องเป็นชื่อเจ้าของและต้องยาวไม่เกิน 8 อักขระ
group	ระบุกลุ่มไฟล์ อย่าใช้ gid สำหรับฟิลด์นี้ ค่าแอ็ตทริบิวต์ ต้องเป็นชื่อกลุ่ม และต้องยาวไม่เกิน 8 อักขระ
mode	ระบุโหมดของไฟล์ ค่าต้องมีสิทธิของ ไฟล์ในรูปแบบฐานแปด คีย์เวิร์ดใดๆ ต่อไปนี้สามารถนำหน้าค่า สิทธิ Items ในรายการถูกคั่นด้วยเครื่องหมายจุลภาค **รายการโหมด** ความหมาย **TCB** ส่วนของ "Trusted Computing Base" ถ้าไฟล์เป็น **SUID** root หรือระบบ **SGID** ไฟล์ต้องเป็น **TCB** **SUID** ไฟล์นี้มีการตั้งค่าบิต ID ผู้ใช้การตั้งค่า นี้ไม่มีความหมายบนรายการ **DIRECTORY** **SGID** ไฟล์นี้มีการตั้งค่าบิต ID กลุ่มการตั้งค่า ถ้าตั้งค่าบนรายการ **DIRECTORY** ค่านี้จะบังคับไฟล์ทั้งหมดที่สร้างในไดเร็กทอรีนั้นมีกลุ่มเดียวกันกับ ไดเร็กทอรี *permissions* ต้องเป็นเลขฐานแปดสามหลัก เช่น 644 **หมายเหตุ:** ถ้าประเภทเป็น SYMLINK โหมดต้องเป็น 777 ไม่มีรายการอื่นที่ใช้ได้

แอ็ตทริบิวต์	คำอธิบาย
link	แสดงรายการฮาร์ดลิงก์ใดๆ ที่ไปที่ไฟล์นี้ ถ้ามีหลายฮาร์ดลิงก์ แต่ละ พาธเต็มที่ไปยังฮาร์ดลิงก์จะถูกคั่นด้วยเครื่องหมายจุลภาค ฮาร์ดลิงก์ต้องอยู่ในไดเร็กทอรีเดียวกับซอร์สไฟล์ ถ้าชนิดของ รายการ คือ **SYMLINK** ดังนั้น **link** จะไม่สามารถใช้ได้ ความยาวสูงสุดของชื่อพาธเต็ม คือ 255 อักขระ
target	ใช้ได้กับ **type=SYMLINK** เท่านั้น นี้คือชื่อไฟล์ของ พาธเต็มไปยังปลายทางของลิงก์ ถ้าลิงก์ที่ถูกสร้างมาจาก /usr/bin ไปยัง /bin ชื่อไฟล์จะเป็น /bin และ ควรเป็น /usr/bin ความยาวสูงสุดของชื่อพาธเต็ม คือ 255 อักขระ
ขนาด	**รายการโหมด** ความหมาย *blank* ถ้าฟิลด์นี้เป็นค่าว่าง ขนาดของชื่อไฟล์จะถูกพิจารณาในตอน ทำการติดตั้ง ข้อเสียเปรียบในการใช้อ็อพชันนี้คือไฟล์ถูกทำให้เสียหายระหว่างการติดตั้ง โดยลูกค้าไม่ได้รับการแจ้งให้ทราบ **VOLATILE** ถ้าขนาดไฟล์ถูกคาดว่าต้องมีการเปลี่ยนแปลงจาการดำเนินการปกติ ค่า สำหรับแอ็ตทริบิวต์นี้ต้องเป็น **VOLATILE** **ขนาด** ขนาดที่แท้จริงของไฟล์ **หมายเหตุ:** อย่ารวมฟิลด์ขนาดสำหรับรายการ **DIRECTORY** หรือ **SYMLINK**
checksum	**รายการโหมด** ความหมาย *blank* ถ้าฟิลด์นี้เป็นค่าว่าง ค่าที่ส่งกลับจากคำสั่ง **sum -r** จะถูกวางในฐานข้อมูล inventory SWVPD เมื่อไฟล์ถูกติดตั้ง **VOLATILE** ระบุขนาดของไฟล์เป็นจำนวนบล็อก ถ้าขนาดไฟล์ถูกคาดว่าต้องมีการเปลี่ยนแปลงจาการดำเนินการปกติ ค่าสำหรับแอ็ตทริบิวต์นี้ต้องเป็น **VOLATILE** **checksum** ผลรวมแท้จริงของไฟล์ที่ส่งกลับโดยคำสั่ง **sum -r** นี้ควรอยู่ในเครื่องหมายคำพูดคู่ **หมายเหตุ:** อย่ารวมฟิลด์เช็คซัมสำหรับรายการ **DIRECTORY** หรือ **SYMLINK**
fileset	ระบุ fileset ที่ไฟล์อยู่

หมายเหตุ: คำสั่ง **sysck** สร้างฮาร์ดลิงก์และลิงก์สัญลักษณ์ระหว่างการติดตั้งถ้าลิงก์เหล่านั้น ไม่มี ลิงก์สัญลักษณ์ส่วน root ควรถูกรวมอยู่ในส่วน root ไฟล์ *Fileset*.**inventory**

ตัวอย่าง fileset.inventory

ตัวอย่าง fileset.inventory ต่อไปนี้แสดงการใช้ **type**

```
/usr/bin:
        owner = bin
        group = bin
        mode = 755
        type = directory
        class = apply,inventory,bos.rte


/usr/bin/tcbck:
        owner = root
        group = security
        mode = TCB,SUID,550
        type = file
        class = apply,inventory,bos.rte.security
        size = 99770
```

```
        checksum = "17077     98 "


/usr/sbin/tsm:
        owner = root
        group = security
        mode = TCB,SUID,555
        links = /usr/sbin/getty,/usr/sbin/login
        class = apply,inventory,bos.rte,security
        size = 55086
        checksum = "57960     54 "
```

ไฟล์ควบคุมการติดตั้งเฉพาะผลิตภัณฑ์ที่ทำเป็นแพ็กเกจใหม่

Fileset.**installed_list**
: ไฟล์ที่สร้างโดยคำสั่ง **installp** เมื่อทำการติดตั้ง fileset จากแพ็กเกจถ้าพบว่า fileset (หรือบางรูปแบบของ fileset) ถูกติดตั้งอยู่แล้วบนระบบในระดับหนึ่ง

ฐานข้อมูลซอฟต์แวร์ถูกค้นหาเพื่อใช้พิจารณาว่า *Fileset* หรือ filesets ใดๆ ที่แสดงรายการในไฟล์ *Fileset*.**namelist** (ถ้ามีอยู่) ได้ถูกติดตั้งไว้แล้ว บนระบบ ถ้าเป็นเช่นนั้น fileset และระดับการติดตั้งจะถูกเขียนไปยัง ไฟล์ *Fileset*.**installed_list**

ถ้าถูกสร้าง *Fileset*.**installed_list** จะพร้อมใช้ในเวลาเรียกใช้ไฟล์เรียกทำงาน **rminstal** และ **instal** ไฟล์ *Fileset*.**installed_list** สามารถอยู่ในไดเร็กทอรีต่อไปนี้ ไดเร็กทอรีการทำงานแพ็กเกจ หรือ *PackageWorkDirectory*:

/usr/lpp/
: *PackageName* สำหรับส่วน usr

/lpp/ *PackageName* สำหรับส่วน root

ไฟล์ *Fileset*.**installed_list** มีรายการหนึ่งบรรทัดสำหรับแต่ละ fileset ที่ ถูกติดตั้ง แต่ละรายการมีชื่อ fileset และระดับ fileset

ดังตัวอย่าง สมมุติว่าขณะที่ `storm.rain 1.2.0.0` fileset กำลังถูกติดตั้ง คำสั่ง **installp** พบว่า `storm.rain 1.1.0.0` ได้ถูกติดตั้งแล้ว คำสั่ง **installp** จะสร้างไฟล์ *PackageWorkDirectory*`/storm.rain.installed_list` ที่มีเนื้อหาต่อไปนี้:

```
storm.rain 1.1.0.0
```

อีกตัวอย่างหนึ่ง สมมุติว่า `Baytown.com` fileset มีไฟล์ `Baytown.com.namelist` ที่มีรายการต่อไปนี้:

```
Pelly.com
GooseCreek.rte
CedarBayou.stream
```

ขณะติดตั้ง `Baytown.com 2.3.0.0` fileset คำสั่ง **installp** พบว่า `Pelly.com 1.2.3.0` และ `CedarBayou.stream 4.1.3.2` ถูกติดตั้งแล้ว คำสั่ง **installp** จะสร้างไฟล์ *PackageWorkDirectory*`/Baytown.com.installed_list` ที่มีเนื้อหาต่อไปนี้:

```
Pelly.obj 1.2.3.0
CedarBayou.stream 4.1.3.2
```

ไฟล์ Fileset.namelist

แอ็ตทริบิวต์	คำอธิบาย
Fileset.**namelist**	ไฟล์นี้จำเป็นต้องมีเมื่อชื่อ fileset เปลี่ยนแปลง หรือ fileset มีไฟล์ที่ถูกแพ็กเกจก่อนหน้านี้ใน filesets เก่า โดยมีชื่อ ของ filesets ทั้งหมดที่ก่อนหน้านี้มีไฟล์ที่ขณะนี้รวมอยู่ใน fileset ที่จะถูกติดตั้ง แต่ละชื่อ fileset ต้องปรากฏบนบรรทัดแยก

ไฟล์ *Fileset*.**namelist** ต้องมีอยู่ในไฟล์ **usr** or **root** part of the**liblpp.a** ไฟล์ *Fileset* . **namelist** จะใช้ได้สำหรับแพ็คเกจการติดตั้งเท่านั้น ใช้ไม่ได้สำหรับแพ็คเกจการอัพเดต

ในตอนเริ่มต้นการติดตั้ง คำสั่ง **installp** จะค้นหา Software Vital Product Data (SWVPD) เพื่อ พิจารณาว่า fileset หรือ fileset ใดๆ ที่แสดงรายการในไฟล์ *Fileset*.**namelist** ถูกติดตั้งไว้แล้วบน ระบบ คำสั่ง **installp** เขียนไปยังไฟล์ *Fileset*.**installed_list** ด้วยชื่อ fileset และ ระดับ fileset ที่พบว่าถูกติดตั้งแล้ว ทำให้ข้อมูลนี้มีใช้ได้สำหรับ ไฟล์เรียกทำงานที่จัดโดยผลิตภัณฑ์

ดังในตัวอย่างแบบง่ายของไฟล์ *Fileset*.**namelist** สมมุติว่า `small.business` fileset แทนที่ fileset ก่อนหน้าที่ชื่อ `family.business` แพ็กเกจผลิตภัณฑ์ `small.business` จะมีไฟล์ `small.business.namelist` ในส่วน usr ของไฟล์ **liblpp.a** ไฟล์ `small.business.namelist` มีรายการต่อไปนี้:

```
family.business
```

ดังตัวอย่างที่ซับซ้อนมากขึ้นไฟล์ *Fileset*.**namelist** จะเป็น fileset ที่ ถูกแบ่งออกเป็น fileset ของไคลเอ็นต์และ fileset ของเซิร์ฟเวอร์ `LawPractice.client` และ `LawPractice.server` filesets แทนที่ `lawoffice.mgr` fileset ก่อนหน้า `LawPractice.server` fileset ยังมีบางไฟล์ จาก `BusinessOffice.mgr` fileset เก่า ไฟล์ `LawPractice.client.namelist` ในไฟล์ **liblpp.a** สำหรับแพ็คเกจ `LawPractice` มีรายการต่อไปนี้:

```
lawoffice.mgr
```

ไฟล์ `LawPractice.server.namelist` ในไฟล์ **liblpp.a** สำหรับแพ็กเกจ `LawPractice` มีรายการต่อไปนี้:

```
lawoffice.mgr
BusinessOffice.mgr
```

ถ้าไฟล์ *Fileset*.**namelist** มีเพียงหนึ่งรายการเท่านั้น และ fileset ปัจจุบันไม่ใช่ การแทนที่โดยตรงสำหรับ fileset ที่แสดงในไฟล์ *Fileset* . **namelist** คุณ ต้องรวมไฟล์ *Fileset*.**rm_inv** ไว้ในไฟล์ **liblpp.a** กระบวนการติดตั้งใช้ ไฟล์ *Fileset*.**namelist** และ ไฟล์ *Fileset*.**rm_inv** เพื่อพิจารณา ว่า fileset เป็นการแทนที่โดยตรงสำหรับอีก fileset หรือไม่ ถ้าไฟล์ *Fileset*.**namelist** มีเพียงหนึ่งรายการเท่านั้น และ ไม่มีไฟล์ *Fileset*.**rm_inv** กระบวนการติดตั้งจะถือว่า fileset ใหม่เป็นการแทนที่โดยตรงสำหรับ fileset เก่า เมื่อ (การแทนที่) fileset ใหม่ถูกติดตั้ง กระบวนการติดตั้ง จะลบไฟล์ทั้งหมดจาก fileset เก่า (ที่ถูกแทนที่) ออกจากระบบ แม้ว่าไฟล์จะไม่รวมอยู่ใน fileset ใหม่

ในตัวอย่างก่อนหน้า `small.business` fileset เป็นการแทนที่โดยตรงสำหรับ `family.business` fileset ดังนั้นไฟล์ `small.business.rm_inv` ไม่ควรมีอยู่ `LawPractice.client` fileset ไม่ใช่การแทนที่โดยตรงสำหรับ `lawoffice.mgr` fileset ดังนั้น ไฟล์ `LawPractice.client.rm_inv` ต้องมีอยู่ แม้ว่าจะว่างเปล่า

ตัวอย่าง 3:

Filesets `bagel.shop.rte` และ `bread.shop.rte` ได้ถูก จัดส่งแยกกันมาหลายปีแล้ว ขณะนี้ `bagel.shop.rte` กำลังจะ จัดส่งเป็นส่วนหนึ่งของ `bread.shop.rte` สำหรับสิ่งนี้ที่จะเกิดขึ้น ไฟล์ `bread.shop.rte.namelist` จะมีลักษณะ:

```
bread.shop.rte
bagel.shop.rte
```

คุณยังต้องจัดส่งไฟล์ bread.shop.rte.rm_inv เปล่าเพื่อระบุว่า ไฟล์ทั้งหมดจาก bagel.shop.rte fileset ควรถูกลบออกจากระบบ

ไฟล์ Fileset.rm_inv

แอ็ตทริบิวต์	คำอธิบาย
Fileset.**rm_inv**	ไฟล์ที่มีรายการของไฟล์, ลิงก์ และไดเร็กทอรีที่จะลบออก จากระบบถ้าถูกพบว่าติดตั้งแล้ว

ไฟล์นี้ถูกใช้เมื่อ fileset ปัจจุบันถูกแพ็กเกจแตกต่างจาก ระดับของ fileset ก่อนหน้าและกระบวนการติดตั้งไม่ควรลบ ไฟล์ที่ติดตั้งก่อนหน้าออกโดยอิงจากรายการ fileset ในฐานข้อมูล inventory

การเปลี่ยนชื่อแบบง่ายสำหรับ fileset ไม่เพียงพอ ที่จะจำเป็นต้องใช้ไฟล์ *Fileset*.**rm_inv** ไฟล์ *Fileset*.**rm_inv** จำเป็น เมื่อ fileset ใหม่เป็นเซ็ตย่อยของ fileset ก่อนหน้า หรือเป็นผสมกันของ ส่วนต่างๆ ของ filesets ก่อนหน้า ถ้าไฟล์ *Fileset*.**namelist** มีอยู่และมีรายการสำหรับ fileset มากกว่าหนึ่งรายการ คุณต้องใช้ไฟล์ *Fileset*.**rm_inv** เพื่อลบระดับของไฟล์ที่ติดตั้งก่อนหน้านี้ออกจากระบบ

ไฟล์ *Fileset*.**rm_inv** ประกอบด้วยข้อมูล ASCII ในรูปแบบ stanza ชื่อของ stanza เป็นชื่อพาธเต็มของไฟล์หรือไดเร็กทอรีที่จะถูกลบออกถ้าถูกพบ บนระบบ ชื่อ stanza ลงท้ายด้วยโคลอน (:) และตามด้วยอักขระขึ้นบรรทัดใหม่ ถ้าแอ็ตทริบิวต์ตามหลังชื่อ stanza แอ็ตทริบิวต์มีรูปแบบเป็น *Attribute*=*Value* แอ็ตทริบิวต์ถูกใช้เพื่อระบุ ฮาร์ดลิงก์และลิงก์สัญลักษณ์ที่จำเป็นต้องถูกลบออก แต่ละแอ็ตทริบิวต์ ถูกแสดงบนบรรทัดแยก รายการต่อไปนี้แสดงแอ็ตทริบิวต์ที่ใช้ได้ที่สัมพันธ์ กับไฟล์ที่แสดงรายการ:

แอ็ตทริบิวต์	คำอธิบาย
links	อย่างน้อยหนึ่งฮาร์ดลิงก์ไปยังไฟล์ ชื่อพาธเต็มของลิงก์ ถูกคั่นด้วยเครื่องหมายจุลภาค
symlinks	อย่างน้อยหนึ่งลิงก์สัญลักษณ์ไปยังไฟล์ ชื่อพาธเต็มของลิงก์ ถูกคั่นด้วยเครื่องหมายจุลภาค

หมายเหตุ: ลิงก์ต้องถูกแสดงรายการสองครั้ง ครั้งแรกเป็น stanza สแตนด์อะโลน และอีกครั้งเป็นแอ็ตทริบิวต์ไปยังไฟล์ที่ถูกลิงก์ไป

ตัวอย่าง สมมุติ U.S.S.R 19.91.0.0 fileset มีไฟล์ต่อไปนี้ อยู่ในไดเร็กทอรี **/usr/lib**: moscow, leningrad, kiev, odessa และ petrograd (ลิงก์สัญลักษณ์ไปยัง leningrad) ผู้พัฒนาผลิตภัณฑ์ตัดสินใจแบ่ง U.S.S.R 19.91.0.0 fileset ออกเป็นสอง filesets: Ukraine.lib 19.94.0.0 และ Russia.lib 19.94.0.0 Ukraine.lib fileset มีไฟล์ kiev และ odessa Russia.lib fileset มี ไฟล์ moscow ไฟล์ leningrad ไม่มีอยู่ต่อไปและแทนที่ด้วยไฟล์ st.petersburg ใน Russia.lib fileset

ไฟล์ Ukraine.lib.rm_inv ต้องมีอยู่เนื่องจาก Ukraine.lib fileset ไม่ใช่การแทนที่โดยตรงสำหรับ U.S.S.R fileset ไฟล์ Ukraine.lib.rm_inv ควรว่างเปล่าเนื่องจาก ไม่มีไฟล์ใดที่จำเป็นต้องถูกลบออกเมื่อ Ukraine.lib fileset ถูกติดตั้งเพื่อคลีนอัพการโอนย้าย U.S.S.R fileset

ไฟล์ Russia.lib.rm_inv ต้องมีอยู่เนื่องจาก Russia.lib fileset ไม่ใช่การแทนที่โดยตรงสำหรับ U.S.S.R fileset ถ้าไฟล์ Russia.lib.rm_inv ถูกใช้เพื่อลบ ไฟล์ leningrad เมื่อ Russia.lib fileset ถูกติดตั้ง ไฟล์ Russia.lib.rm_inv จะมี stanza ต่อไปนี้:

```
/usr/lib/leningrad:
 symlinks = /usr/lib/petrograd
/usr/lib/petrograd:
```

ไฟล์การติดตั้งสำหรับระบบย่อยดิสก์เสริม

ระบบย่อยดิสก์ที่จะไม่ตั้งค่าด้วยไดรเวอร์อุปกรณ์ SCSI หรือที่มีบัสติดตั้ง จำเป็นต้องมีไดรเวอร์อุปกรณ์และวิธีการตั้งค่า ของตนเอง ไฟล์การติดตั้งเหล่านี้มีในดิสเก็ตเสริม (มีมาพร้อมกับอุปกรณ์) และต้องอยู่ในรูปแบบสำรองที่มีไฟล์ **./signature** และไฟล์ **./startup** ไฟล์ลายเซ็น ต้องมี **ปลายทาง** สตริง ไฟล์เริ่มทำงาน ต้องใช้การเรียกคืนโดยชื่อเพื่อแยกไฟล์ที่จำเป็นออก จากดิสเก็ตเสริม และเพื่อรันคำสั่งที่จำเป็นเพื่อให้อุปกรณ์มีสถานะพร้อม ใช้งาน

รูปแบบของสื่อบันทึกเพื่อแจกจ่าย

ประเภทของสื่อบันทึกต่อไปนี้สามารถใช้เพื่อแจกจ่าย แพ็กเกจการติดตั้งผลิตภัณฑ์ซอฟต์แวร์

ส่วนต่อไปนี้อธิบายรูปแบบที่ต้อง ใช้เพื่อแจกจ่ายแพ็กเกจผลิตภัณฑ์หลายแพ็กเกจบนแต่ละสื่อบันทึกเหล่านี้

เทป

เพื่อสแต็กอิมเมจแพ็กเกจผลิตภัณฑ์หลายอิมเมจบน เทปเดียวหรือชุดของเทป ไฟล์บนแต่ละเทปในชุด ต้องสอดคล้องกับรูป แบบต่อไปนี้:

- File 1 ว่าง (สำรองสำหรับเทปที่บูตได้)
- File 2 ว่าง (สำรองสำหรับเทปที่บูตได้)
- File 3 มีตารางไฟล์เนื้อหาที่อธิบาย รูปภาพแพ็กเกจผลิตภัณฑ์บนชุดของเทศ ดังนั้น แต่ละเทปในชุด จะมีสำเนาของตาราง ไฟล์เนื้อหาเดียวกัน ยกเว้นสำหรับส่วนที่แตกต่างกัน ของหมายเลขวอลุ่มเทปในชุดหลายวอลุ่ม
- Files 4 ถึง (*N*+3) มี อิมเมจไฟล์ที่เป็นรูปแบบการสำรองข้อมูลสำหรับแพ็กเกจผลิตภัณฑ์ 1 ถึง *N*
- ไฟล์อิมเมจแพ็กเกจผลิตภัณฑ์ไม่สามารถแตกออกเป็นสอง เทป
- แต่ละไฟล์ตามด้วยการทำเครื่องหมายเทศสิ้นสุดไฟล์

ซีดีรอม

ซีดีรอมที่จะมีรูปภาพแพ็กเกจผลิตภัณฑ์หลาย แพ็กเกจต้องสอดคล้องกับ Rock Ridge Group Protocol แพ็กเกจผลิตภัณฑ์ ควรเก็บอยู่ในไดเร็กทอรีการติดตั้ง ซึ่งต้องมีต่อไปนี้:

- อิมเมจไฟล์ที่มีรูปแบบสำรองข้อมูลของแพ็คเกจผลิตภัณฑ์
- ไฟล์สารบัญชื่อ **.toc** ที่อธิบายอิมเมจแพ็กเกจผลิตภัณฑ์ในไดเร็กทอรี

ซีดีรอมหลายวอลุ่มคือซีดีรอมที่มีโครงสร้างไดเร็กทอรีเพิ่ม เพื่อกำหนดชุดของซีดีรอมให้เป็นหน่วยที่สามารถติดตั้งได้หน่วย เดียว

ซีดีรอมหลายวอลุ่มต้องสอดคล้องกับกฎต่อไปนี้:

- ไดเร็กทอรี **/installp/mvCD** มีโดยมีเนื้อหา ต่อไปนี้:
  1. ไฟล์สารบัญ (.toc) ที่อธิบายอิมเมจแพ็กเกจผลิตภัณฑ์ บนซีดีรอมทั้งหมดของชุด แต่ละวอลุ่มของซีดีรอมต้องมี **.toc** เหมือนกันใน **/installp/mvCD**
  2. ไฟล์ ASCII ชื่อ volume_id ที่ซึ่งบรรทัดแรกประกอบด้วย หมายเลขวอลุ่มฐานสิบของซีดีใน set1
  3. ลิงก์สัญลักษณ์ชื่อ **vol%** ***ใน*** โดยที่ ***n*** คือ หมายเลขวอลุ่มฐานสิบของซีดีในชุด ปลายทางของลิงก์สัญลักษณ์ ต้องเป็นพาธ สัมพัทธ์กับไดเร็กทอรีของแพ็กเกจผลิตภัณฑ์บน วอลุ่มเฉพาะของซีดี ค่ามาตรฐานสำหรับลิงก์สัญลักษณ์คือ **../ppc**

- ไฟล์สารบัญ (**.toc**) ใน **/installp/mvCD** ต้องสอดคล้องกับรูปแบบสารบัญมาตรฐาน คุณสมบัติ พิเศษของ **.toc** หลายวอลุ่ม คือ ตำแหน่งที่ตั้งของแต่ละอิมเมจแพ็คเกจผลิตภัณฑ์ขึ้นต้นด้วยรายการ ไดเร็กทอรี **vol%** ***n*** โดยที่ ***n*** ระบุวอลุ่มที่ มีแพ็คเกจผลิตภัณฑ์เฉพาะนั้น

AIX 5.2 ตัวอย่าง:

fileset A อยู่ในไฟล์ **A.bff** บน volume 1 fileset B อยู่ในไฟล์ **B.bff** บน volume 2 ฟิลด์ในไฟล์ สารบัญใน **/installp/mvCD** มีตำแหน่งที่ตั้ง ของแพ็กเกจแพ็คเกจผลิตภัณฑ์สำหรับ A และ B คือ **vol%1/A.bff** และ **vol%2/B.bff** ตามลำดับ ฟิลด์ใน ไฟล์สารบัญใน **/installp/ppc** ของ volume 1 มีตำแหน่งที่ตั้งของ A เป็น **A.bff** ฟิลด์ใน ไฟล์สารบัญใน **/installp/ppc** ของ volume 2 มีตำแหน่งที่ตั้งของ B เป็น **B.bff**

โครงสร้างไดเร็กทอรีซีดีรอมสำหรับ AIX 5.1 และใหม่กว่าอนุญาตให้ใช้ข้อมูลจำเพาะของโปรแกรมติดตั้งอื่น รวมถึงหลาย แพล็ตฟอร์ม

ดิสเก็ต

เพื่อสแต็กหลายอิมเมจแพ็กเกจผลิตภัณฑ์บนชุดของดิสเก็ต ไฟล์ต่อไปนี้ต้องถูกเขียนลงชุดของดิสเก็ต:

- ไฟล์สารบัญที่อธิบายอิมเมจแพ็กเกจผลิตภัณฑ์จะถูกรวม ในชุด
- แต่ละไฟล์อิมเมจแพ็คเกจผลิตภัณฑ์ที่จะถูกรวมในชุด

ไฟล์ถูกเขียนลงในชุดของดิสเก็ตโดยใช้กฎต่อไปนี้:

- เขียนข้อมูลเป็นแบบสตรีมไปยังดิสเก็ตโดยมีเซ็กเตอร์ ID วอลุ่ม แทรกอยู่ใน block 0 ของแต่ละดิสเก็ตในชุด ข้อมูลจาก บล็อกสุดท้าย ของหนึ่งวอลุ่มจะถูกถือว่าต่อเนื่องกันเชิงตรรกะกับข้อมูลจาก block 1 ของ วอลุ่มถัดไป (เซ็กเตอร์ ID วอลุ่ม ถูกตรวจสอบและข้ามเมื่ออ่าน)
- แต่ละไฟล์เริ่มต้นภายในขอบเขตบล็อก 512 ไบต์
- เขียนไฟล์สารบัญเป็นอันดับแรก เสริมไฟล์นี้เพื่อเติมเซ็กเตอร์ สุดท้ายให้เต็มด้วยอักขระ null (x'00') จำเป็น ต้องใช้อย่าง น้อยหนึ่งอักขระ null เพื่อทำเครื่องหมายสิ้นสุดไฟล์สารบัญ ดังนั้น เซ็กเตอร์ที่มีอักขระ null ทั้งหมดอาจจำเป็น
- หลังจากไฟล์สารบัญแล้ว ให้เขียนแต่ละไฟล์อิมเมจ แพ็คเกจผลิตภัณฑ์ไปยังเซ็กเตอร์ต่อไป เสริมแต่ละไฟล์เพื่อเติมเซ็ก เตอร์สุดท้ายให้เต็ม โดยใช้อักขระ null อักขระ null อาจไม่จำเป็นถ้าไฟล์สิ้นสุดพอดีบน ขอบเขตบล็อก
- Block 0 ของแต่ละดิสเก็ตในชุดมีเซ็กเตอร์ ID วอลุ่ม รูปแบบ ของเซ็กเตอร์นี้คือ:

ตำแหน่ง	คำอธิบาย
Bytes 0:3	เลขกลสำหรับ identification นี้คือเลขจำนวนเต็มฐานสองที่มี ค่าของ 3609823513=x'D7298918' ฐานสิบ
Bytes 4:15	การประทับวันที่และเวลา (ใน ASCII) ที่ทำหน้าที่เป็น identification สำหรับชุดของดิสเก็ต รูปแบบคือ *MonthDayHourMinuteSecondYear Hour* ควรเป็นค่าตั้งแต่ 00 ถึง 23 ฟิลด์วันที่และเวลาทั้งหมดมีสองหลัก ดังนั้น *Month* ควร แสดงเป็น 03 แทน 3 และ *Year* ควรแสดงเป็น 94 แทน 1994
Bytes 16:19	หมายเลขวอลุ่มเลขจำนวนเต็มฐานสองภายในชุดของดิสเก็ตนี้ ดิสเก็ต แรกในชุดคือ x'00000001'
Bytes 20:511	ค่าศูนย์ฐานสอง

ไฟล์ สารบัญ

ชื่อฟิลด์	รูปแบบ	ตัวคั่น	คำอธิบาย
1. วอลุ่ม	อักขระ	ช่องว่าง	สำหรับไฟล์สารบัญของเทปและดิสเก็ต นี้คือ จำนวนวอลุ่มที่เก็บข้อมูลนี้ สำหรับไฟล์สารบัญของดิสก์หรือซีดีรอมคงที่ หมายเลขวอลุ่มคือ 0
2. การประทับวันที่และเวลา	*mmddhhMMssyy*	ช่องว่าง	สำหรับเทปหรือดิสเก็ต นี้คือการประทับเวลาเมื่อ volume 1 ถูกสร้าง สำหรับดิสก์หรือซีดีรอมคงที่ นี้คือการประทับเวลาเมื่อไฟล์ **.toc** ถูกสร้าง โปรดดูที่ รูปแบบการประทับวันที่และเวลา ภายหลังในบทความนี้ เพื่อดูรายละเอียด
3. รูปแบบส่วนหัว	อักขระ	บรรทัดใหม่	หมายเลขที่แสดงรูปแบบของไฟล์สารบัญ รายการ ที่ใช้ได้คือ: • 1 – AIX เวอร์ชัน 3.1 • 2 – เวอร์ชัน 3.2 • 3 – AIX เวอร์ชัน 4.1 หรือใหม่กว่า • B – **mksysb** เทป (ไม่สามารถใช้โดย **installp**)
4. ตำแหน่งของอิมเมจแพ็กเกจผลิตภัณฑ์	อักขระ	ช่องว่าง	สำหรับเทปหรือดิสเก็ต นี้คือสตริงอักขระในรูปแบบ: *vvv*:*bbbbb*:*sssssss*โปรดดูที่ รูปแบบตำแหน่งสำหรับเทปและดิสเก็ต ภายหลังในบทความนี้เพื่อดูรายละเอียด สำหรับดิสก์หรือซีดีรอมคงที่ นี้คือชื่อไฟล์ของไฟล์อิมเมจแพ็คเกจ ผลิตภัณฑ์ โปรดทราบว่านี้คือชื่อไฟล์เท่านั้น และต้องไม่ไม่มีส่วนใดๆ ของชื่อพาธ นำหน้า
5. ข้อมูลเฉพาะแพ็กเกจ	รูปแบบไฟล์ **lpp_name**	บรรทัดใหม่	เนื้อหาของไฟล์ **lpp_name** ที่มีใน อิมเมจแพ็คเกจผลิตภัณฑ์นี้ โปรดดูที่ ไฟล์ข้อมูลแพ็กเกจ lpp_name สำหรับ คำอธิบายละเอียด

หมายเหตุ: รายการ 4 และ 5 ที่อธิบายในตารางก่อนหน้าจะมีซ็ สำหรับแต่ละอิมเมจแพ็คเกจผลิตภัณฑ์ที่อยู่ในสื่อบันทึก

รูปแบบของการประทับวันที่และเวลา

รูปแบบการประทับวันที่และเวลาคือสตริง ASCII ที่ มีรูปแบบต่อไปนี้:

MonthDayHourMinuteSecondYear

Hour ควรเป็นค่าตั้งแต่ 00 ถึง 23 ฟิลด์วันที่และเวลาทั้งหมดมีสองหลัก ดังนั้น *Month* ควร แสดงเป็น 03 แทน 3 และ *Year* ควร แสดงเป็น 94 แทน 1994

รูปแบบตำแหน่งสำหรับเทปและดิสเก็ต

ตำแหน่งมีรูปแบบเป็น *vvv*:*bbbbb*:*sssssss* โดยที่ แต่ละตัวอักษรแทนหลักและมีความหมายดังนี้:

สำหรับเทป

vvv คือหมายเลขวอลุ่มของเทป

bbbbb คือหมายเลขไฟล์บนเทปของอิมเมจแพ็กเกจผลิตภัณฑ์

ssssssss
คือขนาดของไฟล์เป็นไบต์

สำหรับดิสเก็ต

vvv คือหมายเลขวอลุ่มของดิสเก็ต

bbbbb คือหมายเลขบล็อกบนดิสเก็ตที่ไฟล์อิมเมจแพ็คเกจผลิตภัณฑ์ เริ่มต้น

ssssssss

คือขนาดของไฟล์เป็นไบต์ (รวมการเสริมเต็มที่ท้ายของ ขอบเขตบล็อก)

การติดตั้ง AIX ที่เปลี่ยนตำแหน่งได้

ขณะที่การติดตั้ง AIX ที่เปลี่ยนตำแหน่งได้ขณะนี้ได้รับการสนับสนุนในยูทิลิตีการติดตั้ง AIX เป็นทางเลือก (เช่น **installp, instfix, lslpp** และ **lppchk**) การใช้วิธีเปลี่ยนตำแหน่งเป็นสิ่งที่ได้รับความสนใจสำหรับแอ็พพลิเคชันที่จำเป็น ต้องติดตั้งภายใน Workload Partition เนื่องจากการตั้งค่า System WPAR ดีฟอลต์ไม่ได้รวมระบบไฟล์ **/usr** หรือ **/opt** ที่สามารถเขียนได้ ดังนั้น การติดตั้งแอ็พพลิเคชันอาจ ต้องถูกเปลี่ยนปลายทางไปยังตำแหน่งอื่นนอกเหนือจากตำแหน่ง **/usr** หรือ **/opt** ดั้งเดิม

นอกเหนือจากสามารถติดตั้ง filesets ในตำแหน่งการติดตั้งดีฟอลต์แล้ว (นั่นคือ "/") ในขณะนี้ผู้ดูแลระบบยังสามารถติดตั้งแพ็คเกจที่เปลี่ยนตำแหน่งได้ลงในตำแหน่งการติดตั้ง root อื่น นี้ช่วยให้ ผู้ดูแลระบบสามารถทำสิ่งต่อไปนี้:

- ติดตั้งและดูแลรักษาหลายการติดตั้งของแพ็คเกจ installp เดียวกัน ในอินสแตนซ์เดียวของระบบปฏิบัติการ AIX
- ติดตั้งและดูแลรักษาหลายเวอร์ชันของแพ็คเกจ installp เดียวกัน ในอินสแตนซ์เดียวของระบบปฏิบัติการ AIX
- ใช้เครื่องมือการติดตาม installp ดั้งเดิม (เช่น **lppchk, lslpp, instfix** และ **inulag**) เพื่อตรวจสอบและรายงานข้อมูลการติดตั้งบนอินสแตนซ์การติดตั้งที่เปลี่ยนตำแหน่ง ทั้งหมด
- รวมเข้าและแยกตำแหน่งซอฟต์แวร์ที่ติดตั้งล่วงหน้าออกจากระบบที่กำหนด (การโฮสต์ แอ็พพลิเคชัน)

คำศัพท์

พาธติดตั้ง root

ไดเร็กทอรีฐานที่แอ็พพลิเคชันถูกติดตั้ง พาธติดตั้งดีฟอลต์ **installp** คือ "/"

พาธติดตั้งดีฟอลต์

พาธติดตั้ง root ดีฟอลต์ (นั่นคือ "/")

พาธติดตั้งที่เปลี่ยนตำแหน่ง

พาธติดตั้ง root ใดที่ไม่ใช่พาธติดตั้งดีฟอลต์ ตำแหน่ง พาธอาจเป็นพาธที่ใช้ได้พาธใดๆ ที่ไม่ใช่ "/" และที่ มีขนาดไม่ยาวเกิน 512 อักขระ

แอ็พพลิเคชันที่เปลี่ยนตำแหน่งได้

แอ็พพลิเคชันที่สามารถติดตั้งในพาธติดตั้ง root ที่ไม่ใช่ค่าดีฟอลต์

USIL (User Specified Install Location)

พาธอินสแตนซ์ติดตั้งที่เปลี่ยนตำแหน่งซึ่งตั้งค่าโดยผู้ใช้

USIL

User Specified Install Location หรือ USIL คือพาธติดตั้งที่เปลี่ยนตำแหน่ง โดยถูกติดตาม ซึ่งสร้างขึ้นโดยผู้ดูแลระบบ ตำแหน่งนี้ถูกติดตามโดย ระบบและสามารถใช้เป็นพาธติดตั้งทางเลือกที่สนับสนุน การเปลี่ยนตำแหน่ง หลายอินสแตนซ์และ/หรือ เวอร์ชันของซอฟต์แวร์แพ็กเกจเดียวกัน สามารถติดตั้งบนระบบเดียวโดยการมอบหมายแต่ละการติดตั้งไปยัง USIL แยกกัน อินสแตนซ์ USIL ที่มีอยู่แล้วอาจถูกรวมเข้า หรือถอดออกจากระบบใดๆ ที่กำหนด

แต่ละอินสแตนซ์ USIL ดูแลรักษาชุดของ Software Vital Product Data (SWVPD) ของตนเองในส่วน installp ปัจจุบันสามส่วนทั้งหมด:

- *<InstallRoot>***/etc/objrepos**

- *<InstallRoot>*/usr/lib/objrepos
- *<InstallRoot>*/usr/share/lib/objrepos

หมายเหตุ:

1. อ็อบเจ็กต์คลาส SWVPD ปัจจุบันมีต่อไปนี้:
   - product
   - lpp
   - inventory
   - history
   - fix
   - vendor
   - lag
2. แต่ละอินสแตนซ์ USIL จะมิร์เรอร์โครงสร้าง SWVPD ดีฟอลต์ภายใน พาธที่ถูกเปลี่ยนตำแหน่ง

คำสั่งการจัดการ USIL

/usr/sbin/mkusil

สร้างหรือรวมอินสแตนซ์ USIL ใหม่เข้า

mkusil -R *RelocatePath* **-c** *Comments* [**X***Fa*]

-a รวมการติดตั้งที่มีอยู่เป็นอินสแตนซ์ USIL หนึ่ง

-c ความคิดเห็นที่จะรวมในนิยาม USIL (เห็นได้โดยใช้คำสั่ง **lsusil**)

-R พาธไปยังตำแหน่ง USIL ใหม่ ต้องเป็นไดเร็กทอรีที่ถูกต้อง

-X ขยายขนาดพื้นที่ที่ต้องการโดยอัตโนมัติ

/usr/sbin/lsusil

แสดงรายการอินสแตนซ์ USIL ที่มีอยู่

lsusil [**-R** *RelocatePath* | *"***ALL***"*]

-R พาธไปยังตำแหน่ง USIL ที่มีอยู่

/usr/sbin/rmusil

ลบอินสแตนซ์ USIL ที่มีอยู่ออก

rmusil -R *RelocatePath*

-R พาธไปยังตำแหน่ง USIL ที่มีอยู่

หมายเหตุ: คำสั่ง **rmusil** จะลบ การอ้างอิง USIL ใน SWVPD ออกเท่านั้น ไม่มีไฟล์ใดถูกลบออกในพาธติดตั้ง USIL

/usr/sbin/chusil

เปลี่ยนแอ็ตทริบิวต์ของอินสแตนซ์ USIL ที่มีอยู่

chusil -R *RelocatePath* **-c** *NewComments* [**X**]

-c ความคิดเห็นใหม่ที่จะรวมในนิยาม USIL (เห็นได้โดยใช้คำสั่ง **lsusil**)

-R พาธไปยังตำแหน่ง USIL ที่มีอยู่

-X ขยายขนาดโดยอัตโนมัติถ้าจำเป็นต้องใช้พื้นที่

ยูทิลีตีการติดตั้งที่สามารถเปลี่ยนตำแหน่งได้

ในการสงวนการแยกโค้ดไว้ การเปลี่ยนแปลง USIL ทั้งหมดจะถูกแยกไว้ในโมดูลที่คอมไพล์ แยก ยูทิลิตีการติดตั้งที่เปลี่ยนตำแหน่งจะรวม โมดูลระดับผู้ใช้ต่อไปนี้:

- **/usr/sbin/mkusil**
- **/usr/sbin/rmusil**
- **/usr/sbin/lsusil**
- **/usr/sbin/chusil**

หมายเหตุ:

1. แต่ละยูทิลิตีใช้แฟล็ก **-R** *RelocatePath*
2. เมื่อทำงานกับแพ็กเกจ **installp** ที่เปลี่ยนตำแหน่งได้ ต้องใช้ยูทิลิตีด้านบน

ข้อกำหนดสำหรับการสร้างแพ็กเกจแอ็พพลิเคชันที่สามารถเปลี่ยนตำแหน่งได้

การทำแพ็กเกจแอ็พพลิเคชันต้องสนับสนุนการติดตั้งที่เปลี่ยนตำแหน่งได้ ต่อไปนี้คือแนวทางที่แนะนำ:

- แอ็พพลิเคชันแพ็กเกจที่เปลี่ยนตำแหน่งได้ต้องไม่ส่ง (เขียน) อ็อบเจ็กต์ inventory ภายนอกตำแหน่งติดตั้ง root ของตน
- แอ็พพลิเคชันแพ็กเกจที่เปลี่ยนตำแหน่งได้ต้องไม่ส่ง (เขียน) ข้อมูลโดยใช้การกำหนดการทำ แพ็คเกจเองภายนอกตำแหน่งติดตั้ง root ของตน
- แอ็พพลิเคชันแพ็กเกจที่เปลี่ยนตำแหน่งได้ต้องมี แอ็ตทริบิวต์การทำแพ็คเกจที่ขยาย **RELOCATABLE** สำหรับแต่ละ fileset ที่เปลี่ยนตำแหน่งได้ fileset คือหน่วยที่ติดตั้งได้ขนาดเล็กสุดที่สามารถเปลี่ยนตำแหน่งได้
- แอ็พพลิเคชันแพ็กเกจที่เปลี่ยนตำแหน่งได้อาจไม่มีสิ่งจำเป็นต้องมีใน พาธที่ถูกเปลี่ยนเป็นตำแหน่งภายนอก โดยอาจมีสิ่งที่ต้องมีสำหรับ filesets ที่ติดตั้ง ในพาธติดตั้งดีฟอลต์หรือในพาธติดตั้งของตนเอง

ข้อกำหนดสำหรับการเรียกทำงานแอ็พพลิเคชันที่สามารถเปลี่ยนตำแหน่งได้

การออกแบบแอ็พพลิเคชันต้องสนับสนุนการเรียกทำงานจากสภาวะแวดล้อมการติดตั้ง ต่อไปนี้คือแนวทางที่แนะนำ:

- แอ็พพลิเคชันต้องมีวิธีหรือฟังก์ชันเพื่อพิจารณาตำแหน่งติดตั้ง root ที่ซึ่งไม่ต้องขึ้นกับตำแหน่งติดตั้ง
- แอ็พพลิเคชันต้องอ้างอิงคอมโพเนนต์เรียกทำงานเฉพาะแอ็พพลิเคชันทั้งหมด โดยสัมพันธ์กับตำแหน่งติดตั้ง root
- แอ็พพลิเคชันต้องอ้างอิงคอมโพเนนต์ข้อมูลเฉพาะแอ็พพลิเคชันทั้งหมด โดยสัมพันธ์กับตำแหน่งติดตั้ง root หรือต้องถูกออกแบบให้แบ่งใช้ ข้อมูลกับแอ็พพลิเคชันอินสแตนซ์อื่น
- แอ็พพลิเคชันไม่ควรทำการเปลี่ยนแปลงบ่อยๆ กับภายนอก ตำแหน่งติดตั้ง root

ตัวเชื่อมต่อ USIL คลาสอ็อบเจ็กต์ ODM

USIL connector ODM Class Object อยู่ในไฟล์ **/etc/objrepos/usilc** และมีข้อมูลที่ลิงก์ SWVPD ดีฟอลต์กับอินสแตนซ์ USIL ทั้งหมด

ต่อไปนี้คือรายการสำหรับคลาสอ็อบเจ็กต์นี้ที่จะมีอยู่ใน swvpd.cre:

```
/*  User Install Location Connector                                    */
/*  Connects the default install path to all relocated install paths.  */
class   usilc {
        vchar   path[1024];             /* USIL path */
        vchar   comments[2048];   /* USIL Comments */
        long    flags;                      /* USIL flags */
        };
```

การแสดงรายการพาธการติดตั้งทั้งหมดด้วยอ็อพชัน -R "ALL" หรือ -R "all"

คำสั่ง **lslpp** และ **lppchk** สามารถทำการดำเนินการแสดงรายการกับตำแหน่งติดตั้งทั้งหมด ถ้าไวยากรณ์ **-R "ALL"** ถูกใช้

การรวมเข้า/แยกการดำเนินการออก

การดำเนินการรวมเข้าอนุญาตให้ผู้ใช้ผนวกพาธ USIL ที่แยกออกที่มีอยู่เข้าใน SWVPD

ตัวอย่าง ถ้าผู้ดูแลระบบสร้างอินสแตนซ์ USIL มาสเตอร์ด้วยแอ็พพลิเคชัน ที่เปลี่ยนตำแหน่งได้ต่างๆ ที่ติดตั้งสำหรับวัตถุประสงค์การโฮสต์แอ็พพลิเคชัน ผู้ดูแลระบบจะทำสำเนาหรือ NFS เมาท์อินสแตนซ์ USIL นี้กับระบบ ต่างๆ และใช้คุณลักษณะการรวมเข้า เพื่อผนวกอินสแตนซ์ USIL ใน SWVPD การดำเนินการแยกออกจะลบการอ้างอิงไปยังอินสแตนซ์ USIL

การออกไลเซนส์ installp

อินสแตนซ์ USIL ใหม่เริ่มทำงานด้วย LAG (คลาสอ็อบเจ็กต์ ODM ข้อตกลงการอนุญาตใช้สิทธิ์ **installp**) ว่าง การติดตั้ง filesets หรือ LPPs ใดๆ ที่ต้องใช้ไลเซนส์จะต้องมีการยอมรับไลเซนส์ตาม ระเบียบ **installp** ปกติ การยอมรับไลเซนส์ไม่แตกออกตามอินสแตนซ์ USIL

Trusted Computing Base (TCB)

การติดตั้งอินสแตนซ์ USIL ขณะนี้ไม่ได้รับการสนับสนุนบนระบบที่เปิดใช้ TCB

สิ่งที่จำเป็นต้องมีในการเปลี่ยนตำแหน่ง

semantic การทำแพ็กเกจใหม่บ่งชี้ตำแหน่งที่จำเป็นซึ่งเปลี่ยนตำแหน่งได้ ตัวทำแพ็คเกจสามารถระบุว่าสิ่งจำเป็นที่กำหนดควรพบในพาธติดตั้ง ดีฟอลต์หรือในพาธติดตั้งที่เปลี่ยนตำแหน่ง

semantic ที่จำเป็นใหม่ต่อไปนี้ใช้:

- prereq_r = prereq ในพาธติดตั้งที่เปลี่ยนตำแหน่ง
- ifreq_r = ifreq ในพาธติดตั้งที่เปลี่ยนตำแหน่ง
- coreq_r = coreq ในพาธติดตั้งที่เปลี่ยนตำแหน่ง
- instreq_r = instreq ในพาธติดตั้งที่เปลี่ยนตำแหน่ง

ประเภทสิ่งที่ต้องมีที่กำหนดขณะนี้ (**prereq, ifreq, coreq** และ **instreq**) คือสิ่งที่ต้องมีดีฟอลต์ทั้งหมด (สิ่งที่ต้องมีที่ใช้กับตำแหน่ง ติดตั้งดีฟอลต์)

การเปลี่ยน TOC สำหรับแพ็กเกจที่สามารถเปลี่ยนตำแหน่งได้

ต่อไปนี้คือตัวอย่างของส่วนสิ่งต้องมีสิ่งใหม่ในไฟล์ TOC:

```
sscp.rte.1.0.0.5.U.PRIVATE.bff 4 R S sscp {
sscp.rte 01.00.0000.0005 1 N B En_US Sscp
[
*coreq bos.games 1.1.1.1  <-- default requisite in default requisite section
*prereq bos.rte 1.1.1.1   <-- default requisite in default requisite section
%
/usr/bin 20
/etc 20
INSTWORK 72 40
%
%
%
IY99999  1 APAR text here.
%
RELOCATABLE <-- attribute tag to denote relocatable package
%
*prereq bos.rte 1.1.1.1 <-- default requisite in relocated requisite section
*coreq_r bos.games 1.1.1.1 <-- relocated requisite in relocated requisite section
]
}
```

หมายเหตุ:

1. ถ้าส่วนสิ่งที่ต้องมีที่เปลี่ยนตำแหน่งได้แสดงอยู่ระหว่างการติดตั้งที่เปลี่ยนตำแหน่ง จะถูกใช้เป็นส่วนสิ่งที่ต้องมีสำหรับการติดตั้ง
2. ถ้าส่วนสิ่งที่ต้องมีที่เปลี่ยนตำแหน่งได้ไม่มีอยู่ระหว่างการติดตั้ง ที่เปลี่ยนตำแหน่ง ส่วนสิ่งที่ต้องมีดีฟอลต์จะถูกใช้ นี้หมายความว่าสิ่งที่ต้องมีทั้งหมด ที่เป็นค่าดีฟอลต์
3. การติดตั้งดีฟอลต์ (ไม่ถูกเปลี่ยนตำแหน่ง) ไม่ใช้ส่วนสิ่งที่ต้องมีที่ เปลี่ยนตำแหน่งได้

การประมวลผล installp ของแพ็กเกจผลิตภัณฑ์

คำสั่ง	คำอธิบาย
Apply	เมื่อ fileset ในแพ็กเกจการติดตั้งผลิตภัณฑ์ถูกนำใช้ จะถูก ติดตั้งบนระบบ และเขียนทับเวอร์ชันที่มีอยู่แล้วของ fileset นั้น ดังนั้นเป็น *การยอมรับ* เวอร์ชันนั้นของ fileset บนระบบ fileset อาจถูกลบออกถ้าผู้ใช้เห็นว่า fileset ไม่จำเป็นต้องใช้อีกต่อไป เมื่อนำใช้การอัพเดต fileset การอัพเดต จะถูกติดตั้งและข้อมูลถูกบันทึก (ยกเว้นร้องขอเป็นอย่างอื่น) เพื่อที่ สามารถลบการอัพเดตออกในภายหลัง การอัพเดต Fileset ที่ถูกนำใช้สามารถ ยอมรับหรือปฏิเสธได้ภายหลัง
Commit	เมื่อการอัพเดต fileset ถูกยอมรับ ข้อมูลที่บันทึกระหว่าง การนำใช้ถูกลบออกจากระบบ การยอมรับซอฟต์แวร์ที่ถูกนำใช้แล้ว จะไม่ เปลี่ยนแปลงเวอร์ชันที่แอ็คทีฟอยู่ขณะนี้ของ fileset
Reject	เมื่อการอัพเดตถูกปฏิเสธ ข้อมูลที่บันทึกระหว่างการนำใช้ถูกใช้ เพื่อเปลี่ยนแปลงเวอร์ชันแอ็คทีฟของ fileset ไปเป็นเวอร์ชันก่อนหน้าที่ การอัพเดตจะถูกปฏิเสธ จากนั้นข้อมูลที่บันทึกจะถูกลบออกจากระบบ การดำเนินการ ปฏิเสธใช้ได้สำหรับการอัพเดตเท่านั้น ขั้นตอนเหมือนกันหลายขั้นตอนใน การดำเนินการปฏิเสธจะถูกดำเนินการในการดำเนินการ **คลีนอัพ** เมื่อ fileset หรือการอัพเดต fileset ไม่สามารถทำการติดตั้งให้เสร็จสมบูรณ์
Remove	เมื่อ fileset ถูกลบออก fileset และการอัพเดตจะถูกลบออก จากระบบโดยเป็นอิสระจากสถานะที่เป็นอยู่ (applied, committed หรือ broken) การดำเนินการลบออกใช้ได้กับระดับการติดตั้งของ fileset เท่านั้น

ไฟล์เรียกทำงานที่มีภายในแพ็กเกจผลิตภัณฑ์สามารถปรับเปลี่ยน การประมวลผลสำหรับการดำเนินการ apply, reject และ remove ให้เหมาะสม

การติดตั้ง fileset อีกครั้งไม่ได้ดำเนินการเหมือนกับ ที่การลบและการติดตั้ง fileset เดียวกันทำ การดำเนินการติดตั้งอีกครั้ง (โปรดดูที่ **/usr/lib/instl/rminstal**) จะคลีนอัพไฟล์ปัจจุบันจาก เวอร์ชันก่อนหน้าหรือเหมือนกัน แต่ไม่รันสคริปต์ใดๆ ของสคริปต์ **unconfig** หรือ **unpre*** ดังนั้น อย่าเข้าใจว่า สคริปต์ **unconfig** ถูกรัน สคริปต์ **.config** ควรตรวจสอบสภาวะแวดล้อมก่อนที่จะถือว่า unconfig ถูกทำเสร็จสมบูรณ์

ตัวอย่าง สำหรับ **ras.berry.rte** fileset สคริปต์ config เพิ่มบรรทัดไปยังไฟล์ **crontab** ของ root การติดตั้ง **ras.berry.rte** fileset อีกครั้งส่งผลต่อสองรายการ **crontab** เนื่องจากสคริปต์ **unconfig** ไม่ถูกรันในการติดตั้งอีกครั้ง (ซึ่งลบรายการ **crontab**) สคริปต์ config ควรลบรายการและ จากนั้นเพิ่มใหม่อีกครั้งเสมอ

การประมวลผลสำหรับการดำเนินการ Apply

ส่วนนี้อธิบายขั้นตอนที่ทำโดยคำสั่ง **installp** เมื่อ fileset หรือการอัพเดต fileset ถูกนำใช้

1. เรียกคืนไฟล์ข้อมูลแพ็กเกจผลิตภัณฑ์ **lpp_name** สำหรับแพ็กเกจจากอุปกรณ์ที่ระบุ
2. ตรวจสอบว่า filesets ที่ร้องขอมีอยู่บนสื่อบันทึก การติดตั้ง
3. ตรวจสอบระดับของ filesets ที่ร้องขอเพื่อตรวจสอบว่า จะถูกติดตั้งบนระบบแล้ว
4. เรียกคืนไฟล์ควบคุมจากไฟล์ไลบรารีไฟล์เก็บถาวร **liblpp.a** ไปไว้ใน *ไดเร็กทอรีแพ็คเกจ* (แพ็กเกจ **/usr/lpp/***Package_Name* for **usr** หรือ **usr/root** ไฟล์ควบคุมโดยเฉพาะ สำหรับส่วน **root** ของแพ็กเกจ **usr/root** อยู่ใน **/usr/lpp/***Package_Name***/inst_root/liblpp.a**)
5. ตรวจสอบความต้องการใช้พื้นที่ดิสก์
6. ตรวจสอบว่า *สิ่งที่ต้องมี* ที่จำเป็น (filesets ที่จำเป็นในระดับหนึ่งเพื่อใช้หรือติดตั้ง fileset อื่น) ถูกติดตั้งอยู่แล้ว หรืออยู่บนรายการที่จะถูกติดตั้ง
7. พิจารณาว่ามีข้อกำหนดข้อตกลงการอนุญาตใช้สิทธิ์ซึ่งต้องมี เพื่อดำเนินการติดตั้งต่อไปหรือไม่
8. ถ้ามีแพ็กเกจการติดตั้งแทนที่แพ็กเกจการอัพเดต fileset ให้ค้นหา software vital product data (SWVPD) เพื่อดูว่า *Fileset* (fileset ที่จะถูกติดตั้ง) หรือ filesets ใดๆ แสดงรายการในไฟล์ *Fileset***.namelist** ถูกติดตั้งอยู่แล้วบนระบบไม่ว่าที่ระดับใด ถ้า *Fileset* ถูกติดตั้งแล้ว ให้เขียนชื่อ fileset และระดับที่ติดตั้งไปยัง ไฟล์ *Work_Directory*/*Fileset***.installed_list** ถ้า ไม่ได้ติดตั้งระดับของ *Fileset* ดังนั้น ถ้าติดตั้งชุดไฟล์ใดๆ ถูกแสดงรายการใน ไฟล์ *Fileset***.namelist** จะแสดงรายการชุดไฟล์และระดับทั้งหมดในไฟล์ *Work_Directory*/*Fileset***.installed_list** *Work_Directory* จะ เหมือนกับไดเร็กทอรีแพ็กเกจที่มีข้อยกเว้นของส่วน **root** ที่ใช้ **/lpp/***Package_Name*
9. ถ้านี้เป็นแพ็กเกจการติดตั้งแทนที่จะเป็นแพ็คเกจการอัพเดต fileset ให้รันสคริปต์ **/usr/lib/instl/rminstal** เพื่อกระทำสิ่งต่อไปนี้สำหรับแต่ละ fileset ที่จะถูกติดตั้ง

   หมายเหตุ: ยกเว้น ระบุเป็นอย่างอื่น ไฟล์ตรวจหาการมีอยู่ต้องถูกเรียกคืนจาก ไลบรารีไฟล์ควบคุม **liblpp.a**

   a. ถ้า *Fileset***.pre_rm** มีอยู่ ให้รัน *Fileset***.pre_rm** เพื่อดำเนินขั้นตอนที่จำเป็นก่อนลบไฟล์ใดๆ ออกจาก เวอร์ชันนี้ หรือเวอร์ชันที่มีอยู่ของ *Fileset*

   b. ถ้า ***Work_Directory***/*Fileset***.installed_list** มีอยู่ ให้ย้าย ไฟล์ที่มีอยู่ที่แสดงรายการใน *Fileset*.cfgfiles ไปที่ ไดเร็กทอรีบันทึกไฟล์การตั้งค่า (ระบุโดยตัวแปรสภาวะแวดล้อม **MIGSAVE**)

   c. ถ้าเวอร์ชันของ *Fileset* ถูก ติดตั้งแล้ว ให้ลบไฟล์และข้อมูล SWVPD ออก (ยกเว้น ประวัติ) สำหรับ *Fileset*

d. ถ้า **Work_Directory/***Fileset***.installed_list** มีอยู่ และ *Fileset***.rm_inv** มีอยู่ หรือ *Fileset***.namelist** มีมากกว่าหนึ่ง fileset หรือเฉพาะ fileset ที่แสดงรายการใน *Fileset***.namelist** เป็น `bos.obj` ให้ดำเนินการต่อไปนี้:

1) ลบไฟล์และข้อมูล SWVPD inventory สำหรับไฟล์ที่แสดงรายการในไฟล์ *Fileset***.rm_inv**

2) ลบไฟล์และข้อมูล SWVPD inventory สำหรับไฟล์ที่แสดงรายการในไฟล์ *Fileset***.inventory**

3) ลบข้อมูล SWVPD อื่นๆ สำหรับ filesets ใดๆ ที่แสดรายการใน **Fileset.namelist** ซึ่งไม่มีข้อมูล SWVPD inventory ใดๆ อีกต่อไป

e. ถ้า **Work_Directory/***Fileset***.installed_list** มีอยู่และมีหนึ่ง fileset เท่านั้น และ *Fileset***.namelist** มีหนึ่ง fileset เท่านั้น ให้ลบไฟล์และข้อมูล SWVPD ออก (ยกเว้น ประวัติ) สำหรับ fileset นั้น

f. สำหรับแต่ละส่วนของแพ็กเกจผลิตภัณฑ์ (ส่วน **usr** เท่านั้นหรือ **usr** ตามด้วย **root**)

1) ตั้งค่า INUTREE (*U* สำหรับ **usr** และ *M* สำหรับ **root**) และ INUTEMPDIR (ชื่อของตัวแปรสภาวะแวดล้อม ไดเร็กทอรีการทำงานชั่วคราวที่สร้างขึ้น

2) ถ้าโปรแกรมควบคุม **instal** มีอยู่ในไดเร็กทอรีแพ็คเกจ (ไม่แนะนำ) ให้รัน **./instal** มิฉะนั้นรันดีฟอลต์สคริปต์ **/usr/lib/instl/instal** ถ้าโปรแกรมควบคุม **instal** ไม่มีอยู่ใน ไดเร็กทอรีแพ็กเกจ ให้ตั้งค่าตัวแปรสภาวะแวดล้อม **INUSAVEDIR**

3) ถ้าโปรแกรมควบคุม **update** มีอยู่ในไดเร็กทอรี แพ็คเกจ (ไม่แนะนำ) ให้รัน **./update** ถ้าโปรแกรมควบคุม **update** ไม่มีอยู่ในไดเร็กทอรีแพ็คเกจ ให้รันดีฟอลต์สคริปต์ **/usr/lib/instl/update**

4) ถ้าไฟล์ **status** ถูกสร้างเสร็จเรียบร้อย โดย **instal** หรือ **update** ให้ใช้ไฟล์ **status** เพื่อพิจารณาความสำเร็จหรือความล้มเหลวของแต่ละ fileset ถ้าไฟล์ **status** ไม่ถูกสร้าง ให้ถือว่า filesets ที่ร้องขอทั้งหมดในแพ็กเกจไม่สามารถนำใช้

5) ถ้าการดำเนินการนำใช้สำหรับ fileset เสร็จสมบูรณ์ให้อัพเดต Software Vital Product Data (SWVPD) จากนั้นลงทะเบียนข้อกำหนด ข้อตกลงการอนุญาตใช้สิทธิ์ที่เกี่ยวข้องใดๆ ถ้าการดำเนินการนำใช้สำหรับ fileset ไม่สำเร็จ ให้รัน **/usr/lib/instl/cleanup** หรือ **lpp.cleanup** ที่มากับผลิตภัณฑ์จากไดเร็กทอรีแพ็คเกจเพื่อคลีนอัพ filesets ที่ล้มเหลว

การประมวลผลสคริปต์ install หรือ update ดีฟอลต์

ไฟล์เรียกทำงาน **instal** หรือ **update** ถูกเรียกใช้จาก **installp** ที่มี พารามิเตอร์แรกเป็นอุปกรณ์ที่กำลังถูกใช้สำหรับการติดตั้งหรือการอัพเดต พารามิเตอร์ที่สองคือชื่อพาธเต็มไปยังไฟล์ที่มีรายการของ filesets ที่จะถูกติดตั้งหรืออัพเดต ถูกอ้างถึงด้านล่างเป็น **$FILESETLIST** สคริปต์ **instal** และ **update** แบบดีฟอลต์จะถูกเชื่อมเข้าด้วยกัน การประมวลผลจะแตกต่างกันขึ้นอยู่กับว่าถูกใช้เป็น **instal** หรือ **update** ไดเร็กทอรีปัจจุบันเป็นไดเร็กทอรีแพ็กเกจ ไดเร็กทอรีชั่วคราว INUTEMPDIR ถูกสร้างใน **/tmp** เพื่อเก็บไฟล์การทำงาน

โฟลว์ภายในสคริปต์ **instal** และ **update** ดีฟอลต์เป็นดังนี้:

1. ทำสิ่งต่อไปนี้สำหรับแต่ละ fileset ที่แสดงรายการใน **$FILESETLIST**:

a. ถ้า fileset เป็นการอัพเดต ให้เรียกทำงาน ***Fileset*.pre_u** (pre_update) ถ้ามีอยู่ ถ้า fileset ไม่ใช่การอัพเดต ให้เรียกทำงาน ***Fileset*.pre_i** (pre_installation) ถ้ามีอยู่

b. สร้างรายการหลักของไฟล์ที่จะถูกเรียกคืนจาก แพ็คเกจโดยการผนวก ***Fileset*.al** กับไฟล์ใหม่ **INUTEMPDIR/master.al**

c. ถ้านี้เป็นการอัพเดต ไฟล์จะถูกระบุให้ บันทึก และไฟล์ควบคุมไฟล์เก็บถาวร **lpp.acf** จะมีอยู่ บันทึกสมาชิกไฟล์เก็บถาวรของไลบรารีที่กำลังถูกอัพเดต

d. ถ้าการประมวลผลสำเร็จ ให้ผนวก fileset นี้ กับรายการที่จะถูกติดตั้งในไฟล์ **$FILESETLIST.new**

2. ถ้านี้เป็นการอัพเดตและการบันทึกไฟล์ถูกระบุ ให้รัน **inusave** เพื่อบันทึกเวอร์ชันปัจจุบันของไฟล์
3. ถ้าคุณกำลังประมวลผลส่วน root ให้รัน **inucp** เพื่อทำสำเนาไฟล์จากรายการที่นำใช้ไปยังส่วน root ถ้าคุณไม่ได้ กำลังประมวลผลส่วน root ให้รัน **inurest** เพื่อเรียกคืนไฟล์จาก รายการที่นำใช้สำหรับส่วน **usr**
4. ทำสิ่งต่อไปนี้สำหรับแต่ละ fileset ที่แสดงรายการในไฟล์ **$FILESETLIST.new**:

   หมายเหตุ: ความล้มเหลวในขั้นตอนใดๆ จะถูกบันทึก ในไฟล์ **status** และการประมวลผลสำหรับ fileset นั้นสิ้นสุดลง

   a. พิจารณาว่า fileset นี้ถูกติดตั้งที่ระดับเดียวกัน หรือเก่ากว่า หรือ filesets ถูกแสดงรายการใน *Fileset*.**namelist** ถูกติดตั้ง ถ้าเป็นเช่นนั้น ให้เอ็กซ์พอร์ตตัวแปรสภาวะแวดล้อม **INSTALLED_LIST** หรือ **MIGSAVE** นี้เรียกว่า *การโอนย้ายระบบ*

   b. ถ้าคุณกำลังประมวลผลการอัพเดต ให้เรียกใช้ *Fileset*.**post_u** ถ้ามีอยู่ ถ้า *Fileset*.**post_u** ไม่มีอยู่ ให้เรียกใช้ *Fileset*.**post_i** ถ้ามีอยู่

   c. ถ้า *Fileset*.**cfgfiles** มีอยู่ ให้รัน **/usr/lib/instl/migrate_cfg** เพื่อจัดการการประมวลผล ของไฟล์การตั้งค่าตามวิธีการจัดการที่ระบุ

   d. เรียกใช้ **sysck** เพื่อเพิ่มข้อมูล ในไฟล์ *Fileset*.**inventory** ไปยัง Software Vital Product Database (SWVPD)

   e. เรียกใช้คำสั่ง **tcbck** เพื่อ เพิ่มข้อมูล trusted computing base ไปยังระบบถ้าไฟล์ ***Fileset*.tcb** มีอยู่ และแอ็ตทริบิวต์ trusted computing base **tcb_enabled** ถูกตั้งค่าในฐานข้อมูล **/usr/lib/objrepos/PdAt** ODM

   f. เรียกใช้ **errupdate** เพื่อเพิ่มเท็มเพลต ข้อผิดพลาดถ้า *Fileset*.**err** มีอยู่

   g. เรียกใช้ **trcupdate** เพื่อเพิ่มเท็มเพลต รูปแบบรายงานการติดตามถ้า *Fileset*.**trc** มีอยู่

   h. ถ้าอัพเดตหรือถ้า ***Work_Directory*/*Fileset*.installed_list** มีอยู่ ให้เรียกใช้สคริปต์ *Fileset*.**odmdel** และ *Fileset*.*.**odmdel** แต่ละสคริปต์เพื่อประมวลผล คำสั่งการลบฐานข้อมูล ODM

   i. เรียกใช้ **odmadd** ของ *Fileset*.**odmadd** และ *Fileset*.*.**odmadd** ที่มีอยู่แต่ละไฟล์ เพื่อเพิ่มข้อมูลในฐานข้อมูล ODM

   j. ถ้านี้เป็นการอัพเดต ให้เรียกใช้ *Fileset*.**config_u** (การอัพเดตการตั้งค่า fileset) ถ้า มีอยู่ มิฉะนั้น ให้เรียกใช้ *Fileset*.**config** (การตั้งค่า fileset) ถ้ามีอยู่

   k. อัพเดตไฟล์ **สถานะ** ที่ระบุ การประมวลผลสำเร็จสำหรับ fileset

5. ลิงก์ไฟล์ควบคุมที่จำเป็นสำหรับการลบ fileset ในไดเร็กทอรี **deinstl** ของแพ็กเกจเพื่อการใช้งานในอนาคต ไฟล์เหล่านี้มีไฟล์ต่อไปนี้ที่อาจมีอยู่ในไดเร็กทอรี แพ็กเกจ:

   - **lpp.deinstal**
   - *Fileset*.**al**
   - *Fileset*.**inventory**
   - *Fileset*.**pre_d**
   - *Fileset*.**unpre_i**
   - *Fileset*.**unpre_u**
   - *Fileset*.**unpost_i**
   - *Fileset*.**unpost_u**
   - *Fileset*.**unodmadd**
   - *Fileset*.**unconfig**

- *Fileset*. **unconfig_u**
- $SAVEDIR/*Fileset*. ***.rodmadd**
- SAVEDIR/*Fileset*. ***.unodmadd**

การประมวลผลสำหรับการดำเนินการ reject และ cleanup

ส่วนนี้อธิบายขั้นตอนที่ทำโดยคำสั่ง **installp** เมื่อการอัพเดต fileset ถูกปฏิเสธ หรือเมื่อ fileset หรือการอัพเดต fileset ไม่สามารถทำการติดตั้งให้เสร็จสมบูรณ์ สคริปต์ **cleanup** และ **reject** ดีฟอลต์อยู่ใน **/usr/lib/instl** จะถูกลิงก์เข้าด้วยกัน ตรรกะของทั้งสองต่างกันเล็กน้อยขึ้นอยู่กับ ว่าสคริปต์ถูกเรียกใช้เป็น **reject** หรือ **cleanup** สำหรับ **usr**/**root** filesets หรือการอัพเดต fileset ส่วน **root** ถูกประมวลผลก่อนส่วน **usr**

1. ถ้าปฏิเสธ ให้ตรวจสอบสิ่งที่ต้องมีเพื่อให้แน่ใจว่า การอัพเดตผลิตภัณฑ์ที่ขึ้นต่อกันทั้งหมดถูกปฏิเสธเช่นกัน
2. สำหรับแต่ละส่วนของแพ็คเกจ (ตัวอย่าง **usr** และ **root**):
   a. ตั้งค่า **INUTREE** (*U* สำหรับ **usr** และ *M* สำหรับ **root**) และตัวแปรสภาวะแวดล้อม **INUTEMPDIR**
   b. ถ้าไฟล์ควบคุม **reject** มีอยู่ ในไดเร็กทอรีปัจจุบัน (**INULIBDIR**) ให้เรียกใช้ **./lpp.reject** มิฉะนั้น เรียกใช้สคริปต์ดีฟอลต์ **/usr/lib/instl/reject**
3. อัพเดต Software Vital Product Data

ไฟล์เรียกทำงาน **reject** ถูกเรียกใช้ จาก **installp** ที่มีพารามิเตอร์แรกไม่ถูกกำหนด และพารามิเตอร์ที่สองเป็นชื่อพาธเต็มไปยังไฟล์ที่มี รายการของ filesets (ถูกอ้างถึงด้านล่างเป็น **$FILESETLIST**) จะถูกปฏิเสธการอัพเดต

ไฟล์ต่อไปนี้ถูกอ้างอิงโดยสคริปต์ **cleanup** และ **reject** ดีฟอลต์

โฟลว์ภายในสคริปต์ **cleanup** และ **reject** ดีฟอลต์เป็นดังนี้:

1. ทำสิ่งใดสิ่งหนึ่งต่อไปนี้สำหรับแต่ละ fileset ที่แสดงรายการใน **$FILESETLIST**:
   a. ถ้าถูกเรียกใช้เป็น **cleanup** จะ อ่านบรรทัดในไฟล์สถานะ *Package_Name*.**s** เพื่อพิจารณาว่าการติดตั้งในขั้นตอนใดที่ล้มเหลว และ ข้ามไปดำเนินการเลิกทำสำหรับขั้นตอนนั้น การดำเนินการ **คลีนอัพ** จะเริ่มต้นที่ขั้นตอนที่การติดตั้งล้มเหลวเท่านั้น ตัวอย่างเช่น ถ้าการติดตั้ง ชุดไฟล์ล้มเหลวในสคริปต์ *Fileset*.**post_i** ดังนั้น การดำเนินการ cleanup สำหรับชุดไฟล์ดังกล่าวจะเริ่มต้นที่ i เนื่องจากไม่มี การดำเนินการเพื่อยกเลิกการทำจากขั้นตอนต่อมาในการติดตั้ง
   b. เลิกทำการประมวลผลการตั้งค่าใดๆ ที่ถูกดำเนินการระหว่างการติดตั้ง:
   ถ้ามีการปฏิเสธการอัพเดต ให้เรียกใช้ *Fileset*.**unconfig_u** ถ้ามีอยู่ มิฉะนั้น ให้เรียกใช้ *Fileset*.**unconfig** ถ้ามีอยู่
   c. รันไฟล์ *Fileset*.*.**unodmadd**และ/หรือ *Fileset*.**unodmadd** ใดๆ เพื่อลบรายการ Object Data Manager (ODM) ที่เพิ่มระหว่างการติดตั้ง
   d. รัน *Fileset*.*.**rodmadd** และ/หรือ *Fileset*.**rodmadd** ใดๆ ที่อยู่เพื่อแทนที่รายการ ODM ที่ลบระหว่างการติดตั้ง
   e. เรียกใช้ **trcupdate** ถ้า *Fileset*.**undo.trc** มีอยู่เพื่อเลิกทำการเปลี่ยนแปลงเท็มเพลต รูปแบบการติดตามใดๆ ที่ทำระหว่างการติดตั้ง
   f. เรียกใช้ **errupdate** ถ้า *Fileset*.**undo.err** มีอยู่เพื่อเลิกทำการเปลี่ยนแปลงเท็มเพลต รูปแบบข้อผิดพลาดใด ๆ ที่ทำระหว่างการติดตั้ง
   g. เรียกใช้ **tcbck** เพื่อลบข้อมูล trusted computing base ของระบบถ้าไฟล์ *Fileset*.**tcb** มีอยู่ และ trusted computing base attribute **tcb_enabled** ถูกตั้งค่าในฐานข้อมูล **/usr/lib/objrepos/PdAt** ODM
   h. เรียกใช้ **sysck** ถ้า *Fileset*.**inventory** มีอยู่เพื่อเลิกทำการเปลี่ยนแปลงใน ฐานข้อมูลซอฟต์แวร์

i. เลิกทำการประมวลผล post_installation ใดๆ ที่ดำเนินการระหว่าง การติดตั้ง:
ถ้านี้เป็นการอัพเดต ให้เรียกใช้ *Fileset*.**unpost_u** ถ้ามีอยู่ มิฉะนั้น ให้เรียกใช้ *Fileset*.**unpost_i** ถ้า มีอยู่

j. สร้างรายการนำใช้มาสเตอร์ (เรียกว่า **master.al**) จากไฟล์ *Fileset*.**al**

k. เพิ่ม *Fileset* ไปยัง **$FILESETLIST.new**

2. ทำสิ่งต่อไปนี้ถ้า **$INUTEMPDIR/master.al** มีอยู่
   a. เปลี่ยนไดเร็กทอรีเป็น **/** (**root**)
   b. ลบไฟล์ทั้งหมดใน **master.al** ออก
3. ทำสิ่งต่อไปนี้ระหว่างการอ่าน **$FILESETLIST.new**
   a. เรียกใช้ **inurecv** เพื่อกู้คืน ไฟล์ที่บันทึกทั้งหมด
   b. ถ้านี้เป็นการอัพเดต ให้เรียกใช้ *Fileset*.**unpre_u** ถ้ามีอยู่ มิฉะนั้น ให้เรียกใช้ *Fileset*.**unpre_i** ถ้ามีอยู่
   c. ลบไฟล์ควบคุมการติดตั้ง/การอัพเดต
4. ลบไฟล์สถานะ *Package_Name*.**s** ออก

การประมวลผลสำหรับการดำเนินการ remove

ส่วนนี้อธิบายขั้นตอนที่ทำโดยคำสั่ง **installp** เมื่อ fileset ถูกลบออก สำหรับ **usr** / **root** filesets หรือการอัพเดต fileset ส่วน **root** ถูกประมวลผลก่อนส่วน **usr**

1. ตรวจสอบสิ่งที่ต้องมีเพื่อให้แน่ใจว่า filesets ทั้งหมดที่ขึ้นต่อกัน จะถูกลบออกด้วย
2. สำหรับแต่ละส่วนของแพ็กเกจผลิตภัณฑ์ (ตัวอย่างเช่น **usr** หรือ **root**):
   a. ตั้งค่า INUTREE (U สำหรับ **usr** M สำหรับ **root** และ S สำหรับ **share**) และตัวแปรสภาวะแวดล้อม INUTEMPDIR (ไดเร็กทอรีการทำงาน **installp** ที่สร้างใน **/tmp**)
   b. เปลี่ยนไดเร็กทอรีเป็น **INULIBDIR**
   c. ถ้าไฟล์ควบคุม **deinstal** มีอยู่ในไดเร็กทอรีปัจจุบัน ให้รันสคริปต์ **./lpp.deinstal** ถ้าไฟล์ควบคุม **deinstal** ไม่มีอยู่ใน ไดเร็กทอรีปัจจุบัน ให้รันดีฟอลต์สคริปต์ **/usr/lib/instl/deinstal**
3. ลบไฟล์ที่เป็นของ fileset ออกจาก ระบบไฟล์
4. ลบรายการ fileset ออกจาก SWVPD ยกเว้นสำหรับข้อมูล ประวัติ
5. เลิกเรียกใช้การลงทะเบียนข้อกำหนดข้อตกลงการอนุญาตใช้สิทธิ์สำหรับ fileset

ไฟล์เรียกทำงาน **deinstal** ถูกเรียกใช้ จาก **installp** ที่มีพารามิเตอร์แรกเป็นชื่อพาธเต็ม ไปยังไฟล์ที่มีรายการ filesets ที่จะถูกลบออก ถูกอ้างถึง ด้านล่างเป็น **$FILESETLIST**

โฟลว์ภายในสคริปต์ **deinstal** ดีฟอลต์เป็นดังนี้:

1. ทำสิ่งต่อไปนี้สำหรับแต่ละ fileset ที่แสดงรายการในไฟล์ อินพุต **$FILESETLIST**:
2. ถ้า *Fileset*.**unconfig_d** มีอยู่
   ทำงาน *Fileset*.**unconfig_d** เพื่อลบการเปลี่ยนแปลง การตั้งค่าทั้งหมด การเปลี่ยนแปลง Object Data Manager (ODM) และการเปลี่ยนแปลงรูปแบบข้อผิดพลาด และการติดตาม และเพื่อเลิกทำการดำเนินการทั้งหมดที่ทำในสคริปต์หลังการติดตั้งและก่อนก่อนการติดตั้ง สำหรับการอัพเดตและการติดตั้งระดับพื้นฐานทั้งหมด การใช้ไฟล์นี้ *ไม่* แนะนำให้ใช้
3. ถ้า *Fileset*.**unconfig_d** ไม่มีอยู่
   a. สำหรับแต่ละการอัพเดตสำหรับ fileset ให้ดำเนินการต่อไปนี้:

- รันสคริปต์ *Fileset*.**unconfig_u** ทั้งหมด เพื่อเลิกทำการประมวลผลการตั้งค่าอัพเดตใดๆ
- รัน *Fileset*.*.**unodmadd** and *Fileset*.**unodmadd** เพื่อลบรายการ Object Data Manager (ODM) ที่เพิ่มระหว่างการอัพเดต
- รัน *Fileset*.*.**rodmadd** and *Fileset*.**rodmadd** เพื่อเพิ่มรายการ Object Data Manager (ODM) ที่ลบระหว่างการอัพเดต
- รัน **errupdate** ถ้า *Fileset*.**undo.err** มีอยู่เพื่อเลิกทำการเปลี่ยนแปลงเท็มเพลตบันทึกข้อผิดพลาด
- รัน **trcupdate** ถ้า *Fileset*.**undo.trc** มีอยู่เพื่อเลิกทำการเปลี่ยนแปลงเท็มเพลตรายงานการติดตาม
- รัน *Fileset*.**unpost_u** ใดๆ เพื่อ เลิกทำการกำหนดการทำงานหลังการติดตั้งใดๆ เอง

b. สำหรับระดับการติดตั้งตาม fileset ให้ดำเนินการต่อไปนี้:

- รัน *Fileset*.*.**unodmadd**และ/หรือ *Fileset*.**unodmadd** ใดๆ เพื่อลบรายการ Object Data Manager (ODM) ที่เพิ่มระหว่างการติดตั้ง
- รัน *Fileset*.*.**rodmadd** และ/หรือ *Fileset*.**rodmadd** ใดๆ เพื่อเพิ่มรายการ Object Data Manager (ODM) ที่ลบระหว่างการติดตั้ง
- รัน **errupdate** ถ้า *Fileset*.**undo.err** มีอยู่เพื่อเลิกทำการเปลี่ยนแปลงเท็มเพลตบันทึกข้อผิดพลาด
- รัน **trcupdate** ถ้า *Fileset*.**undo.trc** มีอยู่เพื่อเลิกทำการเปลี่ยนแปลงเท็มเพลตรายงานการติดตาม
- รัน *Fileset*.**unconfig_i** เพื่อ เลิกทำการประมวลผลการกำหนดการตั้งค่าใดๆ
- รัน *Fileset*.**unpost_i** เพื่อ เลิกทำการกำหนดค่าหลังการติดตั้งไฟล์เอง

4. ลบไฟล์ และข้อมูลซอฟต์แวร์ที่ติดตั้ง กับ fileset ออก
5. ถ้า *Fileset*.**unconfig_d** ไม่มีอยู่

a. สำหรับแต่ละการอัพเดตสำหรับ fileset นั้น ให้รัน *Fileset*.**unpre_u** ใดๆ เพื่อเลิกทำการกำหนดค่าก่อนการติดตั้งไฟล์เอง

b. สำหรับระดับการติดตั้งตาม fileset ให้รัน *Fileset*.**unpre_i** ใดๆ เพื่อเลิกทำการกำหนดค่า ก่อนการติดตั้งไฟล์ใดๆ เอง

6. ลบไดเร็กทอรีว่างที่สัมพันธ์กับ fileset

หมายเหตุ: ถ้าข้อผิดพลาดถูกส่งกลับมาจากการเรียกใช้ระหว่างการทำงานของไฟล์เรียกทำงาน **deinstal** ข้อผิดพลาดจะถูกบันทึก แต่การทำงานนั้น จะดำเนินต่อไป นี้แตกต่างจากสคริปต์อื่นๆ เนื่องจากการทำงาน สำหรับ fileset นั้นโดยปกติจะถูกยกเลิกทันทีที่พบข้อผิดพลาด อย่างไรก็ตาม เมื่อเริ่มทำการลบ fileset ออก จะไม่สามารถทำการการกู้คืน ดังนั้น การลบออกจึงเป็นความพยายามที่ดีที่สุดเมื่อพบข้อผิดพลาด

ไฟล์ สถานะการติดตั้ง

$INUTEMPDIR/status
ไฟล์ที่มีรายการหนึ่งบรรทัดสำหรับแต่ละ fileset ที่จะ ถูกติดตั้งหรืออัพเดต

คำสั่ง **installp** ใช้ไฟล์ **สถานะ** นี้เพื่อพิจารณาการประมวลผลที่เหมาะสม ถ้าคุณสร้างสคริปต์ การติดตั้ง สคริปต์ของคุณควรสร้างไฟล์ **สถานะ** ที่ มีรูปแบบที่ถูกต้อง แต่ละบรรทัดในไฟล์ **สถานะ** มีรูปแบบดังนี้:

```
StatusCode Fileset
```

โค้ดสถานะ	ความหมาย
s	สำเร็จ อัพเดต SWVPD
f	ล้มเหลว ดำเนินขั้นตอนคลีนอัพ
b	ข้าม ล้มเหลว ไม่จำเป็นต้องคลีนอัพ
i	สิ่งที่ต้องมีล้มเหลว ไม่จำเป็นต้องคลีนอัพ
v	การตรวจสอบ **sysck** ล้มเหลว

ตัวอย่างต่อไปนี้ของไฟล์ **สถานะ** แจ้งคำสั่ง **installp** ให้ทราบว่า การติดตั้ง **tcp.client** และ **tcp.server** filesets ของแพ็กเกจ **bos.net** สำเร็จ และการติดตั้งสำหรับ **nfs.client** fileset ไม่สำเร็จ

```
s bos.net.tcp.client
s bos.net.tcp.server
f bos.net.nfs.client
```

คำสั่งการติดตั้ง ที่ใช้ระหว่างกระบวนการการติดตั้งและอัพเดต

inucp ทำสำเนาไฟล์จากไดเร็กทอรี **/usr/lpp/*Package_Name*/inst_root** ไปยังแผนผังไฟล์ **/** (root) เมื่อติดตั้งส่วน root

inulag ทำหน้าที่เป็นฟไปยังรูทีนย่อยเพื่อจัดการข้อตกลงการอนุญาตใช้สิทธิ์

inurecv เรียกคืนไฟล์ที่บันทึกทั้งหมดสำหรับความล้มเหลวในการติดตั้งหรือการปฏิเสธซอฟต์แวร์ (**installp -r**)

inurest เรียกคืนไฟล์จากสื่อบันทึกการแจกจ่ายไปยังระบบโดยใช้ รายการนำใช้เป็นอินพุต

inusave บันทึกไฟล์ทั้งหมดที่ระบุโดยรายการนำใช้ไปยังไดเร็กทอรีบันทึกที่เป็นของ ผลิตภัณฑ์ซอฟต์แวร์

inuumsg
แสดงข้อความจากไฟล์แค็ตตาล็อกข้อความ `inuumsg.cat` สำหรับผลิตภัณฑ์ซอฟต์แวร์ที่กำลังติดตั้ง

ckprereq
ตรวจสอบความเข้ากันได้ของผลิตภัณฑ์ซอฟต์แวร์ที่มีการขึ้นต่อกันใดๆ โดยใช้ข้อมูลสิ่งที่ต้องมีที่อยู่ในไฟล์ lpp_name และข้อมูล เกี่ยวกับผลิตภัณฑ์ที่ติดตั้งอยู่แล้วที่บนใน SWVPD

sysck ตรวจสอบข้อมูล inventory ระหว่างขั้นตอนการติดตั้งและการอัพเดต

คำสั่ง **sysck** อยู่ในไดเร็กทอรี **/usr/bin** คำสั่งอื่นๆ ที่แสดงรายการก่อนหน้านี้อยู่ใน ไดเร็กทอรี **/usr/sbin**

สำหรับตัวอย่างของการใช้งานนี้ โปรดอ้างอิงสคริปต์การติดตั้ง ดีฟอลต์ **/usr/lib/instl/instal**

Source code control system

Source code control system (SCCS) เป็นระบบของคำสั่ง ทั้งหมดที่อนุญาตให้ผู้ใช้ที่ระบุควบคุมและติดตามการเปลี่ยนแปลงที่เกิดขึ้นกับไฟล์ SCCS ไฟล์ SCCS อนุญาตให้ใช้เวอร์ชันต่างๆ ของไฟล์เดียวกันที่มีอยู่อย่างพร้อมเพียงกัน ซึ่งอาจมีประโยชน์ต่อการพัฒนาโปรเจ็กต์ ที่ต้องการเวอร์ชันจำนวนมากของไฟล์ที่มีขนาดใหญ่

คำสั่ง SCCS จะสนับสนุนอักขระแบบ Multibyte Character Set (MBCS)

คำแนะนำเบื้องต้นเกี่ยวกับ SCCS

คำสั่ง SCCS จะจัดรูปแบบระบบที่สมบูรณ์สำหรับการสร้าง การแก้ไข การแปลง หรือการเปลี่ยนการควบคุมบนไฟล์ SCCS ไฟล์ SCCS คือเท็กซ์ไฟล์ใดๆ ที่ควบคุมด้วยคำสั่ง SCCS ไฟล์ SCCS ทั้งหมดจะมีคำนำหน้า **s.** ซึ่งจะตั้งค่าไฟล์เหล่านั้นออกจากเท็กซ์ไฟล์ปกติ

โปรดทราบ: การใช้คำสั่งที่ไม่ใช่ SCCS เพื่อแก้ไขไฟล์ SCCS อาจทำให้เกิดความเสียหายต่อไฟล์ SCCS ได้

ใช้คำสั่ง SCCS ที่อยู่บนไฟล์ SCCS ถ้าคุณต้องการมองหาโครงสร้างของไฟล์ SCCS ให้ใช้คำสั่ง **pg** หรือคำสั่งที่คล้ายกันเพื่อดูเนื้อหา อย่างไรก็ตาม ห้ามใช้เอดิเตอร์เพื่อเปลี่ยนไฟล์โดยตรง

หากต้องการเปลี่ยนข้อความในไฟล์ SCCS ให้ใช้คำสั่ง SCCS (เช่น คำสั่ง **get**) เพื่อขอรับเวอร์ชันของไฟล์สำหรับการแก้ไข จากนั้นใช้เอดิเตอร์ใดๆ เพื่อแก้ไขข้อความ หลังจากที่เปลี่ยนไฟล์แล้ว ให้ใช้คำสั่ง **delta** เพื่อบันทึกการเปลี่ยนแปลง หากต้องการเก็บเวอร์ชันของไฟล์แยกจากกัน และควบคุมการเข้าถึงเนื้อหา ไฟล์ SCCS ต้องมีโครงสร้างเฉพาะ

ไฟล์ SCCS จะสร้างขึ้นจากส่วนทั้งหมดสามส่วน:

- ตาราง Delta
- เข้าถึงและติดตามแฟล็ก
- เนื้อของข้อความ

ตาราง Delta ในไฟล์ SCCS

แทนที่จะสร้างไฟล์ที่แยกจากกันสำหรับแต่ละเวอร์ชันของไฟล์ ระบบไฟล์ SCCS จะเก็บการเปลี่ยนแปลง สำหรับแต่ละเวอร์ชันของไฟล์เท่านั้น การเปลี่ยนแปลงเหล่านี้จะถูกอ้างถึงเป็น *deltas* การเปลี่ยนแปลง จะถูกติดตามโดยตาราง delta ในไฟล์ SCCS ทุกไฟล์

แต่ละรายการในตาราง delta จะมีข้อมูลเกี่ยวกับผู้สร้างข้อมูล เมื่อข้อมูลเหล่านั้นถูกสร้างขึ้น และสาเหตุของการสร้าง แต่ละ delta จะมี SID เฉพาะ (หมายเลข SCCS IDentification) ได้สูงสุดสี่หลัก หลักแรกคือรีลีส หลักที่สองคือระดับ หลักที่สามคือสาขา และหลักที่สี่คือลำดับ

ตัวอย่างของหมายเลข SID คือ:

```
SID = 1.2.1.4
```

นั่นคือ รีลีส 1 ระดับ 2 สาขา 1 ลำดับ 4

ไม่มีหลักของ SID ที่สามารถมีค่า 0 ได้ ดังนั้นจึงไม่สามารถเป็น SID ตัวอย่างเช่น 2.0 หรือ 2.1.2.0 ได้

แต่ละครั้งที่สร้าง delta ใหม่ ระบบจะกำหนดหมายเลข SID ที่สูงกว่าตามค่าดีฟอลต์ เวอร์ชันของไฟล์จะถูกสร้างโดยใช้ delta ก่อนหน้านี้ทั้งหมด โดยปกติแล้ว ไฟล์ SCCS จะโตตามลำดับ ดังนั้น แต่ละ delta จะระบุด้วยรีลีสและระดับเท่านั้น อย่างไรก็ตาม ไฟล์อ่านมีสาขาและสร้างชุดย่อยของ delta ขึ้นใหม่ จากนั้น ไฟล์จะมีเส้นทาง พร้อมกับ delta ที่ระบุด้วยรีลีสหรือระดับ และสาขาที่มากกว่าหนึ่ง ซึ่งมีรายละเอียดที่ควบคุมส่วนทั้งสี่ส่วนของ SID สำหรับสาขา หมายเลขรีลีสและระดับจะถูกแก้ไข และ deltas ใหม่จะถูกบ่งชี้โดยการเปลี่ยนหมายเลขลำดับ

หมายเหตุ: ไฟล์เวอร์ชันที่สร้างจากสาขา ไม่ได้ใช้ delta ใดๆ ที่วางอยู่หลังจุดของการแบ่งแยก

แฟล็กควบคุมและการติดตามในไฟล์ SCCS

หลังจากที่ตาราง delta ในไฟล์ SCCS รายการของแฟล็กที่ขึ้นต้นด้วย @ (เครื่องหมาย at sign) จะนิยามการเข้าถึงที่หลากหลายและอ็อพชันการติดตามของไฟล์ SCCS ฟังก์ชันแฟล็ก SCCS บางตัวประกอบด้วย:

- การออกแบบผู้ใช้ที่อาจแก้ไขไฟล์
- การล็อกรีลีสบางตัวของไฟล์จากการแก้ไข
- อนุญาตให้เชื่อมเพื่อแก้ไขไฟล์
- เปลี่ยนการอ้างอิงระหว่างกันในไฟล์

เนื้อความของไฟล์ SCCS

เนื้อความของไฟล์ SCCS จะมีข้อความสำหรับเวอร์ชันของไฟล์ ที่แตกต่างกันทั้งหมด ดังนั้น เนื้อความของไฟล์ไม่ได้ดูคล้ายกับเท็กซ์ไฟล์มาตรฐาน ควบคุมอักขระเครื่องหมายวงเล็บในแต่ละส่วนของข้อความ และระบุ delta ที่สร้างขึ้นหรือลบทิ้ง เมื่อระบบ SCCS build เวอร์ชันของไฟล์ที่ระบุเฉพาะ อักขระควบคุมจะบ่งชี้ส่วนของข้อความ ที่สอดคล้องกับแต่ละ delta ส่วนที่เลือกไว้ของข้อความจะถูกใช้เพื่อ build เวอร์ชันที่ระบุเฉพาะนั้น

แฟล็ก sccs และหลักการสำหรับพารามิเตอร์

ส่วนนี้แสดงรายการแฟล็กสำหรับคำสั่ง SCCS

ในกรณีส่วนใหญ่แล้ว คำสั่ง SCCS จะยอมรับพารามิเตอร์อยู่ด้วยกันสองชนิด:

พารามิเตอร์	คำอธิบาย
แฟล็ก	แฟล็กประกอบด้วย - (เครื่องหมายลบ) ตามด้วยอักขระตัวพิมพ์เล็ก ซึ่งในบางครั้งอาจตามด้วยค่า แฟล็กจะควบคุมวิธีการทำงานของคำสั่ง
ไฟล์ หรือ *ไดเร็กทอรี*	พารามิเตอร์เหล่านี้ระบบไฟล์หรือไฟล์ต่างๆ พร้อมกับคำสั่งที่ดำเนินการอยู่ การใช้ชื่อไดเร็กทอรีเป็นอาร์กิวเมนต์จะระบุไฟล์ SCCS ทั้งหมดที่อยู่ในไดเร็กทอรีนั้น

ชื่อไฟล์หรือไดเร็กทอรีไม่สามารถขึ้นต้นด้วย - (เครื่องหมายลบ) ถ้าคุณระบุเครื่องหมายนี้เอง คำสั่งจะอ่านอินพุตแบบมาตรฐาน หรืออินพุตคีย์บอร์ดจนกว่าจะมาถึงอักขระสิ้นสุดไฟล์ ซึ่งมีข้อดี เมื่อใช้ไพพ์ที่อนุญาตให้กระบวนการต่างๆ สื่อสารได้

แฟล็กใดๆ ที่ระบุไว้สำหรับคำสั่งจะนำมาใช้กับไฟล์ทั้งหมดที่อยู่บนบรรทัดรับคำสั่ง และจะประมวลผลก่อนพารามิเตอร์อื่นๆ ในคำสั่งนั้น การจัดวางแฟล็กในบรรทัดรับคำสั่งไม่ใช่สิ่งที่สำคัญ พารามิเตอร์อื่นๆ จะถูกประมวลผลจากซ้ายไปขวา ไฟล์ SCCS บางไฟล์มีแฟล็กที่กำหนดวิธีที่คำสั่งบางคำสั่งทำงานกับไฟล์นี้ โปรดดูคำอธิบายคำสั่ง **admin** ของแฟล็กส่วนหัว SCCS สำหรับข้อมูลเพิ่มเติม

การสร้าง การแก้ไข และการอัพเดตไฟล์ sccs

คุณสามารถ สร้าง แก้ไข และอัพเดตไฟล์ SCCS โดยใช้คำสั่ง **admin, get** และ **delta**

การสร้างไฟล์ SCCS

`admin`
สร้างไฟล์ SCCS หรือเปลี่ยนไฟล์ SCCS ที่มีอยู่

- ในการ สร้างไฟล์ SCCS ว่างชื่อ `s.test.c` ให้ป้อน:

```
admin -n s.test.c
```

โดยใช้ คำสั่ง **admin** ด้วยแฟล็ก **-n** สร้างไฟล์ SCCS เปล่า

- เมื่อต้องการแปลงไฟล์ข้อความที่มีอยู่ไปเป็นไฟล์ SCCS ให้ป้อน:

```
admin -itest.c s.test.c
ไม่มีคีย์เวิร์ดการระบุ SCCS ในไฟล์ (cm7)

คือ
s.test.c test.c
```

ถ้าคุณใช้แฟล็ก **–i** คำสั่ง **admin** สร้าง 1.1 จากไฟล์ที่ระบุ เมื่อ delta 1.1 ถูกสร้าง เปลี่ยนชื่อไฟล์ข้อความ เพื่อที่ไฟล์จะไม่ไปรบกวนคำสั่ง SCCS (ไฟล์แสดงตัวเหมือนเป็นสำเนาสำรอง):

```
mv test.c back.c
```

ข้อความ `There are no SCCS identification keywords in the file (cm7)` ไม่ได้บ่งชี้ถึงข้อผิดพลาด

- เมื่อต้องการเริ่มต้นไฟล์ `test.c` ด้วยหมายเลขรีลีส 3.1 ให้ใช้แฟล็ก **–r** กับคำสั่ง **admin** ดังต่อไปนี้:

```
admin -itest.c -r3 s.test.c
```

การแก้ไขไฟล์ SCCS

ข้อควรใส่ใจ: อย่าแก้ไข ไฟล์ SCCS โดยตรงด้วยคำสั่งที่ไม่ใช่ SCCS หรือไม่แล้วคุณสามารถทำให้ไฟล์ SCCS เสียหาย

get

รับเวอร์ชันที่ระบุของไฟล์ SCCS สำหรับการแก้ไขหรือคอมไพล์

1. เมื่อต้องการแก้ไขไฟล์ SCCS ให้ป้อนคำสั่ง **get** ด้วยแฟล็ก **–e** เพื่อสร้างเวอร์ชัน ที่แก้ไขได้ของไฟล์:

```
get -e s.test.c
1.3
new delta 1.4
67 lines

คือ
p.test.c s.test.c test.c
```

คำสั่ง **get** สร้าง สองไฟล์ใหม่ `p.test.c` และ `test.c`. ไฟล์ที่แก้ไขได้คือไฟล์ `test.c` ไฟล์ `p.test.c` เป็นไฟล์ชั่วคราว ไม่สามารถแก้ไขได้ ที่ใช้โดย SCCS เพื่อติดตามเวอร์ชันไฟล์ ไฟล์จะหายไปเมื่อคุณอัพเดต การเปลี่ยนแปลงของคุณกับไฟล์ SCCS หมายเหตุไว้ว่าคำสั่ง **get** พิมพ์ SID ของเวอร์ชันที่สร้างสำหรับการแก้ไข SID ที่กำหนด ให้กับ delta ใหม่เมื่อคุณอัพเดตการเปลี่ยนแปลง และจำนวนบรรทัด ในไฟล์

2. ใช้เอดิเตอร์ใดๆ เพื่อแก้ไข `test.c` ตัวอย่างเช่น:

```
ed test.c
```

ขณะนี้ คุณสามารถทำงานกับไฟล์จริงของคุณ แก้ไขไฟล์ได้บ่อยเท่าที่คุณต้องการ การเปลี่ยนแปลง ของคุณจะไม่มีผลกับไฟล์ SCCS จนกว่าคุณเลือกที่จะอัพเดตไฟล์

3. เมื่อต้องการแก้ไขเวอร์ชันของไฟล์ SCCS ที่มีหลาย เวอร์ชัน ให้ป้อนคำสั่ง **get** กับแฟล็ก **–r**:

```
get -r1.3 s.test.c
1.3
67 lines

get -r1.3.1.4 s.test.c
1.3.1.4
50 lines
```

การอัพเดตไฟล์ SCCS

เดลต้า

เพิ่มชุดของการเปลี่ยนแปลง (deltas) ให้กับข้อความของไฟล์ SCCS

1. เมื่อต้องการอัพเดตไฟล์ SCCS และสร้าง delta ใหม่ ที่มีการเปลี่ยนแปลงที่คุณได้กระทำ ขณะแก้ไข ให้ใช้คำสั่ง **delta**:

   ```
   $delta s.test.c
   พิมพ์ข้อคิดเห็น ปิดท้ายด้วย EOF หรือบรรทัดว่าง:
   ```

2. คำสั่ง **delta** พร้อมต์คุณ เพื่อขอข้อคิดเห็นที่จะนำมาเชื่อมโยงกับการเปลี่ยนแปลงที่คุณได้ทำ ตัวอย่างเช่น ป้อนข้อคิดเห็นของคุณ แล้วกดแป้น Enter สองครั้ง:

   ```
   No id keywords (cm7)
   1.2
   5 lines inserted
   6 lines deleted
   12 lines unchanged
   ```

   คำสั่ง **delta** อัพเดตไฟล์ s.prog.c ด้วยการเปลี่ยนแปลงที่คุณได้ทำกับไฟล์ `test.c` คำสั่ง **delta** บอกแก่คุณว่า SID ของเวอร์ชันใหม่คือ 1.2 และไฟล์ที่แก้ไขแทรก 5 บรรทัด ถูกลบ ไป 6 บรรทัด และเหลือ 12 บรรทัดที่ไม่มีการเปลี่ยนแปลงจากเวอร์ชันก่อนหน้า

การควบคุมและการติดตามการเปลี่ยนแปลงไฟล์ sccs

คำสั่ง SCCS และระบบไฟล์ใช้เป็นหลักในการควบคุมการเข้าถึงไฟล์ และติดตามผู้ที่เปลี่ยนแปลงไฟล์ เหตุผลที่ถูกเปลี่ยนแปลง และสิ่งที่ถูกเปลี่ยนแปลง

การควบคุมการเข้าถึงไฟล์ SCCS

การเข้าถึงสามชนิดสามารถถูกควบคุมในระบบไฟล์ SCCS :

- การเข้าถึงไฟล์
- การเข้าถึงของผู้ใช้
- การเข้าถึงเวอร์ชัน

ควบคุมการเข้าถึงไฟล์

ไดเร็กทอรีที่มีไฟล์ SCCS ควรถูกสร้างด้วย รหัสสิทธิ 755 (สิทธิ read, write และ execute สำหรับเจ้าของ; สิทธิ read และ execute สำหรับสมาชิกกลุ่มและอื่นๆ) ตัวไฟล์ SCCS เอง ควรถูกสร้างเป็นไฟล์แบบอ่านอย่างเดียว (444) ด้วยสิทธิเหล่านี้ เฉพาะเจ้าของสามารถใช้คำสั่งที่ไม่ใช่ SCCS ในการแก้ไขไฟล์ SCCS ถ้ากลุ่ม สามารถเข้าถึงและปรับเปลี่ยนไฟล์ SCCS ไดเร็กทอรีควรมีสิทธิ write แบบกลุ่ม

ควบคุมการเข้าถึงผู้ใช้

คำสั่ง **admin** พร้อมกับแฟล็ก **–a** สามารถกำหนด กลุ่มของผู้ใช้ที่สามารถทำการเปลี่ยนแปลงกับไฟล์ SCCS ชื่อกลุ่มหรือหมายเลข สามารถใช้ระบุกับแฟล็กนี้ได้เช่นกัน

ควบคุมการเข้าถึงเวอร์ชัน

คำสั่ง **admin** สามารถล็อก หรือป้องกัน เวอร์ชันของไฟล์จากการเข้าถึง โดยคำสั่ง **get** โดย ใช้แฟล็ก header

`-fc`
ตั้งค่าเพดานหมายเลขรีลีสสูงสุดที่ถูกเรียกข้อมูลได้

`-ff`
ตั้งพื้นหมายเลขรีลีสต่ำสุดที่ถูกเรียกข้อมูลได้

`-fl`
ล็อกรีลีสจำเพาะจากการเรียกข้อมูล

การติดตามการเปลี่ยนแปลงกับไฟล์ SCCS

มีสามวิธีในการติดตามการเปลี่ยนแปลงกับ SCCS:

- ข้อคิดเห็นที่สัมพันธ์กับแต่ละ delta
- หมายเลข Modification Request (MR)
- คำสั่ง SCCS

การติดตามการเปลี่ยนแปลงกับข้อคิดเห็น delta

หลังจากไฟล์ SCCS ถูกอัพเดตและ delta ใหม่ถูกสร้าง ระบบพร้อมต์ของข้อคิดเห็นเพื่อนำไปเชื่อมโยงกับ delta ข้อคิดเห็นเหล่านี้ มีความยาวอักขระได้มากถึง 512 อักขระและสามารถถูกแก้ไขด้วยคำสั่ง **cdc**

`cdc`
เปลี่ยนข้อคิดเห็นที่สัมพันธ์กับ delta

คำสั่ง **get** พร้อมกับแฟล็ก **-l** พิมพ์ตาราง delta และข้อคิดเห็น delta ทั้งหมดสำหรับเวอร์ชันของไฟล์ นอกเหนือจากการเก็บข้อคิดเห็น ที่สัมพันธ์กับ delta ตาราง delta เก็บข้อมูลเวลาและวันที่ ของการแก้ไขล่าสุด user ID จริงในเวลาที่มีการเปลี่ยนแปลง หมายเลขอนุกรมของ delta และต้นกำเนิด และหมายเลข MR ที่สัมพันธ์ กับ delta โดยอัตโนมัติ

การติดตามการเปลี่ยนแปลงกับหมายเลขคำร้องขอการแก้ไข

คำสั่ง **admin** พร้อมกับแฟล็ก **-fv** พร้อมต์ของหมายเลข MR ในแต่ละครั้งที่มีการสร้าง delta โปรแกรมสามารถถูกระบุด้วยแฟล็ก **-fv** เพื่อตรวจสอบความถูกต้องของหมายเลข MR เมื่อมีความ พยายามสร้าง delta ในไฟล์ SCCS ถ้าโปรแกรม MR validity-checking ส่งกลับค่าจบการทำงานที่ไม่ใช่ศูนย์ การอัพเดตจะไม่สำเร็จ

โปรแกรม MR validity-checking ถูกสร้างโดย ผู้ใช้ ซึ่งสามารถถูกเขียนเพื่อติดตามการเปลี่ยนแปลงกับไฟล์ SCCS และทำดัชนี เทียบกับฐานข้อมูลหรือระบบติดตามอื่น

การติดตามการเปลี่ยนแปลงด้วยคำสั่ง SCCS

`sccsdiff`
เปรียบเทียบไฟล์ SCCS สองไฟล์และพิมพ์ความแตกต่างไปที่เอาต์พุตมาตรฐาน

คำสั่ง **delta** พร้อมกับ แฟล็ก **-p** ทำงานเหมือนกับคำสั่ง **sccsdiff** เมื่อไฟล์ถูกอัพเดต ทั้งสองคำสั่งนี้อนุญาตให้คุณ ดูสิ่งที่เปลี่ยนแปลงไประหว่างเวอร์ชัน

`prs`
จัดรูปแบบและพิมพ์ส่วนที่ระบุของ SCCS ไปที่เอาต์พุต มาตรฐาน

คำสั่งนี้อนุญาตให้คุณค้นหาความแตกต่างในไฟล์สองเวอร์ชัน

การตรวจหาและการซ่อมไฟล์ sccs ที่เสียหาย

คุณสามารถตรวจหาและซ่อมแซมไฟล์ SCCS ที่เสียหาย โดยใช้คำสั่ง **admin**

โพรซีเดอร์

1. ตรวจสอบไฟล์ SCCS เป็นประจำสำหรับ ความเสียหายที่เกิดขึ้นได้ ทุกครั้งที่ไฟล์ SCCS ถูกเปลี่ยนแปลงโดยปราศจากการใช้คำสั่ง SCCS ที่ถูกต้อง อาจมีความเสียหายเกิดขึ้นกับไฟล์ ระบบไฟล์ SCCS ตรวจหาความเสียหายนี้โดย คำนวณเช็คซัม และเปรียบเทียบกับข้อมูลที่เก็บในตาราง เดลต้า ตรวจสอบความเสียหายโดยการรันคำสั่ง **admin** ด้วยแฟล็ก **–h** บน SCCS หรือไดเร็กทอรี SCCS ทั้งหมด ตามที่แสดง:

   ```
   admin -h s.file1 s.file2 ...
   ```

   หรือ

   ```
   admin -h directory1 directory2 ...
   ```

   ถ้าคำสั่ง **admin** ค้นหาไฟล์โดยที่เช็คซัมที่คำนวณไม่เท่ากับ เช็คซัมที่แสดงในส่วนหัวไฟล์ SCCS คำสั่งจะแสดง ข้อความนี้:

   ```
   ERROR [s.filename]:
   1255-057 ไฟล์เสียหาย (co6)
   ```

2. ถ้าไฟล์เสียหาย ให้พยายามแก้ไขไฟล์อีกครั้ง หรืออ่านข้อมูลสำเนาสำรอง เมื่อเช็คซัมได้ทำการคำนวณซ้ำ ความเสียหายที่คงอยู่ จะตรวจไม่พบโดยคำสั่ง **admin**

   หมายเหตุ: การใช้คำสั่ง **admin** ด้วยแฟล็ก **–z** กับไฟล์ที่เสียหาย สามารถป้องกันการตรวจหาความเสียหายในอนาคต

3. หลังจากแก้ไขไฟล์ รันคำสั่ง **admin** ด้วยแฟล็ก **–z** และซ่อมแซม ชื่อไฟล์:

   ```
   admin -z s.file1
   ```

รายการของคำสั่ง sccs เพิ่มเติม

คำสั่ง SCCS ดังต่อไปนี้ทำให้ระบบสำหรับ การจัดการไฟล์ SCCS สมบูรณ์:

ข้อควรใส่ใจ:: การใช้คำสั่งที่ไม่ใช่ SCCS กับไฟล์ SCCS อาจทำให้ไฟล์ SCCS เสียหายได้

คำสั่ง	คำอธิบาย
rmdel	เอา delta ล่าสุดบนสาขาออกจากไฟล์ SCCS
sact	แสดงสถานะการแก้ไขไฟล์ SCCS ปัจจุบัน
sccs	โปรแกรมดูแลระบบสำหรับระบบ SCCS คำสั่ง **sccs** มีชุดของ pseudo-commands ที่ดำเนินเซอร์วิส SCCS ส่วนใหญ่
sccshelp	อธิบายข้อความแสดงข้อผิดพลาดหรือคำสั่ง SCCS
unget	ยกเลิกผลของการใช้คำสั่ง **get –e** ก่อนหน้า
val	เปลี่ยนไฟล์ SCCS เพื่อดูว่าเช็คซัมที่คำนวณตรงกับเช็คซัม ในรายการในส่วนหัวหรือไม่
vc	แทนที่ค่าที่กำหนดในตำแหน่งของคีย์เวิร์ดที่เหมือนกัน
what	ค้นหาไฟล์ระบบตามรูปแบบและแสดงข้อความตามที่ กำหนด

รูทีนย่อย ตัวอย่างโปรแกรม และไลบรารี

หัวข้อนี้ให้ข้อมูลเกี่ยวกับสิ่งที่รูทีนย่อย เป็น วิธีใช้รูทีนย่อย และที่เก็บรูทีนย่อย

รูทีนย่อยจะเก็บอยู่ในไลบรารีเพื่อเก็บรักษาพื้นที่เก็บข้อมูล และทำให้โปรแกรมเชื่อมต่อกับกระบวนการได้อย่างมีประสิทธิภาพมากขึ้น *ไบรารี* คือไฟล์ข้อมูลที่มีสำเนาของจำนวนไฟล์แต่ละไฟล์ และควบคุมข้อมูลที่อนุญาตให้ไฟล์เหล่านั้นสามารถเข้าถึงได้ ไลบรารีจะอยู่ในไดเร็กทอรี **/usr/ccs/lib** และ **/usr/lib** ตามระเบียบแล้ว ไฟล์ส่วนใหญ่จะมีชื่อที่อยู่ในรูปแบบ **lib***name***.a** โดยที่ *name* จะระบุไลบรารีเฉพาะ

คำสั่ง include ทั้งหมดควรอยู่ใกล้กับบรรทัดแรกที่จะถูกคอมไพล์ ตามปกติแล้วจะอยู่ในส่วนที่มีการประกาศที่อยู่ก่อน **main( )** และต้องเกิดขึ้นก่อนที่จะใช้ฟังก์ชันไลบรารีใดๆ ตัวอย่างเช่น ใช้คำสั่งต่อไปนี้เพื่อสอดแทรกไฟล์ **stdio.h** :

```
#include <stdio.h>
```

คุณไม่จำเป็นต้องทำสิ่งใดเป็นพิเศษเพื่อใช้รูทีนย่อยจากไลบรารี Standard C (**libc.a**) คำสั่ง **cc** จะค้นหาไลบรารีนี้ให้โดยอัตโนมัติ สำหรับรูทีนย่อยที่โปรแกรมต้องการ อย่างไรก็ตาม ถ้าคุณใช้รูทีนย่อยจากไลบรารีอื่น คุณต้องแจ้งให้คอมไพล์เลอร์ทราบเพื่อค้นหาไลบรารีนั่น ถ้าโปรแกรมของคุณใช้รูทีนย่อยจากไลบรารี **lib***name***.a** ให้คอมไพล์โปรแกรมของคุณด้วยแฟล็ก **-l***name* (L ตัวพิมพ์เล็ก) ตัวอย่างต่อไปนี้จะคอมไพล์โปรแกรม `myprog.c` ซึ่งใช้รูทีนย่อยจากไลบรารี **libdbm.a** :

```
cc myprog.c -ldbm
```

คุณสามารถระบุแฟล็ก **-l** (L ตัวพิมพ์เล็ก) ที่มากกว่าหนึ่งแฟล็กได้ แต่ละแฟล็กจะถูกประมวลผลตามลำดับที่ระบุ

ถ้าคุณกำลังใช้รูทีนย่อยที่เก็บอยู่ใน Berkeley Compatibility Library ให้เชื่อมโยงไลบรารี **libbsd.a** *ก่อน* การเชื่อมโยง กับไลบรารี **libc.a** ดังแสดงในตัวอย่างต่อไปนี้:

```
cc myprog.c -lbsd
```

เมื่อข้อผิดพลาดเกิดขึ้น รูทีนย่อยจำนวนมากจะส่งคืนค่า -1 และตั้งค่าตัวแปรภายนอกที่ชื่อ **errno** เพื่อระบุข้อผิดพลาด ไฟล์ **sys/errno.h** จะประกาศตัวแปร **errno** ไว้ และกำหนดค่าคงที่สำหรับเงื่อนไขข้อผิดพลาดที่อาจเป็นไปได้แต่ละครั้ง

ในเอกสารคู่มือนี้ รูทีนทั้งหมดที่ระบบเรียกจะถูกอธิบายไว้เป็น *รูทีนย่อย* และจะถูกแก้ปัญหาจากไลบรารี **libc.a** โปรแกรมมิงที่อินเตอร์เฟสกับที่ระบบเรียกจะเป็นการเรียกเฉพาะสำหรับรูทีนย่อยนั้น ตราบที่โปรแกรม C Language ยังคงเกี่ยวข้องอยู่ การเรียกของระบบจะยังคงเป็นการเรียกรูทีนย่อย ข้อแตกต่างที่แท้จริงระหว่างการเรียกของระบบ และรูทีนย่อยคือ ชนิดของการดำเนินการที่โปรแกรมดำเนินการ เมื่อโปรแกรมเรียกใช้งานการเรียกของระบบ การสับเปลี่ยนการป้องกันโดเมนจะเข้าแทนที่ ดังนั้น รูทีนที่เรียกจะมีสิทธิเข้าถึงข้อมูลพิเศษของเคอร์เนลระบบ ดังนั้น รูทีนจะดำเนินการในโหมดเคอร์เนล เพื่อดำเนินการภารกิจในฐานะของโปรแกรม ด้วยวิธีนี้ การเข้าถึงข้อมูลระบบพิเศษจะถูกจำกัดสำหรับชุดของรูทีนที่ได้ถูกกำหนดไว้ก่อน ซึ่งมีการดำเนินการที่สามารถควบคุมได้

หมายเหตุ:

1. รายการต่อไปนี้จะแสดงถึงรูทีน wString ที่ล้าสมัยแล้วสำหรับ **libc.a** แบบ 64 บิต ซึ่งรวมถึงรูทีนที่เทียบเท่ากับ **libc.a** แบบ 64 บิต รูทีนสำหรับ **libc.a** แบบ 32 บิตสามารถค้นหาได้ในรูทีนย่อย wstring

32 บิตเท่านั้น	เทียบเท่ากับ 64 บิต
wstrcat	wcscat
wstrchr	wcschr
wstrcmp	wcscoll
wstrcpy	wcscpy
wstrcspn	wcscspn
wstrdup	Not available and has no equivalents in the 64 bit libc.a
wstrlen	wcslen
wstrncat	wcsncat

wstrncpy	wcsncpy
wstrpbrk	wcspbrk
wstrrchr	wcsrchr
wstrspn	wcsspn
wstrtok	wcstok

2. โปรแกรมทั้งหมดที่จัดการกับอักขระแบบมัลติไบต์ อักขระ wide หรือข้อมูลที่ระบุเฉพาะสำหรับโลแคล ต้องเรียกรูทีนย่อย **setlocale** ที่จุดเริ่มต้นของโปรแกรม
3. โปรแกรมมิงในสภาพแวดล้อม multithreaded ต้องการรูทีนย่อย reentrant เพื่อ รับประกัน data integrity

หลักการที่เกี่ยวข้อง:

"เครื่องมือและยูทิลีตี" ในหน้า 2
ส่วนนี้ให้ภาพรวมของเครื่องมือและยูทิลิตีที่คุณสามารถใช้เพื่อพัฒนาโปรแกรมภาษาที่คอมไพล์ด้วย C

ข้อมูลที่เกี่ยวข้อง:

รายการของเซอร์วิสการจัดการข้อมูเวลา

ภาพรวมไฟล์ส่วนหัว

itrunc

printf

scanf

setlocate

sqrt

ชนิดข้อมูล long double floating-point 128 บิต

ระบบปฏิบัติการ AIX สนับสนุนชนิดข้อมูล 128 บิต long double ที่ให้ความละเอียดสูงกว่าชนิดข้อมูลดีฟอลต์ 64 บิต long double ชนิดข้อมูล 128 บิต สามารถจัดการได้ถึง 31 ดิจิตสำคัญ (เทียบกับ 17 ที่จัดการโดย 64 บิต long double) อย่างไรก็ตาม ในขณะที่ ชนิดข้อมูลนี้สามารถเก็บค่าตัวเลขได้แม่นยำมากกว่าชนิดข้อมูล 64 บิต ไม่เก็บตัวเลขที่มีขนาดใหญ่กว่า

ประเด็นพิเศษต่อไปนี้มีผลกับการใช้ ชนิดข้อมูล long double 128 บิต :

- การคอมไพล์โปรแกรมที่ใช้ชนิดข้อมูล long double 128 บิต
- เป็นไปตามมาตรฐาน IEEE 754
- การนำใช้รูปแบบ long double 128 บิต
- ค่าของแมโครตัวเลข

การคอมไพล์โปรแกรมที่ใช้ชนิดข้อมูล long double 128 บิต

ในการคอมไพล์โปรแกรมภาษา C ที่ใช้ชนิดข้อมูล long double 128 บิต ให้ใช้คำสั่ง **xlc128** คำสั่งนี้เป็น alias ของคำสั่ง **xlc** ที่มีการสนับสนุนสำหรับชนิดข้อมูล 128 บิต คำสั่ง **xlc** สนับสนุนเฉพาะชนิดข้อมูล long double 64 บิต เท่านั้น

ไลบรารีภาษา C มาตรฐาน **libc.a** จัดให้มีการแทนรูทีน **libc.a** ซึ่งมีความอ่อนไหว ต่อขนาดของ long double ลิงก์กับไลบรารี **libc.a** เมื่อทำการคอมไพล์แอ็พพลิเคชันที่ใช้ชนิดข้อมูล long double 64 บิต ลิงก์แอ็พพลิเคชันที่ใช้ค่า long double 128 บิต กับไลบรารี **libc128.a** และ **libc.a** ทั้งสอง เมื่อทำการลิงก์ ขอให้แน่ใจว่าได้ระบุไลบรารี **libc128.a** ก่อนไลบรารี **libc.a** ในลำดับการค้นหาไลบรารี

เป็นไปตามมาตรฐาน IEEE 754

การใช้ 64 บิต ของชนิดข้อมูล long double นั้นเข้ากันได้โดยสมบูรณ์กับมาตรฐาน IEEE 754 แต่การใช้ 128 บิต เข้ากันไม่ได้ ใช้การใช้ 64 บิต ในแอ็พพลิเคชันที่ต้องยึดตาม มาตรฐาน IEEE 754

การใช้ 128 บิต ต่างจากมาตรฐาน IEEE สำหรับ long double ในเรื่องต่อไปนี้:

- สนับสนุนเฉพาะโหมด round-to-nearest ถ้า แอ็พพลิเคชันเปลี่ยนโหมดการปัดเศษ ผลลัพธ์ไม่ถูกกำหนด
- ไม่สนับสนุนตัวเลขพิเศษ IEEE โดยสมบูรณ์ NaN และ INF
- ไม่สนับสนุนแฟล็กสถานะ IEEE สำหรับโอเวอร์โฟลว์ อันเดอร์โฟลว์ และเงื่อนไขอื่นๆ แฟล็กเหล่านี้ไม่มีความหมายสำหรับการใช้ long double 128 บิต
- ชนิดข้อมูล long double 128 บิตไม่สนับสนุน API คณิตศาสตร์ต่อไปนี้: **atanhl, cbrtl, copysignl, exp2l, expm1l, fdiml, fmal, fmaxl, fminl, hypotl, ilogbl, llrintl, llroundl, log1pl, log2l, logbl, lrintl, lroundl, nanl, nearbyintl, nextafterl, nexttoward, nexttowardf, nexttowardl, remainderl, remquol, rintl, roundl, scalblnl, scalbnl, tgammal** และ **truncl**

การนำรูปแบบ long double 128 บิตไปใช้งาน

ตัวเลข long double 128 บิตประกอบด้วยคู่ของตัวเลข double-precision 64 บิต ที่จัดลำดับ สมาชิกแรกของคู่ที่จัดลำดับ มีส่วน high-order ของตัวเลข และสมาชิกที่สองมีส่วน low-order ค่าของจำนวน long double คือผลรวมของ จำนวน 64 บิต สอง จำนวน

แต่ละชุดของจำนวน 64 บิต สองจำนวนคือจำนวน double-precision floating-point ที่มีเครื่องหมาย เลขชี้กำลัง และ significand โดยทั่วไป สมาชิก low-order มี magnitude ที่น้อยกว่า 0.5 หน่วยในตำแหน่งสุดท้าย ของส่วน high ดังนั้นค่าของ จำนวน 64 บิต สองจำนวนจะไม่ซ้อนกัน และ ทั้ง significand ของจำนวน low-order จะเพิ่มตำแหน่งความสำคัญห่างจาก จำนวน high-order

การแทนนี้ส่งผลให้เกิดหลายประเด็นที่ ต้องพิจารณาในการใช้ตัวเลขเหล่านี้:

- ช่วงของเลขชี้กำลังจะเหมือนกันของ double precision แม้ว่าความถูกต้องมีค่ามากกว่า magnitude ของจำนวนที่แสดงได้ จะเหมือนกับ double precision 64 บิต
- เมื่อค่าสัมบูรณ์ของ magnitude ลดลง (เข้าใกล้ช่วงการดีนอร์มัล) ความถูกต้องเพิ่มที่มีอยู่ในส่วน low-order จะลดลงเช่นกัน เมื่อค่าที่จะแสดงอยู่ในช่องดีนอร์มัล ก
- จำนวนบิตแท้จริงของ precision อาจ แตกต่างกัน ถ้าส่วน low-order น้อยกว่า 1 ULP ของส่วน high-order บิต significant (ที่เป็น 0 ทั้งหมดหรือ 1 ทั้งหมด) จะถูกแสดงเป็นนัยระหว่าง significant ของ ตัวเลข high-order และ low-order อัลกอริธึมเฉพาะที่ต้องขึ้นกับจำนวนบิตใน significand แบบคงที่อาจล้มเหลวเมื่อใช้จำนวน long double 128 บิต

ค่าของแมโครตัวเลข

เนื่องจากวิธีของหน่วยเก็บข้อมูลสำหรับชนิดข้อมูล long double ที่มากกว่า หนึ่งจำนวนอาจเป็นค่าที่แน่นอนที่พร้อมใช้งาน เป็นแมโคร การแทนจำนวน long double 128 บิตหมายถึงแมโครต่อไปนี้ที่ต้องการโดย C มาตรฐานในไฟล์ **values.h** จะไม่มีความหมายที่ชัดเจน:

- จำนวนบิตใน mantissa (**LDBL_MANT_DIG**)
- Epsilon (**LBDL_EPSILON**)
- ค่าจำกัดที่แสดงได้สูงสุด (**LDBL_MAX**)

จำนวนบิตใน mantissa

จำนวนบิตใน significand ไม่เป็นค่าคงที่ แต่สำหรับจำนวนที่จัดรูปแบบอย่างถูกต้อง (ยกเว้นในช่วงดีนอร์มัล) จำนวน สูงสุดที่มีคือ 106 ดังนั้น ค่าของแมโคร **LDBL_MANT_DIG** คือ 106

Epsilon

ANSI C standard กำหนดค่าของ epsilon เป็น ส่วนต่างระหว่าง 1.0 และค่าที่แสดงได้อย่างน้อยที่สุดที่มากกว่า 1.0 นั่นคือ *b*** (1–*p*) โดยที่ *b* คือฐาน (2) และ *p* คือ จำนวน *b* ดิจิตที่เป็นฐานในจำนวน นิยามนี้ ต้องการให้จำนวนของดิจิต *b* ฐานคงที่ ซึ่งไม่เป็น จริงสำหรับจำนวน long double 128 บิต

ค่าที่แทนได้ที่เล็กที่สุดที่มากกว่า 1.0 คือจำนวนนี้:

```
0x3FF0000000000000, 0x0000000000000001
```

ความแตกต่างระหว่างค่านี้กับ 1.0 คือ จำนวนนี้:

```
0x0000000000000001, 0x0000000000000000
0.4940656458412465441765687928682213E-323
```

เนื่องจากโดยปกติจำนวน 128 บิต จะมีอย่างน้อย 106 บิตความแม่นยำ ค่าต่ำสุดที่เหมาะสมสำหรับ *p* คือ 106 ดังนั้น *b*** (1–*p*) และ 2**(–105) จะได้ค่านี้:

```
0x3960000000000000, 0x0000000000000000
0.2465190328815661891911651766508707 0E-31
```

ทั้งสองค่าเป็นไปตามนิยามของ epsilon ที่เป็นไปตามมาตรฐาน c รูทีนย่อย long double ใช้ค่าที่สองเพราะแสดงลักษณะความแม่นยำที่จัดหาโดยการใช้ 128 บิตได้ดีกว่า

ค่า long double สูงสุด

ค่าของแมโคร **LDBL_MAX** เป็น ตัวเลข 128 บิต long double ที่ใหญ่ที่สุดซึ่งสามารถถูกคูณด้วย 1.0 และ ให้ผลตัวเลขดั้งเดิม ค่านี้ยังเป็นค่าที่มีขอบเขตใหญ่ที่สุดที่ ถูกสร้างได้โดยการดำเนินการ primitive เช่นการคูณและการหาร:

```
0x7FEFFFFFFFFFFFFF, 0x7C8FFFFFFFFFFFFF
0.17976931348623159077293051907890025 75e+309
```

รายการของรูทีนย่อยการจัดการอักขระ

ฟังก์ชันและแมโครการจัดการอักขระทดสอบ และแปลอักขระ ASCII

ฟังก์ชันและแมโครเหล่านี้มีสามประเภท:

- การทดสอบอักขระ
- การแปลอักขระ
- การจัดการอักขระเบ็ตเตล็ด

Character Testing จะแสดง บางรูทีนย่อยการจัดการอักขระ

การทดสอบอักขระ

ใช้ฟังก์ชันและแมโครดังต่อไปนี้เพื่อระบุ ชนิดอักขระ เครื่องหมายวรรคตอน ตัวหนังสือ และค่าฟังก์ชัน case-querying ขึ้นกับตารางการเทียบกันปัจจุบัน

รูทีนย่อย **ctype** มีฟังก์ชันดังต่อไปนี้:

`isalpha`
: อักขระเป็นตัวหนังสือ?

`isalnum`
: อักขระเป็นตัวหนังสือและตัวเลข?

`isupper`
: อักขระเป็นตัวพิมพ์ใหญ่?

`islower`
: อักขระเป็นตัวพิมพ์เล็ก?

`isdigit`
: อักขระเป็นดิจิต?

`isxdigit`
: อักขระเป็นเลขฐานสิบหก?

`isspace`
: อักขระเป็นอักขระช่องว่าง?

`ispunct`
: อักขระเป็นเครื่องหมายวรรคตอน?

`isprint`
: อักขระเป็นอักขระการพิมพ์ รวมทั้งพื้นที่?

`isgraph`
: อักขระเป็นอักขระการพิมพ์ ที่แยกช่องว่าง?

`iscntrl`
: อักขระเป็นอักขระควบคุม?

`isascii`
: อักขระเป็นอักขระจำนวนเต็ม ASCII?

การแปลอักขระ

รูทีนย่อย **conv** มีฟังก์ชันดังต่อไปนี้:

`toupper`
: แปลงตัวอักษรตัวพิมพ์เล็กเป็นตัวพิมพ์ใหญ่

`_toupper`
: (แมโคร) แปลงอักษรตัวพิมพ์เล็กเป็นตัวพิมพ์ใหญ่

tolower
แปลงตัวอักษรตัวพิมพ์ใหญ่เป็นตัวพิมพ์เล็ก

_tolower
(แมโคร) แปลงอักษรตัวพิมพ์ใหญ่เป็นตัวพิมพ์เล็ก

toascii
แปลงจำนวนเต็มเป็นอักขระ ASCII

การจัดการอักขระเบ็ตเตล็ด

getc, fgetc,getchar, getw
รับอักขระหรือคำจากสตรีมอินพุต

putc,putchar, fputc, putw
เขียนอักขระหรือคำลงในสตรีม

รายการของรูทีนย่อยการสร้างโปรแกรมที่สามารถเรียกทำงานได้

รายการของเซอร์วิสการสร้างโปรแกรมเรียกทำงานประกอบด้วย รูทีนย่อยที่สนับสนุนกลุ่มคำสั่ง

คำสั่งและรูทีนย่อยเหล่านี้ อนุญาตให้คุณสร้าง คอมไพล์ และทำงานกับไฟล์เพื่อทำให้โปรแกรมของคุณ ทำงาน

รูทีนย่อย	คำอธิบาย
_end, _text, _edata	กำหนดตำแหน่งสุดท้ายของโปรแกรม
confstr	ระบุค่าปัจจุบันของตัวแปรระบบที่ระบุที่กำหนด เป็นสตริง
getopt	รับตัวอักษรแฟล็กจากอาร์กิวเมนต์เวกเตอร์
ldopen, ldaopen	เปิดอ็อบเจ็กต์ไฟล์ทั่วไป
ldclose, ldaclose	ปิดอ็อบเจ็กต์ไฟล์ทั่วไป
ldahread	อ่านส่วนหัวการเก็บถาวรของไฟล์เก็บถาวร
ldfhread	อ่านส่วนหัวไฟล์ของอ็อบเจ็กต์ไฟล์ทั่วไป
ldlread, ldlinit, ldlitem	อ่านและจัดการรายการตัวเลขบรรทัดของฟังก์ชันอ็อบเจ็กต์ไฟล์ทั่วไป
ldshread, ldnshread	อ่านส่วนหัวของอ็อบเจ็กต์ไฟล์ทั่วไป
ldtbread	อ่านรายการตารางสัญลักษณ์ของอ็อบเจ็กต์ไฟล์ทั่วไป
ldgetname	เรียกคืนชื่อสัญลักษณ์จากรายการตารางสัญลักษณ์หรือจาก ตารางสตริง
ldlseek, ldnseek	ค้นหารายการหมายเลขบรรทัดของส่วนของอ็อบเจ็กต์ไฟล์ทั่วไป
ldohseek	ค้นหาส่วนหัวไฟล์ทางเลือกของอ็อบเจ็กต์ไฟล์ทั่วไป
ldrseek, ldnrseek	ค้นหาข้อมูลการเปลี่ยนตำแหน่งสำหรับส่วนของอ็อบเจ็กต์ไฟล์ ทั่วไป
ldsseek, ldnsseek	ค้นหาส่วนของอ็อบเจ็กต์ไฟล์ทั่วไป
ldtbseek	ค้นหาตารางสัญลักษณ์ของอ็อบเจ็กต์ไฟล์ทั่วไป
ldtbindex	ส่งกลับดัชนีของรายการตารางสัญลักษณ์ อ็อบเจ็กต์ไฟล์ทั่วไป
load	โหลดและเชื่อมอ็อบเจ็กต์โมดูลลงในกระบวนการปัจจุบัน
unload	ยกเลิกการโหลดอ็อบเจ็กต์ไฟล์
loadbind	จัดเตรียมการแก้ปัญหารันไทม์จำเพาะของสัญลักษณ์เลื่อนเวลาของโมดูล
loadquery	ส่งกลับข้อมูลข้อผิดพลาดจากรูทีนย่อย **load** หรือรูทีนย่อย **exec** และจัดเตรียมรายการของอ็อบเจ็กต์ไฟล์ ที่โหลดสำหรับกระบวนการปัจจุบัน
monitor	เริ่มและหยุดการทำงานการทำโปรไฟล์กระทำการ
nlist	รับรายการจากรายการชื่อ
regcmp, regex	คอมไพล์และจับคู่รูปแบบนิพจน์ทั่วไป
setjmp, longjmp	เก็บตำแหน่ง
sgetl, sputl	เข้าถึงข้อมูลตัวเลข long ในแบบไม่ขึ้นกับเครื่อง
sysconf	ระบุค่าปัจจุบันของข้อจำกัดระบบหรือตัวเลือกที่ระบุ

รายการของไฟล์และรูทีนย่อยไดเร็กทอรี

ระบบจัดเตรียม เซอร์วิสในการสร้างไฟล์ ย้ายข้อมูลเข้าออกไฟล์ และอธิบายข้อบังคับ และโครงสร้างของระบบไฟล์

รูทีนย่อยจำนวนมากเหล่านี้เป็นพื้นฐาน สำหรับคำสั่งระบบที่มีชื่อเหมือนกัน อย่างไรก็ตามคุณสามารถใช้รูทีนย่อย เหล่านี้เพื่อเขียนคำสั่งหรือยูทิลิตีใหม่เพื่อช่วยในกระบวนการ พัฒนาโปรแกรม หรือเพื่อรวมไว้ในแอ็พพลิเคชันโปรแกรม

ระบบจัดเตรียมรูทีนย่อยสำหรับ:

การควบคุมไฟล์

`access, accessx หรือ faccessx`
กำหนดความสามารถในการเข้าถึงของไฟล์

`fclear`
ล้างพื้นที่ในไฟล์

`fcntl, dup หรือ dup2`
เปิดไฟล์ descriptors ที่เปิด

`fsync`
เขียนการเปลี่ยนแปลงในไฟล์ไปที่พื้นที่จัดเก็บข้อมูลถาวร

`getenv`
ส่งกลับค่าของตัวแปรสภาวะแวดล้อม

`getutent, getutid, getutline, putuline, setutent, endutent หรือ utmpname`
เข้าถึงรายการไฟล์ utmp

`getutid_r, getutline_r, pututline_r, setutent_r, endutent_r หรือ utmpname_r`
เข้าถึงรายการไฟล์ utmp

`lseek หรือ llseek`
ย้ายตัวชี้ read-write ในไฟล์ที่เปิด

`lockfx, lockf หรือ flock`
ควบคุมการล็อกไฟล์ descriptor ที่เปิด

`mknod หรือ mkfifo`
สร้างไฟล์ regular, FIFO หรือ special

`mktemp หรือ mkstemp`
สร้างชื่อไฟล์เฉพาะ

`open, openx หรือ creat`
ส่งกลับไฟล์ descriptor และสร้างไฟล์

`pclose`
ปิดไพพ์ที่เปิด

`pipe`
สร้างแชนเนลระหว่างกระบวนการ

popen
กำหนดค่าเริ่มต้นไพพ์ให้กับกระบวนการ

pathconf, fpathconf
เรียกข้อมูลคุณลักษณะการนำไฟล์ไปใช้

putenv
ตั้งค่าตัวแปรสภาวะแวดล้อม

read, readx, readv, readvx
อ่านจากไฟล์หรืออุปกรณ์

rename
เปลี่ยนชื่อไดเร็กทอรีหรือไฟล์ภายในระบบไฟล์

statx, stat, fstatx, fstat, fullstat, fullstat
รับสถานะไฟล์

tmpfile
สร้างไฟล์ชั่วคราว

tmpnam หรือ tempnam
สร้างชื่อสำหรับไฟล์ชั่วคราว

truncate, ftruncate
ทำให้ไฟล์สั้นลง

umask
รับและตั้งค่าค่าของมาสก์การสร้างไฟล์

utimes หรือ utime
ตั้งค่าเวลาการเข้าถึงหรือปรับเปลี่ยนไฟล์

write, writex, writev, writevx
เขียนไปที่ไฟล์หรืออุปกรณ์

การทำงานกับไดเร็กทอรี

chdir
เปลี่ยนไดเร็กทอรีทำงานปัจจุบัน

chroot
เปลี่ยนไดเร็กทอรี root ที่มีประสิทธิผล

getwd, getcwd
รับชื่อพาธไดเร็กทอรีปัจจุบัน

glob
สร้างรายการของชื่อพาธกับไฟล์ที่สามารถเข้าถึงได้

globfree
ฟรีหน่วยความจำทั้งหมดที่สัมพันธ์กับพารามิเตอร์ *pglob*

link
สร้างรายการไดเร็กทอรีเพิ่มเติมสำหรับไฟล์ที่มีอยู่

mkdir
สร้างไดเร็กทอรี

opendir, readdir, telldir, seekdir, rewinddir, closedir
การดำเนินการบนไดเร็กทอรี

readdir_r
อ่านไดเร็กทอรี

rmdir
เอาไดเร็กทอรีออก

scandir, alphasort
การสแกนไดเร็กทอรี

readlink
อ่านวอลุ่มของการเชื่อมโยงสัญลักษณ์

remove
ทำให้ไฟล์เข้าถึงไม่ได้ตามชื่อที่ระบุ

symlink
สร้างการเชื่อมโยงสัญลักษณ์ไปที่ไฟล์

unlink
เอารายการไดเร็กทอรีออก

การจัดการกับระบบไฟล์

confstr
ระบุค่าปัจจุบันของตัวแปรระบบที่ระบุที่กำหนด โดยสตริง

fscntl
จัดการการดำเนินการควบคุมระบบไฟล์

getfsent, getfsspec, getfsfile, getfstype, setfsent หรือ endfsent
รับข้อมูลเกี่ยวกับระบบไฟล์

getvfsent, getvfsbytype, getvfsbyname, getvfsbyflag, setvfsent, endvfsent
รับข้อมูลเกี่ยวกับรายการระบบไฟล์เสมือน

mnctl
ส่งกลับข้อมูลสถานะการเชื่อมต่อ

quotactl
จัดการโควต้าดิสก์

statfs, fstatfs
รับสถานะของระบบไฟล์ของไฟล์

`sysconf`
รายงานค่าปัจจุบันของข้อจำกัดหรือตัวเลือกระบบ

`sync`
อัพเดตข้อมูลระบบไฟล์ทั้งหมดลงดิสก์

`umask`
รับและตั้งค่าค่าของมาสก์การสร้างไฟล์

`vmount`
เชื่อมต่อระบบไฟล์

`umount, uvmount`
เอาระบบไฟล์เสมือนออกจากทรีไฟล์

รายการ FORTRAN BLAS ระดับ 1: รูทีนย่อย Vector-vector

Level 1: รูทีนย่อย vector-vector รวมถึง:

รูทีนย่อย	คำอธิบาย
SDOT, DDOT	ส่งกลับผลคูณ dot ของสองเวกเตอร์
CDOTC, ZDOTC	ส่งกลับผลคูณ complex dot ของสองเวกเตอร์ คอนจูเกตค่าแรก
CDOTU, ZDOTU	ส่งกลับผลคูณ complex dot ของสองเวกเตอร์
SAXPY, DAXPY, CAXPY, ZAXPY	ส่งกลับค่าคงที่คูณเวกเตอร์บวกหนึ่งเวกเตอร์
SROTG, DROTG, CROTG, ZROTG	สร้าง Givens plane rotation
SROT, DROT, CSROT, ZDROT	ใช้ plane rotation
SCOPY, DCOPY, CCOPY, ZCOPY	คัดลอกเวกเตอร์ *X* ไปที่ *Y*
SSWAP, DSWAP, CSWAP, ZSWAP	แลกเปลี่ยนเวกเตอร์ *X* และ *Y*
SNRM2, DNRM2, SCNRM2, DZNRM2	ส่งกลับ Euclidean norm ของ *N*-vector ที่เก็บใน *X*() โดยการเพิ่มพื้นที่จัดเก็บข้อมูล *INCX*
SASUM, DASUM, SCASUM, DZASUM	ส่งครับผลรวมของค่าสัมบูรณ์ของคอมโพเนนต์เวกเตอร์
SSCAL, DSCAL, CSSCAL, CSCAL, ZDSCAL, ZSCAL	สเกลเวกเตอร์ตามค่าคงที่
ISAMAX, IDAMAX, ICAMAX, IZAMAX	ค้นหาดัชนีของอิลิเมนต์ที่มีค่าสัมบูรณ์สูงสุด
SDSDOT	ส่งกลับผลคูณ dot ของสองเวกเตอร์บวกค่าคงที่
SROTM, DROTM	ใช้การแปลงสภาพ Givens ที่ปรับเปลี่ยน
SROTMG, DROTMG	สร้างการแปลงสภาพ Givens ที่ปรับเปลี่ยน

รายการ FORTRAN BLAS ระดับ 2: รูทีนย่อย Matrix-vector

Level 2: รูทีนย่อย matrix-vector รวมถึง:

รูทีนย่อย	คำอธิบาย
SGEMV, DGEMV,CGEMV, ZGEMV	ดำเนินการ matrix-vector กับเมทริกทั่วไป
SGBMV, DGBMV, CGBMV, ZGBMV	ดำเนินการ matrix-vector กับเมทริกที่ถูกแบนด์ทั่วไป
CHEMV, ZHEMV	ดำเนินการ matrix-vector โดยใช้เมทริก Hermitian
CHBMV, ZHBMV	ดำเนินการ matrix-vector โดยใเมทริกแบนด์ Hermitian
CHPMV,ZHPMV	ดำเนินการ matrix-vector โดยใช้เมทริก Hermitian ที่ถูกแพ็ก
SSYMV , DSYMV	ดำเนินการ matrix-vector โดยใช้เมทริกแบบสมมาตร
SSBMV , DSBMV	ดำเนินการ matrix-vector โดยใช้เมทริกแบนด์แบบสมมาตร
SSPMV , DSPMV	ดำเนินการ matrix-vector โดยใช้เมทริกแบบสมมาตรที่ถูกแพ็ก
STRMV, DTRMV, CTRMV, ZTRMV	ดำเนินการ matrix-vector โดยใช้เมทริกสามเหลี่ยม
STBMV, DTBMV, CTBMV, ZTBMV	ดำเนินการ matrix-vector โดยใช้เมทริกแบนด์สามเหลี่ยม
STPMV, DTPMV, CTPMV, ZTPMV	ดำเนินการ matrix-vector บนเมทริกสามเหลี่ยมที่ถูกแพ็ก
STRSV, DTRSV, CTRSV, ZTRSV	แก้ปัญหาระบบของสมการ

รูทีนย่อย	คำอธิบาย
STBSV, DTBSV, CTBSV, ZTBSV	แก้ปัญหาระบบของสมการ
STPSV, DTPSV, CTPSV, ZTPSV	แก้ปัญหาระบบของสมการ
SGER, DGER	ดำเนินการ rank 1
CGERU, ZGERU	ดำเนินการ rank 1
CGERC,ZGERC	ดำเนินการ rank 1
CHER, ZHER	ดำเนินการ Hermitian rank 1
CHPR,ZHPR	ดำเนินการ Hermitian rank 1
CHPR2,ZHPR2	ดำเนินการ Hermitian rank 2
SSYR, DSYR	ดำเนินการ rank 1 แบบสมมาตร
SSPR, DSPR	ดำเนินการ rank 1 แบบสมมาตร
SSYR2 , DSYR2	ดำเนินการ rank 2 แบบสมมาตร
SSPR2 ,DSPR2	ดำเนินการ rank 2 แบบสมมาตร

รายการ FORTRAN BLAS ระดับ 3: รูทีนย่อย Matrix-matrix

Level 3: รูทีนย่อย matrix–matrix รวมถึง:

รูทีนย่อย	คำอธิบาย
SGEMM, DGEMM, CGEMM, ZGEMM	ดำเนินการ matrix–matrix บนเมทริกทั่วไป
SSYMM, DSYMM,CSYMM, ZSYMM	ดำเนินการ matrix–matrix บนเมทริกที่สมมาตร
CHEMM,ZHEMM	ดำเนินการ matrix–matrix บนเมทริก Hermitian
SSYRK, DSYRK,CSYRK, ZSYRK	ดำเนินการ rank k ที่สมมาตร
CHERK, ZHERK	ดำเนินการ Hermitian rank k
SSYR2K, DSYR2K, CSYR2K, ZSYR2K	ดำเนินการ rank 2k ที่สมมาตร
CHER2K,ZHER2K	ดำเนินการ Hermitian rank 2k
STRMM, DTRMM,CTRMM, ZTRMM,	ดำเนินการ matrix–matrix บนเมทริกสามเหลี่ยม
STRSM, DTRSM, CTRSM, ZTRSM	แก้ปัญหาสมการเมทริกบางสมการ

รายการของรูทีนย่อยการจัดการกับตัวเลข

ฟังก์ชันเหล่านี้ดำเนินการจัดการเชิงตัวเลข:

ฟังก์ชัน	คำอธิบาย
a64l, l64a	แปลงค่าระหว่าง long integers และสตริง base–64 ASCII
abs, div, labs, ldiv, imul_dbl, umul_dbl, llabs, lldiv	คำนวณค่าสัมบูรณ์ การหาร และการคูณ จำนวนเต็ม
asin, asinl, acos, acosl, atan, atanl, atan2, atan2l	ฟังก์ชันคำนวณตรีโกณมิติผกผัน
asinh, acosh, atanh	คำนวณฟังก์ชันไฮเปอร์โบลิกผกผัน
atof, atoff, strtod, strtold, strtof	แปลงสตริง ASCII เป็นตัวเลข floating point
bessel: j0, j1, jn, y0, y1, yn	คำนวณฟังก์ชัน bessel
class, finite, isnan, unordered	ระบุชนิดของฟังก์ชัน floating point
copysign, nextafter, scalb, logb, ilogb	คำนวณฟังก์ชัน floating–point ไบนารี
nrand48, mrand48, jrand48, srand48, seed48, lcong48	สร้างลำดับ pseudo–random
lrand48_r, mrand48_r, nrand48_r, seed48_r, or **srand48_r**	สร้างลำดับ pseudo–random
drem หรือ **remainder**	คำนวณ IEEE remainder
ecvt, fcvt, gcvt	แปลงตัวเลข floating–point เป็นสตริง
erf, erfl, erfc, erfcl	ฟังก์ชันข้อผิดพลาดในการคำนวณและ complementary error
exp, expl, expm1, log, logl, log10, log10l, log1p, pow, powl	ฟังก์ชันคำนวณ การยกกำลัง log และ power
floor, floorl, ceil, ceill, nearest, **trunc, rint, itrunc, uitrunc, fmod, fmodl, fabs, fabsl**	ปัดตัวเลข floating–point
fp_any_enable, fp_is_enabled, fp_enable_all, **fp_enable, fp_disable_all, fp_disable**	อนุญาตการดำเนินการกับสถานะ floating–point exception
fp_clr_flag, fp_set_flag, fp_read_flag, หรือ **fp_swap_flag**	อนุญาตการดำเนินการกับสถานะ floating–point exception
fp_invalid_op, fp_divbyzero, fp_overflow, **fp_underflow, fp_inexact, fp_any_xcp**	ทดสอบเพื่อดูว่า floating–point exception ได้เกิดขึ้นหรือไม่
fp_iop_snan, fp_iop_infsinf, fp_iop_infdinf,	

ฟังก์ชัน	คำอธิบาย
fp_iop_zrdzr, fp_iop_infmzr, fp_iop_invcmp	ทดสอบเพื่อดูว่า floating-point exception ได้เกิดขึ้นหรือไม่
fp_read_rnd,fp_swap_rnd	อ่านและตั้งค่าโหมดการปัดค่า IEEE
frexp, frexpl, ldexp, ldexpl, modf, modfl	จัดการตัวเลข floating point
l64a_r	แปลงจำนวนเต็ม base-64 long เป็นสตริง
lgamma, lgammal, gamma	คำนวณลอการิทึมของฟังก์ชัน gamma
hypot, cabs	คำนวณฟังก์ชันระยะ Euclidean และค่าสัมบูรณ์
l3tol, ltol3	แปลงค่าระหว่าง 3-ไบต์ integers และ long integers
madd, msub, mult, mdiv, pow, gcd, invert,	
rpow, msqrt, mcmp, move, min, omin,	
fmin, m_in, mout, omout, fmout, m_out, sdiv, itom	จัดเตรียมคณิตศาสตร์จำนวนเต็มที่ความละเอียดหลายตำแหน่ง
rand, srand	สร้างตัวเลขสุ่ม
rand_r	สร้างตัวเลขสุ่ม
random, srandom, initstate, setstate	สร้างตัวเลขสุ่มที่ดีกว่า
rsqrt	คำนวณ reciprocal ของสแควร์รูทของตัวเลข
sin, cos, tan	คำนวณฟังก์ชันตรีโกณมิติและตรีโกณมิติผกผัน
sinh, sinhl, cosh, coshl, tanh, tanhl	คำนวณฟังก์ชันไฮเปอร์โบลิก
sqrt, sqrtl, cbrt	คำนวณฟังก์ชันสแควร์รูทและคิวบ์รูท
strtol, strtoll, strtoul, strtoull, atol, atoi	แปลงสตริงเป็นจำนวนเต็ม

รายการของรูทีนย่อยการจัดการตัวเลขจำนวนเต็ม long long

รูทีนย่อยต่อไปนี้ทำหน้าที่ในการจัดการตัวเลข ของเลขจำนวนเต็มที่เก็บในรูปแบบข้อมูล long long integer:

รูทีนย่อย	คำอธิบาย
llabs	คำนวณค่าสัมบูรณ์ของ long long integer
lldiv	คำนวณหาผลหารและเศษเหลือของการหาร long long integers สองจำนวน
strtoll	แปลงสตริงเป็น long long integer ที่มีเครื่องหมาย
strtoull	แปลงสตริงเป็น long long integer ที่ไม่มีเครื่องหมาย
wcstoll	แปลงสตริง wide character เป็น long long integer ที่มีเครื่องหมาย
wcstoull	แปลงสตริง wide character เป็น long long integer ที่ไม่มีเครื่องหมาย

รายการของรูทีนย่อยการจัดการตัวเลข long double 128 บิต

รูทีนย่อยดังต่อไปนี้ดำเนินการจัดการเชิงตัวเลขของ ตัวเลข floating-point ที่เก็บในชนิดข้อมูล long double 128-บิต

รูทีนย่อย เหล่านี้ไม่สนับสนุนชนิดข้อมูล long double 64-บิต แอ็พพลิเคชัน ที่ใช้ชนิดข้อมูล long double 64-บิต ควรใช้รูทีนย่อย ความเที่ยงตรงสองเท่าที่ตรงกัน

รูทีนย่อย	คำอธิบาย
acosl	คำนวณ cosine ผกผันของตัวเลข floating-point ในรูปแบบ long double
asinl	คำนวณ sine ผกผันของตัวเลข floating-point ในรูปแบบ long double
atan2l	คำนวณค่า principal ของ arc tangent ของ *x*/*y* ซึ่งคอมโพเนนต์ถูกแสดงในรูปแบบ long double
atanl	คำนวณ tangent ผกผันของตัวเลข floating-point ในรูปแบบ long double
ceill	คำนวณค่าอินทริกรัลน้อยที่สุดที่ไม่น้อยกว่าตัวเลข floating-point ที่ระบุ ในรูปแบบ long double
coshl	คำนวณไฮเปอร์โบลิก cosine ของตัวเลข floating-point ในรูปแบบ long double
cosl	คำนวณ cosine ของตัวเลข floating-point ในรูปแบบ long double
erfcl	คำนวณค่า 1 ลบด้วยฟังก์ชันข้อผิดพลาดของตัวเลข floating-point ในรูปแบบ long double
erfl	คำนวณฟังก์ชันข้อผิดพลาดของตัวเลข floating-point ในรูปแบบ long double
expl	คำนวณฟังก์ชันยกกำลังของตัวเลข floating-point ในรูปแบบ long double
fabsl	คำนวณค่าสัมบูรณ์ของตัวเลข floating-point ในรูปแบบ long double
floorl	คำนวณค่าอินทริกรัลมากที่สุดที่ไม่มากกว่าตัวเลข floating-point ที่ระบุ ในรูปแบบ long double
fmodl	คำนวณเศษ long double ของเศษส่วน *x*/*y* โดยที่ *x* และ *y* เป็นตัวเลข floating-point ในรูปแบบ long double
frexpl	แสดงตัวเลข floating-point ในรูปแบบ double format เป็นเศษส่วน ธรรมดาและอินทริกรัลยกกำลัง 2 เก็บค่าจำนวนเต็มและส่งกลับเศษ

รูทีนย่อย	คำอธิบาย
ldexpl	คูณตัวเลข floating-point ในรูปแบบ long double ด้วยอินทริกรัล ยกกำลัง 2
lgammal	คำนวณลอการิทึมฐาน e ของค่าสัมบูรณ์ของฟังก์ชันแกมมา ของตัวเลข floating-point ในรูปแบบ long double
log10l	คำนวณลอการิทึมฐาน 10 ของตัวเลข floating-point ในรูปแบบ long double
logl	คำนวณลอการิทึมฐาน e ของตัวเลข floating-point ในรูปแบบ long double
modfl	เก็บส่วนอินทริกรัลของจำนวนจริงในตัวแปร long double และส่งกลับเศษของจำนวนจริง
powl	คำนวณค่าของ *x* ยกกำลัง *y* โดยที่ทั้งสองตัวเลขเป็นตัวเลข floating-point ในรูปแบบ long double
sinhl	คำนวณไฮเปอร์โบลิก sine ของตัวเลข floating-point ในรูปแบบ long double
sinl	คำนวณ sine ของตัวเลข floating-point ในรูปแบบ long double
sqrtl	คำนวณสแควร์รูทของตัวเลข floating-point ในรูปแบบ long double
strtold	แปลงสตริงเป็นตัวเลข floating-point ในรูปแบบ long double
tanl	คำนวณ tangent ของตัวเลข floating-point ในรูปแบบ long double
tanhl	คำนวณไฮเปอร์โบลิก tangent ของตัวเลข floating-point ในรูปแบบ long double

รายการของรูทีนย่อยกระบวนการ

จากการเสนอเธรด รูทีนย่อยของกระบวนการบางส่วนได้ถูก ขยายการทำงานและรูทีนย่อยอื่นได้ถูกเพิ่มเข้ามา เธรด ไม่ใช่กระบวนการ ในขณะนี้เป็นเอนทิตีที่จัดตารางเวลาได้

สำหรับสัญญาณ ตัวจัดการมีอยู่ที่ระดับกระบวนการ แต่แต่ละเธรดสามารถกำหนดมาสก์สัญญาณ ตัวอย่างบางส่วนของรูทีนย่อย ที่เปลี่ยนแปลงหรือรูทีนย่อยใหม่ คือ: **getprocs, getthrds, ptrace, getpri, setpri, yield** และ **sigprocmask**

รูทีนย่อยถูกแสดงในประเภทต่อไปนี้:

การกำหนดค่าเริ่มต้นของกระบวนการ

`exec:, execl, execv, execle, execve, execlp, execvp` หรือ `exect`
ดำเนินการโปรแกรมใหม่ในกระบวนการที่เรียก

`fork` หรือ `vfork`
สร้างกระบวนการใหม่

`reboot`
เริ่มระบบใหม่

`siginterrupt`
ตั้งค่ารูทีนย่อยให้เริ่มระบบใหม่เมื่อถูกอินเตอร์รัปต์โดยสัญญาณจำเพาะ

การหยุดกระบวนการชั่วคราว

`pause`
หยุดกระบวนการชั่วคราวจนกว่ากระบวนการนั้นจะได้รับสัญญาณ

`wait, wait3, waitpid`
หยุดกระบวนการชั่วคราวจนกว่ากระบวนการ child หยุดหรือจบการทำงาน

การยุติโปรเซส

`abort`
ยุติกระบวนการปัจจุบันและสร้างดัมพ์หน่วยความจำโดยส่งสัญญาณ **SIGOT**

`exit, atexit หรือ _exit`
ยุติกระบวนการ

`, unatexit,`
ฟังก์ชันที่ไม่ได้รีจิสเตอร์ที่ก่อนหน้านี้ถูกรีจิสเตอร์โดยรูทีนย่อย **atexit** ถ้าพบฟังก์ชันที่อ้างถึง ฟังก์ชันนั้นจะถูกลบออกจากรายการฟังก์ชันที่เรียกไว้ในช่วงการยกเลิกโปรแกรมปกติ

`kill หรือ killpg`
หยุดการทำงาน กระบวนการหรือกลุ่มกระบวนการปัจจุบันด้วยสัญญาณ

การระบุกระบวนการและเธรด

`ctermid`
รับชื่อพาธสำหรับเทอร์มินัลที่ควบคุมกระบวนการปัจจุบัน

`cuserid`
รับชื่อผู้ใช้ตัวอักษรและตัวเลขที่สัมพันธ์กับกระบวนการปัจจุบัน

`getpid, getpgrp หรือ getppid`
รับ process ID process group ID หรือ parent process ID ตามลำดับ

`getprocs`
รับรายการตารางกระบวนการ

`getthrds`
รับรายการตารางเธรด

`setpgid หรือ setpgrp`
ตั้งค่า process group ID

`setsid`
สร้างเซสชันและตั้งค่า process group IDs

`uname หรือ unamex`
รับชื่อของระบบปฏิบัติการปัจจุบัน

การทำบัญชีกระบวนการ

`acct`
เปิดใช้งานและปิดใช้งานบัญชีผู้ใช้กระบวนการ

`ptrace`
ติดตามการดำเนินการของกระบวนการ

การจัดสรรรีซอร์สกระบวนการ

`brk หรือ sbrk`
เปลี่ยนการจัดสรรพื้นที่เซ็กเมนต์ข้อมูล

`getdtablesize`
รับขนาดตาราง descriptor

`getrlimit, setrlimit หรือ vlimit`
จำกัดการใช้ทรัพยากรระบบโดยกระบวนการปัจจุบัน

`getrusage, times หรือ vtimes`
แสดงข้อมูลเกี่ยวกับการใช้ทรัพยากร

`plock`
ล็อกกระบวนการ ข้อความ และข้อมูลลงในหน่วยความจำ

`profil`
เริ่มและหยุดการทำงานการสุ่มตัวอย่างแอดเดรสโปรแกรมสำหรับการทำโปรไฟล์การดำเนินการ

`ulimit`
ตั้งค่าข้อจำกัดกระบวนการผู้ใช้

การจัดลำดับความสำคัญกระบวนการ

`getpri`
ส่งกลับลำดับความสำคัญการจัดตารางเวลาของกระบวนการ

`getpriority, setpriority หรือ nice`
รับหรือตั้งค่าค่าลำดับความสำคัญของกระบวนการ

`setpri`
ตั้งค่าลำดับความสำคัญการจัดตารางเวลากระบวนการกับค่าคงที่

`yield`
ให้โพรเซสเซอร์กับกระบวนการที่มีลำดับความสำคัญสูงกว่า

การเธรดกระบวนการและเธรด

`compare_and_swap`
อัพเดตหรือส่งกับตัวแปร single word แบบ atom โดยมีเงื่อนไข

`fetch_and_add`
อัพเดตตัวแปร single word แบบ atom

`fetch_and_and และ fetch_and_or`
ตั้งค่าหรือเคลียร์บิตในตัวแปร single word แบบ atom

`semctl`
ควบคุมการดำเนินการ semaphore

`semget`
รับชุดของ semaphores

`semop`
ดำเนินการ semaphore

การมาสก์และส่งสัญญาณกระบวนการ

raise
ส่งสัญญาณไปที่โปรแกรมกระทำการ

sigaction, sigvec หรือ signal
ระบุการดำเนินการเมื่อสัญญาณมาถึง

sigemptyset, sigfillset, sigaddset, sigdelset หรือ sigismember
สร้างและจัดการมาสก์สัญญาณ

sigpending
ระบุชุดของสัญญาณที่ถูกบล็อกไม่ให้มาถึง

sigprocmask, sigsetmask หรือ sigblock
ตั้งค่ามาสก์สัญญาณ

sigset, sighold, sigrelse หรือ sigignore
เพิ่มการบริการสัญญาณและจัดเตรียมการจัดการสัญญาณ

sigsetjmp หรือ siglongjmp
บันทึกและคืนค่าบริบทสแต็กและมาสก์สัญญาณ

sigstack
ตั้งค่าบริบทสแต็กสัญญาณ

sigsuspend
เปลี่ยนชุดของสัญญาณที่บล็อก

ssignal หรือ gsignal
นำหน่วยบริการสัญญาณซอฟต์แวร์ไปใช้

ข้อความกระบวนการ

msgctl
จัดเตรียมการดำเนินการควบคุมข้อความ

msgget
แสดงตัวระบุคิวข้อความ

msgrcv
อ่านข้อความจากคิว

msgsnd
ส่งข้อความไปที่คิวข้อความ

msgxrcv
รับข้อความที่ขยาย

psignal
พิมพ์ข้อความสัญญาณระบบ

รายการรูทีนย่อยโปรแกรมมิงแบบมัลติเธรด

โปรแกรมมิงในสภาพแวดล้อม multithreaded ต้องการรูทีนย่อย reentrant เพื่อ รับประกัน data integrity

ใช้รูทีนย่อยดังต่อไปนี้แทนเวอร์ชัน non-reentrant:

รูทีนย่อย	**คำอธิบาย**
asctime_r	แปลงค่าเวลาเป็นอาร์เรย์อักขระ
getgrnam_r	ส่งกลับรายการกลุ่มถัดไปในฐานข้อมูลผู้ใช้ที่ตรงกับชื่อที่ระบุ
getpwuid_r	ส่งกลับรายการถัดไปที่ตรงกับ user ID ที่ระบุในฐานข้อมูล use

รายการดังต่อไปนี้แสดงรูทีนย่อย non-reentrant ใน libc

รูทีนย่อย	**คำอธิบาย**		
asctime	getgrent	gsignal	setkey
auditread	getgrgid	hcreate	setlogmask
closelog	getgrnam	hdestroy	setnetent
crypt	getgroupsbyuser	hsearch	setnetgrent
ctime	getgrset	inet_ntoa	setprotoent
dirname	gethostbyaddr	initstate	setpwent
drand48	gethostbyname	innetgr	setpwfile
ecvt	gethostent	iso_addr	setrpcent
endttyent	getlogin	iso_ntoa	setservent
encrypt	getnetbyaddr	jrand48	setstate
asctime	getnetbyname	l64a	setttyent
endfsent	getnetent	lcong48	setutent
endfsent	getnetgrent	link_ntoa	setutxent
endgrent	getopt	localtime	srand48
endhostent	getprotobyname	lrand48	srandom
endnetent	getprotobynumber	mrand48	ssignal
endnetgrent	getprotoent	mtime	strerror
endprotoent	getpwent	ndutent	strtok
endpwent	getpwnam	nrand48	syslog
endrpcent	getpwuid	ns_ntoa	ttyname
endservent	getrpcbyname	openlog	utmpname
endttyent	getrpcbynumber	pututline	wcstok
endutxent	getrpcent	pututxline	
erand48	getservbyname	rand	

รูทีนย่อย	คำอธิบาย		
ether_aton	getservbyport	random	
ether_ntoa	getservent	rcmd	
fcvt fgetgrent	getttyent	rcmd2	
fgetpwent	getttynam	readdir	
getdate	getuinfo	rexec	
getfsent	getutent	re_comp	
getfsent	getutid	re_exec	
getfsfile	getutline	seed48	
getfsfile	getutxent	setfsent	
getfsspec	getutxid	setgrent	
getfstype	getutxline	sethostent	

รายการของรูทีนย่อยไลบรารี workbench ของโปรแกรมเมอร์

ไลบรารี Programmers Workbench (**libPW.a**) มีรูทีนที่ถูกจัดเตรียมเฉพาะสำหรับความเข้ากันได้กับ โปรแกรมที่มีอยู่

ไม่แนะนำให้ใช้กับโปรแกรมใหม่ อินเตอร์เฟสเหล่านี้ มาจาก AT&T PWB Toolchest

รูทีน	คำอธิบาย
any (*Character*, *String*)	ระบุว่า *String* มี *Character* หรือไม่
anystr (*String1*, *String2*)	ระบุออฟเซ็ตใน *String1* ของอักขระแรก ที่มีอยู่เช่นกันใน *String2*
balbrk (*String*, *Open*, *Close*, *End*)	ระบุออฟเซ็ตใน *String* ของอักขระแรก ในสตริง *End* ที่เกิดขึ้นภายนอก สตริงที่สมดุลตามที่กำหนดโดย *Open* และ *Close*
cat (*Destination*, *Source1*, *Source0*)	เชื่อมต่อสตริง *Source* และคัดลอก ไปที่ *Destination*
clean_up ( )	รูทีน cleanup เริ่มต้น
curdir (*String*)	นำชื่อพาธแบบเต็มของไดเร็กทอรีปัจจุบันใส่ใน *String*
dname (*p*)	ระบุไดเร็กทอรีใดที่มีไฟล์ *p*
fatal (*Message*)	ตัวจัดการข้อผิดพลาดอเนกประสงค์
fdfopen (*fd*, *Mode*)	เหมือนกับรูทีนย่อย **stdio fdopen**
giveup (*Dump*)	บังคับ core dump
imatch (*pref*, *String*)	ระบุว่าสตริง *pref* เป็น ซับสตริงเริ่มต้นของ *String*
lockit (*LockFile*, *Count*, *pid*)	สร้างล็อกไฟล์
move (*String1*, *String2*, *n*)	คัดลอกอักขระ *n* แรกจาก *String1* ไปที่ *String2*
patoi (*String*)	แปลง *String* ไปเป็นจำนวนเต็ม
patol (*String*)	Converts *String* to long.
repeat (*Destination*, *String*, *n*)	ตั้งค่า *Destination* กับ *String* ทำซ้ำ *n* ครั้ง
repl (*String*, *Old*, *New*)	แทนที่แต่ละตำแหน่งที่มีอักขระ *Old* ใน *String* ด้วยอักขระ *New*
satoi (*String*, **ip*)	แปลง *String* ไปเป็นจำนวนเต็มและบันทึกไว้ใน **ip*
setsig ( )	ส่งสัญญาณเพื่อให้รับโดย **setsig1**
setsig1 (*Signal*)	รูทีนการจัดการสัญญาณอเนกประสงค์
sname (*String*)	รับตัวชี้ไปยังชื่อแบบง่ายของชื่อพาธแบบเต็ม *String*
strend (*String*)	ค้นหาจุดสิ้นสุดของ *String*
trnslat (*s*, *old*, *new*, *Destination*)	คัดลอกสตริง *s* ลงใน *Destination* และแทนที่อักขระใน *old* ด้วยอักขระ ที่ตรงกันใน *new*
unlockit (*lockfile*, *pid*)	ลบล็อกไฟล์
userdir (*uid*)	รับไดเร็กทอรีเข้าสู่ระบบของผู้ใช้

รูทีน	คำอธิบาย
userexit (*code*)	รูทีน exit ของผู้ใช้เริ่มต้น
username (*uid*)	รับชื่อเข้าสู่ระบบของผู้ใช้
verify (*String1*, *String2*)	ระบุออฟเซ็ตในสตริง *String1* ของ อักขระแรกที่ไม่อยู่ใน *String2*
xalloc (*asize*)	จัดสรรหน่วยความจำ
xcreat (*name*, *mode*)	สร้างไฟล์
xfree (*aptr*)	ฟรีหน่วยความจำ
xfreeall ( )	ฟรีหน่วยความจำทั้งหมด
xlink (*f1*, *f2*)	การเชื่อมโยงไฟล์
xmsg (*file*, *func*)	เรียกรูทีน **fatal** ที่มีข้อความแสดงข้อผิดพลาด ที่เหมาะสม
xpipe (*t*)	สร้างไพพ์
xunlink (*f*)	เอารายการไดเรกทอรีออก
xwrite (*fd*, *buffer*, *n*)	เขียน *n* ไบต์กับไฟล์ที่สัมพันธ์กับ *fd* จาก *buffer*
zero (*p*, *n*)	ทำให้เป็นค่าศูนย์ *n* ไบต์เริ่มต้นที่แอดเดรส *p*
zeropad (*s*)	แทนที่ช่องว่างเริ่มต้นด้วยอักขระ 0 (ศูนย์) ในสตริง *s*

ไฟล์

/usr/lib/libPW.a

มีรูทีนที่จัดเตรียมเฉพาะสำหรับความเข้ากันได้กับโปรแกรมที่มีอยู่

รายการของรูทีนย่อยการรักษาความปลอดภัยและการตรวจสอบ

ส่วนนี้แสดงรายการรูทีนย่อยการรักษาความปลอดภัยและการตรวจสอบ

รูทีนย่อยการควบคุมการเข้าถึง

รูทีนย่อย	คำอธิบาย
acl_chg หรือ **acl_fchg**	เปลี่ยนข้อมูลค่าควบคุมการเข้าใช้บนไฟล์
acl_get หรือ **acl_fget**	รับข้อมูลค่าควบคุมการเข้าใช้ของไฟล์
acl_put หรือ **acl_fput**	ตั้งค่าข้อมูลค่าควบคุมการเข้าใช้ของไฟล์
acl_set หรือ **acl_fset**	ตั้งค่ารายการฐานของข้อมูลค่าควบคุมการเข้าใช้ของไฟล์
aclx_convert	แปลงข้อมูลค่าควบคุมการเข้าใช้จากหนึ่งชนิด ACL ไปเป็นชนิดอื่น
aclx_get หรือ **aclx_fget**	รับข้อมูลค่าควบคุมการเข้าใช้ของไฟล์ถ้า ACL ที่เกี่ยวข้อง เป็นชนิด AIXC
aclx_gettypeinfo	เรียกคุณลักษณะ ACL ที่กำหนดให้กับชนิด ACL
aclx_gettypes	เรียกรายการของชนิด ACL ที่สนับสนุนสำหรับระบบไฟล์ที่สัมพันธ์ กับพาธที่จัดเตรียม
aclx_print หรือ **aclx_printStr**	แปลงข้อมูลค่าควบคุมการเข้าใช้ไบนารีลงในรูปแบบไม่ใช่ไบนารี, สามารถอ่านได้
aclx_put หรือ **aclx_fput**	เก็ขข้อมูลค่าควบคุมการเข้าใช้สำหรับอ็อบเจ็กต์ระบบไฟล์
aclx_scan หรือ **aclx_scanStr**	แปลงข้อมูลค่าควบคุมการเข้าใช้ที่อยู่ในรูปแบบไม่ใช่ไบนารี , ข้อความ ที่อ่านได้ เป็นข้อมูลไบนารี ACL ไบนารีที่มีรูปแบบประเภท ACL โดยเฉพาะ
chacl หรือ **fchac l**	เปลี่ยนสิทธิ์ของไฟล์
chmod หรือ **fchmod**	เปลี่ยนสิทธิการเขาถึงของไฟล์
chown, fchown, chownx หรือ **fchownx**	เปลี่ยนความเป็นเจ้าของไฟล์
frevoke	เรียกคืนการเข้าถึงไฟล์โดยกระบวนการอื่น
revoke	เรียกคืนการเข้าถึงไฟล์
statacl หรือ **fstatacl**	เรียกข้อมูลค่าควบคุมการเข้าใช้สำหรับไฟล์

รูทีนย่อยการตรวจสอบ

รูทีนย่อย	คำอธิบาย
audit	เปิดใช้งานและปิดใช้งานการตรวจสอบระบบ
auditbin	กำหนดไฟล์ที่จะอยู่ในเร็กคอร์ดการตรวจสอบ
auditevents	รับหรือตั้งค่าสถานะของการตรวจสอบเหตุการณ์ระบบ
auditlog	ผนวกเร็กคอร์ดการตรวจสอบกับไฟล์ audit bin
auditobj	รับหรือตั้งค่าโหมดการตรวจสอบของอ็อบเจ็กต์ข้อมูลระบบ
auditpack	บีบอัดและคลายการบีบอัด audit bins
auditproc	รับหรือตั้งค่าภาวะการตรวจสอบของกระบวนการ
auditread หรือ **auditread_r**	อ่านเร็กคอร์ดการตรวจสอบ
auditwrite	เขียนเร็กคอร์ดการตรวจสอบ

รูทีนย่อยการระบุและการพิสูจน์ตัวตน

รูทีนการรับรองความถูกต้องของผู้ใช้มีศักยภาพในการเก็บรหัสผ่านและรหัสผ่านที่เข้ารหัส ในหน่วยความจำ ซึ่งอาจเปิดเผยรหัสผ่านและรหัสผ่านที่เข้ารหัสใน coredumps

รูทีนย่อย	คำอธิบาย
authenticate	รับรองความถูกต้อง ชื่อและรหัสผ่านผู้ใช้
ckuseracct	ตรวจสอบความถูกต้องของบัญชีผู้ใช้
ckuserID	รับรองความถูกต้องผู้ใช้
crypt, encrypt หรือ **setkey**	เข้ารหัสหรือถอดรหัสข้อมูล
genpagvalue	สร้างค่า PAG เฉพาะ system-wide สำหรับชื่อ PAG ที่กำหนด เช่น afs
getpagvalue64	เรียกคืนค่า PAG 64-บิตสำหรับกระบวนการ
setpagvalue64	เก็บค่า PAG 64-บิตสำหรับกระบวนการ .
getgrent, getgrgid, getgrnam, setgrent หรือ **endgrent**	เข้าถึงข้อมูลกลุ่มพื้นฐานในฐานข้อมูลผู้ใช้
getgrgid_r	รับรายการฐานข้อมูลกลุ่มสำหรับ group ID ในสภาพแวดล้อม multithreaded
getgrnam_r	ค้นหาฐานข้อมูลกลุ่มสำหรับชื่อในสภาพแวดล้อม multithreaded
getgroupattr, IDtogroup, nextgroup หรือ **putgroupattr**	เข้าถึงขอมูลกลุ่มในฐานข้อมูลผู้ใช้
getlogin	รับชื่อเข้าสู่ระบบของผู้ใช้
getlogin_r	รับชื่อเข้าสู่ระบบของผู้ใช้ในสภาพแวดล้อม multithreaded
getpass	อ่านรหัสผ่าน
getportattr หรือ **putportattr**	เข้าถึงข้อมูลพอร์ตในฐานข้อมูลพอร์ต
getpwent, getpwuid, getpwnam, putpwent, setpwent หรือ **endpwent**	เข้าถึงข้อมูลผู้ใช้พื้นฐานในฐานข้อมูลผู้ใช้
getuinfo	ค้นหาค่าที่สัมพันธ์กับผู้ใช้
getuserattr, IDtouser, nextuser หรือ **putuserattr**	เข้าถึงข้อมูลผู้ใช้ในฐานข้อมูลผู้ใช้
getuserpw, putuserpw, หรือ **putuserpwhist**	เข้าถึงข้อมูลการรับรองความถูกต้องของผู้ใช้
loginfailed	บันทึกการพยายามเข้าสู่ระบบที่ไม่สำเร็จ
loginrestrictions	ระบุว่าผู้ใช้ได้รับอนุญาตให้เข้าถึงระบบหรือไม่
loginsuccess	บันทึกการเข้าสู่ระบบที่สำเร็จ
newpass	สร้างรหัสผ่านใหม่สำหรับผู้ใช้
passwdexpired	ตรวจสอบรหัสผ่านของผู้ใช้เพื่อระบุว่าหมดอายุแล้วหรือไม่
setpwdb หรือ **endpwdb**	เปิดหรือปิดฐานข้อมูลการพิสูจน์ตัวตน
setuserdb หรือ **enduserdb**	เปิดหรือปิดฐานข้อมูลผู้ใช้
ระบบ	รันคำสั่งเชลล์
tcb	เปลี่ยนสถานะ Trusted Computing Base ของไฟล์

รูทีนย่อยกระบวนการ

รูทีนย่อย	คำอธิบาย
getgid หรือ **getegid**	รับ real หรือ group ID ของกระบวนการที่เรียก
getgroups	รับชุดกลุ่มที่เหมือนกันของกระบวนการปัจจุบัน
getpcred	รับข้อมูลประจำตัวความปลอดภัยของกระบวนการปัจจุบัน
getpenv	รับสภาพแวดล้อมกระบวนการปัจจุบัน
getuid หรือ **geteuid**	รับ real หรือ effective user ID ของกระบวนการปัจจุบัน
initgroups	กำหนดค่าเริ่มต้น group ID เสริมของกระบวนการปัจจุบัน
kleenup	เคลียร์ค่าสภาพแวดล้อมรันไทม์ของกระบวนการ
setgid, setrgid, setegid หรือ **setregid**	ตั้งค่า group IDs ของกระบวนการที่เรียก
setgroups	ตั้งค่า group ID เสริมของกระบวนการปัจจุบัน
setpcred	ตั้งค่าข้อมูลประจำตัวของกระบวนการปัจจุบัน
setpenv	ตั้งค่าสภาพแวดล้อมกระบวนการปัจจุบัน
setuid, setruid, setuid หรือ **setreuid**	ตั้งค่า user ID ของกระบวนการ
usrinfo	รับและตั้งค่าข้อมูลผู้ใช้เกี่ยวกับเจ้าของของกระบวนการปัจจุบัน

รายการของรูทีนย่อยการจัดการกับสตริง

ฟังก์ชันการจัดการสตริงรวมถึง:

ฟังก์ชันการจัดการสตริงรวมถึง:

- ค้นหาตำแหน่งอักขระภายในสตริง
- ค้นหาลำดับของอักขระภายในสตริง
- คัดลอกสตริง
- เชื่อมต่อสตริง
- เปรียบเทียบสตริง
- แปลสตริง
- วัดผลสตริง

เมื่อใช้ฟังก์ชันสตริงเหล่านี้ คุณไม่จำเป็นต้อง รวมไฟล์ header file ในโปรแกรมหรือระบุแฟล็กพิเศษให้กับคอมไพลเลอร์

ฟังก์ชันดังต่อไปนี้จัดการข้อมูลสตริง:

`bcopy, bcmp, bzero, ffs`
: การดำเนินการ bit และ byte ของสตริง

`gets, fgets`
: รับสตริงจากสตรีม

`puts, fputs`
: เขียนสตริงลงสตรีม

`compile, step, advance`
: คอมไพล์และจับคู่รูปแบบนิพจน์ทั่วไป

`strlen, strchr, strrchr, strpbrk, strspn, strcspn, strstr, strtok`
: การดำเนินการกับสตริง

`jcode`
: ทำการแปลงสตริงบนโค้ดการประมวลผล 8-บิต

varargs
จัดการรายการพารามิเตอร์ความยาวตัวแปร

ตัวอย่าง: โปรแกรมสำหรับการดำเนินการกับอักขระ

ส่วนนี้มีตัวอย่างโปรแกรมมิงสำหรับการดำเนินการกับ อักขระ

```
/*
โปรแกรมนี้ออกแบบมาเพื่อสาธิตการใช้รูทีนย่อย "การจัดประเภท
และการแปลงอักขระ" เนื่องจากเรากำลังทำงานกับ
อักขระ ซึ่งเป็นตำแหน่งปกติที่ใช้ในการสาธิตการใช้
รูทีนย่อย getchar และรูทีนย่อย putchar จะไลบรารี stdio
วัตถุประสงค์ของโปรแกรมคือ:
-อ่านอินพุตจาก "stdin"
-ตรวจสอบว่า อักขระทั้งหมดนั่นคือ ascii และสามารถพิมพ์ได้
-แปลงอัพขระตัวพิมพ์ใหญ่ทั้งหมดให้เป็นตัวพิมพ์เล็ก
-ยกเว้นพื้นที่ว่างจำนวนมาก
-รายงานสถิติโดยพิจารณาถึงชนิดของอักขระ
รูทีนย่อยต่อไปนี้จะถูกสาธิตโดยโปรแกรมตัวอย่างนี้:

- getchar

- putchar

- isascii (ctype)

- iscntrl (ctype)

- isspace (ctype)

- isalnum (ctype)

- isdigit (ctype)

- isalpha (ctype)

- isupper (ctype)

- islower (ctype)

- ispunct (ctype)

- tolower (conv)

- toascii ( conv)
*/
```

```
#include <stdio.h>  /* The mandatory include file  */
#include <ctype.h>  /* Included for character classification
subroutines */
/* The various statistics gathering counters */
int asciicnt, printcnt, punctcnt, uppercnt, lowercnt,
digcnt, alnumcnt, cntrlcnt, spacecnt, totcnt, nonprntcnt,linecnt, tabcnt ;
main()
{
int ch ; /* The input character is read in to this */
char c , class_conv() ;
asciicnt=printcnt=punctcnt=uppercnt=lowercnt=digcnt==0;
cntrlcnt=spacecnt=totcnt=nonprntcnt=linecnt=tabcnt=0;
alnumcnt=0;
while ( (ch =getchar()) != EOF )
{
totcnt++;
c = class_conv(ch) ;
putchar(c);
}
printf("The number lines of of input were %d\n",linecnt);
printf(" The character wise breakdown follows :\n");
printf(" TOTAL  ASCII  CNTRL   PUNCT   ALNUM  DIGITS UPPER
LOWER   SPACE   TABCNT\n");
printf("%5d %5d %5d %5d %5d %5d %5d %5d %5d %5d\n",totcnt,
asciicnt, cntrlcnt, punctcnt, alnumcnt, digcnt, uppercnt,lowercnt, spacecnt, tabcnt );
}
char class_conv(ch)
char ch;
{
if (isascii(ch)) {
asciicnt++;
if ( iscntrl(ch) && ! isspace(ch)) {
nonprntcnt++ ;
cntrlcnt++ ;
return(' ');
}
else if ( isalnum(ch)) {
alnumcnt++;
if (isdigit(ch)){
digcnt++;
return(ch);
}
else if (isalpha(ch)){
if ( isupper(ch) ){
uppercnt++ ;
return(tolower(ch));
```

```
}
else if ( islower(ch) ){

lowercnt++;
return(ch);

}
else {

/*
We should never be in this situation since an alpha character can only be
either uppercase or lowercase.
*/

fprintf(stderr,"Classification error for %c \n",ch);
return(NULL);

}

}
else if (ispunct(ch) ){

punctcnt++;
return(ch);

}
else if ( isspace(ch) ){

spacecnt++;
if ( ch == '\n' ){
linecnt++;
return(ch);

}
while ( (ch == '\t' ) || ( ch == ' ' ) ) {
if ( ch == '\t' ) tabcnt ++ ;
else if ( ch == ' ' ) spacecnt++ ;
totcnt++;
ch = getchar();

}
ungetc(ch,stdin);
totcnt--;
return(' ');

}
else {

/*
We should never be in this situation any ASCII character
can only belong to one of the above classifications.
*/
fprintf(stderr,"Classification error for %c \n",ch);
return(NULL);
}

}
else
{

fprintf(stdout,"Non Ascii character encountered \n");
return(toascii(ch));

}
```

```
}
```

ตัวอย่าง: การค้นหาและเรียงลำดับโปรแกรม

ส่วนนี้มีตัวอย่างของการค้นหาและเรียงลำดับโปรแกรม

```
/**โปรแกรม นี้ สาธิต วิธีการ ใช้ ฟังก์ชัน ต่อไปนี้:

รูทีนย่อย -qsort (ไลบรารีรูทีนการเรียงลำดับแบบด่วน)

รูทีนย่อย -bsearch (ไลบรารีรูทีนการค้นหาแบบไบนารี)

รูทีนย่อย -fgets, fopen, fprintf, malloc, sscanf และ strcmp

โปรแกรมอ่านไฟล์ อินพุต สองไฟล์ พร้อมกับเร็กคอร์ด ใน
รูปแบบ สตริง และพิมพ์หรือแสดง:

-records from file2, which are excluded in file1

-records from file1, which are excluded in file2

โปรแกรม จะ อ่าน เร็กคอร์ด อินพุต จาก ไฟล์ทั้งสองไฟล์
ในอาร์เรย์สองตัว ซึ่งจะเรียงลำดับตาม
ลำดับทั่วไปโดยใช้รูทีนย่อย qsort แต่ละอิลิเมนท์ของ
อาร์เรย์หนึ่งค่าที่ถูกค้นหารายการสำเนาใน
อาร์เรย์อื่นโดยใช้รูทีนย่อย bsearch ถ้าไอเท็ม
ในทั้งสองอาร์เรย์ ข้อความจะบ่งชี้ว่า
ไม่พบ กระบวนการจะทำการแลกเปลี่ยนซ้ำ
เพื่อขอรับรายการสำรองของการแยกออกไปนี้
**/
#include <stdio.h>      /*the library file to be included for
                         /*standard input and output*/

#include <search.h>     /*the file to be included for qsort*/
#include <sys/errno.h>  /*the include file for interpreting

                        /*predefined error conditions*/

#define MAXRECS  10000          /*array size limit*/

#define MAXSTR   256            /*maximum input string length*/
#define input1 "file1"          /*one input file*/
#define input2 "file2"          /*second input file*/
#define out1   "o_file1"        /*output file1*/
#define out2   "o_file2"        /*output file2*/

main()
{

char *arr1[MAXRECS] , *arr2[MAXRECS] ;/*the arrays to store

input records*/

unsigned int num1 , num2;       /*to keep track of the number of

                                /*input records. Unsigned int

                                /*declaration ensures
                                /*compatability

                                /*with qsort library routine.*/

int i ;
int compar();           /*the function used by qsort and

                        /*bsearch*/
```

```
extern int errno ; /*to capture system call failures*/
FILE *ifp1 , *ifp2, *ofp1, *ofp2; /*the file pointers for

input and output */

void *bsearch() ;       /*the library routine for binary search*/
void qsort();           /*the library routine for quick sort*/
char*malloc() ;         /*memory allocation subroutine*/
void exit() ;

num1 = num2 = 0;

/**Open the input and output files for reading or writing
**/

if ( (ifp1 = fopen( input1 , "r" ) ) == NULL )
{

(void) fprintf(stderr,"%s could not be opened\n",input1);
exit(-1);

}

if (( ifp2 = fopen( input2 , "r" )) == NULL )
{

(void) fprintf(stderr,"%s could not be opened\n",input2);
exit(-1);

}

if (( ofp1 = fopen(out1,"w" )) == NULL )
{

(void) fprintf(stderr,"%s could not be opened\n",out1);
exit(-1);

}

if (( ofp2 = fopen(out2,"w")) == NULL )
{

(void) fprintf(stderr,"%s could not be opened\n", out2);
exit(-1);

}

/**Fill the arrays  with data from input files. Readline
function returns the number of input records.**/

if ( (i = readline( arr1 , ifp1 ))  < 0 )
{

(void) fprintf(stderr,"o data in %s. Exiting\n",input1);
exit(-1);

}
num1 = (unsigned) i;
if ( (i = readline ( arr2 , ifp2)) < 0 )
{

(void) fprintf(stderr,"No data in %s.  Exiting\n",input2);
exit(-1);

}
num2 = (unsigned) i;
```

```
/**
The arrays can now be sorted using qsort subroutine
**/

qsort( (char *)arr1 , num1 ,  sizeof (char *)  , compar);
qsort( (char *)arr2 , num2 ,  sizeof (char *)  , compar);

/**When the two arrays are sorted in a common order, the
program builds a list of elements found in one but not
in the other, using bsearch.

Check that each element in array1 is in array2
**/

for ( i= 0 ; i < num1 ; i++ )
{

if ( bsearch((void *)&arr1[i] , (char *)arr2,num2,

sizeof(char *) , compar) == NULL )

{
(void)  fprintf(ofp1,"%s",arr1[i]);

}

} /**One list of exclusions is complete**/

/**Check that each element in array2 is in array1**/

for ( i = 0 ; i < num2 ; i++ )
{
  if ( bsearch((void *)&arr2[i], (char *)arr1, num1

, sizeof(char *) , compar) == NULL )

{
(void) fprintf(ofp2,"%s",arr2[i]);

}

}

/**Task completed, so return**/

return(0);

}

/**The function reads in records from an input
 file and fills in the details into the two arrays.**/

readline ( char **aptr, FILE *fp )

{

char str[MAXSTR] , *p ;
int i=0 ;

/**Read the input file line by line**/

while ( fgets(str , sizeof(str) , fp ))
{

/**Allocate sufficient memory. If the malloc subroutine
fails, exit.**/

if ( (p = (char *)malloc ( sizeof(str))) == NULL )
{
```

```
(void) fprintf(stderr,"Insufficient Memory\n");
return(-1);

}
else

{
if ( 0 > strcpy(p, str))

{
(void) fprintf(stderr,"Strcpy failed \n");
return(-1);
}
i++ ; /*increment number of records count*/

}

} /**End of input file reached**/
return(i);/*return the number of records read*/

}

/**We want to sort the arrays based only on the contents of the first field of
the input records. So we get the first  field using SSCANF**/

compar( char **s1 , char **s2 )
{

char st1[100] , st2[100] ;
(void) sscanf(*s1,"%s" , st1) ;
(void) sscanf(*s2,"%s" , st2) ;

/**Return the results of string comparison to the calling procedure**/

return(strcmp(st1 , st2));

}
```

รายการของไลบรารีระบบปฏิบัติการ

ส่วนนี้แสดงรายการไลบรารีของระบบปฏิบัติการ

ไลบรารี	คำอธิบาย
/usr/lib/libbsd.a	ไลบรารี Berkeley
/lib/profiled/libbsd.a	ไลบรารี Berkeley ที่ทำโปรไฟล์
/usr/ccs/lib/libcurses.a	ไลบรารี Curses
/usr/ccs/lib/libc.a	ไลบรารี Standard I/O, ไลบรารี standard C
/lib/profiled/libc.a	ไลบรารี Standard I/O, ไลบรารี standard C ที่ทำโปรไฟล์
/usr/ccs/lib/libdbm.a	ไลบรารีการจัดการฐานข้อมูล
/usr/ccs/lib/libi18n.a	ไลบรารีเค้าโครง
/usr/lib/liblvm.a	ไลบรารี LVM (Logical Volume Manager)
/usr/ccs/lib/libm.a	ไลบรารีคณิตศาสตร์
/usr/ccs/lib/libp/libm.a	ไลบรารีคณิตศาสตร์ที่ทำโปรไฟล์
/usr/lib/libodm.a	ไลบรารี ODM (Object Data Manager)
/usr/lib/libPW.a	ไลบรารี Programmers Workbench
/usr/lib/libpthreads.a	ไลบรารี POSIX compliant Threads
/usr/lib/libqb.a	ไลบรารี Queue Backend
/usr/lib/librpcsvc.a	ไลบรารี RPC (Remote Procedure Calls)
/usr/lib/librts.a	ไลบรารี Run-Time Services
/usr/lib/libs.a	ฟังก์ชันความปลอดภัย
/usr/lib/libsm.a	ไลบรารีการจัดการระบบ
/usr/lib/libsrc.a	ไลบรารี SRC (System Resource Controller)

ไลบรารี	คำอธิบาย
/usr/lib/libmsaa.a	ไลบรารีคณิตศาสตร์ SVID (System V Interface Definition)
/usr/ccs/lib/libp/libmsaa.a	ไลบรารีคณิตศาสตร์ SVID (System V Interface Definition) ที่ทำโปรไฟล์
/usr/ccs/lib/libtermcap.a	Terminal I/O
/usr/lib/liby.a	ไลบรารี YP (Yellow Pages)
/usr/lib/lib300.a	รูทีนย่อยกราฟิกสำหรับเวิร์กสเตชัน DASI 300
/usr/lib/lib300s.a	รูทีนย่อยกราฟิกสำหรับเวิร์กสเตชัน DASI 300s
/usr/lib/lib300S.a	รูทีนย่อยกราฟิกสำหรับเวิร์กสเตชัน DASI 300S
/usr/lib/lib4014.a	รูทีนย่อยกราฟิกสำหรับเวิร์กสเตชัน Tektronix 4014
/usr/lib/lib450.a	รูทีนย่อยกราฟิกสำหรับเวิร์กสเตชัน DASI 450
/usr/lib/libcsys.a	เซอร์วิสสวนขยายเคอร์เนล
/usr/ccs/lib/libdbx.a	ไลบรารีโปรแกรมดีบัก
/usr/lib/libgsl.a	ไลบรารี Graphics Support
/usr/lib/libieee.a	ไลบรารี IEEE floating point
/usr/lib/libIM.a	ไลบรารีการประมวลผลไฟล์ Stanza
/usr/ccs/lib/libl.a	ไลบรารี lex
/usr/lib/libogsl.a	ไลบรารีสนับสนุนกราฟิกแบบเก่า
/usr/lib/liboldX.a	ไลบรารี X10
/usr/lib/libplot.a	คณิตศาสตร์การลงจุด
/usr/lib/librpcsvc.a	เซอร์วิส RPC
/usr/lib/librs2.a	รูทีนย่อยจำเพาะฮาร์ดแวร์ **sqrt** และ **itrunc**
/usr/lib/libxgsl.a	เพิ่มประสิทธิภาพกราฟิก Enhanced X-Windows
/usr/lib/libX11.a	ไลบรารีรันไทม์ X11
/usr/lib/libXt.a	ไลบรารีทูลคิต X11
/usr/lib/liby.a	ไลบรารีรันไทม์ yacc

System Management Interface Tool (SMIT)

System Management Interface Tool (SMIT) คืออินเตอร์เฟสคำสั่งแบบโต้ตอบและอินเตอร์เฟสคำสั่งเชิงหน้าจอที่สามารถขยายได้

ซึ่งจะแสดงพร้อมต์ให้ผู้ใช้กรอกข้อมูลที่ต้องการเพื่อสร้างสตริงคำสั่ง และนำเสนอการเลือกที่ได้ถูกกำหนดไว้ก่อนที่เหมาะสมและดีฟอลต์รันไทม์ที่พร้อมใช้งาน วิธีนี้ช่วยป้องกันผู้ใช้จากซอร์สต่างๆ ของงานพิเศษหรือข้อผิดพลาด ซึ่งรวมถึงรายละเอียดของไวยากรณ์คำสั่งที่ซับซ้อน ค่าพารามิเตอร์ที่ถูกต้อง การสะกดคำสั่งของระบบ หรือชื่อพาธเชลล์แบบกำหนดเอง

คุณยังสามารถสร้างและใช้ฐานข้อมูลสำรองแทนการแก้ไขดีฟอลต์ฐานข้อมูลระบบของ SMIT

ส่วนต่อไปนี้กล่าวถึง SMIT โดยละเอียด:

ภารกิจใหม่ที่ประกอบด้วยคำสั่งตั้งแต่หนึ่งคำสั่งขึ้นไป หรือการเติมสคริปต์เชลล์ **ksh** สามารถเพิ่มให้กับ SMIT ได้ทุกเวลาโดยการเพิ่มอินสแตนซ์ของอ็อบเจ็กต์หน้าจอที่ได้ถูกกำหนดไว้ก่อนให้กับฐานข้อมูลของ SMIT อ็อบเจ็กต์หน้าจอเหล่านี้ (อธิบายด้วยไฟล์ stanza) จะถูกใช้โดย Object Data Manager (ODM) เพื่ออัพเดตฐานข้อมูล SMIT ฐานข้อมูลนี้ควบคุมลักษณะการทำงานแบบรันไทม์ของ SMIT

ข้อมูลที่เกี่ยวข้อง:

dspmsg

gencat

ksh

man

odmadd

odmcreate

odmget

smit

spaths

ชนิดของหน้าจอ SMIT

มีชนิดหน้าจอหลักอยู่ด้วยกันสามหน้าจอ ที่พร้อมใช้งานสำหรับ System Management Interface Tool (SMIT) หน้าจอที่เกิดขึ้นในลำดับชั้น ที่เข้ากันได้กับหน้าจอเมนู หน้าจอของตัวเลือก และหน้าจอไดอะล็อก

ขณะที่ดำเนินการภารกิจ ผู้ใช้จะเดินข้ามเมนูตั้งแต่หนึ่งเมนูขึ้นไป จากนั้นเดินข้ามตัวเลือกตั้งแต่ศูนย์ขึ้นไป และท้ายสุดเดินข้ามหนึ่งไดอะล็อก

ตารางต่อไปนี้จะแสดงชนิดของหน้าจอ SMIT ซึ่งผู้ใช้มองเห็นได้บนหน้าจอแต่ละหน้าจอ และสิ่งที่ SMIT จะทำกับหน้าจอแต่ละหน้าจอภายใน:

ชนิดหน้าจอ	สิ่งที่ผู้ใช้เห็นบนหน้าจอ	สิ่งที่ SMIT ทำกับแต่ละหน้าจอ ภายใน
Menu	รายการของตัวเลือก	ใช้ตัวเลือกเพื่อเลือกหน้าจอถัดไปที่ต้องการแสดง
Selector	รายการของตัวเลือกหรือฟิลด์รายการ อย่างใดอย่างหนึ่ง	ขอรับค่าข้อมูลสำหรับหน้าจอถัดไป เลือกไดอะล็อกหรือตัวเลือกสำรอง
Dialog	ลำดับของฟิลด์รายการ	ใช้ข้อมูลจากฟิลด์รายการเพื่อสร้างและรันสตริงคำสั่งภารกิจเป้าหมาย

เมนูจะแสดงรายการของภารกิจย่อยสำรอง การเลือกสามารถนำไปสู่หน้าจอของเมนอื่นๆ หรือนำไปสู่หน้าจอของตัวเลือกหรือหน้าจอไดอะล็อก ตัวเลือก จะถูกใช้เพื่อขอรับรายการข้อมูลที่จำเป็นต่อหน้าจอถัดไป และยังสามารถนำมาใช้เพื่อเลือกหน้าจอของตัวเลือกจำนวนมาก หรือหน้าจอไดอะล็อกที่ต้องการใช้ถัดไป หน้าจอไดอะล็อกคือที่ที่อินพุตยังคงเหลืออยู่ ซึ่งถูกร้องขอจากผู้ใช้ และที่ที่ภารกิจที่เลือกไว้ถูกรัน

เมนูคือ entry point พื้นฐานไปยัง SMIT และสามารถตามด้วยเมนู ตัวเลือก หรือไดอะล็อกอื่นๆ ตัวเลือกสามารถตามด้วยไดอะล็อก ไดอะล็อกคือพาเนลรายการสุดท้ายในลำดับของ SMIT

หน้าจอเมนู

เมนู SMIT คือรายการของไอเท็มที่ผู้ใช้สามารถเลือกได้ เมนูไอเท็มคือภารกิจหรือคลาสของภารกิจ ที่สามารถดำเนินการได้จาก SMIT ผู้ใช้ที่เริ่มต้นด้วยเมนู SMIT หลักจะเลือกไอเท็มที่กำหนดช่วงของภารกิจของระบบไว้แบบกว้างๆ การเลือกจากเมนูถัดไปและเมนูลำดับถัดมาจะเน้นที่ตัวเลือกของผู้ใช้ จนกว่าไดอะล็อกสุดท้ายจะแสดงขึ้นเพื่อเก็บรวบรวมข้อมูล สำหรับผลการทำงานของภารกิจเฉพาะ

การออกแบบเมนูจะช่วยให้ผู้ใช้ SMIT มีขอบเขตที่แคบลง ของตัวเลือกกับภารกิจเฉพาะ การออกแบบของคุณสามารถทำได้ง่ายๆ เหมือนกับเมนูและไดอะล็อกใหม่ ซึ่งพ่วงต่อกับกิ่งของ SMIT ที่มีอยู่ หรือสามารถทำให้ซับซ้อนเหมือนกับลำดับชั้นใหม่ทั้งหมดของเมนู ตัวเลือก และไดอะล็อก ที่ขึ้นต้นด้วยเมนูแอ็พพลิเคชัน SMIT

ณ รันไทม์ SMIT จะได้รับอ็อบเจ็กต์เมนูทั้งหมดพร้อมกับ ID ที่กำหนดไว้ (ค่า descriptor **id**) จากที่เก็บอ็อบเจ็กต์ที่ระบุ หากต้องการเพิ่มลงใน SMIT เฉพาะ ให้เพิ่มอ็อบเจ็กต์เมนูที่มีค่า ID เท่ากับค่าของ descriptor **id** ของอ็อบเจ็กต์ที่ไม่มีหัวเรื่องในเมนูเดียวกัน

สร้างเมนูโดยกำหนดเมนูในไฟล์ stanza จากนั้นประมวลผลไฟล์ด้วยคำสั่ง **odmadd** นิยามของเมนูจะถูกคอมไพล์ลงในกลุ่มของอ็อบเจ็กต์เมนู จำนวนของเมนู ตัวเลือก และไดอะล็อกสามารถกำหนดลงในไฟล์ ตั้งแต่หนึ่งไปขึ้นไปได้

คำสั่ง	คำอธิบาย
odmadd	เพิ่มนิยามเมนูลงในที่เก็บอ็อบเจ็กต์ที่ระบุ
/usr/lib/objrepos	ที่เก็บอ็อบเจ็กต์ที่เป็นค่าดีฟอลต์สำหรับขอมูลระบบ และสามารถนำมาใช้เพื่อเก็บอ็อบเจ็กต์ที่คอมไพล์แล้วของคุณ

ณ รันไทม์ของ SMIT อ็อบเจ็กต์จะเรียกข้อมูลจากฐานข้อมูล SMIT โดยเฉพาะ

หมายเหตุ: คุณควรสำรองข้อมูลไดเร็กทอรี **/usr/lib/objrepos** เสมอ ก่อนที่จะลบหรือเพิ่มอ็อบเจ็กต์หรือคลาสอ็อบเจ็กต์ใดๆ ความเสียหายที่ไม่ได้คาดคิดไว้ซึ่งเกิดขึ้นกับอ็อบเจ็กต์หรือคลาสที่จำเป็นสำหรับการดำเนินการของระบบ อาจเป็นสาเหตุทำให้เกิดปัญหา เกี่ยวกับระบบได้

หน้าจอตัวเลือก

ตัวเลือก SMIT จะแสดงพร้อมต์ให้ผู้ใช้ระบุรายการเฉพาะ โดยปกติคืออ็อบเจ็กต์ของระบบ (เช่น เครื่องพิมพ์) หรือแอ็ตทริบิวต์ของอ็อบเจ็กต์ (เช่น โหมดเครื่องพิมพ์แบบอนุกรมหรือเครื่องพิมพ์แบบขนาน) ข้อมูลนี้จะถูกใช้โดย SMIT ในไดอะล็อกถัดไป

ตัวอย่างเช่น ตัวเลือกสามารถพร้อมต์ให้ผู้ใช้ป้อนชื่อของโลจิคัลวอลุ่ม ที่ต้องการเปลี่ยนคุณสมบัติของโลจิคัลวอลุ่ม ซึ่งสามารถใช้เป็นพารามิเตอร์ในฟิลด์ `sm_cmd_hdr.cmd_to_discover_postfix` ของไดอะล็อกถัดไปสำหรับการกำหนดค่าเริ่มต้นฟิลด์รายการ เช่นเดียวกัน ค่าตัวเลือกยังสามารถนำมาใช้เป็นค่าสำหรับฟิลด์ `sm_cmd_opt.cmd_to_list_postfix` ในลำดับถัดมา ซึ่งยังสามารถใช้เป็นค่าของฟิลด์รายการเริ่มต้น ลำดับถัดมาได้โดยตรง ในกรณีนี้ ความสอดคล้องกันแบบโลจิคัลต้องการให้รายการนี้ถูกเลือก ก่อนไดอะล็อก หรือถูกพักค่าคงที่ไว้ อย่างใดอย่างหนึ่ง ขณะที่อยู่ในไดอะล็อก

ออกแบบตัวเลือกเพื่อร้องขอส่วนหนึ่งของข้อมูล จากผู้ใช้ ตัวเลือกจะอยู่ระหว่างเมนูกับไดอะล็อก เมื่อใช้ ตัวเลือกสามารถเก็บเป็นโครงสร้างพร้อมกันเป็นชุด เพื่อรวบรวมส่วนของข้อมูลหลายๆ ส่วน ก่อนที่ไดอะล็อกจะแสดง

ตัวเลือกยังมีพร้อมต์ที่แสดงด้วยภาษาที่ปรับเข้ากับผู้ใช้ และพื้นที่การตอบกลับสำหรับอินพุตของผู้ใช้ หรือรายการที่ป๊อปอัพอย่างใดอย่างหนึ่ง เพื่อเลือกค่า นั่นคือ ฟิลด์คำถามหนึ่งฟิลด์ และฟิลด์คำตอบอีกหนึ่งฟิลด์ โดยทั่วไป ฟิลด์คำถามจะแสดงขึ้น และผู้ใช้ SMIT จะป้อนค่าลงในพื้นที่การตอบกลับ โดยพิมพ์ค่าหรือเลือกค่าจากรายการ หรืออ็อพชัน

หากต้องการกำหนดให้รายการของตัวเลือกให้กับผู้ใช้ ขณะรันไทม์ อ็อบเจ็กต์ตัวเลือกสามารถมีคำสั่งที่เชื่อมโยง (นิยามในฟิลด์ `sm_cmd_opt.cmd_to_list`) ที่แสดงรายการของตัวเลือกที่ถูกต้อง รายการไม่ใช่ฮาร์ดโค้ด แต่ถูกพัฒนาด้วยคำสั่งที่เชื่อมกับ เอาต์พุตมาตรฐาน ผู้ใช้จะได้รับรายการนี้โดยเลือกฟังก์ชัน **F4 (Esc+4)=List** ของอินเตอร์เฟส SMIT

ในตัวเลือก ghost (`sm_cmd_hdr.ghost="y"`) คำสั่งที่กำหนดในฟิลด์ `sm_cmd_opt.cmd_to_list` จะรันโดยอัตโนมัติ หากมีอยู่ หน้าจอตัวเลือกจะไม่แสดง ณ เวลานี้ และผู้ใช้จะมองเห็นรายการป๊อปอัพเท่านั้น

แอ็พพลิเคชันของตัวเลือกแบบ super-ghost จะอนุญาตให้แตกกิ่งที่ตามด้วยการเลือกเมนู ซึ่งกิ่งจะถูกใช้ขึ้นอยู่กับสถานะของระบบ และไม่ใช่อินพุตของผู้ใช้ ในกรณีนี้ descriptor **cmd_to_classify** ในตัวเลือก super-ghost สามารถนำมาใช้เพื่อขอรับข้อมูลที่ต้องการ และเลือกหน้าจอที่ถูกต้องเพื่อแสดงถัดไป

สร้างตัวเลือกโดยกำหนดเมนูในไฟล์ stanza จากนั้นประมวลผลไฟล์ด้วยคำสั่ง **odmadd** หลายๆ เมนู ตัวเลือก และไดอะล็อก สามารถกำหนดได้ในไฟล์เดี่ยว คำสั่ง **odmadd** จะเพิ่มตัวเลือกแต่ละตัว ให้กับที่เก็บอ็อบเจ็กต์ที่ระบุ ไดเร็กทอรี **/usr/lib/objrepos** คือที่เก็บอ็อบเจ็กต์ดีฟอลต์สำหรับข้อมูลระบบ และถูกใช้เพื่อเก็บอ็อบเจ็กต์ที่คอมไพล์แล้วของคุณ ณ รันไทม์ของ SMIT อ็อบเจ็กต์จะเรียกข้อมูลจากฐานข้อมูล SMIT โดยเฉพาะ

หมายเหตุ: สำรองข้อมูลไดเร็กทอรี **/usr/lib/objrepos** เสมอ ก่อนที่จะลบหรือเพิ่มอ็อบเจ็กต์ใดๆ หรือคลาสอ็อบเจ็กต์ใดๆ ความเสียหายที่ไม่ได้คาดคิดไว้ซึ่งเกิดขึ้นกับอ็อบเจ็กต์หรือคลาสที่จำเป็นสำหรับการดำเนินการของระบบ อาจเป็นสาเหตุทำให้เกิดปัญหา เกี่ยวกับระบบได้

หน้าจอไดอะล็อก

ไดอะล็อกใน SMIT คืออินเตอร์เฟสไปยังคำสั่ง หรือภารกิจที่ผู้ใช้ดำเนินการ แต่ละไดอะล็อกจะเรียกทำงานคำสั่งตั้งแต่หนึ่งคำสั่งขึ้นไป ฟังก์ชัน shell หรืออื่นๆ คำสั่งสามารถรันจากจำนวนของไดอะล็อกใดๆ

หากต้องการออกแบบไดอะล็อก คุณจำเป็นต้องรู้สตริงคำสั่ง ที่คุณต้องการสร้างและอ็อพชันคำสั่ง และตัวถูกดำเนินการ ที่คุณต้องการค่าที่ผู้ใช้ระบุ ในไดอะล็อกที่แสดง อ็อพชันและตัวถูกดำเนินการเหล่านี้แต่ละตัว จะแสดงถึงโดยพรอมต์ที่แสดงในภาษาที่ปรับให้เข้ากับผู้ใช้ และพื้นที่การตอบกลับสำหรับอินพุตผู้ใช้ แต่ละอ็อพชันและตัวถูกดำเนินการ จะแสดงโดยอ็อบเจ็กต์อ็อพชันคำสั่งไดอะล็อกที่อยู่ในฐานข้อมูล Object Data Manager (ODM) ไดอะล็อกทั้งหมดจะถูกพักไว้พร้อมกันโดยอ็อบเจ็กต์ส่วนหัวของไดอะล็อก

ผู้ใช้ SMIT จะป้อนค่าลงในพื้นที่การตอบกลับ โดยพิมพ์ค่า หรือเลือกค่าจากรายการหรืออ็อพชัน หากต้องการกำหนดรายการของตัวเลือกให้กับผู้ใช้ขณะรันไทม์ อ็อบเจ็กต์ไดอะล็อกแต่ละตัวสามารถมีคำสั่งที่เชื่อมโยง ซึ่งแสดงตัวเลือกที่ถูกต้อง คำสั่งที่เชื่อมโยงจะถูกกำหนดอยู่ในฟิลด์ `sm_cmd_opt.cmd_to_list` ผู้ใช้จะได้รับรายการนี้โดยเรียกใช้งานฟังก์ชัน **F4 (Esc + 4) =List** ของอินเตอร์เฟส SMIT ซึ่งเป็นสาเหตุทำให้ SMIT รันคำสั่งที่กำหนดไว้ในฟิลด์ `cmd_to_list` ที่เชื่อมโยง และเพื่อใช้เอาต์พุตมาตรฐาน และไฟล์ **stderr** สำหรับการพัฒนารายการ

หน้าจอไดอะล็อกสามารถมีอ็อพชันต่อไปนี้ซึ่งได้กำหนดให้กับรายการต่างๆ:

อ็อพชัน	ฟังก์ชัน
#	หมายถึงค่าตัวเลขที่คาดการณ์ไว้
*	หมายถึงรายการบังคับ
+	หมายถึงการแสดงรายการของตัวเลือกที่สามารถขอรับได้โดยใช้ปุ่ม F4

ในไดอะล็อก ghost หน้าจอไดอะล็อกจะไม่แสดงขึ้น ไดอะล็อกจะรัน หากผู้ใช้กดปุ่ม **Enter** โดยทันทีในหน้าจอไดอะล็อกเพื่อรันไดอะล็อก

สร้างไดอะล็อกโดยกำหนดไดอะล็อกไว้ในไฟล์ stanza จากนั้นประมวลผลไฟล์ด้วยคำสั่ง **odmadd** หลายๆ เมนู ตัวเลือก และไดอะล็อก สามารถกำหนดได้ในไฟล์เดี่ยว คำสั่ง **odmadd** จะเพิ่มนิยามของไดอะล็อกแต่ละนิยาม ให้กับที่เก็บอ็อบเจ็กต์ที่ระบุ

ไดเร็กทอรี **/usr/lib/objrepos** คือที่เก็บอ็อบเจ็กต์ที่เป็นค่าดีฟอลต์สำหรับข้อมูลระบบ และสามารถนำมาใช้เพื่อเก็บอ็อบเจ็กต์ที่คอมไพล์แล้วของคุณได้ ณ รันไทม์ของ SMIT อ็อบเจ็กต์จะเรียกข้อมูลจากฐานข้อมูล SMIT โดยเฉพาะ

หมายเหตุ: สำรองข้อมูลไดเร็กทอรี **/usr/lib/objrepos** เสมอ ก่อนที่จะลบหรือเพิ่มอ็อบเจ็กต์ใดๆ หรือคลาสอ็อบเจ็กต์ใดๆ ความเสียหายที่ไม่ได้คาดคิดไว้ซึ่งเกิดขึ้นกับอ็อบเจ็กต์หรือคลาสที่จำเป็นสำหรับการดำเนินการของระบบ อาจเป็นสาเหตุทำให้เกิดปัญหา เกี่ยวกับระบบได้

คลาสอ็อบเจ็กต์ SMIT

คลาสอ็อบเจ็กต์ System Management Interface Tool (SMIT) จะสร้างด้วย Object Data Manager (ODM) ที่นิยามรูปแบบทั่วไป หรือเร็กคอร์ดชนิดข้อมูลสำหรับอ็อบเจ็กต์แต่ละตัวที่เป็นอินสแตนซ์ของคลาสอ็อบเจ็กต์นั้น

ดังนั้น คลาสอ็อบเจ็กต์ SMIT คือเร็กคอร์ดชนิดข้อมูล และอ็อบเจ็กต์ SMIT คือเร็กคอร์ดเฉพาะของชนิดนั้น

เมนู SMIT, ตัวเลือก, และหน้าจอไดอะล็อก จะอธิบายถึงด้วยอ็อบเจ็กต์ที่เป็นอินสแตนซ์ของหนึ่งในสี่ของคลาสอ็อบเจ็กต์:

- **sm_menu_opt**
- **sm_name_hdr**
- **sm_cmd_hdr**
- **sm_cmd_opt**

ตารางต่อไปนี้แสดงอ็อบเจ็กต์ที่ใช้เพื่อสร้างชนิดหน้าจอแต่ละชนิด :

ชนิดหน้าจอ	คลาสอ็อบเจ็กต์	การใช้ของอ็อบเจ็กต์ (กรณีทั่วไป)
Menu	**sm_menu_opt**	1 สำหรับหัวเรื่องของหน้าจอ
	sm_menu_opt	1 สำหรับรายการแรก
	sm_menu_opt	1 สำหรับรายการที่สอง
	...	...
	sm_menu_opt	หนึ่งสำหรับรายการสุดท้าย
Selector	**sm_name_hdr**	1 สำหรับหัวเรื่องของหน้าจอและแอ็ตทริบิวต์อื่นๆ
	sm_cmd_opt	1 สำหรับฟิลด์รายการหรือรายการป๊อปอัพ
Dialog	**sm_cmd_hdr**	1 สำหรับหัวเรื่องของหน้าจอและสตริงคำสั่ง
	sm_cmd_opt	1 สำหรับฟิลด์รายการ
	sm_cmd_opt	1 สำหรับฟิลด์รายการสำรอง
	...	...
	sm_cmd_opt	1 สำหรับฟิลด์รายการล่าสุด

แต่ละอ็อบเจ็กต์ประกอบด้วยลำดับของฟิลด์ที่มีชื่อ และค่าที่เชื่อมโยง ซึ่งจะแสดงอยู่ในรูปแบบ stanza ในไฟล์ ASCII ที่สามารถใช้ด้วยคำสั่ง **odmadd** เพื่อกำหนดหรือขยายฐานข้อมูล SMIT Stanza ในไฟล์ควรถูกแยกออกด้วยบรรทัดว่าง ตั้งแต่หนึ่งบรรทัดขึ้นไป

หมายเหตุ: ความคิดเห็นในไฟล์อินพุต ODM (ไฟล์ ASCII stanza) ถูกใช้ด้วยคำสั่ง **odmadd** ต้องเป็นคำสั่งเดี่ยวบนบรรทัด ที่ขึ้นต้นด้วย # (เครื่องหมายปอนด์ หรือ * (เครื่องหมายดอกจัน) ในหนึ่งคอลัมน์ เฉพาะความคิดเห็น * (เครื่องหมายดอกจัน) สามารถอยู่บนบรรทัดเดียวกันได้ตามบรรทัดของ stanza และต้องอยู่หลังค่า descriptor

ต่อไปนี้คือตัวอย่างของ stanza สำหรับอ็อบเจ็กต์ **sm_menu_opt** :

```
sm_menu_opt:                    *name of object class
  id             = "top_menu"  *object's (menu screen) name
  id_seq_num     = "050"
  next_id        = "commo"     *id of objects for next menu screen
  text           = "Communications Applications & Services"
  text_msg_file  = ""
  text_msg_set   = 0
  text_msg_id    = 0
  next_type      = "m"         *next_id specified another menu
  alias          = ""
  help_msg_id           = ""
  help_msg_loc          = ""
  help_msg_base         = ""
  help_msg_book         = ""
```

หมายเหตุ *ObjectClass.Descriptor* จะถูกใช้เพื่ออธิบายถึงค่าของฟิลด์ของอ็อบเจ็กต์ สำหรับตัวอย่าง ในอ็อบเจ็กต์ **sm_menu_opt** ที่อยู่ก่อนหน้า ค่าของ **sm_menu_opt.id** คือ top_menu

ต่อไปนี้คือตัวอย่างของ stanza สำหรับอ็อบเจ็กต์ **sm_name_hdr** :

```
sm_name_hdr:                     *---- used for selector screens
  id                      = ""  *the name of this selector screen
  next_id                 = ""  *next sm_name_hdr or sm_cmd_hdr
  option_id               = ""  *specifies one associated sm_cmd_opt
  has_name_select         = ""
  name                    = "" *title for this screen
  name_msg_file         = ""
  name_msg_id           = 0
  type                    = ""
  ghost                   = ""
  cmd_to_classify         = ""
  cmd_to_classify_postfix = ""
  raw_field_name          = ""
  cooked_field_name       = ""
  next_type               = ""
  help_msg_id           = ""
  help_msg_loc          = ""
  help_msg_base         = ""
  help_msg_book         = ""
```

ต่อไปนี้คือตัวอย่างของ stanza สำหรับอ็อบเจ็กต์**sm_cmd_hdr** :

```
sm_cmd_hdr:                      *---- used for dialog screens
  id                      = ""  *the name of this dialog screen
  option_id               = "" *defines associated set of sm_cmd_opt objects
  has_name_select         = ""
  name                    = "" *title for this screen
  name_msg_file         = ""
  name_msg_set          = 0
```

```
  name_msg_id           = 0
  cmd_to_exec             = ""
  ask                     = ""
  exec_mode               = ""
  ghost                   = ""
  cmd_to_discover         = ""
  cmd_to_discover_postfix = ""
  name_size               = 0
  value_size              = 0
  help_msg_id           = ""
  help_msg_loc          = ""
  help_msg_base         = ""
  help_msg_book         = ""
```

ต่อไปนี้คือตัวอย่างของ stanza สำหรับอ็อบเจ็กต์ **sm_cmd_opt** :

```
sm_cmd_opt:                 *---- used for selector and dialog screens
  id                    = ""  *name of this object
  id_seq_num            = ""  *"0" if associated with selector screen
  disc_field_name       = ""
  name                  = ""  *text describing this entry
  name_msg_file         = ""
  name_msg_set          = 0
  name_msg_id           = 0
  op_type               = ""
  entry_type            = ""
  entry_size            = 0
  required              = ""
  prefix                = ""
  cmd_to_list_mode      = ""
  cmd_to_list           = ""
  cmd_to_list_postfix   = ""
  multi_select          = ""
  value_index           = 0
  disp_values           = ""
  values_msg_file       = ""
  values_msg_set        = 0
  values_msg_id         = 0
  aix_values            = ""
  help_msg_id           = ""
  help_msg_loc          = ""
  help_msg_base         = ""
  help_msg_book         = ""
```

อ็อบเจ็กต์ SMIT ทั้งหมดมีฟิลด์ id ที่จัดเตรียมชื่อที่ถูกใช้เพื่อมองหาอ็อบเจ็กต์นั้น อ็อบเจ็กต์ **sm_menu_opt** จะถูกใช้สำหรับหัวเรื่องของเมนู และยังถูกค้นหาได้โดยใช้ฟิลด์ next_id อ็อบเจ็กต์ **sm_menu_opt** และ **sm_name_hdr** ยังมีฟิลด์ next_id ที่ชี้ไปยังฟิลด์ id ของอ็อบเจ็กต์อื่นๆ เหล่านี้คือวิธีการลิงก์ระหว่างหน้าจอที่แสดงอยู่ในฐานข้อมูล SMIT เช่นเดียวกัน มีฟิลด์ option_id ในอ็อบเจ็กต์ **sm_name_hdr** และ **sm_cmd_hdr** ที่ชี้ไปยังฟิลด์ id ของอ็อบเจ็กต์ **sm_cmd_opt** ที่เชื่อมโยง

หมายเหตุ: ฟิลด์อ็อบเจ็กต์ **sm_cmd_hdr.option_id** จะเท่ากับฟิลด์อ็อบเจ็กต์ **sm_cmd_opt.id** แต่ละฟิลด์ ซึ่งกำหนดลิงก์ระหว่างอ็อบเจ็กต์ **sm_cmd_hdr** และอ็อบเจ็กต์ **sm_cmd_opt** ที่เชื่อมโยง

ไดอะล็อกตั้งแต่สองไดอะล็อกขึ้นไปสามารถแบ่งใช้อ็อบเจ็กต์ **sm_cmd_opt** เนื่องจาก SMIT ใช้ตัวดำเนินการ ODM **LIKE** เพื่อมองหาอ็อบเจ็กต์ด้วยค่าฟิลด์ sm_cmd_opt.id ที่เหมือนกัน SMIT อนุญาตให้ใช้ได้สูงสุดห้า ID (คั่นด้วยเครื่องหมายจุลภาค) ที่ต้องการระบุในฟิลด์ sm_cmd_hdr.option_id ดังนั้น อ็อบเจ็กต์ **sm_cmd_opt** ที่มีค่าฟิลด์ sm_cmd_opt.id ใดๆ ที่แตกต่างกันห้าจุดสามารถเชื่อมโยงกับอ็อบเจ็กต์ **sm_cmd_hdr**

ตารางต่อไปนี้แสดงวิธีที่ค่าของฟิลด์ sm_cmd_hdr.option_id เกี่ยวข้องกับค่าของฟิลด์ sm_cmd_opt.id และ sm_cmd_opt.id_seq_num

> **หมายเหตุ:** ค่าในฟิลด์ sm_cmd_opt.id_seq_num จะถูกใช้เพื่อเรียงลำดับอ็อบเจ็กต์ที่ดึงข้อมูลสำหรับการแสดงผลหน้าจอ

ID ของอ็อบเจ็กต์เพื่อดึงข้อมูล (sm_cmd_hdr.option_id)	อ็อบเจ็กต์ที่ถูกดึง (sm_cmd_opt.id)	แสดงลำดับของอ็อบเจ็กต์ที่ถูกดึงข้อมูล (sm_cmd_opt.id_seq_num)
″demo.[AB]″	″demo.A″	″10″
	″demo.B″	″20″
	″demo.A″	″30″
	″demo.A	″40″
″demo.[ACD]″	″demo.A″	″10″
	″demo.C″	″20″
	″demo.A″	″30″
	″demo.A″	″40″
	″demo.D″	″50″
″demo.X,demo.Y,demo.Z″	″demo.Y″	″20″
	″demo.Z″	″40″
	″demo.X″	″60″
	″demo.X″	″80″

ฐานข้อมูล SMIT

อ็อบเจ็กต์ SMIT จะถูกสร้างด้วยตัวช่วยสร้าง ODM และเก็บอยู่ในไฟล์ในฐานข้อมูลที่กำหนดไว้ ดีฟอลต์ฐานข้อมูล SMIT ประกอบด้วยไฟล์ทั้งหมดแปดไฟล์ :

- **sm_menu_opt**
- **sm_menu_opt.vc**
- **sm_name_hdr**
- **sm_name_hdr.vc**
- **sm_cmd_hdr**
- **sm_cmd_hdr.vc**
- **sm_cmd_opt**

- **sm_cmd_opt.vc**

ไฟล์จะถูกเก็บตามค่าดีฟอลต์ในไดเร็กทอรี **/usr/lib/objrepos** ซึ่งควรบันทึกและเรียกคืนพร้อมกันได้

นามแฝง SMIT และพาธด่วน

ส่วนนี้อธิบายเกี่ยวกับนามแฝง SMIT และพาธด่วน

อ็อบเจ็กต์ System Management Interface Tool (SMIT) **sm_menu_opt** สามารถนำมาใช้เพื่อกำหนดวิธีลัด เมื่อป้อนด้วยคำสั่ง **smit** เพื่อเริ่มต้น SMIT ซึ่งสามารถนำพาผู้ใช้ไปยังเมนู ตัวเลือก หรือไดอะล็อกได้โดยตรง นามแฝงของอ็อบเจ็กต์ไม่เคยแสดงให้เห็น การใช้วิธีลัดอนุญาตให้ผู้ใช้ผ่านเมนูหลัก SMIT และอ็อบเจ็กต์อื่นๆ ที่อยู่ในอินเตอร์เฟส SMIT พาธไปยังเมนู ตัวเลือก หรือไดอะล็อกนั้น หมายเลขของวิธีลัดใดๆ สามารถชี้ไปยังเมนู ตัวเลือก หรือไดอะล็อกเดียวกันได้

อ็อบเจ็กต์ **sm_menu_opt** ถูกนำมาใช้เพื่อกำหนดวิธีลัดโดยตั้งค่าฟิลด์ `sm_menu_opt.alias` ให้มีค่า "y" ในกรณีนี้ อ็อบเจ็กต์ **sm_menu_opt** จะถูกนำมาใช้เพื่อกำหนดวิธีลัดโดยเฉพาะ วิธีลัด หรือชื่อนามแฝงจะถูกระบุด้วยค่าที่อยู่ในฟิลด์ `sm_menu_opt.id` เนื้อหาของฟิลด์ `sm_menu_opt.next_id` จะชี้ไปยังอ็อบเจ็กต์เมนู อ็อบเจ็กต์ส่วนหัวของตัวเลือก หรืออ็อบเจ็กต์ส่วนหัวของไดอะล็อกอื่นๆ ซึ่งขึ้นอยู่กับว่า ค่าของฟิลด์ `sm_menu_opt.next_type` คือค่า "m" (เมนู) "n" (ตัวเลือก) หรือ "d" (ไดอะล็อก)

อ็อบเจ็กต์ **sm_menu_opt** ที่ไม่มีนามแฝงสำหรับหัวเรื่องของเมนู (`next_type="m"`) ควรมีค่าเฉพาะของฟิลด์ `sm_menu_opt.next_id` เนื่องจากฟิลด์นี้จะถูกใช้เป็นวิธีลัดโดยอัตโนมัติ

ถ้าคุณต้องการให้สองรายการเมนูชี้ไปยังเมนูตัวสืบทอดเดียวกัน หนึ่งฟิลด์ `next_id` ควรชี้ไปยังนามแฝง ซึ่งจะชี้ไปยังเมนูตัวสืบทอดตามลำดับ

สร้างนามแฝงและวิธีลัดโดยนินามไว้ในไฟล์ stanza จากนั้นประมวลผลไฟล์ด้วยคำสั่ง **odmadd** หลายๆ เมนู ตัวเลือก และไดอะล็อก สามารถกำหนดได้ในไฟล์เดี่ยว คำสั่ง **odmadd** จะเพิ่มนิยามนามแฝงแต่ละชื่อให้กับที่เก็บอ็อบเจ็กต์ที่ระบุไว้ไดเร็กทอรี **/usr/lib/objrepos** คือที่เก็บอ็อบเจ็กต์ที่เป็นค่าดีฟอลต์สำหรับข้อมูลระบบ และสามารถนำมาใช้เพื่อเก็บอ็อบเจ็กต์ที่คอมไพล์แล้วของคุณได้ ณ รันไทม์ของ SMIT อ็อบเจ็กต์จะเรียกข้อมูลจากฐานข้อมูล SMIT โดยเฉพาะ

> **หมายเหตุ:** คุณควรสำรองข้อมูลไดเร็กทอรี **/usr/lib/objrepos** ก่อนที่จะลบหรือเพิ่มอ็อบเจ็กหรือคลาสอ็อบเจ็กต์ใดๆ ความเสียหายที่ไม่ได้คาดคิดไว้ซึ่งเกิดขึ้นกับอ็อบเจ็กต์หรือคลาสที่จำเป็นสำหรับการดำเนินการของระบบ อาจเป็นสาเหตุทำให้เกิดปัญหา เกี่ยวกับระบบได้

SMIT information command descriptors

System Management Interface Tool (SMIT) สามารถใช้ descriptor ต่างๆ ที่กำหนดไว้ในอ็อบเจ็กต์เพื่อขอรับข้อมูล เช่น ค่าเวลาที่ใช้ในการรันปัจจุบัน ซึ่งจำเป็นต้องใช้เพื่อดำเนินการผ่านโครงสร้างอินเตอร์เฟส SMIT

descriptor เหล่านี้แต่ละตัวจะถูกกำหนดรูปแบบของสตริงคำสั่งบางส่วน เพื่อรันและเรียกข้อมูลที่ต้องการ

descriptor ที่สามารถตั้งค่าคำสั่งสำหรับการค้นพบของข้อมูลที่ต้องการคือ :

- descriptor **cmd_to_discover** ที่เป็นส่วนหนึ่งของคลาสอ็อบเจ็กต์ **sm_cmd_hdr** ซึ่งใช้เพื่อกำหนดส่วนหัวของไดอะล็อก
- descriptor **cmd_to_classify** ที่เป็นส่วนหนึ่งของคลาสอ็อบเจ็กต์ **sm_name_hdr** จะถูกนำมาใช้เพื่อกำหนดส่วนหัวของตัวเลือก

- descriptor **cmd_to_list** ที่เป็นส่วนหนึ่งของคลาสอ็อบเจ็กต์ **sm_cmd_opt** ถูกนำมาใช้เพื่อกำหนดรายการอ็อพชันของตัวเลือกที่เชื่อมโยงกับ รายการอ็อพชันคำสั่งไดอะล็อกที่เชื่อมโยงกับฟิลด์รายการไดอะล็อก

SMIT จะเรียกใช้สตริงคำสั่งที่ระบุโดย descriptor **cmd_to_list, cmd_to_classify** หรือ **cmd_to_discover** โดยสร้าง child process ในครั้งแรก ข้อผิดพลาดมาตรฐาน (strerr) และเอาต์พุตของ child process จะเปลี่ยนทิศทางใหม่ไปยัง SMIT ผ่านไพพ์ SMIT ถัดไปจะเรียกใช้งานรูทีนย่อย **setenv(**″**ENV=**″**)** ใน child process เพื่อป้องกันคำสั่งที่ระบุอยู่ในไฟล์ **$HOME/.env** ของผู้ใช้จากการันแบบอัตโนมัติ เรียกใช้งาน shell ใหม่ ท้ายสุด SMIT จะเรียกรูทีนย่อยของระบบ **execl** เพื่อเริ่มต้น **ksh** โดยใช้สตริงคำสั่งเป็นค่าพารามิเตอร์ **ksh –c** ถ้าค่าออกไม่ใช่ 0 SMIT จะแจ้งเตือนผู้ใช้ที่คำสั่งล้มเหลว

SMIT จะสร้างชื่อพาธของไฟล์บันทึกการทำงาน และค่าติดตั้งของบรรทัดรับคำสั่ง **verbose, trace** และแฟล็ก **debug** ที่พร้อมใช้งานในสภาวะแวดล้อม shell ของคำสั่งที่รัน ค่าเหล่านี้จะจัดเตรียมไว้ผ่านตัวแปรสภาวะแวดล้อมต่อไปนี้ :

- **_SMIT_LOG_FILE**
- **_SMIT_SCRIPT_FILE**
- **_SMIT_VERBOSE_FLAG**
- **_SMIT_TRACE_FLAG**
- **_SMIT_DEBUG_FLAG**

การมีอยู่หรือไม่มีอยู่ของแฟล็กที่สอดคล้องกันจะถูกบ่งชี้ด้วยค่า 0 หรือ 1 ตามลำดับ

วิธีง่ายๆ เพื่อดูค่าติดตั้งปัจจุบันคือ การเรียกใช้ฟังก์ชัน shell หลังจากที่เริ่มต้น SMIT จากนั้นรันสตริงคำสั่ง **env | grep _SMIT**

ค่าทั้งหมดที่เขียนลงไฟล์บันทึกการทำงานควรผสานกัน และควรตามด้วยการล้างข้อมูล ยกเว้นแต่ว่า การดำเนินการนี้เกิดขึ้นโดยอัตโนมัติ

cmd_to_discover descriptor

เมื่อ SMIT สร้างไดอะล็อกขึ้น SMIT จะขอรับอ็อบเจ็กต์ **sm_cmd_hdr** (ส่วนหัวของไดอะล็อก) และเนื้อความของไดอะล็อกที่เชื่อมโยง (อ็อบเจ็กต์ **sm_cmd_opt** ตั้งแต่หนึ่งตัวขึ้นไป) จากที่เก็บอ็อบเจ็กต์ อย่างไรก็ตาม อ็อบเจ็กต์ **sm_cmd_opt** ยังสามารถกำหนดค่าเริ่มต้นด้วยค่ารันไทม์ปัจจุบัน ถ้าฟิลด์ `sm_cmd_hdr.cmd_to_discover` ไม่ว่าง (″″) SMIT จะรันคำสั่งที่ระบุอยู่ในฟิลด์เพื่อขอรับค่ารันไทม์ปัจจุบัน

สตริงคำสั่ง **ksh** ที่ถูกต้องใดๆ สามารถนำมาใช้เป็นค่า descriptor **cmd_to_discover** ได้ คำสั่งควรสร้างรูปแบบของเอาต์พุตต่อไปนี้ ซึ่งเป็นเอาต์พุตมาตรฐาน:

```
#name_1:name_2: ... :name_n\n
value_1:value_2: ... :value_n
```

ในเอาต์พุตมาตรฐานของคำสั่ง อักขระแรกจะเป็น # (เครื่องหมายปอนด์) เสมอ \n (อักขระบรรทัดใหม่) จะใช้เพื่อแยกบรรทัดสำหรับชื่อออกจาก บรรทัดสำหรับค่า ชื่อและค่าหลายค่าจะถูกคั่นด้วย : (โคลอน) และชื่อหรือค่าใดๆ สามารถเป็นสตริงเปล่า (ในรูปแบบเอาต์พุต จะปรากฏขึ้นเป็นสองโคลอนโดยไม่มีช่องว่างคั่นระหว่างชื่อหรือค่านั้น) SMIT จะรักษาค่าภายในปัจจุบันที่ตั้งค่าในรูปแบบนี้ ซึ่งจะถูกใช้เพื่อส่งผ่านคู่ของค่าสำหรับชื่อ จากหน้าจอหนึ่งไปยังหน้าจอถัดไป

> **หมายเหตุ:** ถ้าค่าสอดแทรก : (โคลอน) ไว้ : ต้องอยู่นำหน้า #! (เครื่องหมายปอนด์ เครื่องหมายอัศเจรีย์) มิฉะนั้น SMIT จะอาน : (โคลอน) ในฐานะตัวคั่นฟิลด์

เมื่อ SMIT รันคำสั่งที่ระบุในฟิลด์ cmd_to_discover แล้ว SMIT จะดักจับ stdout ของคำสั่งและโหลดคู่ของค่าสำหรับชื่อ (name_1 and value_1 name_2 and value_2 และอื่นๆ) ลงใน descriptor **disp_values** และ **aix_values** ของอ็อบเจ็กต์ **sm_cmd_opt** (อ็อพชันคำสั่งไดอะล็อก) โดยจับคู่ชื่อกับ descriptor **sm_cmd_opt.disc_field_name** ในอ็อบเจ็กต์ **sm_cmd_opt**

สำหรับอ็อบเจ็กต์ **sm_cmd_opt** (อ็อพชันคำสั่งไดอะล็อก) ที่แสดงค่าจากตัวเลือกที่อยู่ก่อนหน้า descriptor **disc_field_name** สำหรับอ็อบเจ็กต์ของอ็อพชันคำสั่งไดอะล็อกต้องถูกตั้งค่าเป็น "_rawname" or "_cookedname" (หรือชื่อสำรองต้องถูกใช้เพื่อแทนทับชื่อดีฟอลต์) เพื่อบ่งชี้ถึงค่าที่ต้องการใช้ ในกรณีนี้ descriptor **disc_field_name** ของอ็อบเจ็กต์ **sm_cmd_opt** (อ็อพชันคำสั่งไดอะล็อก) ควรเป็นฟิลด์ที่ไม่มีรายการ ถ้าค่าเฉพาะควรส่งผ่านไปยังคำสั่ง descriptor **required** สำหรับอ็อบเจ็กต์ **sm_cmd_opt** (อ็อพชันคำสั่งไดอะล็อก) ต้องตั้งค่าเป็น y (ใช่) หรือหนึ่งในค่าสำรองอื่นๆ

ในกรณีของการกำหนดค่าเริ่มต้นให้กับฟิลด์อ็อพชัน อนุญาตให้ค่าปัจจุบันสำหรับ descriptor **cmd_to_discover** (นั่นคือ คู่ของค่าสำหรับชื่อใดๆ จากชุดของค่าปัจจุบันของไดอะล็อก) ของฟิลด์รายการ เพื่อระบุค่าที่ได้ถูกกำหนดไว้ล่วงหน้าเพื่อใช้เป็นค่าดีฟอลต์ หรือค่าเริ่มต้นสำหรับฟิลด์รายการที่สอดคล้องกัน ณ เวลาของการกำหนดค่าเริ่มต้นสำหรับไดอะล็อก เมื่อฟิลด์รายการไดอะล็อกตรงกับชื่อในชุดของค่าปัจจุบันของไดอะล็อก (ผ่าน **sm_cmd_opt.disc_field_name**) การตรวจสอบจะถูกทำขึ้นเพื่อกำหนดว่าเป็นอ็อพชันสำหรับฟิลด์ (**sm_cmd_opt.op_type = "r"**) และมีค่าที่ได้ถูกกำหนดไว้ก่อน (**sm_cmd_opt.aix_values !=** "") หรือไม่ ถ้าใช่ ชุดของค่าอ็อพชันนี้จะถูกนำมาเปรียบเทียบกับค่าปัจจุบันสำหรับ **disc_field_name** จากชุดของค่าปัจจุบัน ถ้าพบการจับคู่ ค่าของอ็อพชันที่ถูกจับคู่จะกลายเป็นค่าดีฟอลต์ **sm_cmd_opt.value_index** ซึ่งจะถูกตั้งค่าไว้ในดัชนี) ค่าที่ถูกแปลแล้วซึ่งสอดคล้องกัน (**sm_cmd_opt.disp_values**) จะแสดงขึ้น ถ้ามีอยู่ ถ้าไม่พบการจับคู่ ข้อผิดพลาดจะถูกรายการ และค่าปัจจุบันจะกลายเป็นค่าดีฟอลต์และเป็นค่าเดียวสำหรับริง

ในหลายๆ กรณี คำสั่ง discovery จะมีอยู่แล้ว สำหรับอุปกรณ์ และพื้นที่หน่วยเก็บ แบบอย่างของการเพิ่ม ลบ เปลี่ยน และแสดงจะมีอยู่ ตัวอย่างเช่น หากต้องการเพิ่ม (**mk**) ไดอะล็อกจำเป็นต้องมีเพื่อหาคุณสมบัติ ไดอะล็อกสามารถมีคำสั่ง discovery ซึ่งเป็นคำสั่ง show (**ls**) ที่มีพารามิเตอร์ที่ร้องขอค่าดีฟอลต์ SMIT ใช้เอาต์พุตมาตรฐานของคำสั่ง show (**ls**) เพื่อกรอกค่าดีฟอลต์ที่แนะนำ อย่างไรก็ตาม สำหรับอ็อบเจ็กต์ที่มีค่าดีฟอลต์ ซึ่งเป็นค่าคงที่ที่รู้จักกันในช่วงของการพัฒนา (นั่นคือ ค่าที่ไม่ได้อ้างอิงตามสถานะปัจจุบันของเครื่องที่กำหนดไว้) ค่าดีฟอลต์อาจถูกกำหนดค่าเริ่มต้นในเร็กคอร์ดไดอะล็อกของตนเองได้ ในกรณีนี้ อาจไม่ต้องการ **cmd_to_discover** จากนั้น ไดอะล็อกจะแสดงขึ้น เมื่อฟิลด์ทั้งหมดถูกกรอกข้อมูลแล้ว และไดอะล็อกถูก commit แล้ว คำสั่ง add (**mk**) จะถูกเรียกใช้งาน

สำหรับตัวอย่างอื่น การเปลี่ยนไดอะล็อก (**ch**) สามารถมีได้โดยมีค่าเป็นคำสั่ง discovery นั่นคือ คำสั่ง show (**ls**) เพื่อขอรับค่าปัจจุบันสำหรับอินสแตนซ์ที่ได้กำหนดไว้ เช่น อุปกรณ์เฉพาะ SMIT ใช้เอาต์พุตมาตรฐานของคำสั่ง of the show (**ls**) เพื่อกรอกค่าก่อนที่จะแสดงไดอะล็อก คำสั่ง show (**ls**) ที่ใช้สำหรับการค้นพบในอินสแตนซ์นี้สามารถเป็นตัวเดียวกันกับที่ใช้สำหรับการค้นพบ ในตัวอย่างคำสั่ง add (**mk**) ยกเว้น ชุดอื่นๆ ของอ็อพชัน

cmd_to_*_postfix descriptors

การเชื่อมโยงกับเหตุการณ์ที่เกิดขึ้นของ descriptor **cmd_to_discover, cmd_to_classify** หรือ **cmd_to_list** คือ descriptor สำรองที่นิยาม postfix สำหรับสตริงคำสั่งที่ได้กำหนดไว้โดย descriptor**cmd_to_discover, cmd_to_classify** หรือ **cmd_to_list** postfix คือสตริงอักขระที่นิยามแฟล็กและพารามิเตอร์ซึ่งถูกผนวกเข้ากับคำสั่ง ก่อนที่จะเรียกใช้งาน

descriptor ที่สามารถนำมาใช้เพื่อนิยาม postfix ที่ต้องการผนวกเข้ากับคำสั่งคือ:

- descriptor **cmd_to_discover_postfix** ที่นิยาม postfix สำหรับ descriptor **cmd_to_discover** ในอ็อบเจ็กต์ **sm_cmd_hdr** ที่นิยามส่วนหัวของไดอะล็อก

- descriptor **cmd_to_classify_postfix** ที่กำหนด postfix สำหรับ descriptor **cmd_to_classify** ในอ็อบเจ็กต์ **sm_name_hdr** ที่นิยามส่วนหัวของตัวเลือก
- descriptor **cmd_to_list_postfix** ที่นิยาม postfix สำหรับ descriptor **cmd_to_list** ในอ็อบเจ็กต์ **sm_cmd_opt** ที่นิยามฟิลด์รายการของตัวเลือกที่เชื่อมโยงกับตัวเลือกหรือฟิลด์รายการไดอะล็อก ที่เชื่อมโยงกับไดอะล็อก

ต่อไปนี้คือตัวอย่างของวิธีการใช้ descriptor postfix เพื่อระบุแฟล็กและค่าพารามิเตอร์ * (เครื่องหมายดอกจัน) ในตัวอย่างอาจเป็น `list`, `classify` หรือ `discover` ก็ได้

สมมุติว่า `cmd_to_*` มีค่าเท่ากับ `"DEMO -a"` และ `cmd_to_*_postfix` มีค่าเท่ากับ `"-l _rawname -n stuff -R _cookedname"` และค่าปัจจุบันจะมีค่าเป็น:

```
#name1:_rawname:_cookedname::stuff\n
value1:gigatronicundulator:parallel:xxx:47
```

จากนั้น สตริงคำสั่งที่สร้างขึ้นจะมีค่าเป็น:

```
DEMO -a -l 'gigatronicundulator' -n '47' -R 'parallel'
```

การครอบด้วย `''` (เครื่องหมายอัญประกาศเดี่ยว) สามารถครอบค่า descriptor postfix ได้ เพื่ออนุญาตให้จัดการกับค่าพารามิเตอร์ ด้วยพื้นที่ที่ฝังตรึงไว้

การสร้างและการประมวลผลคำสั่ง SMIT

ไดอะล็อกแต่ละตัวใน System Management Interface Tool (SMIT) จะสร้างและเรียกใช้งานเวอร์ชันของคำสั่งมาตรฐาน

คำสั่งที่ต้องการเรียกใช้งานโดยไดอะล็อกจะถูกกำหนดโดย descriptor **cmd_to_exec** ในอ็อบเจ็กต์ **sm_cmd_hdr** ที่กำหนดส่วนหัวของไดอะล็อก

การสร้างงานที่ไดอะล็อกกำหนด

สำหรับการสร้างคำสั่งที่กำหนดไว้ใน descriptor **sm_cmd_hdr.cmd_to_exec** SMIT จะใช้การส่งค่าสองที่สแกนผ่านชุดไดอะล็อกของอ็อบเจ็กต์ **sm_cmd_opt** เพื่อเก็บรวบรวมค่าที่นำหน้า และค่าพารามิเตอร์ ค่าพารามิเตอร์ที่เก็บรวบรวมจะสอดแทรกพารามิเตอร์เหล่านั้นที่ผู้ใช้เปลี่ยนจากค่าที่แสดงในตอนต้น และพารามิเตอร์พร้อมกับ descriptor **sm_cmd_opt.required** ที่ตั้งค่าเป็น `"y"`

การส่งค่าแรกจะเก็บรวบรวมค่าทั้งหมดของอ็อบเจ็กต์ **sm_cmd_opt** (ตามลำดับ) ที่ descriptor **prefix** คือสตริงเปล่า (`""`) หรือสตริงที่ขึ้นต้นด้วย - (เครื่องหมายลบ) พารามิเตอร์เหล่านี้ไม่ใช่พารามิเตอร์ที่คำนึงถึงตำแหน่ง และจะถูกเพิ่มในทันทีที่ด้านหลังชื่อคำสั่ง พร้อมกับเนื้อหาของ descriptor **prefix** สำหรับพารามิเตอร์

การส่งค่าที่สองจะเก็บรวบรวมค่าทั้งหมดของอ็อบเจ็กต์ **sm_cmd_opt** ที่เหลืออยู่ (ตามลำดับ) ที่ descriptor **prefix** คือ – (เส้นประสองเส้น) พารามิเตอร์เหล่านี้คือพารามิเตอร์ที่คำนึงถึงตำแหน่ง และจะถูกเพิ่มหลังจากที่อ็อพชันที่ถูกแฟล็ก เก็บรวบรวมการส่งค่าแรกแล้ว

> **หมายเหตุ:** SMIT จะเรียกใช้ค่าที่คุณป้อนลงในฟิลด์คำนำหน้า ถ้าค่าในฟิลด์คำนำหน้าคืออักขระเชลล์ที่สงวนไว้ ตัวอย่างเช่น * (เครื่องหมายดอกจัน) คุณต้องระบุอักขระที่ตามมาด้วย —' (เส้นประ เส้นประ เครื่องหมายอัญประกาศเดี่ยว) จากนั้น เมื่อระบบประเมินผลอักขระ จะไม่มีข้อผิดพลาดสำหรับอักขระเชลล์

ค่าพารามิเตอร์คำสั่งในไดอะล็อกจะถูกกรอกข้อมูลโดยอัตโนมัติ เมื่อ descriptor **disc_field_name** ของอ็อบเจ็กต์ **sm_cmd_opt** ตรงกับชื่อของค่าที่สร้างขึ้นโดยตัวเลือกที่มาก่อน หรือคำสั่ง discovery ที่มาก่อน ค่าพารามิเตอร์เหล่านี้จะเป็นค่าดีฟอลต์ และตามปกติแล้ว จะไม่ถูกเพิ่มลงในบรรทัดรับคำสั่ง การกำหนดค่าเริ่มต้น descriptor **sm_cmd_opt.required** ให้มีค่าเป็น "y" หรือ "+" อาจเป็นสาเหตุทำให้ค่าเหล่านี้ถูกเพิ่มลงในบรรทัดรับคำสั่ง เมื่อค่าเหล่านั้นไม่ถูกเปลี่ยนแปลงในไดอะล็อก ถ้าค่า descriptor **sm_cmd_opt.required** คือ "?" ค่าที่สอดคล้องกันจะถูกนำมาใช้เท่านั้น หากฟิลด์รายการที่เชื่อมโยงไม่ใช่ค่าว่าง ค่าพารามิเตอร์เหล่านั้นจะถูกสร้างลงในบรรทัดรับคำสั่ง ซึ่งเป็นส่วนหนึ่งของกระบวนการส่งค่าสองค่าตามปกติ

การนำหน้าและการตามหลังด้วยพื้นที่ว่าง (ช่องว่างและแท็บ) จะถูกลบออกจากค่าพารามิเตอร์ ยกเว้นเมื่อ descriptor **sm_cmd_opt.entry_type** ถูกตั้งค่าเป็น "r" ถ้าค่าพารามิเตอร์ของผลลัพธ์คือสตริงเปล่า จะไม่มีการดำเนินการใดๆ ที่ถูกนำมาใช้ เว้นแต่ descriptor **sm_cmd_opt.prefix** จะขึ้นต้นด้วยแฟล็กอ็อพชัน การล้อมรอบด้วยเครื่องหมายอัญประกาศเดี่ยวจะถูกเพิ่มลงในค่าพารามิเตอร์หาก descriptor **prefix** ไม่ได้ตั้งค่าเป็น "–" (เส้นประสองเส้น) พารามิเตอร์แต่ละตัวจะถูกวางอยู่หลังคำนำหน้าที่เชื่อมโยงในทันที โดยไม่มีช่องว่างระหว่างกลาง และหาก descriptor **multi_select** มีค่า "m" โทเค็นที่ถูกคั่นด้วยพื้นที่ว่างในฟิลด์รายการจะถูกใช้เป็นตัวคั่นพารามิเตอร์

การประมวลผลงานที่ไดอะล็อกกำหนด

SMIT จะรันสตริงคำสั่งที่ระบุใน descriptor **sm_cmd_hdr.cmd_to_exec** โดยสร้าง child process เป็นอันดับแรก ข้อผิดพลาดมาตรฐานและเอาต์พุตมาตรฐานของ child process จะถูกจัดการตามที่ระบุไว้โดยเนื้อหาของ descriptor **sm_cmd_hdr.exec_mode** SMIT ถัดไปจะรันรูทีนย่อย **setenv("ENV=")** ใน child process เพื่อป้องกันคำสั่งที่ระบุในไฟล์ **$HOME/.env** ของผู้ใช้จากการรันโดยอัตโนมัติ เมื่อเชลล์ใหม่ถูกเรียกใช้งาน ท้ายสุด SMIT จะเรียกรูทีนย่อย **execl** เพื่อเริ่มต้นเชลล์ **ksh** โดยใช้สตริงคำสั่งเป็นค่าพารามิเตอร์ **ksh -c**

SMIT จะสร้างชื่อพาธของไฟล์บันทึกการทำงาน และค่าติดตั้งของถ้อยคำบรรทัดรับคำสั่ง trace และแฟล็กดีบักที่พร้อมใช้งานในสภาวะแวดล้อมเชลล์ของคำสั่งที่ทำงาน ค่าเหล่านี้จะถูกจัดเตรียมไว้ พร้อมกับตัวแปรสภาวะแวดล้อมต่อไปนี้:

- **_SMIT_LOG_FILE**
- **_SMIT_SCRIPT_FILE**
- **_SMIT_VERBOSE_FLAG**
- **_SMIT_TRACE_FLAG**
- **_SMIT_DEBUG_FLAG**

การมีอยู่หรือไม่มีอยู่ของแฟล็กที่สอดคล้องกันจะถูกบ่งชี้ด้วยค่า 0 หรือ 1 ตามลำดับ

นอกจากนั้น ตัวแปรสภาวะแวดล้อม **SMIT** จัดเตรียมข้อมูลเกี่ยวกับสภาวะแวดล้อม SMIT ที่แอ็คทีฟ ตัวแปรสภาวะแวดล้อม **SMIT** สามารถมีค่าต่อไปนี้ได้:

ค่า	สภาวะแวดล้อม SMIT
a	SMIT ในอินเตอร์เฟส ASCII
d	SMIT ในอินเตอร์เฟส Distributed SMIT (DSMIT)
m	SMIT ในอินเตอร์เฟสหน้าต่าง (ซึ่งยังเรียกว่า Motif)

วิธีง่ายๆ เพื่อดูค่าติดตั้งปัจจุบันคือ การเรียกใช้ฟังก์ชันเชลล์หลังจากที่เริ่มต้น SMIT จากนั้นรันสตริงคำสั่ง **env | grep SMIT**

คุณสามารถปิดใช้งานฟังก์ชันคีย์ F9=Shell โดยตั้งค่าตัวแปรสภาวะแวดล้อม **SMIT_SHELL=n**

ค่าทั้งหมดที่เขียนลงไฟล์บันทึกการทำงานควรต่อท้าย และควรตามด้วยการล้างข้อมูล ซึ่งการดำเนินการนี้ไม่ได้เกิดขึ้นโดยอัตโนมัติ

คุณสามารถแทนที่ค่าดีฟอลต์การเปลี่ยนทิศทางเอาต์พุต SMIT ของกระบวนการภารกิจ (child) โดยตั้งค่าฟิลด์ `sm_cmd_hdr.exec_mode` ให้มีค่า `"i"` ค่าติดตั้งนี้กำหนดให้การจัดการเอาต์พุตควบคุมภารกิจ เนื่องจากกระบวนการภารกิจจะสืบทอดข้อผิดพลาดมาตรฐานอย่างง่ายๆ และ descriptor ไฟล์เอาต์พุตมาตรฐาน

คุณสามารถทำให้ SMIT ปิดระบบ และแทนที่ SMIT ด้วยภารกิจเป้าหมายโดยตั้งค่าฟิลด์ `sm_cmd_hdr.exec_mode` ให้มีค่า `"e"`

การเพิ่มงานเข้ากับฐานข้อมูล SMIT

ขณะที่พัฒนาอ็อบเจ็กต์ใหม่สำหรับฐานข้อมูล System Management Interface Tool (SMIT) ขอแนะนำให้คุณตั้งค่าโดยแยกฐานข้อมูลสำหรับการทดสอบ เพื่อการพัฒนา

โพรซีเดอร์

หากต้องการสร้างฐานข้อมูลสำหรับการทดสอบ ให้ปฏิบัติดังนี้:

1. สร้างไดเร็กทอรีสำหรับใช้ทดสอบ ตัวอย่างเช่น คำสั่งต่อไปนี้จะสร้างไดเร็กทอรี `/home/smit/test`:

   ```
   mkdir /home/smit /home/smit/test
   ```

2. กำหนดให้ไดเร็กทอรีสำหรับการทดสอบเป็นไดเร็กทอรีปัจจุบัน:

   ```
   cd /home/smit/test
   ```

3. กำหนดไดเร็กทอรีสำหรับการทดสอบให้เป็นดีฟอลต์ที่เก็บอ็อบเจ็กต์ โดยตั้งค่าตัวแปรสภาวะแวดล้อม **ODMDIR** ให้เป็น `.` (ไดเร็กทอรีปัจจุบัน):

   ```
   export ODMDIR= .
   ```

4. สร้างฐานข้อมูล SMIT ใหม่ในไดเร็กทอรีสำหรับการทดสอบ:

   ```
   cp  /usr/lib/objrepos/sm_* .
   ```

หากต้องการเพิ่มภารกิจให้กับฐานข้อมูล SMIT:

1. ออกแบบไดอะล็อกสำหรับคำสั่งที่คุณต้องการให้ SMIT สร้างขึ้น
2. ออกแบบลำดับชั้นของเมนูและตัวเลือกที่ต้องการ เพื่อนำพาผู้ใช้ SMIT ไปยังไดอะล็อก และกำหนดตำแหน่งและวิธีการลิงก์ลำดับชั้นนี้กับฐานข้อมูล SMIT ที่มีอยู่ กลยุทธ์ต่อไปนี้ อาจช่วยคุณประหยัดเวลาหากคุณกำลังพัฒนาส่วนขยายฐานข้อมูล SMIT ในครั้งแรก:
   a. สตาร์ท SMIT (รันคำสั่ง **smit**) มองหาเมนู ตัวเลือก และหน้าจอไดอะล็อกที่มีอยู่ซึ่งใช้ดำเนินการกับภารกิจที่คล้ายกับภารกิจที่คุณต้องการเพิ่ม และค้นหาหน้าจอเมนูที่คุณต้องการเพิ่มภารกิจใหม่
   b. ออกจาก SMIT จากนั้นลบไฟล์บันทึกการทำงาน SMIT ที่มีอยู่ แทนที่จะลบไฟล์บันทึกการทำงาน คุณสามารถใช้แฟล็ก **-l** ของคำสั่ง **smit** เพื่อระบุไฟล์บันทึกการทำงานอื่นๆ เมื่อสตาร์ท SMIT ในขั้นตอนต่อไปนี้ ซึ่งจะอนุญาตให้คุณแยก trace สำหรับเอาต์พุตของเซสชัน SMIT ถัดไปของคุณ
   c. สตาร์ท SMIT อีกครั้งด้วยแฟล็กคำสั่ง **-t** และมองหาหน้าจอที่คุณต้องการเพิ่มภารกิจใหม่อีกครั้ง การกระทำเช่นนี้จะบันทึกการทำงาน ID อ็อบเจ็กต์ที่เข้าถึงสำหรับหน้าจอแต่ละหน้าจอในขั้นตอนถัดไป
   d. มองหาไฟล์บันทึกการทำงาน SMIT เพื่อกำหนด ID สำหรับแต่ละคลาสอ็อบเจ็กต์ที่ใช้เป็นส่วนหนึ่งของเมนู

e. ใช้ ID คลาสอ็อบเจ็กต์ด้วยคำสั่ง **odmget** เพื่อดึงข้อมูล stanza สำหรับอ็อบเจ็กต์เหล่านี้ stanza สามารถใช้เป็นตัวอย่างคร่าวๆ เพื่อช่วยแนะนำคุณในเรื่องของการนำไปปฏิบัติ และศึกษาจากประสบการณ์ของผู้อื่น

f. ดูไฟล์บันทึกการทำงาน SMIT สำหรับสตริงคำสั่งที่ใช้ เมื่อรันผ่านหน้าจอเพื่อดูว่า เครื่องมือพิเศษกำลังถูกนำไปใช้ประโยชน์ (เช่น สคริปต์ **sed** หรือ **awk** ฟังก์ชัน shell **ksh** การกำหนดตัวแปรสภาวะแวดล้อม และอื่นๆ) ขณะที่ป้อนสตริงคำสั่ง ให้ระลึกไว้ว่า สตริงคำสั่งเหล่านั้นถูกประมวลผลสองครั้ง: ครั้งแรกโดยคำสั่ง **odmadd** และครั้งที่สองโดย shell **ksh** โปรดระวังว่า เมื่อใช้ escape meta-character ชนิดพิเศษ เช่น \ หรืออักขระเครื่องหมายอัญประกาศ (' และ ") หมายเหตุ เอาต์พุตคำสั่ง **odmget** ไม่ได้จับคู่อินพุตกับคำสั่ง **odmadd** โดยเฉพาะอย่างยิ่ง เมื่ออักขระเหล่านี้หรือค่าสตริงหลายบรรทัดถูกนำมาใช้

3. โค้ดอ็อบเจ็กต์ไดอะล็อก เมนู และตัวเลือกด้วยการกำหนดลงใน อ็อบเจ็กต์ ASCII สำหรับรูปแบบไฟล์ stanza ที่ต้องการโดยคำสั่ง **odmadd**
4. เพิ่มอ็อบเจ็กต์ไดอะล็อก เมนู และตัวเลือกให้กับฐานข้อมูลสำหรับการทดสอบของ SMIT ด้วยคำสั่ง **odmadd** โดยใช้ชื่อของอ็อบเจ็กต์ ASCII สำหรับไฟล์ stanza ของคุณที่อยู่ในตำแหน่งของ `test_stanzas`:

   ```
   odmadd test_stanzas
   ```

5. ทดสอบและดีบักส่วนที่เพิ่มของคุณโดยรัน SMIT โดยใช้ฐานข้อมูลการทดสอบบนโลคัล:

   ```
   smit -o
   ```

เมื่อคุณกำลังเสร็จสิ้นการทดสอบ ให้เรียกคืนไดเร็กทอรี **/etc/objrepos** ที่เป็นดีฟอลต์ที่เก็บอ็อบเจ็กต์โดยตั้งค่าตัวแปรสภาวะแวดล้อม **ODMDIR** ให้มีค่า **/etc/objrepos**:

```
export ODMDIR=/etc/objrepos
```

การดีบักส่วนขยายฐานข้อมูล SMIT

ส่วนนี้อธิบายกระบวนการของการดีบักส่วนขยายฐานข้อมูล SMIT

งานหรือเงื่อนไขที่จำเป็นต้องมี

1. เพิ่มภารกิจให้กับฐานข้อมูล SMIT
2. ทดสอบภารกิจ

โพรซีเดอร์

1. ระบุปัญหา โดยใช้หนึ่งในแฟล็กต่อไปนี้:
   - รันคำสั่ง **smit -v** ถ้าปัญหานั้นใช้กับ SMIT descriptor ต่อไปนี้:
     - **cmd_to_list**
     - **cmd_to_classify**
     - **cmd_to_discover**
   - รันคำสั่ง **smit -t** ถ้าปัญหานั้นใช้กับเร็กคอร์ดฐานข้อมูล SMIT แต่ละเร็กคอร์ด
   - รันคำสั่ง **smit -l** เพื่อสร้างไฟล์บันทึกการทำงานสำรอง ใช้ไฟล์บันทึกการทำงานสำรอง เพื่อแยกข้อมูลเซสชันปัจจุบัน
2. แก้ไขฐานข้อมูล SMIT ที่มีข้อมูลที่ไม่ถูกต้องอยู่
3. รีสตาร์ทภารกิจ SMIT

การสร้างข้อมูลวิธีใช้ SMIT สำหรับงานใหม่

วิธีใช้ System Management Interface Tool (SMIT) คือส่วนขยายของโปรแกรม SMIT

ซึ่งเป็นชุดของวิธีใช้ที่ออกแบบมาเพื่อให้ข้อมูลแบบออนไลน์เกี่ยวกับคอมโพเนนต์ของ SMIT ที่ใช้เพื่อสร้างไดอะล็อกและเมนูต่างๆ วิธีใช้ SMIT อยู่ในฐานข้อมูล ในรูปของโค้ด SMIT ที่สามารถเรียกทำงานได้ในฐานข้อมูล SMIT มีสองวิธีในการดึงข้อมูลวิธีใช้ SMIT:

เมธอดเหล่านี้แต่ละวิธีมีวิธีต่างๆ ที่ใช้ในการดึงข้อมูลวิธีใช้ SMIT จากฐานข้อมูลวิธีใช้ SMIT

เมธอด Man pages

งานหรือเงื่อนไขที่จำเป็นต้องมี

สร้างภารกิจ SMIT ใหม่ที่จำเป็นต้องมีข้อมูลวิธีใช้

โพรซีเดอร์

1. สำหรับการใช้เอดิเตอร์ให้สร้างไฟล์และป้อนข้อความวิธีใช้ภายในไฟล์ ไฟล์นี้ต้องอยู่ในรูปแบบที่ระบุด้วยคำสั่ง **man** ให้วางชุดของข้อมูลวิธีใช้หนึ่งชุดลงในไฟล์
2. กำหนดไฟล์ข้อความวิธีใช้ตามที่ระบุไว้ด้วยคำสั่ง **man**
3. วางไฟล์ข้อความวิธีใช้ลงในตำแหน่งที่ถูกต้องในไดเร็กทอรีย่อย **manual**
4. ทดสอบไฟล์ที่สร้างขึ้นใหม่เพื่อมั่นใจว่า ไฟล์นั้นทำงานได้โดยใช้คำสั่ง **man**
5. ให้หาตำแหน่งไฟล์ที่มีอ็อบเจ็กต์ ASCII ที่มีรูปแบบไฟล์ stanza สำหรับภารกิจ SMIT ใหม่
6. ให้หาตำแหน่งฟิลด์ descriptor วิธีใช้ในอ็อบเจ็กต์ stanzas ของไฟล์
7. ตั้งค่าฟิลด์ descriptor วิธีใช้ `help_msg_loc` ให้เหมือนกับหัวเรื่องของไฟล์ข้อความวิธีใช้ หัวเรื่องสำหรับเท็กซ์ไฟล์ยังเป็นพารามิเตอร์ที่ส่งผ่านไปยังคำสั่ง **man** ตัวอย่างเช่น:

   ```
   help_msg_loc = "xx", where "xx" = title string name
   ```

   ตัวอย่างนี้เรียกใช้งานคำสั่ง **man** ด้วยชื่อสตริงหัวเรื่อง `xx`
8. ไม่ต้องระบุค่าในฟิลด์ descriptor วิธีใช้ที่ว่างอยู่

เมธอดแค็ตตาล็อกข้อความ

งานหรือเงื่อนไขที่จำเป็นต้องมี

สร้างภารกิจ SMIT ใหม่ที่จำเป็นต้องมีข้อมูลวิธีใช้

โพรซีเดอร์

1. ใช้เอดิเตอร์ใดๆ เพื่อสร้างไฟล์และป้อนข้อความวิธีใช้ลงในไฟล์ ไฟล์ **.msg** ต้องอยู่ในรูปแบบที่ระบุด้วยตัวช่วยข้อความ

   หมายเหตุ: ไฟล์ **.msg** ที่มีอยู่ยังสามารถนำมาใช้ได้
2. กำหนดข้อความวิธีใช้แต่ละข้อความให้เป็นหมายเลขเซ็ต (Set #) และหมายเลขข้อความ (MSG#) ซึ่งอนุญาตให้ระบบดึงข้อมูลของข้อความวิธีใช้ที่ถูกต้อง

3. ใช้คำสั่ง **gencat** เพื่อแปลงไฟล์ **.msg** ไปเป็นไฟล์ **.cat** วางไฟล์ **.cat** ลงในไดเร็กทอรีที่ถูกต้องตามตัวแปรสภาวะแวดล้อม **NLSPATH**
4. ทดสอบข้อความวิธีใช้โดยใช้คำสั่ง **dspmsg**
5. ให้หาตำแหน่งไฟล์ที่มีอ็อบเจ็กต์ ASCII ที่มีรูปแบบไฟล์ stanza สำหรับภารกิจ SMIT ใหม่
6. ให้หาตำแหน่งฟิลด์ descriptor วิธีใช้ในอ็อบเจ็กต์ stanzas ของไฟล์
7. สำหรับอ็อบเจ็กต์ stanza แต่ละตัว ให้หาตำแหน่งฟิลด์ descriptor วิธีใช้ `help_msg_id` ป้อนค่า Set# และ Msg# สำหรับข้อความลงในไฟล์ **.msg** ค่าเหล่านี้ต้องอยู่ในรูปแบบ ตัวช่วยข้อความ ตัวอย่างเช่น หากต้องการดึงข้อความ #14 สำหรับ set #2 ให้ตั้งค่า:

   ```
   help_msg_id - "2,14"
   ```
8. ตั้งค่าฟิลด์ descriptor วิธีใช้ `help_msg_loc` ให้เป็นชื่อไฟล์ของไฟล์ที่มีข้อความวิธีใช้
9. ไม่ต้องระบุค่าลงในฟิลด์ descriptor วิธีใช้ที่ว่างอยู่

คลาสอ็อบเจ็กต์ sm_menu_opt (เมนู SMIT)

รายการแต่ละรายการบนเมนูจะถูกระบุด้วยอ็อบเจ็กต์ **sm_menu_opt**

เมนูที่แสดงขึ้นจะแสดงถึงชุดของอ็อบเจ็กต์ ที่มีค่าเหมือนกันสำหรับ **id** บวกกับอ็อบเจ็กต์ **sm_menu_opt** ที่ใช้สำหรับหัวเรื่อ ซึ่งมีค่า **next_id** เท่ากับค่า **id** ของอ็อบเจ็กต์อื่น

> **หมายเหตุ:** ขณะที่โค้ดอ็อบเจ็กต์ในคลาสอ็อบเจ็กต์นี้ให้ตั้งค่าสตริงเปล่าที่ไม่ได้ใช้ให้เป็น "" (เครื่องหมายอัญประกาศคู่) และตั้งค่าฟิลด์เลขจำนวนเต็มที่ไม่ได้ใช้ให้มีค่า 0

descriptors สำหรับอ็อบเจ็กต์ **sm_menu_opt** คือ:

Descriptor	นิยาม
id	ID หรือชื่อของอ็อบเจ็กต์ ค่าของ **id** คือสตริงที่มีความยาวอักขระสูงสุด 64 ตัวอักษร ID ควรมีค่าเฉพาะกับแอ็พพลิเคชันทั้งสองแอ็พพลิเคชัน และเป็นค่าเฉพาะภายในฐานข้อมูล SMIT ที่ใช้งานโดยเฉพาะ โปรดดูนิยาม **next_id** และ **alias** สำหรับอ็อบเจ็กต์นี้สำหรับข้อมูลที่เกี่ยวข้อง
id_seq_num	ตำแหน่งของรายการนี้จะเกี่ยวข้องกับรายการอื่นๆ บนเมนู อ็อบเจ็กต์ **sm_menu_opt** ไม่มีหัวเรื่องจะถูกเรียงลำดับตามฟิลด์สตริงนี้ ค่าของ **id_seq_num** คือสตริงที่มีความยาวอักขระสูงสุด 16 ตัวอักษร
next_id	ชื่อวิธีลัดของเมนูถัดไป ถ้าค่าสำหรับ descriptor **next_type** ของอ็อบเจ็กต์นี้คือ "m" (เมนู) **next_id** ของเมนูควรเป็นค่าเฉพาะแอ็พพลิเคชันของคุณ และเป็นค่าเฉพาะภายในฐานข้อมูล SMIT ที่ใช้โดยเฉพาะ อ็อบเจ็กต์ **sm_menu_opt** ที่ไม่ใช่นามแฝงที่มีค่า **id** ที่ตรงกับค่า **next_id** จะมีรูปแบบเป็นชุดของการเลือกสำหรับเมนูนั้น ค่าของ **next_id** คือสตริงที่มีความยาวอักขระสูงสุด 64 ตัวอักษร
text	คำอธิบายของภารกิจที่ถูกแสดงเป็นรายการเมนู ค่าของ **text** คือสตริงที่มีความยาวอักขระสูงสุด 1024 ตัวอักษร สตริงนี้สามารถจัดรูปแบบด้วยอักขระ \n (ขึ้นบรรทัดใหม่) แบบฝัง
text_msg_file	ชื่อไฟล์ (ไม่ใช่ชื่อพาธเต็ม) นั่นคือแค็ตตาล็อก Message Facility สำหรับสตริง **text** ค่าของ **text_msg_file** คือสตริงที่มีความยาวอักขระสูงสุด 1024 ตัวอักษร แคตาล็อกข้อความ จำเป็นต้องมีสำหรับแอ็พพลิเคชันโปรแกรมที่สามารถพัฒนาได้ด้วย Message Facility ตั้งค่าให้เป็น "" ถ้าคุณไม่ได้ใช้ Message Facility
text_msg_set	ID ชุด Message Facility สำหรับสตริง **text** ID ชุดสามารถนำมาใช้เพื่อบ่งชี้เซ็ตย่อยของแคตาล็อกเดี่ยว ค่าของ **text_msg_set** คือเลขจำนวนเต็ม ตั้งค่าให้เป็น 0 ถ้าคุณไม่ได้ใช้ Message Facility
text_msg_id	ID Message Facility สำหรับสตริง **text** ค่าของ **text_msg_id** คือเลขจำนวนเต็ม ตั้งค่าให้เป็น 0 ถ้าคุณไม่ได้ใช้ Message Facility

Descriptor	นิยาม
next_type	ชนิดของอ็อบเจ็กต์ถัดไป ถ้ารายการนี้ถูกเลือกไว้ ค่าที่ใช้ได้คือ: "**m**" เมนู อ็อบเจ็กต์ถัดไปคือเมนู (**sm_menu_opt**) "**d**" ไดอะล็อก อ็อบเจ็กต์ถัดไปคือไดอะล็อก (**sm_cmd_hdr**) "**n**" ชื่อ อ็อบเจ็กต์ถัดไปคือตัวเลือก (**sm_name_hdr**) "**i**" รายละเอียด อ็อบเจ็กต์นี้ถูกใช้เพื่อใส่ช่องว่างหรือตัวคั่นบรรทัดอื่นๆ ในเมนู หรือเพื่อแสดงหัวข้อที่ไม่ได้นำไปสู่ภารกิจที่สามารถเรียกใช้งานได้ แต่วิธีใช้จะถูกจัดเตรียมผ่าน descriptor **help_msg_loc** ของอ็อบเจ็กต์นี้
alias	กำหนดว่า ค่าของ descriptor **id** สำหรับอ็อบเจ็กต์เมนูนี้คือนามแฝงสำหรับวิธีลัดที่มีอยู่ซึ่งระบุในฟิลด์ `next_id` ของอ็อบเจ็กต์นี้ ค่าของ descriptor **alias** ต้องมีค่า "`n`" สำหรับอ็อบเจ็กต์เมนู
help_msg_id	ระบุหมายเลขชุดข้อความ Message Facility และหมายเลข message ID พร้อมกับตัวคั่นที่เป็นเครื่องหมายจุลภาคหรือสตริงตัวเลขที่เท่ากับแท็กตัวระบุ SMIT
help_msg_loc	ชื่อไฟล์ที่ส่งออกเป็นพารามิเตอร์ไปยังคำสั่ง **man** สำหรับดึงข้อความวิธีใช้ หรือชื่อไฟล์ของไฟล์ที่มีข้อความวิธีใช้อยู่ ค่าของ **help_msg_loc** คือสตริงที่มีความยาวอักขระสูงสุด 1024 ตัวอักษร
help_msg_base	ชื่อพาธที่ผ่านการรับรองโดยสมบูรณ์ของไลบรารีที่ SMIT อ่านสำหรับชื่อไฟล์ที่เชื่อมโยงกับหนังสือคู่มือที่ถูกต้อง
help_msg_book	มีสตริงที่มีค่าของไฟล์ชื่อที่อยู่ในไฟล์ไลบรารี ซึ่งบ่งชี้ด้วย **help_msg_base**

คลาสอ็อบเจ็กต์ sm_menu_opt ที่ใช้สำหรับนามแฝง

นามแฝง SMIT จะถูกระบุโดยอ็อบเจ็กต์ **sm_menu_opt**

คำอธิบายสำหรับคลาสอ็อบเจ็กต์ **sm_menu_opt** และค่าติดตั้งที่ระบุเป็นนามแฝงคือ:

Descriptor	นิยาม
id	ID หรือชื่อของวิธีลัดใหม่หรือนามแฝงของวิธีลัด ค่าของ **id** คือสตริงที่มีความยาวอักขระสูงสุด 64 ตัวอักษร ID ควรมีค่าเฉพาะกับแอ็พพลิเคชันของคุณ และเป็นค่าเฉพาะภายในฐานข้อมูล SMIT ที่ใช้งาน
id_seq_num	ตั้งค่าเป็น "" (สตริงเปล่า)
next_id	ระบุ **id_seq_num** ของอ็อบเจ็กต์เมนู ที่ชี้โดยนามแฝง ค่าของ **next_id** คือสตริงที่มีความยาวอักขระสูงสุด 64 ตัวอักษร
text	ตั้งค่าเป็น "" (สตริงเปล่า)
text_msg_file	ตั้งค่าเป็น "" (สตริงเปล่า)
text_msg_set	ตั้งค่าเป็น 0
text_msg_id	ตั้งค่าเป็น 0
next_type	ชนิดหน้าจอของวิธีลัด ค่าของสตริง **next_type** ค่าที่ใช้ได้ มีดังนี้: "**m**" เมนู ฟิลด์ `next_id` จะระบุวิธีลัดของหน้าจอเมนู "**d**" ไดอะล็อก ฟิลด์ `next_id` จะระบุวิธีลัดของหน้าจอไดอะล็อก "**n**" ชื่อ ฟิลด์ `next_id` ที่ระบุวิธีลัดของหน้าจอตัวเลือก
alias	นิยามอ็อบเจ็กต์นี้เป็นวิธีลัดนามแฝง descriptor **alias** สำหรับนามแฝงต้องตั้งค่าเป็น "`y`" (ใช่)
help_msg_id	ตั้งค่าเป็น "" (สตริงเปล่า)
help_msg_loc	ตั้งค่าเป็น "" (สตริงเปล่า)
help_msg_base	ตั้งค่าเป็น "" (สตริงเปล่า)
help_msg_book	ตั้งค่าเป็น "" (สตริงเปล่า)

คลาสอ็อบเจ็กต์ sm_name_hdr (ส่วนหัวตัวเลือก SMIT)

หน้าจอของตัวเลือกจะถูกระบุโดยอ็อบเจ็กต์สองตัวคือ: อ็อบเจ็กต์ **sm_name_hdr** ซึ่งระบุหัวเรื่องของหน้าจอและข้อมูลอื่นๆ และอ็อบเจ็กต์ **sm_cmd_opt** ที่ระบุชนิดหน่วยข้อมูลที่ต้องการขอรับ

> **หมายเหตุ:** ขณะที่โค้ดอ็อบเจ็กต์ในคลาสอ็อบเจ็กต์ ให้ตั้งค่าสตริงเปล่าที่ไม่ได้ใช้ให้เป็น "" (เครื่องหมายอัญประกาศคู่) และตั้งค่าฟิลด์เลขจำนวนเต็มที่ไม่ได้ใช้ให้มีค่า 0

ในหน้าจอ ส่วนหัวของตัวเลือก SMIT (**sm_name_hdr**) ที่มี type = "c" หากคุณระบุค่าโดยใช้ : (โคลอน) (ตัวอย่างเช่น tty:0) SMIT จะแทรก #! (เครื่องหมายปอนด์ เครื่องหมายอัศเจรีย์) ที่ด้านหน้าของ : เพื่อบ่งบอกว่า : ไม่ใช่ตัวคั่นฟิลด์ SMIT จะลบ #! หลังจากวิเคราะห์คำส่วนที่เหลือของค่า ก่อนที่จะส่งค่าไปยัง descriptor **cmd_to_classify** หากต้องการเพิ่มเติมข้อมูลใดๆ ใน descriptor **cmd_to_classify** ให้แทรก #! อีกครั้ง ที่ด้านหน้าของ :

descriptor สำหรับคลาสอ็อบเจ็กต์ **sm_name_hdr** คือ:

Descriptor	นิยาม
id	ID หรือชื่อของอ็อบเจ็กต์ ฟิลด์ `id` สามารถทำให้ปรากฏเป็น ID วิธีลัดภายนอกได้ ยกเว้น **has_name_select** จะถูกตั้งค่าเป็น "y" (ใช่) ค่าของ **id** คือสตริงที่มีความยาวอักขระสูงสุด 64 ตัวอักษร ID ควรมีค่าเฉพาะกับแอ็พพลิเคชันของคุณ และเป็นค่าเฉพาะภายในระบบของคุณ
next_id	ระบุอ็อบเจ็กต์ส่วนหัวสำหรับหน้าจอลำดับถัดมา ตั้งค่าฟิลด์ `id` ของอ็อบเจ็กต์ **sm_cmd_hdr** หรืออ็อบเจ็กต์ **sm_name_hdr** ที่ตามด้วยตัวเลือกนี้ ฟิลด์ `next_type` จะอธิบายไว้ด้านล่างนี้จะระบุคลาสอ็อบเจ็กต์ที่ถูกบ่งชี้ ค่าของ **next_id** คือสตริงที่มีความยาวอักขระสูงสุด 64 ตัวอักษร
option_id	ระบุเนื้อความของตัวเลือกนี้ ตั้งค่าฟิลด์ `id` ของอ็อบเจ็กต์ **sm_cmd_opt** ค่าของ **option_id** คือสตริงที่มีความยาวอักขระสูงสุด 64 ตัวอักษร
has_name_select	ระบุว่า หน้าจอนี้ต้องนำหน้าด้วยหน้าจอของตัวเลือก ค่าที่ใช้ได้ มีดังนี้: "" **หรือ** "**n**" ไม่ใช่ ในกรณีนี้คือค่าดีฟอลต์ **id** ของอ็อบเจ็กต์นี้สามารถนำมาใช้เป็นวิธีลัดได้ แม้ว่าจะนำหน้าด้วยหน้าจอของตัวเลือก "**y**" ใช่ ตัวเลือกต้องอยู่นำหน้าอ็อบเจ็กต์นี้ ค่าติดตั้งนี้จะป้องกัน **id** ของอ็อบเจ็กต์จากการใช้เป็นวิธีลัดไปยังหน้าจอไดอะล็อกที่สอดคล้องกัน
name	ข้อความที่แสดงเป็นหัวเรื่องของหน้าจอของตัวเลือก ค่าของ **name** คือสตริงที่มีความยาวอักขระสูงสุด 1024 ตัวอักษร สตริงสามารถจัดรูปแบบด้วยอักขระแบบฝัง `\n` (ขึ้นบรรทัดใหม่)
name_msg_file	ชื่อไฟล์ (ที่ไม่ใช่ชื่อพาธเต็ม) นั่นคือแคตาล็อก Message Facility สำหรับสตริง **name** ค่าของ **name_msg_file** คือสตริงที่มีความยาวอักขระสูงสุด 1024 ตัวอักษร แคตาล็อกข้อความ จำเป็นต้องมีสำหรับแอ็พพลิเคชันโปรแกรมที่สามารถพัฒนาได้ด้วย Message Facility
name_msg_set	ID ชุด Message Facility สำหรับสตริง **name** ID ชุดสามารถนำมาใช้เพื่อบ่งชี้เซ็ตย่อยของแคตาล็อกเดี่ยว ค่าของ **name_msg_set** คือเลขจำนวนเต็ม
name_msg_id	ID Message Facility สำหรับสตริง **name** ค่าของ **name_msg_id** คือเลขจำนวนเต็ม
type	เมธอดที่ต้องการใช้เพื่อประมวลผลตัวเลือก ค่าของ **type** คือสตริงที่มีความยาวอักขระสูงสุด 1 ตัวอักษร ค่าที่ใช้ได้คือ: "" **หรือ** "**j**" ID ถัดไป อ็อบเจ็กต์ที่ตามหลังอ็อบเจ็กต์นี้จะเป็นอ็อบเจ็กต์ที่ระบุโดยค่าของ descriptor **next_id** descriptor **next_id** คือสตริงที่กำหนดไว้อย่างครบถ้วน ซึ่งถูกกำหนดค่าเริ่มต้น ณ เวลาของการพัฒนา "**r**" ชื่อ Cat raw ในกรณีนี้ descriptor **next_id** จะถูกนิยามไว้เป็นบางส่วน ณ เวลาของการพัฒนา และถูกนิยามไว้เป็นบางส่วน ณ เวลารันไทม์ โดยอินพุตของผู้ใช้ ค่าของ descriptor **next_id** ที่ถูกกำหนดไว้ ณ เวลาของการพัฒนาจะต่อกับค่าที่เลือกโดยผู้ใช้เพื่อสร้างค่า **id** ในการค้นหาถัดไป (นั่นคือ ไดอะล็อกหรือตัวเลือกที่ต้องการแสดง) "**c**" ชื่อ Cat cooked ค่าที่เลือกโดยผู้ใช้จำเป็นต้องใช้ในการประมวลผล สำหรับข้อมูลเพิ่มเติม ค่านี้จะถูกส่งไปยังคำสั่งที่มีชื่ออยู่ใน descriptor **cmd_to_classify** จากนั้น เอาต์พุตจากคำสั่งจะต่อกับค่าของ descriptor **next_id** เพื่อสร้าง descriptor **id** ในการค้นหาถัดไป (นั่นคือ ไดอะล็อกหรือตัวเลือกที่ต้องการแสดง)

Descriptor	นิยาม
ghost	ระบุว่า ให้แสดงหน้าจอของตัวเลือกนี้ หรือเฉพาะพาเนลรายการแบบป๊อปอัพ ที่สร้างโดยคำสั่งในฟิลด์ `cmd_to_list` ค่าของ **ghost** จะมีค่าเป็นสตริง ค่าที่ใช้ได้ มีดังนี้: "" **หรือ** "**n**" ไม่ใช่ ให้แสดงหน้าจอของตัวเลือกนี้ "**y**" ใช่ ให้แสดงเฉพาะพาเนลแบบป๊อปอัพที่สร้างด้วยสตริงคำสั่ง โดยใช้ฟิลด์ `cmd_to_list` และ `cmd_to_list_postfix` ในอ็อบเจ็กต์ **sm_cmd_opt** ที่เชื่อมโยง ถ้าไม่มีค่า **cmd_to_list** SMIT จะยอมรับว่า อ็อบเจ็กต์นี้คือ super-ghost (ไม่มีข้อมูลแสดง) รันคำสั่ง **cmd_to_classify** และดำเนินการต่อไป
cmd_to_classify	สตริงคำสั่งที่ต้องการใช้ หากต้องการ เพื่อแบ่งแยกประเภทค่าของฟิลด์ `name` ของอ็อบเจ็กต์ **sm_cmd_opt** ที่เชื่อมโยงกับตัวเลือกนี้ ค่าของ **cmd_to_classify** คือสตริงที่มีความยาวอักขระสูงสุด 1024 ตัวอักษร อินพุตใน **cmd_to_classify** ที่ได้มาจากฟิลด์ `entry` จะถูกเรียกว่า "raw name" และเอาต์พุตของ **cmd_to_classify** จะถูกเรียกว่า "cooked name" ก่อนหน้า AIX เวอร์ชัน 4.2.1 คุณสามารถสร้างเฉพาะหนึ่งค่าด้วย **cmd_to_classify** ถ้ามีค่าที่สอดแทรกโคลอน ค่านั้นจะเป็นค่า escaped โดยอัตโนมัติ ใน AIX 4.2.1 และเวอร์ชันถัดมา คุณสามารถสร้างค่าหลายค่าได้ด้วย **cmd_to_classify** แต่โคลอนจะไม่ใช่ escaped อีกต่อไป โคลอนจะถูกใช้เป็นตัวคั่นด้วยคำสั่งนี้ ณ เวลานี้ ถ้าคุณใช้โคลอนในค่าของคุณ คุณต้องสงวนโคลอนเหล่านี้ไว้ด้วยตนเอง
cmd_to_classify_postfix	postfix ที่ต้องการตีความและเพิ่มลงในสตริงคำสั่งในฟิลด์ `cmd_to_classify` ค่าของ **cmd_to_classify_postfix** คือสตริงที่มีความยาวอักขระสูงสุด 1024 ตัวอักษร
raw_field_name	ชื่อสำรองสำหรับคาดิบนี้ ค่าของ **raw_field_name** คือสตริงที่มีความยาวอักขระสูงสุด 1024 ตัวอักษร ค่าดีฟอลต์คือ `"_rawname"`
cooked_field_name	ชื่อสำรองสำหรับค่า cooked ค่าของ **cooked_field_name** คือสตริงที่มีความยาวอักขระสูงสุด 1024 ตัวอักษร ค่าดีฟอลต์คือ `"cookedname"`
next_type	ชนิดของหน้าของที่ตามหลังตัวเลือกนี้ ค่าที่ใช้ได้ มีดังนี้: "**n**" ชื่อ หน้าจอของตัวเลือกที่ตามหลัง โปรดดูคำอธิบายของ **next_id** ที่ด้านบนสำหรับข้อมูลที่เกี่ยวข้อง "**d**" ไดอะล็อก หน้าจอไดอะล็อกที่ตามหลัง โปรดดูคำอธิบายของ **next_id** ที่ด้านบนสำหรับข้อมูลที่เกี่ยวข้อง
help_msg_id	ระบุหมายเลขชุดข้อความ Message Facility และหมายเลข message ID พร้อมกับตัวคั่นที่เป็นเครื่องหมายจุลภาคหรือสตริงตัวเลขที่เท่ากับแท็กตัวระบุ SMIT
help_msg_loc	ชื่อไฟล์ที่ส่งออกเป็นพารามิเตอร์ไปยังคำสั่ง **man** สำหรับดึงข้อความวิธีใช้ หรือชื่อไฟล์ของไฟล์ที่มีข้อความวิธีใช้อยู่ ค่าของ **help_msg_loc** คือสตริงที่มีความยาวอักขระสูงสุด 1024 ตัวอักษร
help_msg_base	ชื่อพาธที่ผ่านการรับรองโดยสมบูรณ์ของไลบรารีที่ SMIT อ่านสำหรับชื่อไฟล์ที่เชื่อมโยงกับหนังสือคู่มือที่ถูกต้อง
help_msg_book	มีสตริงที่มีค่าของไฟล์ชื่อที่อยู่ในไฟล์ไลบรารี ซึ่งบ่งชี้ด้วย **help_msg_base**

คลาสอ็อพชัน sm_cmd_opt (อ็อพชันคำสั่ง SMIT dialog/selector)

แต่ละอ็อบเจ็กต์ในไดอะล็อก ยกเว้นอ็อบเจ็กต์ส่วนหัวของไดอะล็อก จะสอดคล้องกับแฟล็ก อ็อพชัน หรือแอ็ตทริบิวต์ของคำสั่ง ที่ไดอะล็อกดำเนินการ

อ็อบเจ็กต์เหล่านี้ตั้งแต่หนึ่งตัวขึ้นไปจะถูกสร้างขึ้นสำหรับแต่ละไดอะล็อก SMIT ไดอะล็อก ghost อาจไม่มีอ็อบเจ็กต์ของอ็อพชันคำสั่งไดอะล็อกที่เชื่อมโยง หน้าจอตัวเลือกแต่ละหน้าจอจะประกอบด้วยหนึ่งอ็อบเจ็กต์ส่วนหัวของตัวเลือก และหนึ่งอ็อบเจ็กต์ของอ็อพชันคำสั่งของตัวเลือก

> **หมายเหตุ:** ขณะที่โค้ดอ็อบเจ็กต์ในคลาสอ็อบเจ็กต์ ให้ตั้งค่าสตริงเปล่าที่ไม่ได้ใช้ให้เป็น "" (เครื่องหมายอัญประกาศคู่) และตั้งค่าฟิลด์เลขจำนวนเต็มที่ไม่ได้ใช้ให้มีค่า 0

อ็อบเจ็กต์ของอ็อพชันคำสั่งไดอะล็อกและอ็อบเจ็กต์ของอ็อพชันคำสั่งของตัวเลือก คืออ็อบเจ็กต์ **sm_cmd_opt** descriptor สำหรับคลาสอ็อบเจ็กต์ **sm_cmd_opt** และฟังก์ชันของ descriptor เหล่านั้นคือ:

Descriptor	ฟังก์ชัน
id	ID หรือชื่อของอ็อบเจ็กต์ **id** ของไดอะล็อกที่เชื่อมโยงหรืออ็อบเจ็กต์ส่วนหัวของตัวเลือกสามารถนำมาใช้เป็นวิธีลัดไปยัง ID หรือชื่อของอ็อบเจ็กต์ และอ็อบเจ็กต์ไดอะล็อกอื่นๆ ในไดอะล็อก ค่าของ **id** คือสตริงที่มีความยาวอักขระสูงสุด 64 ตัวอักษร อ็อบเจ็กต์ไดอะล็อกทั้งหมด ที่ปรากฏอยู่ในหนึ่งไดอะล็อกต้องมี ID เดียวกัน และ ID ไม่ควรซ้ำกันในแอ็พพลิเคชันของคุณ และไม่ซ้ำกันภายในฐานข้อมูล SMIT เฉพาะที่ใช้
id_seq_num	ตำแหน่งของรายการนี้จะเกี่ยวข้องกับรายการอื่นบนไดอะล็อก อ็อบเจ็กต์ **sm_cmd_opt** ในไดอะล็อกจะถูกเรียงลำดับบนฟิลด์สตริงนี้ ค่าของ **id_seq_num** คือสตริงที่มีความยาวอักขระสูงสุด 16 ตัวอักษร เมื่ออ็อบเจ็กต์นี้เป็นส่วนหนึ่งของหน้าจอไดอะล็อก สตริง "0" จะเป็นค่าที่ไม่ถูกต้องสำหรับฟิลด์นี้ เมื่ออ็อบเจ็กต์นี้เป็นส่วนหนึ่งของหน้าจอตัวเลือก descriptor **id_seq_num** ต้องถูกตั้งค่าเป็น 0
disc_field_name	สตริงที่ควรจับคู่หนึ่งในฟิลด์ชื่อที่อยู่ในเอาต์พุตของคำสั่ง **cmd_to_discover** ในส่วนหัวของไดอะล็อก ค่าของ **disc_field_name** คือสตริงที่มีความยาวอักขระสูงสุด 64 ตัวอักษร ค่าของ descriptor **disc_field_name** สามารถนิยามได้โดยใช้ชื่อแบบ raw หรือ cooked จากตัวเลือกที่อยู่ก่อนหน้าแทนที่จะนิยามโดยคำสั่ง **cmd_to_discover** ในอ็อบเจ็กต์ส่วนหัวที่เชื่อมโยง ถ้า descriptor ถูกนิยามด้วยอินพุตจากตัวเลือกที่อยู่ก่อนหน้า descriptor นั้นต้องถูกตั้งค่าเป็น "_rawname" หรือ "_cookedname" อย่างใดอย่างหนึ่ง หรือต้องตั้งค่าให้สอดคล้องกับค่า `sm_name_hdr.cooked_field_name` หรือค่า `sm_name_hdr.raw_field_name` ถ้าใช้เพื่อนิยามชื่อดีฟอลต์ใหม่
name	สตริงที่ปรากฏขึ้นบนไดอะล็อกหรือหน้าจอตัวเลือก ที่เป็นชื่อฟิลด์ ซึ่งจะมองเห็นเป็นคำถามหรือพร้อมต์ส่วนของอ็อบเจ็กต์ คำอธิบายภาษาดั้งเดิมของแฟล็ก อ็อพชันหรือพารามิเตอร์ของคำสั่งที่ระบุในฟิลด์ `cmd_to_exec` ของอ็อบเจ็กต์ส่วนหัวของไดอะล็อกที่เชื่อมโยง ค่าของ **name** คือสตริงที่มีความยาวอักขระสูงสุด 1024 ตัวอักษร
name_msg_file	ชื่อไฟล์ (ที่ไม่ใช่ชื่อพาธเต็ม) นั่นคือแคตาล็อก Message Facility สำหรับสตริง **name** ค่าของ **name_msg_file** คือสตริงที่มีความยาวอักขระสูงสุด 1024 ตัวอักษร แค็ตตาล็อกข้อความ จำเป็นต้องมีสำหรับแอ็พพลิเคชันโปรแกรมที่สามารถพัฒนาได้ด้วย Message Facility ตั้งค่าเป็น "" (สตริงเปล่า) ถ้าไม่ได้ใช้
name_msg_set	ID ชุด Message Facility สำหรับสตริง **name** ค่าของ **name_msg_set** คือเลขจำนวนเต็ม ตั้งค่าเป็น 0 หากไม่ได้ใช้
name_msg_id	Message Facility message ID สำหรับสตริง **name** ค่าของ **name_msg_id** คือเลขจำนวนเต็ม ตั้งค่าเป็น 0 หากไม่ได้ใช้
op_type	ชนิดของการดำเนินการสำรองที่สนับสนุนฟิลด์นี้ ค่าของ **op_type** คือสตริง ค่าที่ใช้ได้ มีดังนี้: "" หรือ "**n**" – ซึ่งเป็นกรณีดีฟอลต์ ไม่มีการดำเนินการสำรอง (การเลือกลิสต์หรือกลุ่ม) ที่สนับสนุนฟิลด์นี้ "**l**" – การดำเนินการเลือกลิสต์ที่ถูกจัดเตรียม หน้าต่างป๊อปอัพ จะแสดงรายการของไอเท็มที่สร้างขึ้นโดยการรันคำสั่งในฟิลด์ `cmd_to_list` ของอ็อบเจ็กต์นี้ เมื่อผู้ใช้เลือกฟังก์ชัน **F4=List** ของอินเตอร์เฟส SMIT "**r**" – การดำเนินการเลือกแบบเป็นกลุ่มที่จัดเตรียมไว้ สตริงในฟิลด์ `disp_values` หรือ `aix_values` ถูกตีความเป็นชุดของรายการที่ถูกต้องซึ่งคั่นด้วยเครื่องหมายจุลภาค ผู้ใช้สามารถใช้แท็บหรือแท็บย้อนกลับผ่าน ค่าเหล่านี้เพื่อทำการเลือกได้ และฟังก์ชันอินเตอร์เฟส **F4=List** สามารถนำมาใช้ได้ในกรณีนี้ หาก SMIT จะแปลงสภาพกลุ่มไปเป็นลิสต์ ตามความต้องการ ค่า "N", "L" และ "R" สามารถนำมาใช้เป็นค่า **op_type** ซึ่งเป็นค่าตัวพิมพ์เล็ก "n", "l" และ "r" อย่างไรก็ตาม สำหรับค่าตัวพิมพ์ใหญ่ ถ้าคำสั่ง **cmd_to_exec** รันอยู่ และส่งคืนค่าด้วยค่าออกที่มีค่า 0 ฟิลด์รายการที่สอดคล้องกันจะถูกล้างข้อมูลให้เป็นสตริงเปล่า

Descriptor	ฟังก์ชัน
entry_type	ชนิดของค่าที่ต้องการโดยฟิลด์รายการ ค่าของ **entry_type** คือสตริง ค่าที่ใช้ได้ มีดังนี้: "" หรือ "**n**" – ไม่มีรายการ ซึ่งค่าปัจจุบันไม่สามารถแก้ไขผ่านการพิมพ์โดยตรงได้ ฟิลด์เป็นการแสดงข้อมูลเท่านั้น "**t**" – รายการข้อความ อินพุตสำหรับตัวอักษรผสมตัวเลขที่สามารถป้อนได้ "**#**" – รายการตัวเลข ซึ่งเป็นอักขระตัวเลขเท่านั้น 0, 1, 2, 3, 4, 5, 6, 7, 8 หรือ 9 สามารถป้อนได้ - (เครื่องหมายลบ) หรือ + (เครื่องหมายบวก) สามารถป้อนเป็นอักขระตัวแรกได้ "**x**" – รายการเลขฐานสิบหก ซึ่งอินพุตแบบเลขฐานสิบหกสามารถป้อนได้ "**f**" – รายการไฟล์ ซึ่งชื่อไฟล์สามารถป้อนได้ "**r**" – รายการข้อความดิบ ซึ่งอินพุตแบบตัวอักษรผสมตัวเลขสามารถป้อนได้ การนำหน้าและการตามด้วยช่องว่าง จะถูกนำมาพิจารณาเป็นส่วนสำคัญ และไม่ต้องลบออกจากฟิลด์
entry_size	จำกัดจำนวนของอักขระที่ผู้ใช้สามารถพิมพ์ได้ในฟิลด์รายการ ค่าของ **entry_size** คือเลขจำนวนเต็ม ค่าของ 0 จะดีฟอลต์เป็นขนาดของค่าสูงสุดที่อนุญาต
required	กำหนดว่า ฟิลด์คำสั่งต้องส่งไปยังคำสั่ง **cmd_to_exec** ที่กำหนดไว้ในอ็อบเจ็กต์ส่วนหัวของไดอะล็อกที่เชื่อมโยง ค่าของ **required** คือสตริง ถ้าอ็อบเจ็กต์เป็นส่วนหนึ่งของหน้าจอตัวเลือก ฟิลด์ `required` ต้องตั้งค่าเป็น "" (สตริงเปล่า) ถ้าอ็อบเจ็กต์ เป็นส่วนหนึ่งของหน้าจอไดอะล็อก ค่าที่ถูกต้องคือ: "" หรือ "**n**" – ไม่ใช่ อ็อพชันที่เพิ่มไปยังสตริงคำสั่งในคำสั่ง **cmd_to_exec** หากผู้ใช้เปลี่ยนค่าที่แสดงไว้ตอนเริ่มต้น นี่คือกรณีดีฟอลต์ "**y**" – ใช่ ค่าของฟิลด์ `prefix` และค่าของฟิลด์รายการจะถูกส่งไปยังคำสั่ง **cmd_to_exec** เสมอ "**+**" – ค่าของฟิลด์ `prefix` และค่าของฟิลด์รายการ จะถูกส่งไปยังคำสั่ง **cmd_to_exec** เสมอ ฟิลด์รายการต้องมีอักขระที่ไม่ใช่อักขระว่างอย่างน้อยหนึ่งตัว SMIT จะไม่อนุญาตให้ผู้ใช้รันภารกิจ จนกว่าจะทำตามเงื่อนไข "**?**" – ยกเว้น เมื่อมีค่าว่าง ค่าของฟิลด์ `prefix` และค่าของฟิลด์รายการจะถูกส่งไปยังฟิลด์ `cmd_to_exec` เว้นเสียแต่ฟิลด์รายการจะเป็นฟิลด์ว่าง
prefix	ในกรณีง่ายสุด ให้นิยามแฟล็กที่ส่งไปพร้อมกับค่าฟิลด์รายการให้กับคำสั่ง **cmd_to_exec** ที่กำหนดไว้ในอ็อบเจ็กต์ส่วนหัวของไดอะล็อกที่เชื่อมโยง ค่าของ **prefix** คือสตริงที่มีความยาวอักขระสูงสุด 1024 ตัวอักษร การใช้ฟิลด์นี้ขึ้นอยู่กับการตั้งค่าของฟิลด์ `required` เนื้อหาของฟิลด์ `prefix` และเนื้อหาที่ฟิลด์ `entry` เชื่อมโยง **หมายเหตุ:** ถ้าฟิลด์ `prefix` ตั้งค่าเป็น – (เส้นประ เส้นประ) เนื้อหาของฟิลด์รายการที่เชื่อมโยงจะถูกผนวกเข้ากับจุดสิ้นสุดของคำสั่ง **cmd_to_exec** ถ้าฟิลด์ `prefix` ถูกตั้งค่าเป็น –' (เส้นประ เส้นประ เครื่องหมายอัญประกาศเดี่ยว) เนื้อหาของฟิลด์รายการที่เชื่อมโยงจะถูกผนวกเข้ากับจุดสิ้นสุดของคำสั่ง **cmd_to_exec** ในเครื่องหมายอัญประกาศเดี่ยว
cmd_to_list_mode	กำหนดจำนวนของรายการจากลิสต์ที่ควรใช้ ลิสต์จะถูกสร้างโดยคำสั่ง ที่ระบุในฟิลด์ `cmd_to_list` ของอ็อบเจ็กต์นี้ ค่าของ **cmd_to_list_mode** คือสตริงที่มีความยาวอักขระสูงสุด 1 ตัวอักษร ค่าที่ใช้ได้ มีดังนี้: "" หรือ "**a**" – ขอรับฟิลด์ทั้งหมด นี่คือกรณีดีฟอลต์ "**1**" – ขอรับฟิลด์แรก "**2**" – ขอรับฟิลด์ที่สอง "**r**" – ช่วง การรันสตริงคำสั่งในฟิลด์ `cmd_to_list` จะส่งคืนช่วง (เช่น **1..99**) แทนการส่งคืนลิสต์ ช่วงใช้สำหรับแสดงข้อมูลเท่านั้น ซึ่งจะแสดงอยู่ที่ลิสต์แบบป๊อปอัพ แต่ไม่เปลี่ยนฟิลด์รายการที่เชื่อมโยง
cmd_to_list	สตริงคำสั่งถูกใช้เพื่อขอรับลิสต์ของค่าที่ถูกต้องสำหรับฟิลด์ค่า ค่าของ **cmd_to_list** คือสตริงที่มีค่าความยาวอักขระสูงสุด 1024 ตัวอักษร คำสั่งนี้ควรแสดงค่าที่คั่นด้วยอักขระ \n (ขึ้นบรรทัดใหม่)

Descriptor	ฟังก์ชัน
cmd_to_list_postfix	postfix ที่ต้องการตีความและเพิ่มลงในสตริงคำสั่งซึ่งระบุอยู่ในฟิลด์ `cmd_to_list` อ็อบเจ็กต์ไดอะล็อก ค่าของ **cmd_to_list_postfix** คือสตริงที่มีความยาวอักขระสูงสุด 1024 ตัวอักษร ถ้าบรรทัดแรกขึ้นต้นด้วย # (เครื่องหมายปอนด์) ตามด้วยช่องว่า รายการนั้นจะถูกสร้างขึ้นโดยไม่สามารถเลือกได้ ซึ่งจะมีประโยชน์สำหรับส่วนหัวของคอลัมน์ บรรทัดถัดมาที่ขึ้นต้นด้วย # จะนำหน้าด้วยช่องว่า ซึ่งถูกใช้เป็นความคิดเห็นและการต่อเนื่องของรายการที่นำหน้า
multi_select	กำหนดว่า ผู้ใช้สามารถสร้างการเลือกจำนวนมากจากลิสต์ของค่าที่ถูกต้อง ซึ่งสร้างโดยคำสั่งที่อยู่ในฟิลด์ `cmd_to_list` ของอ็อบเจ็กต์ไดอะล็อก ค่าของ **multi_select** คือสตริง ค่าที่ใช้ได้ มีดังนี้: "" – ไม่ใช่ ผู้ใช้สามารถเลือกได้เฉพาะค่าหนึ่งค่าจากลิสต์ นี่คือกรณีดีฟอลต์ "," – ใช่ ผู้ใช้สามารถเลือกรายการจำนวนมากได้จากลิสต์ เมื่อคำสั่งถูกสร้างขึ้น เครื่องหมายจุลภาคจะถูกแทรกอยู่ระหว่างรายการแต่ละรายการ "**y**" – ใช่ ผู้ใช้สามารถเลือกค่าจำนวนมากได้จากรายการ เมื่อคำสั่งถูกสร้างขึ้น คำนำหน้าอ็อพชันจะถูกแทรกอยู่หน้าสตริง ของรายการที่เลือกไว้ "**m**" – ใช่ ผู้ใช้สามารถเลือกรายการจำนวนมาก จากลิสต์ได้ เมื่อคำสั่งถูกสร้างขึ้น คำนำหน้าอ็อพชันจะถูกแทรกอยู่ก่อนรายการที่เลือกไว้แต่ละรายการ
value_index	สำหรับอ็อพชันแบบกลุ่ม ดัชนีที่มีค่าศูนย์ในอาร์เรย์ของฟิลด์ `disp_value` หมายเลข **value_index** บ่งชี้ถึงค่า ที่แสดงเป็นค่าดีฟอลต์ในฟิลด์รายการในผู้ใช้ ค่าของ **entry_size** คือเลขจำนวนเต็ม
disp_values	อาร์เรย์ของค่าที่ถูกต้องในอ็อพชันแบบกลุ่มที่ต้องการแสดงให้กับผู้ใช้ ค่าของฟิลด์ `disp_values` คือสตริงที่มีความยาวอักขระสูงสุด 1024 ตัวอักษร ค่าที่ถูกต้องคือค่าที่แยกด้วย , (เครื่องหมายจุลภาค) โดยไม่มีช่องว่างนำหน้า หรือตามหลังเครื่องหมายจุลภาค
values_msg_file	ชื่อไฟล์ (ไม่ใช่ชื่อพาธเต็ม) นั่นคือแค็ตตาล็อก Message Facility สำหรับค่าในฟิลด์ `disp_values` ถ้าค่าเหล่านั้นถูกกำหนดค่าเริ่มต้นในช่วงของการพัฒนา ค่าของฟิลด์ `values_msg_file` คือสตริงที่มีความยาวอักขระสูงสุด 1024 ตัวอักษร แคตาล็อกข้อความ จำเป็นต้องมีสำหรับแอ็พพลิเคชันโปรแกรมที่สามารถพัฒนาได้ด้วย Message Facility
values_msg_set	ID ชุดของ Message Facility สำหรับค่าในฟิลด์ `disp_values` ตั้งค่าเป็น 0 ถ้าไม่ได้ใช้
values_msg_id	Message Facility message ID สำหรับค่าในฟิลด์ `disp_values` ตั้งค่าเป็น 0 ถ้าไม่ได้ใช้
aix_values	ถ้าสำหรับอ็อพชันแบบกลุ่ม อาร์เรย์ของค่าถูกระบุไว้ ดังนั้น องค์ประกอบแต่ละตัวจะสอดคล้องกับองค์ประกอบในอาร์เรย์ **disp_values** ที่อยู่ในตำแหน่งเดียวกัน ให้ใช้หากค่าภาษาดั้งเดิมใน **disp_values** ไม่ใช่อ็อพชันจริงที่ต้องการใช้สำหรับคำสั่ง ค่าของฟิลด์ `aix_values` คือสตริงที่มีความยาวอักขระสูงสุด 1024 ตัวอักษร
help_msg_id	ระบุหมายเลขชุดข้อความ Message Facility และหมายเลข message ID พร้อมกับตัวคั่นที่เป็นเครื่องหมายจุลภาคหรือสตริงตัวเลขที่เท่ากับแท็กตัวระบุ SMIT
help_msg_loc	ชื่อไฟล์ที่ส่งออกเป็นพารามิเตอร์ไปยังคำสั่ง **man** สำหรับดึงข้อความวิธีใช้ หรือชื่อไฟล์ของไฟล์ที่มีข้อความวิธีใช้อยู่ ค่าของ **help_msg_loc** คือสตริงที่มีความยาวอักขระสูงสุด 1024 ตัวอักษร
help_msg_base	ชื่อพาธที่ผ่านการรับรองโดยสมบูรณ์ของไลบรารีที่ SMIT อ่านสำหรับชื่อไฟล์ที่เชื่อมโยงกับหนังสือคู่มือที่ถูกต้อง
help_msg_book	มีสตริงที่มีค่าของไฟล์ชื่อที่อยู่ในไฟล์ไลบรารี ซึ่งบ่งชี้ด้วย **help_msg_base**

คลาสอ็อบเจ็กต์ sm_cmd_hdr (ส่วนหัวไดอะล็อก SMIT)

อ็อบเจ็กต์ส่วนหัวของไดอะล็อกคืออ็อบเจ็กต์ **sm_cmd_hdr** อ็อบเจ็กต์ส่วนหัวของไดอะล็อกจำเป็นต้องมีสำหรับไดอะล็อกแต่ละส่วน และชี้ไปยังอ็อบเจ็กต์สำหรับอ็อพชันคำสั่งของไดอะล็อกที่เชื่อมโยงกับไดอะล็อก

หมายเหตุ: ขณะที่โค้ดอ็อบเจ็กต์ในคลาสอ็อบเจ็กต์นี้ให้ตั้งค่าสตริงเปล่าที่ไม่ได้ใช้ให้เป็น "" (เครื่องหมายอัญประกาศคู่) และตั้งค่าฟิลด์เลขจำนวนเต็มที่ไม่ได้ใช้ให้มีค่า 0

descriptor สำหรับคลาสอ็อบเจ็กต์ **sm_cmd_hdr** คือ:

Descriptor	ฟังก์ชัน
id	ID หรือชื่อของอ็อบเจ็กต์ ค่าของ **id** คือสตริงที่มีความยาวอักขระสูงสุด 64 ตัวอักษร ฟิลด์ `id` สามารถนำมาใช้เป็น ID วิธีลัด ยกเว้นว่ามีตัวเลือกที่เชื่อมโยงกับไดอะล็อก ID ควรมีค่าเฉพาะกับแอ็พพลิเคชันของคุณ และเป็นค่าเฉพาะภายในระบบของคุณ
option_id	**id** ของอ็อบเจ็กต์ **sm_cmd_opt** (ฟิลด์ไดอะล็อก) ที่ส่วนหัวนี้อ้างถึง ค่าของ **option_id** คือสตริงที่มีความยาวอักขระสูงสุด 64 ตัวอักษร
has_name_select	ระบุว่า หน้าจอนี้ต้องนำหน้าด้วยหน้าจอตัวเลือก หรือหน้าจอเมนู ค่าที่ถูกต้อง คือ: "" **หรือ** "**n**" ไม่ใช่ ในกรณีนี้คือค่าดีฟอลต์ "**y**" ใช่ ตัวเลือกอยู่หน้าอ็อบเจ็กต์นี้ ค่าติดตั้งนี้จะป้องกัน **id** ของอ็อบเจ็กต์จากการใช้เป็นวิธีลัดไปยังหน้าจอไดอะล็อกที่สอดคล้องกัน
name	ข้อความที่แสดงเป็นหัวเรื่องของหน้าจอไดอะล็อก ค่าของ **name** คือสตริงที่มีความยาวอักขระสูงสุด 1024 ตัวอักษร ข้อความอธิบายถึงภารกิจที่ดำเนินการโดยไดอะล็อก สตริงสามารถจัดรูปแบบด้วยอักขระแบบฝัง **\n** (ขึ้นบรรทัดใหม่)
name_msg_file	ชื่อไฟล์ (ที่ไม่ใช่ชื่อพาธเต็ม) นั่นคือแค็ตตาล็อก Message Facility สำหรับสตริง **name** ค่าของ **name_msg_file** คือสตริงที่มีความยาวอักขระสูงสุด 1024 ตัวอักษร แค็ตตาล็อกข้อความ จำเป็นต้องมีสำหรับแอ็พพลิเคชันโปรแกรมที่สามารถพัฒนาได้ด้วย Message Facility
name_msg_set	ID ชุด Message Facility สำหรับสตริง **name** ID ชุดสามารถนำมาใช้เพื่อบ่งชี้เซ็ตย่อยของแค็ตตาล็อกเดี่ยว ค่าของ **name_msg_set** คือเลขจำนวนเต็ม
name_msg_id	ID Message Facility สำหรับสตริง **name** Message ID สามารถสร้างขึ้นได้ด้วยเครื่องมือแยกข้อความที่ Message Facility เป็นเจ้าของ ค่าของ **name_msg_id** คือเลขจำนวนเต็ม
cmd_to_exec	ส่วนแรกของสตริงคำสั่งซึ่งสามารถเป็นคำสั่ง หรือคำสั่ง และอ็อพชันที่แก้ไข ซึ่งเรียกใช้งานภารกิจของไดอะล็อก อ็อพชันอื่นๆ จะถูกผนวกผ่านการโต้ตอบกับผู้ใช้ด้วยอ็อบเจ็กต์ของอ็อพชันคำสั่ง (**sm_cmd_opt**) ที่เชื่อมโยงกับหน้าจอไดอะล็อก ค่าของ **cmd_to_exec** คือสตริงที่มีความยาวอักขระสูงสุด 1024 ตัวอักษร
ask	กำหนดว่า ผู้ใช้จะได้รับพร้อมต์ เพื่อให้พิจารณาตัวเลือกใหม่สำหรับการเรียกใช้งานภารกิจหรือไม่ ค่าที่ถูกต้อง คือ: "" **หรือ** "**n**" ไม่ใช่ ผู้ใช้ไม่ได้รับพร้อมต์สำหรับการยืนยัน ภารกิจจะถูกดำเนินการ เมื่อไดอะล็อกถูก commit ตัวเลือกนี้เป็นค่ากำหนดดีฟอลต์สำหรับ descriptor **ask** "**y**" ใช่ ผู้ใช้จะได้รับพร้อมต์เพื่อยืนยันภารกิจที่ต้องการดำเนินการ ภารกิจจะดำเนินการหลังจากที่ผู้ใช้ยืนยันเท่านั้น การพร้อมต์ให้ผู้ใช้ยืนยันการเรียกทำงานนี้ จะมีประโยชน์ก่อนภารกิจการลบผลการทำงาน ซึ่งรีซอร์สที่ลบจะยากหรือเป็นไปไม่ได้ที่จะกู้คืน หรือเมื่อไม่มีไดอะล็อกที่สามารถแสดงได้ซึ่งเชื่อมโยงกับภารกิจ (เมื่อฟิลด์ `ghost` มีค่าเป็น "y")

Descriptor	ฟังก์ชัน
exec_mode	กำหนดตัวจัดการอินพุตมาตรฐาน เอาต์พุตมาตรฐาน และไฟล์ **stderr** ในระหว่างการประมวลผลภารกิจ ค่าของ **exec_mode** คือสตริง ค่าที่ถูกต้อง คือ: **"" หรือ "p"** โหมดไพพ์ ซึ่งเป็นค่ากำหนดดีฟอลต์สำหรับ descriptor **exec_mode** คำสั่งจะเรียกใช้งานด้วยเอาต์พุตมาตรฐาน และไฟล์ **stderr** จะเปลี่ยนทิศทางผ่านไพพ์ไปยัง SMIT SMIT จะจัดการเอาต์พุตจากคำสั่ง เอาต์พุตจะถูกบันทึก และสามารถเลื่อนได้โดยผู้ใช้ หลังจากที่ภารกิจทำงานเสร็จสิ้นแล้ว ขณะที่ภารกิจกำลังทำงาน เอาต์พุตจะถูกเลื่อนลงตามความต้องการ **"n"** ไม่มีโหมดไพพ์แบบไม่เลื่อน จะทำงานเหมือนกับโหมด "p" ยกเว้นว่า เอาต์พุตจะไม่สามารถเลื่อนได้ขณะที่รันภารกิจอยู่ หน้าจอแรกของเอาต์พุตจะถูกแสดงตามที่ได้สร้างขึ้น จากนั้นปล่อยให้อยู่ในตำแหน่งนั้น ขณะที่รันภารกิจ เอาต์พุตจะถูกบันทึก และสามารถเลื่อนได้โดยผู้ใช้ หลังจากที่ภารกิจทำงานเสร็จสิ้นแล้ว **"i"** โหมดสืบทอด คำสั่งจะเรียกทำงานด้วยอินพุตมาตรฐาน เอาต์พุตมาตรฐาน และไฟล์ **stderr** โดยจะถูกสืบทอดโดย child process ในขณะที่รันภารกิจ โหมดนี้จะกำหนดให้ควบคุมอินพุตและเอาต์พุตกับคำสั่งที่ถูกเรียกใช้งาน ค่านี้จะทำตามคำสั่งที่ต้องการเขียนลงในไฟล์ **/dev/tty** ดำเนินการกำหนดแอดเดรสของเคอร์เซอร์ หรือใช้การดำเนินการไลบรารี **libcur** หรือ **libcurses** **"e"** โหมดออก/เรียกใช้งาน เป็นสาเหตุทำให้ SMIT เรียกใช้งาน (เรียกใช้งานรูทีนย่อย **execl**) สตริงคำสั่งที่ระบุในกระบวนการปัจจุบัน ซึ่งจะยกเลิก SMIT นี่เป็นการทำตามคำสั่งเพื่อรันคำสั่งที่ทำงานร่วมกันไม่ได้กับ SMIT (ซึ่งเปลี่ยนโหมดการแสดงผลหรือขนาดฟอนต์ เป็นต้น) คำเตือนจะกำหนดว่า SMIT ควรออกก่อนที่จะรันคำสั่ง **"E"** เช่นเดียวกับ **"e"** แต่คำเตือนจะไม่ได้กำหนดไว้ ก่อนที่จะออกจาก SMIT **"P", "N" หรือ "I"** โหมดการสำรองข้อมูล ทำงานเหมือนกับโหมด "p", "n" และ "i" ยกเว้นว่า ถ้าคำสั่ง **cmd_to_exec** ทำงานอยู่ และส่งคืนค่าออกที่มีค่า 0 SMIT จะสำรองข้อมูลอยู่ใกล้กับเมนูที่อยู่ก่อนหน้า (ถ้ามี) หรืออยู่ใกล้กับตัวเลือกที่อยู่ก่อนหน้า (ถ้ามี) หรือส่งค่าให้กับบรรทัดรับคำสั่ง
ghost	บ่งชี้ว่า ไดอะล็อกที่แสดงตามปกติไม่ควรแสดงขึ้น ค่าของ **ghost** จะมีค่าเป็นสตริง ค่าที่ถูกต้อง คือ: **"" หรือ "n"** ไม่มี ไดอะล็อกที่เชื่อมโยงกับภารกิจจะถูกแสดงขึ้น นี่คือค่ากำหนดดีฟอลต์ **"y"** ใช่ ไดอะล็อกที่เชื่อมโยงกับภารกิจจะไม่ถูกแสดง เนื่องจากไม่มีข้อมูลเพิ่มเติมที่จำเป็นจากผู้ใช้ คำสั่งที่ระบุใน descriptor **cmd_to_exec** จะถูกเรียกทำงานในทันทีที่ผู้ใช้เลือกภารกิจ
cmd_to_discover	สตริงคำสั่งจะถูกใช้เพื่อค้นหาค่าดีฟอลต์หรือค่าปัจจุบันของอ็อบเจ็กต์ ที่กำลังถูกจัดการ ค่าของ **cmd_to_discover** คือสตริงที่มีความยาวอักขระสูงสุด 1024 ตัวอักษร คำสั่งจะถูกเรียกใช้งานก่อนที่ไดอะล็อกจะแสดงขึ้น และเอาต์พุตจะถูกดึงข้อมูล เอาต์พุตของคำสั่ง ต้องอยู่ในรูปแบบเครื่องหมายโคลอน
cmd_to_discover_postfix	postfix ที่ต้องการตีความและเพิ่มลงในสตริงคำสั่งในฟิลด์ cmd_to_discover ค่าของ **cmd_to_discover_postfix** คือสตริงที่มีความยาวอักขระสูงสุด 1024 ตัวอักษร
help_msg_id	ระบุหมายเลขชุดข้อความ Message Facility และหมายเลข message ID พร้อมกับตัวคั่นที่เป็นเครื่องหมายจุลภาคหรือสตริงตัวเลขที่เท่ากับแท็กตัวระบุ SMIT
help_msg_loc	ชื่อไฟล์ที่ส่งออกเป็นพารามิเตอร์ไปยังคำสั่ง **man** สำหรับดึงข้อความวิธีใช้ หรือชื่อไฟล์ของไฟล์ที่มีข้อความวิธีใช้อยู่ ค่าของ **help_msg_loc** คือสตริงที่มีความยาวอักขระสูงสุด 1024 ตัวอักษร
help_msg_base	ชื่อพาธที่ผ่านการรับรองโดยสมบูรณ์ของไลบรารีที่ SMIT อ่านสำหรับชื่อไฟล์ที่เชื่อมโยงกับหนังสือคู่มือที่ถูกต้อง

Descriptor	ฟังก์ชัน
help_msg_book	มีสตริงที่มีค่าของไฟล์ชื่อที่อยู่ในไฟล์ไลบรารี ซึ่งบ่งชี้ด้วย **help_msg_base**

โปรแกรมตัวอย่าง SMIT

โปรแกรมตัวอย่างต่อไปนี้ได้รับการออกแบบมาเพื่อช่วยคุณเขียน stanza ที่เป็นของคุณเอง

ถ้าคุณเพิ่ม stanza เหล่านี้ให้กับไดเร็กทอรี SMIT ซึ่งมาพร้อมกับระบบปฏิบัติการ stanza เหล่านี้จะสามารถเข้าถึงได้ผ่าน SMIT โดยเลือกรายการ **Applications** ในเมนูหลัก SMIT โปรแกรมสาธิตทั้งหมดทำงานได้ ยกเว้นโปรแกรมสาธิตที่ 3 ซึ่งไม่ได้ติดตั้งภาษาใดๆ ไว้

```
#------------------------------------------------------------
# Intro:
# Unless you are creating a new SMIT database, first you need
# to decide where to insert the menu for your application.
# Your new menu will point to other menus, name headers, and
# dialogs. For this example, we are inserting a pointer to the
# demo menu under the "Applications" menu option. The next_id for
# the Applications menu item is "apps", so we begin by creating a
# menu_opt with "apps" as its id.
#------------------------------------------------------------
sm_menu_opt:
      id           = "apps"
      id_seq_num   = "010"
      next_id      = "demo"
      text         = "SMIT Demos"
      next_type    = "m"

sm_menu_opt:
      id              = "demo"
      id_seq_num   = "010"
      next_id      = "demo_queue"
      text         = "Demo 1: Add a Print Queue"
   next_type                = "n"

sm_menu_opt:
      id             = "demo"
   id_seq_num           = "020"
      next_id         = "demo_mle_inst_lang_hdr"
      text            = "Demo 2: Add Language for Application Already Installed"
   next_type                = "n"

#----
# Since demo_mle_inst_lang_hdr is a descriptive, but not very
# memorable name, an alias with a simpler name can be made to
# point to the same place.
#----
sm_menu_opt:
      id             = "demo_lang"
      next_id        = "demo_mle_inst_lang_hdr"
   next_type                = "n"
      alias          = "y"

sm_menu_opt:
```

```
   id_seq_num            = "030"
      id              = "demo"
      next_id         = "demo_lspv"
      text            = "Demo 3: List Contents of a Physical Volume"
      text_msg_file   = "smit.cat"
   next_type                = "n"

sm_menu_opt:
      id_seq_num      = "040"
      id              = "demo"
      next_id         = "demo_date"
      text            = "Demo 4: Change / Show Date, Time"
      text_msg_file   = "smit.cat"
   next_type                = "n"

#------------------------------------------------------------------
# Demo 1
# ------
# Goal: Add a Print Queue. If the printers.rte package is not
#       installed, install it automatically. If the user is
#       running MSMIT (SMIT in a windows interface), launch a
#       graphical program for this task. Otherwise, branch to
#       the SMIT print queue task.
#
# Topics:      1. cooked output & cmd_to_classify
#               2. SMIT environment variable (msmit vs. ascii)
#               3. ghost name_hdr
#              4. super-ghost name_hdr
#              5. creating an "OK / cancel" option
#              6. dspmsg for translations
#              7. exit/exec mode
#              8. id_seq_num for a name_hdr option
#------------------------------------------------------------------
#----
# Topics: 1,4
# Note that the next_id is the same as the id. Remember that the
# output of the cmd_to_classify is appended to the next_id,
# since the type is "c", for cooked. So, the next_id will be
# either demo_queue1 or demo_queue2. None of the output of the
# name_hdr is displayed, and there is no cmd_to_list in the
# demo_queue_dummy_opt, making this name_hdr a super-ghost.
#----
sm_name_hdr:
      id                       = "demo_queue"
      next_id      = "demo_queue"
      option_id        = "demo_queue_dummy_opt"
      name                     = "Add a Print Queue"
   name_msg_file         = "smit.cat"
      name_msg_set             = 52
      name_msg_id              = 41
      type                     = "c"
      ghost                    = "y"
   cmd_to_classify          = "\
x()
{
```

```
    # Check to see if the printer file is installed.
    lslpp -l printers.rte 2>/dev/null 1>/dev/null
    if [[ $? != 0 ]]
    then
      echo 2
    else
      echo 1
    fi
}
x"
   next_type                = "n"

#----
# Topics: 2,4
# Having determined the printer software is installed, we want
# to know if the gui program should be run or if we should
# branch to the ascii SMIT screen for this task. To do this, we
# check the value of the environment variable SMIT, which is "m"
# for windows (Motif) or "a" for ascii. Here again we tack the
# output of the cmd_to_classify onto the next_id.
#----
sm_name_hdr:
      id                        = "demo_queue1"
      next_id                   = "mkpq"
      option_id        = "demo_queue_dummy_opt"
   has_name_select = ""
      ghost                     = "y"
   next_type                = "n"
      type                      = "c"
   cmd_to_classify          = "\
gui_check()
{
   if [ $SMIT = \"m\" ]; then
     echo gui
   fi
}
      gui_check"

sm_name_hdr:
      id               = "mkpqgui"
      next_id          = "invoke_gui"
   next_type                = "d"
      option_id        = "demo_queue_dummy_opt"
      ghost                    = "y"

#----
# Topics: 7
# Note: the exec_mode of this command is "e", which
# exits SMIT before running the cmd_to_exec.
#----
sm_cmd_hdr:
      id               = "invoke_gui"
      cmd_to_exec      = "/usr/bin/X11/xprintm"
      exec_mode        = "e"
      ghost                    = "y"
```

```
sm_cmd_opt:
      id                              = "demo_queue_dummy_opt"
      id_seq_num                      = 0

#----
# Topics: 3,5
# The printer software is not installed. Install the software
# and loop back to demo_queue1 to check the SMIT environment
# variable. This is a ghost name_hdr. The cmd_to_list of the
# sm_cmd_opt is displayed immediately as a pop-up option
# instead of waiting for the user to input a response. In this
# ghost, the cmd_opt is a simple OK/cancel box that prompts the
# user to press return.
#----
sm_name_hdr:
      id                          = "demo_queue2"
      next_id                     = "demo_queue1"
      option_id                   = "demo_queue_opt"
      name                        = "Add a Print Queue"
   name_msg_file         = "smit.cat"
      name_msg_set                = 52
      name_msg_id                 = 41
      ghost                       = "y"
   cmd_to_classify           = "\
install_printers ()
{

  # Install the printer package.
  /usr/lib/assist/install_pkg \"printers.rte\" 2>&1 >/dev/null
  if [[ $? != 0 ]]
  then
    echo "Error installing printers.rte"
    exit 1
  else
    exit 0
  fi
}
install_printers "
   next_type                 = "n"

#----
# Topics: 5,6,8
# Here a cmd_opt is used as an OK/cancel box. Note also that the
# command dspmsg is used to display the text for the option. This
# allows for translation of the messages.
# Note: the id_seq_num for the option is 0. Only one option is
#       allowed per name_hdr, and its id_seq_num must be 0.
#----
sm_cmd_opt:
      id                          = "demo_queue_opt"
   id_seq_num                = "0"
   disc_field_name       = ""
      name                        = "Add a Print Queue"
   name_msg_file         = "smit.cat"
```

```
      name_msg_set              = 52
      name_msg_id               = 41
   op_type               = "l"
      cmd_to_list               = "x()\
{
if [ $SMIT = \"a\" ] \n\
then \n\
dspmsg -s 52 smit.cat 56 \
'Press Enter to automatically install the printer software.\n\
Press F3 to cancel.\n\
'\n\
else \n\
dspmsg -s 52 smit.cat 57 'Click on this item to automatically install
the printer software.\n' \n\
fi\n\
} \n\
x"
   entry_type                = "t"
      multi_select = "n"

#-----------------------------------------------------------------
#
# Demo 2
# ------
# Goal: Add a Language for an Application Already Installed. It
#       is often clearer to the user to get some information
#       before displaying the dialog screen. Name Headers
#       (sm_name_hdr) can be used for this purpose. In this
#       example, two name headers are used to determine the
#       language to install and the installation device. #       dialog has entries for the rest of the information needed
#       to perform the task.
#
# Topics:
#       1. Saving output from successive name_hdrs with
#          cooked_field_name
#               2. Using getopts inside cmd_to_exec to process cmd_opt
#          info
#               3. Ring list vs. cmd_to_list for displaying values
#          cmd_opts
#-----------------------------------------------------------------

#----
# Topic: 1
# This is the first name_hdr. It is called by the menu_opt for
# this function. We want to save the user's input for later use
# in the dialog. The parameter passed into the cmd_to_classify
# comes from the user's selection/entry. Cmd_to_classify cleans
# up the output and stores it in the variable specified by
# cooked_field_name. This overrides the default value for the
# cmd_to_classify output, which is _cookedname. The default must
# be overridden because we also need to save the output of the
# next name_hdr.
#----
sm_name_hdr:
    id                        = "demo_mle_inst_lang_hdr"
```

```
    next_id                  = "demo_mle_inst_lang"
    option_id                = "demo_mle_inst_lang_select"
    name                     = "Add Language for Application Already Installed"
    name_msg_file         = "smit.cat"
    name_msg_set          = 53
    name_msg_id              = 35
    type                     = "j"
    ghost                    = "n"
    cmd_to_classify          = "\
        foo() {
            echo $1 | sed -n \"s/[^[]*\\[\\([^]]*\\).*/\\1/p\"
        }
        foo"
    cooked_field_name        = "add_lang_language"
    next_type                = "n"
    help_msg_id              = "2850325"

sm_cmd_opt:
    id                       = "demo_mle_inst_lang_select"
    id_seq_num               = "0"
    disc_field_name          = "add_lang_language"
    name                     = "LANGUAGE translation to install"
    name_msg_file         = "smit.cat"
    name_msg_set          = 53
    name_msg_id              = 20
    op_type               = "l"
    entry_type            = "n"
    entry_size            = 0
    required                 = ""
    prefix                   = "-l "
    cmd_to_list_mode         = "a"
    cmd_to_list              = "/usr/lib/nls/lsmle -l"
    help_msg_id              = "2850328"

#----
# Topic:1
# This is the second name_hdr. Here the user's input is passed
# directly through the cmd_to_classify and stored in the
# variable add_lang_input.
#----
sm_name_hdr:
    id                       = "demo_mle_inst_lang"
    next_id                  = "demo_dialog_add_lang"
    option_id                = "demo_add_input_select"
    has_name_select          = "y"
    name                     = "Add Language for Application Already Installed"
    name_msg_file         = "smit.cat"
    name_msg_set          = 53
    name_msg_id              = 35
    type                     = "j"
    ghost                    = "n"
    cmd_to_classify          = "\
        foo() {
            echo $1
        }
```

```
        foo"
    cooked_field_name         = "add_lang_input"
    next_type                 = "d"
    help_msg_id               = "2850328"

sm_cmd_opt:
    id                        = "demo_add_input_select"
    id_seq_num                = "0"
    disc_field_name       = "add_lang_input"
    name                  = "INPUT device/directory for software"
    name_msg_file         = "smit.cat"
    name_msg_set          = 53
    name_msg_id           = 11
    op_type               = "l"
    entry_type                = "t"
    entry_size            = 0
    required              = "y"
    prefix                = "-d "
    cmd_to_list_mode      = "1"
    cmd_to_list           = "/usr/lib/instl/sm_inst list_devices"
    help_msg_id           = "2850313"

#----
# Topic: 2
# Each of the cmd_opts formats its information for processing
# by the getopts command (a dash and a single character, followed
# by an optional parameter). The colon following the letter in
# the getopts command means that a parameter is expected after
# the dash option. This is a nice way to process the cmd_opt
# information if there are several options, especially if one of
# the options could be left out, causing the sequence of $1, $2,
# etc. to get out of order.
#----
sm_cmd_hdr:
    id              = "demo_dialog_add_lang"
    option_id       = "demo_mle_add_app_lang"
    has_name_select = ""
    name                      = "Add Language for Application Already Installed"
    name_msg_file         = "smit.cat"
    name_msg_set          = 53
    name_msg_id               = 35
    cmd_to_exec     = "\
       foo()
       {
       while getopts d:l:S:X Option \"$@\"
       do
          case $Option in
             d) device=$OPTARG;;
             l) language=$OPTARG;;
             S) software=$OPTARG;;
             X) extend_fs="-X";;
          esac
       done

       if [[ `/usr/lib/assist/check_cd -d $device` = '1' ]]
```

```
       then
          /usr/lib/assist/mount_cd $device
          CD_MOUNTED=true
       fi

       if [[ $software = \"ALL\" ]]
       then
          echo "Installing all software for $language..."
       else
          echo "Installing $software for $language..."
       fi
       exit $RC
       }
       foo"
    ask                       = "y"
    ghost                     = "n"
    help_msg_id               = "2850325"

sm_cmd_opt:
    id                    = "demo_mle_add_app_lang"
    id_seq_num                = "0"
    disc_field_name           = "add_lang_language"
    name                      = "LANGUAGE translation to install"
    name_msg_file         = "smit.cat"
    name_msg_set          = 53
    name_msg_id               = 20
    entry_type            = "n"
    entry_size            = 0
    required              = "y"
    prefix                    = "-l "
    cmd_to_list_mode          = "a"
    help_msg_id               = "2850328"

#----
# Topic: 2
# The prefix field precedes the value selected by the user, and
# both the prefix and the user-selected value are passed into
# the cmd_to_exec for getopts processing.
#----
sm_cmd_opt:
    id                    = "demo_mle_add_app_lang"
    id_seq_num            = "020"
    disc_field_name       = "add_lang_input"
    name                  = "INPUT device/directory for software"
    name_msg_file         = "smit.cat"
    name_msg_set          = 53
    name_msg_id           = 11
    entry_type            = "n"
    entry_size            = 0
    required              = "y"
    prefix                = "-d "
    cmd_to_list_mode      = "1"
    cmd_to_list           = "/usr/lib/instl/sm_inst list_devices"
    help_msg_id           = "2850313"
```

```
sm_cmd_opt:
    id                    = "demo_mle_add_app_lang"
    id_seq_num            = "030"
    name                  = "Installed APPLICATION"
    name_msg_file         = "smit.cat"
    name_msg_set          = 53
    name_msg_id           = 43
    op_type               = "l"
    entry_type            = "n"
    entry_size            = 0
    required              = "y"
    prefix                = "-S "
    cmd_to_list_mode      = ""
    cmd_to_list           = "\
       list_messages ()
       {
          language=$1
          device=$2
          lslpp -Lqc | cut -f2,3 -d':'
       }
       list_messages"
    cmd_to_list_postfix   = "add_lang_language add_lang_input"
    multi_select          = ","
    value_index           = 0
    disp_values           = "ALL"
    help_msg_id           = "2850329"

#----
# Topic: 3
# Here, instead of a cmd_to_list, there is a comma-delimited set
# of Ring values in the disp_values field. This list is displayed
# one item at a time as the user presses tab in the cmd_opt entry
# field. However, instead of passing a yes or no to the cmd_hdr,
# it is more useful to use the aix_values field to pass either
# a -X or nothing. The list in the aix_values field must match
# one-to-one with the list in the disp_values field.
#----
sm_cmd_opt:
      id_seq_num = "40"
   id                    = "demo_mle_add_app_lang"
   disc_field_name       = ""
      name = "EXTEND file systems if space needed?"
   name_msg_file         = "smit.cat"
   name_msg_set          = 53
      name_msg_id = 12
      op_type = "r"
   entry_type            = "n"
   entry_size            = 0
   required              = "y"
      multi_select = "n"
   value_index           = 0
      disp_values = "yes,no"
        values_msg_file = "sm_inst.cat"
      values_msg_set = 1
      values_msg_id = 51
```

```
      aix_values = "-X,"
      help_msg_id = "0503005"

#-------------------------------------------------------------
#
# Demo 3
# ------
# Goal: Show Characteristics of a Logical Volume. The name of the
#   logical volume is entered by the user and passed to the
#   cmd_hdr as _rawname.
#
# Topics:       1. _rawname
#               2. Ringlist & aix_values
#-------------------------------------------------------------

#----
# Topic: 1
# No rawname is needed because we have only one name_hdr and
# we can use the default variable name _rawname.
#----
sm_name_hdr:
      id = "demo_lspv"
      next_id = "demo_lspvd"
      option_id = "demo_cmdlvmpvns"
   has_name_select = ""
      name = "List Contents of a Physical Volume"
   name_msg_file         = "smit.cat"
      name_msg_set = 15
      name_msg_id = 100
   type                       = "j"
   ghost                    = ""
   cmd_to_classify          = ""
   raw_field_name           = ""
   cooked_field_name        = ""
   next_type                  = "d"
      help_msg_id = "0516100"

sm_cmd_opt:
   id_seq_num                 = "0"
      id = "demo_cmdlvmpvns"
      disc_field_name = "PVName"
      name = "PHYSICAL VOLUME name"
   name_msg_file         = "smit.cat"
      name_msg_set = 15
      name_msg_id = 101
   op_type                = "l"
   entry_type                 = "t"
   entry_size             = 0
      required = "+"
   cmd_to_list_mode       = "1"
      cmd_to_list = "lsvg -o|lsvg -i -p|grep -v '[:P]'| \
            cut -f1 -d' '"
   cmd_to_list_postfix    = ""
      multi_select = "n"
      help_msg_id = "0516021"
```

```
#----
# Topic: 1
# The cmd_to_discover_postfix passes in the name of the physical
# volume, which is the raw data selected by the user in the
# name_hdr - _rawname.
#----
sm_cmd_hdr:
      id = "demo_lspvd"
      option_id = "demo_cmdlvmlspv"
   has_name_select           = "y"
      name = "List Contents of a Physical Volume"
   name_msg_file         = "smit.cat"
      name_msg_set = 15
      name_msg_id = 100
      cmd_to_exec = "lspv"
      ask = "n"
      cmd_to_discover_postfix = "_rawname"
      help_msg_id = "0516100"

sm_cmd_opt:
      id_seq_num = "01"
      id = "demo_cmdlvmlspv"
      disc_field_name = "_rawname"
      name = "PHYSICAL VOLUME name"
   name_msg_file         = "smit.cat"
      name_msg_set = 15
      name_msg_id = 101
   op_type               = "l"
   entry_type                = "t"
   entry_size            = 0
      required = "+"
   cmd_to_list_mode      = "1"
      cmd_to_list = "lsvg -o|lsvg -i -p|grep -v '[:P]'| \
            cut -f1 -d' '"
      help_msg_id = "0516021"

#----
# Topic: 2
# Here a ringlist of 3 values matches with the aix_values we
# want to pass to the sm_cmd_hdr's cmd_to_exec.
#----
sm_cmd_opt:
      id_seq_num = "02"
      id = "demo_cmdlvmlspv"
      disc_field_name = "Option"
      name = "List OPTION"
   name_msg_file         = "smit.cat"
      name_msg_set = 15
      name_msg_id = 92
      op_type = "r"
   entry_type            = "n"
   entry_size            = 0
      required = "n"
   value_index           = 0
```

```
      disp_values = "status,logical volumes,physical \
                     partitions"
      values_msg_file = "smit.cat"
      values_msg_set = 15
      values_msg_id = 103
      aix_values = "  ,-l,-p"
      help_msg_id = "0516102"

#-------------------------------------------------------------
#
# Demo 4
# ------
# Goal: Change / Show Date & Time
#
# Topics:       1. Using a ghost name header to get variable
#                  values for the next dialog screen.
#               2. Using a cmd_to_discover to fill more than one
#                  cmd_opt with initial values.
#               3. Re-ordering parameters in a cmd_to_exec.
#-------------------------------------------------------------

#----
# Topic: 1
# This ghost name_hdr gets two values and stores them in the
# variables daylight_y_n and time_zone for use in the cmd_opts
# for the next dialog. The output of cmd_to_classify is colon-
# delimited, as is the list of field names in cooked_field_name.
#----
sm_name_hdr:
      id = "demo_date"
      next_id = "demo_date_dial"
      option_id = "date_sel_opt"
   name_msg_set            = 0
   name_msg_id             = 0
      ghost                     = "y"
      cmd_to_classify           = "\
if [ $(echo $TZ | awk '{ \
  if (length($1) <=6 ) {printf(\"2\")} \
  else {printf(\"1\")} }') = 1 ] \n\
then\n\
    echo $(dspmsg smit.cat -s 30 18 'yes')\":$TZ\"\n\
else\n\
    echo $(dspmsg smit.cat -s 30 19 'no')\":$TZ\"\n\
fi #"
      cooked_field_name = "daylight_y_n:time_zone"

sm_cmd_opt:
   id_seq_num               = "0"
      id = "date_sel_opt"

#----
# Topic: 2,3
# Here the cmd_to_discover gets six values, one for each of the
# editable sm_cmd_opts for this screen. The cmd_to_discover
# output is two lines, the first with a # followed by a list of
```

```
# variable names, and the second line the list of values. ทั้งสอง
# lists are colon-delimited. We also see here the cmd_to_exec
# takeing the parameters from the cmd_opts and reordering them
# when calling the command.
#----
sm_cmd_hdr:
      id = "demo_date_dial"
      option_id = "demo_chtz_opts"
   has_name_select           = "y"
      name = "Change / Show Date & Time"
   name_msg_file         = "smit.cat"
      name_msg_set = 30
      name_msg_id = 21
      cmd_to_exec = "date_proc () \
# MM dd hh mm ss yy\n\
# dialogue param order   #  3  4  5  6  7  2\n\
{\n\
date \"$3$4$5$6.$7$2\"\n\
}\n\
date_proc "
      exec_mode = "P"
      cmd_to_discover = "disc_proc() \n\
{\n\
TZ=\"$1\"\n\
echo '#cur_month:cur_day:cur_hour:cur_min:cur_sec:cur_year'\n\
date +%m:%d:%H:%M:%S:%y\n\
}\n\
disc_proc"
   cmd_to_discover_postfix = ""
      help_msg_id = "055101"

#----
# The first two cmd_opts get their initial values
# (disc_field_name) from the name_hdr.
#----
sm_cmd_opt:
      id_seq_num = "04"
      id = "demo_chtz_opts"
      disc_field_name = "time_zone"
      name = "Time zone"
   name_msg_file         = "smit.cat"
      name_msg_set = 30
      name_msg_id = 16
   required              = "y"

sm_cmd_opt:
      id_seq_num = "08"
      id = "demo_chtz_opts"
      disc_field_name = "daylight_y_n"
      name = "Does this time zone go on daylight savings time?\n"
   name_msg_file         = "smit.cat"
      name_msg_set = 30
      name_msg_id = 17
   entry_size            = 0
```

```
#----
# The last six cmd_opts get their values from the
# cmd_to_discover.
#----
sm_cmd_opt:
      id_seq_num = "10"
      id = "demo_chtz_opts"
      disc_field_name = "cur_year"
      name = "YEAR (00-99)"
   name_msg_file         = "smit.cat"
      name_msg_set = 30
      name_msg_id = 10
      entry_type = "#"
      entry_size = 2
      required = "+"
      help_msg_id = "055102"

sm_cmd_opt:
      id_seq_num = "20"
      id = "demo_chtz_opts"
      disc_field_name = "cur_month"
      name = "MONTH (01-12)"
   name_msg_file         = "smit.cat"
      name_msg_set = 30
   name_msg_id           = 11
      entry_type = "#"
      entry_size = 2
      required = "+"
      help_msg_id = "055132"

sm_cmd_opt:
      id_seq_num = "30"
      id = "demo_chtz_opts"
      disc_field_name = "cur_day"
      name = "DAY (01-31)\n"
   name_msg_file         = "smit.cat"
      name_msg_set = 30
      name_msg_id = 12
      entry_type = "#"
      entry_size = 2
      required = "+"
      help_msg_id = "055133"

sm_cmd_opt:
      id_seq_num = "40"
      id = "demo_chtz_opts"
      disc_field_name = "cur_hour"
      name = "HOUR (00-23)"
   name_msg_file         = "smit.cat"
      name_msg_set = 30
      name_msg_id = 13
      entry_type = "#"
      entry_size = 2
      required = "+"
      help_msg_id = "055134"
```

```
sm_cmd_opt:
      id_seq_num = "50"
      id = "demo_chtz_opts"
      disc_field_name = "cur_min"
      name = "MINUTES (00-59)"
   name_msg_file         = "smit.cat"
      name_msg_set = 30
      name_msg_id = 14
      entry_type = "#"
      entry_size = 2
      required = "+"
      help_msg_id = "055135"

sm_cmd_opt:
      id_seq_num = "60"
      id = "demo_chtz_opts"
      disc_field_name = "cur_sec"
      name = "SECONDS (00-59)"
   name_msg_file         = "smit.cat"
      name_msg_set = 30
      name_msg_id = 15
      entry_type = "#"
      entry_size = 2
      required = "+"
      help_msg_id = "055136"
```

คอนโทรลเลอร์รีซอร์สระบบ

บทความนี้แสดงข้อมูลเกี่ยวกับ System Resource Controller (SRC) ซึ่งจะช่วยจัดการ และควบคุมระบบย่อยที่ซับซ้อน

SRC คือตัวควบคุมระบบย่อย โปรแกรมเมอร์ระบบย่อยผู้ที่เป็นเจ้าของกระบวนการ daemon ตั้งแต่หนึ่งกระบวนการขึ้นไป สามารถใช้เซอร์วิส SRC เพื่อกำหนดอินเตอร์เฟสการจัดการระบบที่สอดคล้องกัน สำหรับแอ็พพลิเคชันของโปรแกรมเมอร์เอง SRC มีชุดของคำสั่งเดี่ยวเพื่อเริ่มต้น หยุดทำงาน ติดตาม รีเฟรช และเคียวรีสถานะของระบบย่อย

นอกจากนี้ SRC ยังมีตัวช่วยแจ้งเตือนข้อผิดพลาด ซึ่งคุณสามารถใช้ตัวช่วยนี้เพื่อรวมเมธอดการกู้คืนที่ระบุเฉพาะ สำหรับระบบย่อยเข้าด้วยกัน ชนิดของรายละเอียดการกู้คืนที่สอดแทรกไว้จะถูกจำกัดไว้เท่านั้น ด้วยข้อกำหนดที่เมธอดการแจ้งเตือนคือสตริงในไฟล์ และสามารถเรียกทำงานได้

โปรดอ้างถึงข้อมูลต่อไปนี้ เพื่อศึกษาเพิ่มเติมเกี่ยวกับข้อกำหนดโปรแกรมมิง SRC:

ระบบย่อยโต้ตอบกับ SRC

SRC จะนิยามระบบย่อยเป็นโปรแกรม หรือชุดของโปรแกรมที่เกี่ยวข้องที่ออกแบบมาเป็นยูนิตซึ่งดำเนินการกับฟังก์ชันที่เกี่ยวข้อง โปรดดู "System Resource Controller" ใน *Operating system and device management* สำหรับคำอธิบายโดยละเอียดของคุณสมบัติของระบบย่อย

เซิร์ฟเวอร์ย่อยคือกระบวนการที่เป็นเจ้าของ และถูกควบคุมโดยระบบย่อย

SRC จะดำเนินการกับอ็อบเจ็กต์ที่อยู่ในคลาสอ็อบเจ็กต์ SRC ระบบย่อยจะถูกนิยามให้กับ SRC เป็นอ็อบเจ็กต์ระบบย่อย เซิร์ฟเวอร์ย่อยนิยามเป็นอ็อบเจ็กต์ชนิดเซิร์ฟเวอร์ย่อย โครงสร้างที่เชื่อมโยงกับแต่ละชนิดของอ็อบเจ็กต์ได้ถูกกำหนดไว้ก่อนในไฟล์ **usr/include/sys/srcobj.h**

SRC สามารถออกคำสั่ง SRC กับอ็อบเจ็กต์ที่ระบบย่อย เซิร์ฟเวอร์ย่อย และระดับของกลุ่มระบบย่อย กลุ่มระบบย่อย คือกลุ่มของระบบย่อยที่ระบุโดยผู้ใช้ การจัดกลุ่มระบบย่อยอนุญาตให้ระบบย่อยจำนวนมาก ถูกควบคุมโดยการเรียกใช้คำสั่งเดี่ยว กลุ่มของระบบย่อย ยังอาจแบ่งใช้เมธอดการแจ้งเตือนทั่วไป

SRC จะสื่อสารกับระบบย่อยโดยส่งสัญญาณ และแลกเปลี่ยนคำร้องขอและตอบกลับแพ็กเก็ต นอกจากสัญญาณแล้ว SRC ยังจดจำซ็อกเก็ต และประเภทการสื่อสาร IPC message-queue จำนวนของรูทีนย่อยจะพร้อมใช้งานเป็น SRC API เพื่อช่วยในการโปรแกรมมิงการสื่อสารระหว่างระบบย่อย และ SRC SRC API ยังสนับสนุนโปรแกรมมิงการสื่อสารระหว่างไคลเอ็นต์โปรแกรม และ SRC

SRC และคำสั่ง init

SRC จะเป็นอิสระจากคำสั่ง **init** อย่างไรก็ตาม SRC มีวัตถุประสงค์เพื่อขยายการทำงาน แบบแตกกระบวนการ (process-spawning) ที่จัดเตรียมไว้คำสั่งนี้ นอกจากจัดเตรียมจุดเดี่ยวของการควบคุม เพื่อเริ่มต้น หยุดทำงาน ติดตาม รีเฟรช และเคียวรีสถานะของระบบย่อยแล้ว SRC สามารถควบคุมการดำเนินการของระบบย่อยแต่ละระบบ สนับสนุนการควบคุมระบบรีโมต และบันทึกการทำงานระบบย่อยที่ล้มเหลว

สำหรับการดำเนินการ เฉพาะเวลาที่คำสั่ง **init** และ SRC โต้ตอบกันเกิดขึ้นเมื่อ **srcmstr** (SRC ต้นฉบับ) daemon จะถูกฝังอยู่ภายในไฟล์ **inittab** ตามค่าดีฟอลต์แล้ว **srcmstr** daemon จะอยู่ในไฟล์ **inittab** ในกรณีนี้ คำสั่ง **init** จะสตาร์ท **srcmstr** daemon ที่จุดเริ่มต้นทำงานของระบบ เนื่องด้วยกระบวนการอื่นๆ ทั้งหมด คุณต้องมีสิทธิผู้ใช้ root หรืออยู่ในกลุ่มของระบบเพื่อเรียกใช้งาน **srcmstr** daemon

การคอมไพล์โปรแกรมเพื่อโต้ตอบกับ srcmstr daemon

หากต้องการเปิดใช้งานโปรแกรมเพื่อโต้ตอบกับ **srcmstr** daemon ไฟล์ **/usr/include/spc.h** ควรถูกสอดแทรกและโปรแกรมควรถูกคอมไพล์ด้วยไลบรารี **libsrc.a** การสนับสนุนนี้ไม่มีความต้องการ หากระบบย่อยใช้สัญญาณเพื่อสื่อสารกับ SRC

การดำเนินการ SRC

หากต้องการใช้การทำงานของ SRC ระบบย่อยต้องโต้ตอบกับ **srcmstr** daemon ด้วยสองวิธี:

- อ็อบเจ็กต์ระบบย่อยต้องถูกสร้างไว้สำหรับระบบย่อยในคลาสอ็อบเจ็กต์ระบบย่อย SRC
- ถ้าระบบย่อยใช้สัญญาณอยู่ จึงไม่มีความต้องการที่จะใช้รูทีนย่อย SRC อย่างไรก็ตาม ถ้าระบบย่อยใช้ message queues หรือซ็อกเก็ต ระบบย่อยนั้นต้องตอบกลับ เพื่อหยุดทำงานคำร้องขอโดยใช้รูทีนย่อย SRC

ระบบย่อย SRC ทั้งหมดต้องสนับสนุนคำสั่ง **stopsrc** SRC จะใช้คำสั่งนี้เพื่อหยุดทำงานระบบย่อย และเซิร์ฟเวอร์ย่อยด้วยสัญญาณ **SIGNORM** (หยุดทำงานแบบปกติ) **SIGFORCE** (หยุดทำงานแบบบังคับ) หรือ **SIGCANCEL** (ยกเลิกระบบ)

ส่วนสนับสนุนระบบย่อยเป็นตัวเลือกสำหรับคำสั่ง **startsrc, lssrc -l, traceson, tracesoff** และ **refresh** สถานะยาวและการรายงานสถานะเซิร์ฟเวอร์ย่อย และกลไลการแจ้งเตือน SRC

ความสามารถของ SRC

SRC มีส่วนสนับสนุนต่อไปนี้สำหรับโปรแกรมเมอร์ระบบย่อย :

- อินเตอร์เฟสคำสั่งทั่วไปเพื่อสนับสนุนการเริ่มต้น การหยุดทำงาน และการส่งคำร้องขอไปยังระบบย่อย
- จุดศูนย์กลางของการควบคุมสำหรับระบบย่อยและกลุ่มของระบบย่อย
- รูปแบบทั่วไปสำหรับคำร้องขอไปยังระบบย่อย
- นิยามของเซิร์ฟเวอร์ย่อย ดังนั้น แต่ละเซิร์ฟเวอร์ย่อยจะสามารถจัดการได้ เนื่องจากถูกนิยามให้เป็นระบบย่อยโดยเฉพาะ
- ความสามารถในการนิยามเมธอดการแจ้งเตือนข้อผิดพลาดที่ระบุเฉพาะสำหรับระบบย่อย
- ความสามารถในการนิยามการตอบกลับที่ระบุเฉพาะสำหรับระบบย่อยเพื่อร้องขอสถานะ ส่วนสนับสนุนการติดตาม และรีเฟรชคอนฟิกูเรชัน
- จุดเดี่ยวของการควบคุมสำหรับการให้บริการคำร้องขอระบบย่อย ในสภาวะแวดล้อมการคำนวณเน็ตเวิร์ก

ข้อมูลที่เกี่ยวข้อง:

mknotify

mkserver

mksys

refresh

rmnotify

srcmstr

tracesoff

traceson

spc.h

srcobj.h

อ็อบเจ็กต์ SRC

System Resource Controller (SRC) กำหนดและจัดการกับคลาสอ็อบเจ็กต์สามคลาส :

คลาสอ็อบเจ็กต์เหล่านี้จะแสดงโดเมนที่ SRC ดำเนินการตามหน้าที่ ชุดของ descriptor คลาสอ็อบเจ็กต์ที่ได้ถูกกำหนดไว้ก่อน ประกอบด้วยชุดของคอนฟิกูเรชันระบบย่อยที่เป็นไปได้ ซึ่งสนับสนุนโดย SRC

> **หมายเหตุ:** เฉพาะคลาสอ็อบเจ็กต์ระบบย่อย SRC จำเป็นต้องมี ใช้ชนิดเซิร์ฟเวอร์ย่อยและคลาสอ็อบเจ็กต์การแจ้งเตือน เป็นระบบย่อยที่ต้องการพึ่งพา

คลาสอ็อบเจ็กต์ระบบย่อย

คลาสอ็อบเจ็กต์ระบบย่อยมี descriptor สำหรับระบบย่อย SRC ทั้งหมด ระบบย่อยต้องถูกปรับแต่งในคลาสนี้ ก่อนที่จะสามารถจดจำได้โดย SRC

The descriptors for the Subsystem object class are defined in the **SRCsubsys** structure of the **/usr/include/sys/srcobj.h** file. ตาราง descriptor อ็อบเจ็กต์ระบบย่อย และค่าดีฟอลต์แสดงภาพในรูปแบบสั้นๆ ของ descriptor ระบบย่อยพร้อมกับแฟล็กคำสั่ง **mkssys** และ **chssys** ที่เชื่อมโยงกับ descriptor แต่ละตัว

ตารางที่ 79. descriptor อ็อบเจ็กต์ระบบย่อย และค่าดีฟอลต์

Descriptor	ค่าดีฟอลต์	แฟล็ก
ชื่อระบบย่อย		-s
พาธไปยังคำสั่งระบบย่อย		-p
อาร์กิวเมนต์คำสั่ง		-a
การประมวลผลลำดับความสำคัญ	20	-E
อินสแตนซ์จำนวนมาก	NO	-Q -q
ID ผู้ใช้		-u
ชื่อเหมือน (คีย์)		-t
การดำเนินการสตาร์ท	ONCE	-O -R
stdin	/dev/console	-i
stdout	/dev/console	-o
stderr	/dev/console	-e
ประเภทการสื่อสาร	ซ็อกเก็ต	-K -I -S
ชนิดข้อความของระบบย่อย		-m
คีย์การสื่อสาร IPC queue		-l
ชื่อกลุ่ม		-G
สัญญาณ **SIGNORM**		-n
สัญญาณ **SIGFORCE**		-f
แสดงผล	ใช่	-D -d
ช่วงเวลารอ	20 วินาที	-w
Auditid		

descriptor ของอ็อบเจ็กต์ระบบย่อยจะถูกกำหนดดังนี้:

descriptors อ็อบเจ็กต์ **นิยาม**

ชื่อระบบย่อย
ระบุชื่อของอ็อบเจ็กต์ระบบย่อย ชื่อไม่สามารถมีขนาดเกิน 30 ไบต์ ซึ่งรวมถึงเทอร์มิเนเตอร์ null (29 ตัวอักษรสำหรับชุดอักขระไบต์เดี่ยว หรือ 14 ตัวอักษรสำหรับชุดอักขระแบบมัลติไบต์) descriptor นี้ต้องเป็นแบบ POSIX-compliant ฟิลด์นี้เป็นฟิลด์บังคับ

พาธคำสั่งของระบบย่อย
ระบุชื่อพาธเต็มสำหรับโปรแกรมที่เรียกใช้งาน ด้วยคำสั่งสตาร์ทระบบย่อย ชื่อพาธไม่สามารถมีขนาดเกิน 200 ไบต์ ซึ่งรวมถึงเทอร์มิเนเตอร์ (199 ตัวอักษรสำหรับชุดอักขระไบต์เดี่ยว หรือ 99 ตัวอักษรสำหรับชุดอักขระแบบมัลติไบต์) ชื่อพาธต้องเป็นแบบ POSIX-compliant ฟิลด์นี้เป็นฟิลด์บังคับ

อาร์กิวเมนต์คำสั่ง
ระบุอาร์กิวเมนต์ใดๆ ที่ต้องส่งไปยังคำสั่งที่สตาร์ทระบบย่อย อาร์กิวเมนต์ไม่สามารถมีขนาดเกิน 200 ไบต์ ซึ่งรวมถึงเทอร์มิเนเตอร์ null (199 ตัวอักษรสำหรับชุดอักขระไบต์เดี่ยว หรือ 99 ตัวอักษรสำหรับชุดอักขระแบบมัลติไบต์) อาร์กิวเมนต์จะถูกวิเคราะห์คำโดย **srcmstr** daemon ตามกฎเดียวกันกับที่ใช้โดย shell ตัวอย่างเช่น สตริงที่ถูกอ้างจะถูกส่งผ่านเป็นอาร์กิวเมนต์เดี่ยว และที่ว่างภายนอกสตริงที่ถูกอ้างจะเป็นตัวคั่นอาร์กิวเมนต์

การประมวลผลลำดับความสำคัญ
ระบุการประมวลผลลำดับความสำคัญของระบบย่อยที่ต้องการรัน ระบบย่อยจะสตาร์ทด้วย **srcmstr** daemon ที่รันด้วยลำดับความสำคัญนี้ ค่าดีฟอลต์คือ 20

descriptors อ็อบเจ็กต์	นิยาม
อินสแตนซ์จำนวนมาก	ระบุจำนวนของอินสแตนซ์ของระบบย่อยที่สามารถรันได้ในหนึ่งครั้ง ค่า NO (แฟล็ก **-Q**) ระบุว่า มีเพียงหนึ่งอินสแตนซ์ของระบบย่อยเท่านั้นที่สามารถรันได้ในหนึ่งครั้ง ความพยายามในการสตาร์ทระบบย่อยนี้หากระบบย่อยที่รันอยู่เกิดความล้มเหลว จะมีความพยายามในการสตาร์ทระบบย่อยบนคีย์ IPC message queue เดียวกัน ค่า YES (แฟล็ก **-q**) ระบุว่า ระบบย่อยจำนวนมากอาจใช้ IPC message queue เดียวกัน และมีอินสแตนซ์ของระบบย่อยเดียวกันจำนวนมาก ค่าดีฟอลต์คือ NO
ID ผู้ใช้	ระบุ ID ผู้ใช้ (ตัวเลข) ภายใต้ระบบย่อย ที่รันอยู่ ค่า 0 บ่งชี้ถึงผู้ใช้ root ฟิลด์นี้เป็นฟิลด์บังคับ
ชื่อเหมือน	ระบุสตริงอักขระที่ต้องการใช้เป็นชื่อสำรอง สำหรับระบบย่อย สตริงอักขระไม่สามารถมีค่าเกิน 30 ไบต์ ซึ่งประกอบด้วยเทอร์มิเนเตอร์ null (29 ตัวอักษรสำหรับชุดอักขระไบต์เดี่ยว หรือ 14 ตัวอักขระ สำหรับชุดอักขระแบบมัลติไบต์) ฟิลด์นี้เป็นฟิลด์เผื่อเลือก
การดำเนินการสตาร์ท	ระบุว่า **srcmstr** daemon ควรรีสตาร์ทระบบย่อยหลังจากที่สิ้นสุดแบบผิดปกติ ค่า RESPAWN (แฟล็ก **-R**) ระบุว่า **srcmstr** daemon ควรรีสตาร์ทระบบย่อย ค่า ONCE (แฟล็ก **-O**) ระบุว่า **srcmstr** daemon ไม่ควรพยายามรีสตาร์ทระบบที่ล้มเหลว มีข้อจำกัดเกี่ยวกับ respawn สองข้อที่รีสตาร์ท ภายในช่วงเวลาที่รอ ถ้าระบบที่ล้มเหลวไม่สามารถรีสตาร์ทได้เป็นผลสำเร็จ อ็อพชันของเมธอดการแจ้งเตือนจะถูกนำมาศึกษา ค่าดีฟอลต์คือ ONCE
อินพุตมาตรฐานของไฟล์/อุปกรณ์	ระบุไฟล์หรืออุปกรณ์ที่ระบบย่อยได้รับอินพุต ดีฟอลต์คือ **/dev/console** ฟิลด์นี้ไม่สามารถมีขนาดเกิน 200 ไบต์ ซึ่งรวมถึงเทอร์มิเนเตอร์ null (199 อักขระสำหรับชุดอักขระไบต์เดี่ยว หรือ 99 ตัวอักษรสำหรับชุดอักขระแบบมัลติไบต์) ฟิลด์นี้จะถูกละเว้น หากประเภทการสื่อสารคือซ็อกเก็ต
เอาต์พุตมาตรฐานของไฟล์/อุปกรณ์	ระบุไฟล์หรืออุปกรณ์ที่ระบบย่อยส่งเอาต์พุต ฟิลด์นี้ไม่สามารถมีขนาดเกิน 200 ไบต์ ซึ่งรวมถึงเทอร์มิเนเตอร์ null (199 ตัวอักษร สำหรับชุดอักขระไบต์เดี่ยว หรือ 99 ตัวอักษรสำหรับชุดอักขระแบบมัลติไบต์) ดีฟอลต์คือ **/dev/console**
ข้อผิดพลาดมาตรฐานของไฟล์/อุปกรณ์	ระบุไฟล์หรืออุปกรณ์ที่ระบบเขียนข้อความแสดงความผิดพลาด ฟิลด์นี้ไม่สามารถมีขนาดเกิน 200 ไบต์ ซึ่งรวมถึงเทอร์มิเนเตอร์ null (199 ตัวอักษร สำหรับชุดอักขระไบต์เดี่ยว หรือ 99 ตัวอักษรสำหรับชุดอักขระแบบมัลติไบต์) ความล้มเหลวจะถูกจัดการเป็นส่วนหนึ่งของเมธอดการแจ้งเตือน ดีฟอลต์คือ **/dev/console** **หมายเหตุ:** ข้อผิดพลาดร้ายแรงจะส่งไปยังบันทึกข้อผิดพลาด
ประเภทการสื่อสาร	ระบุเมธอดการสื่อสารระหว่าง **srcmstr** daemon กับระบบย่อย ซึ่งสามารถกำหนดได้สามชนิดคือ: IPC (**-I**) ซ็อกเก็ต (**-K**) หรือสัญญาณ (**-S**) ค่าดีฟอลต์คือซ็อกเก็ต
คีย์การสื่อสารแบบ IPC queue	ระบุค่าเลขฐานสิบที่สอดคล้องกับคีย์ IPC message queue ที่ **srcmstr** daemon ใช้เพื่อสื่อสารกับระบบย่อย ฟิลด์นี้เป็นฟิลด์บังคับสำหรับระบบย่อยที่สื่อสารโดยใช้ IPC message queues ใช้รูทีนย่อย **ftok** ด้วยชื่อพาธที่ผ่านการรับรองโดยสมบูรณ์ และ ID พารามิเตอร์เพื่อมั่นใจว่า เป็นคีย์เฉพาะ **srcmstr** daemon จะสร้าง message queue ก่อนที่จะเริ่มต้นระบบย่อย
ชื่อกลุ่ม	กำหนดให้ระบบย่อยเป็นสมาชิกของกลุ่ม ฟิลด์นี้ไม่สามารถมีค่าเกิน 30 ไบต์ ซึ่งประกอบด้วยเทอร์มิเนเตอร์ null (29 ตัวอักษรสำหรับชุดอักขระไบต์เดี่ยว หรือ 14 ตัวอักษรสำหรับชุดอักขระแบบมัลติไบต์) ฟิลด์นี้เป็นฟิลด์เผื่อเลือก
ชนิดข้อความระบบย่อย	ระบุ mtype ของข้อความที่อยู่บน message queue ของระบบย่อย ระบบย่อยใช้ค่านี้เพื่อดึงข้อความโดยใช้รูทีนย่อย **msgrcv** หรือ **msgxrcv** ฟิลด์นี้เป็นฟิลด์บังคับหากคุณกำลังใช้ message queues
ค่าสัญญาณเดี่ยว SIGNORM	ระบุค่าที่ต้องการส่งไปยังระบบย่อย เมื่อส่งคำร้องขอหยุดให้หยุดการทำงานแบบปกติ ฟิลด์นี้เป็นฟิลด์บังคับของระบบย่อยโดยใช้ประเภทการสื่อสารแบบสัญญาณ
ค่าสัญญาณเดี่ยว SIGFORCE	ระบุค่าที่ต้องการส่งไปยังระบบย่อยเมื่อส่งคำร้องขอให้หยุดทำงานแบบบังคับ ฟิลด์นี้เป็นฟิลด์บังคับของระบบย่อยโดยใช้ประเภทการสื่อสารแบบสัญญาณ
แสดงค่า	บ่งชี้ถึงสถานะของระบบที่ไม่ได้ดำเนินการซึ่งสามารถแสดงได้บนเอาต์พุต **lssrc -a** หรือ **lssrc -g** แฟล็ก **-d** จะบ่งชี้ถึงการแสดง แฟล็ก **-D** บ่งชี้ถึงการไม่แสดง ค่าดีฟอลต์คือ **-d** (แสดง)
ช่วงเวลารอ	ระบุเวลาในหน่วยวินาทีที่ระบบต้องเสร็จสิ้นการรีสตาร์ท หรือหยุดคำร้องขอก่อนที่การดำเนินการสำรองจะถูกใช้ ค่าดีฟอลต์คือ 20 วินาที
Auditid	ระบุ ID การตรวจสอบระบบย่อย สร้างโดย **srcmstr** daemon แบบอัตโนมัติเมื่อนิยามระบบย่อย ฟิลด์นี้จะถูกใช้โดยระบบรักษาความปลอดภัย ถ้ากำหนดไว้ ฟิลด์นี้ไม่สามารถตั้งค่าหรือเปลี่ยนแปลงได้โดยโปรแกรม

คลาสอ็อบเจ็กต์ชนิดเซิร์ฟเวอร์ย่อย

อ็อบเจ็กต์ต้องถูกปรับแต่งในคลาสนี้ ถ้าระบบย่อยมีเซิร์ฟเวอร์ย่อย และระบบย่อยคาดหวังจะได้รับคำสั่งที่เกี่ยวข้องกับเซิร์ฟเวอร์ย่อยจาก **srcmstr** daemon

คลาสอ็อบเจ็กต์นี้มี descriptor 3 ตัว ซึ่งถูกกำหนดในโครงสร้าง **SRCsubsvr** ของไฟล์ **srcobj.h** :

Descriptor	นิยาม
ID เซิร์ฟเวอร์ย่อย (คีย์)	ระบุชื่อของ identifier อ็อบเจ็กต์ชนิดของเซิร์ฟเวอร์ย่อย ชุดของชื่อชนิดเซิร์ฟเวอร์ย่อย จะกำหนดค่าที่อนุญาตให้ใช้สำหรับแฟล็ก **–t** ของคำสั่งเซิร์ฟเวอร์ย่อย ความยาวชื่อไม่สามารถมีขนาดเกิน 30 ไบต์ ซึ่งรวมถึงการยกเลิก null (29 ตัวอักษรสำหรับชุดอักขระไบต์เดี่ยว หรือ 14 ตัวอักษรสำหรับชุดอักขระแบบมัลติไบต์)
ความเป็นเจ้าของชื่อระบบย่อย	ระบุชื่อของระบบย่อยที่เป็นเจ้าของอ็อบเจ็กต์เซิร์ฟเวอร์ย่อย ฟิลด์นี้จะถูกกำหนดไว้ด้วยลิงก์ไปยังคลาสอ็อบเจ็กต์ของระบบย่อย SRC
จุดโค้ด	ระบุหมายเลขทศนิยามที่ระบุถึงเซิร์ฟเวอร์ย่อย จุดโค้ดจะส่งผ่านไปยัง ระบบย่อยที่ควบคุมเซิร์ฟเวอร์ย่อยในฟิลด์ `object` ของโครงสร้าง **subreq** ของโครงสร้างคำร้องขอ SRC ถ้าชื่ออ็อบเจ็กต์เซิร์ฟเวอร์ย่อยยังถูกจัดเตรียมไว้ในคำสั่ง **srcmstr** daemon จะส่งต่อจุดโค้ดไปยังระบบย่อยในฟิลด์ `objname` ของโครงสร้าง **subreq** โปรดดู "ตัวอย่างโครงสร้างคำร้องขอ SRC" ในไฟล์เอกสารคู่มือ **spc.h** สำหรับตัวอย่างขององค์ประกอบเหล่านี้

คำสั่งที่อ้างถึงเซิร์ฟเวอร์ย่อยจะระบุเซิร์ฟเวอร์ย่อยแต่ละตัวเป็นชนิดของเซิร์ฟเวอร์ย่อยที่กำหนดชื่อแล้ว เป็นชนิดของเซิร์ฟเวอร์ย่อยที่กำหนดชื่อแล้ว และยั้งสามารถผนวกชื่อกับ อินสแตนซ์ชนิดของเซิร์ฟเวอร์แต่ละอินสแตนซ์ SRC daemon ใช้ชนิดของเซิร์ฟเวอร์ย่อย เพื่อกำหนดการควบคุมระบบย่อยสำหรับเซิร์ฟเวอร์ย่อย แต่ไม่ได้ตรวจสอบชื่อเซิร์ฟเวอร์ย่อย

คลาสอ็อบเจ็กต์การแจ้งเตือน

คลาสนี้จะแสดงกลไลสำหรับ **srcmstr** daemon เพื่อเรียกใช้งานรูทีนที่ระบบย่อยจัดเตรียมไว้ เมื่อตรวจพบความล้มเหลวของระบบย่อย เมื่อ SRC daemon ได้รับสัญญาณ **SIGCHLD** ที่บ่งชี้ถึงการยกเลิกการประมวลผลระบบย่อย daemon นั้นจะตรวจสอบสถานะของระบบย่อย (รักษาไว้ด้วย **srcmstr** daemon) เพื่อพิจารณาว่า การยกเลิกมีต้นเหตุมาจาก **stopsrc** ถ้าไม่ได้ออกคำสั่ง **stopsrc** การยกเลิกจะถูกตีความเป็นการยกเลิกแบบผิดปกติ ถ้าการดำเนินการรีสตาร์ทในนิยามไม่ได้ระบุ respawn หรือถ้าความพยายาม respawn เกิดความล้มเหลว **srcmstr** daemon จะพยายามอ่านอ็อบเจ็กต์ที่เชื่อมโยงกับชื่อระบบย่อยจากคลาสอ็อบเจ็กต์การแจ้งเตือน ถ้าพบอ็อบเจ็กต์ใดๆ เมธรอดที่เชื่อมโยงกับระบบย่อย จะทำงาน

ถ้าไม่พบอ็อบเจ็กต์ระบบย่อยในคลาสอ็อบเจ็กต์ การแจ้งเตือน **srcmstr** daemon จะพิจารณาว่า ระบบย่อยเป็นของกลุ่มหรือไม่ ถ้าใช่ **srcmstr** daemon จะพยายามอ่านอ็อบเจ็กต์ของชื่อกลุ่มจากคลาสอ็อบเจ็กต์การแจ้งเตือน ถ้าพบอ็อบเจ็กต์ใดๆ เมธอดที่เชื่อมโยงกับอ็อบเจ็กต์จะถูกเรียกใช้งาน ด้วยวิธีนี้ กลุ่มของระบบย่อยสามารถแบ่งใช้เมธอดทั่วไป

> **หมายเหตุ:** เมธอดการแจ้งเตือนระบบย่อยจะอยู่นำหน้า เมธอดการแจ้งเตือนกลุ่ม ดังนั้น ระบบย่อยสามารถเป็นเจ้าของกลุ่ม ที่สตาร์ทพร้อมกัน แต่ยังมีรูทีนการกู้คืนหรือการล้างข้อมูลโดยเฉพาะที่กำหนดไว้

อ็อบเจ็กต์การแจ้งเตือนถูกกำหนดไว้ด้วยสอง descriptor:

Descriptor	นิยาม
ชื่อระบบย่อย หรือ ชื่อกลุ่ม	ระบุชื่อของระบบย่อยหรือกลุ่มที่เมธอดการแจ้งเตือนได้ถูกกำหนดไว้
เมธอดการแจ้งเตือน	ระบุชื่อพาธเต็มให้กับรูทีนที่เรียกใช้งาน เมื่อ **srcmstr** daemon ตรวจพบการยกเลิกแบบผิดปกติของระบบย่อย หรือกลุ่ม

การแจ้งเตือนมีประโยชน์เมื่อการกู้คืนหรือการล้างข้อมูลโดยเฉพาะ ต้องการดำเนินการก่อนที่ระบบย่อยจะรีสตาร์ทได้ และยังเป็นเครื่องมือสำหรับการเก็บรวบรวมข้อมูลเพื่อกำหนดสาเหตุของระบบที่หยุดทำงาน แบบผิดปกติ

แจ้งเตือนอ็อบเจ็กต์ที่สร้างด้วยคำสั่ง **mknotify** หากต้องการแก้ไขเมธอดการแจ้งเตือน อ็อบเจ็กต์การแจ้งเตือนต้องถูกลบออกโดยใช้คำสั่ง **rmnotify** จากนั้นอ็อบเจ็กต์การแจ้งเตือนใหม่จะถูกสร้างขึ้น

คำสั่ง	นิยาม
mknotify	เพิ่มเมธอดการแจ้งให้กับ SRC ฐานข้อมูลคอนฟิกูเรชัน
rmnotify	ลบเมธอดการแจ้งออกจากฐานขอมูลคอนฟิกูเรชัน SRC

srcmstr daemon จะบันทึกการทำงานกิจกรรมการกู้คืน ระบบย่อยจะรับผิดชอบต่อการรายงานความล้มเหลวของระบบย่อย

ประเภทการสื่อสาร SRC

System Resource Controller (SRC) สนับสนุนประเภทการสื่อสารอยู่สามประเภทคือ : สัญญาณ ซ็อกเก็ต และ interprocess communication (IPC) message queues

ประเภทการสื่อสารที่เลือกไว้จะเป็นตัวกำหนดระดับของระบบย่อยที่ใช้ประโยชน์ของฟังก์ชัน SRC

หมายเหตุ: ระบบย่อยทั้งหมด โดยไม่พิจารณาถึงประเภทการสื่อสาร ที่ระบุในอ็อบเจ็กต์สภาวะแวดล้อมของระบบย่อยต้องมีความสามารถในการสนับสนุนประเภทการสื่อสารแบบสัญญาณ ที่ถูกจำกัด รูทีนตัวจับสัญญาณต้องถูกนิยามไว้เพื่อจัดการกับสัญญาณ **SIGTERM** (หยุดทำงาน ยกเลิก) สัญญาณ **SIGTERM** ที่บ่งชี้ถึงระบบย่อยควรล้างข้อมูลรีซอร์สทั้งหมดและควรยกเลิก

โปรดอ้างถึงส่วนต่อไปนี้เพื่อศึกษาเกี่ยวกับประเภทการสื่อสาร SRC:

ตาราง การสื่อสารระหว่าง srcmstr daemon กับระบบย่อยจะสรุปการดำเนินการตามประเภทการสื่อสารที่เชื่อมโยงกับฟังก์ชัน SRC

ฟังก์ชัน	Using IPC or sockets	Using signals
start		
ระบบย่อย	SRC forks และ execs เพื่อสร้างกระบวนการของระบบย่อย	SRC forks และ execs เพื่อสร้างกระบวนการของระบบย่อย
เซิร์ฟเวอร์ย่อย	ใช้ IPC message queue หรือซ็อกเก็ตเพื่อส่งคำร้องขอไปยังระบบย่อย	ไม่สนับสนุน
หยุดทำงานแบบปกติ		
ระบบย่อย	ใช้ IPC message queue หรือซ็อกเก็ตเพื่อส่งคำร้องขอไปยังระบบย่อย	ส่ง **SIGNORM** ไปยังระบบย่อย
เซิร์ฟเวอร์ย่อย	ใช้ IPC message queue หรือซ็อกเก็ตเพื่อส่งคำร้องขอไปยังระบบย่อย	ไม่สนับสนุน
หยุดทำงานแบบบังคับ		
ระบบย่อย	ใช้ IPC message queue หรือซ็อกเก็ตเพื่อส่งคำร้องขอไปยังระบบย่อย	ส่ง **SIGFORCE** ไปยังระบบย่อย
เซิร์ฟเวอร์ย่อย	ใช้ IPC message queue หรือซ็อกเก็ตเพื่อส่งคำร้องขอไปยังระบบย่อย	ไม่สนับสนุน
หยุดทำงาน ยกเลิก		
ระบบย่อย	ส่ง **SIGTERM** แล้วตามด้วย **SIGKILL** ไปยังกลุ่มกระบวนการของระบบย่อย	ส่ง **SIGTERM** แล้วตามด้วย **SIGKILL** ไปยังกลุ่มกระบวนการของระบบย่อย

ฟังก์ชัน	Using IPC or sockets	Using signals
สถานะ แบบย่อ		
ระบบย่อย	นำไปปฏิบัติโดย SRC (ไม่มีคำร้องขอระบบย่อย)	นำไปปฏิบัติโดย SRC (ไม่มีคำร้องขอระบบย่อย)
เซิร์ฟเวอร์ย่อย	ใช้ IPC message queue หรือซ็อกเก็ตเพื่อส่งคำร้องขอไปยังระบบย่อย	ไม่สนับสนุน
สถานะ ยาว		
ระบบย่อย	ใช้ IPC message queue หรือซ็อกเก็ตเพื่อส่งคำร้องขอไปยังระบบย่อย	ไม่สนับสนุน
เซิร์ฟเวอร์ย่อย	ใช้ IPC message queue หรือซ็อกเก็ตเพื่อส่งคำร้องขอไปยังระบบย่อย	ไม่สนับสนุน
traceon/traceoff		
ระบบย่อย	ใช้ IPC message queue หรือซ็อกเก็ตเพื่อส่งคำร้องขอไปยังระบบย่อย	ไม่สนับสนุน
เซิร์ฟเวอร์ย่อย	ใช้ IPC message queue หรือซ็อกเก็ตเพื่อส่งคำร้องขอไปยังระบบย่อย	ไม่สนับสนุน
รีเฟรช		
ระบบย่อย	ใช้ IPC message queue หรือซ็อกเก็ตเพื่อส่งคำร้องขอไปยังระบบย่อย	ไม่สนับสนุน
เซิร์ฟเวอร์ย่อย	ใช้ IPC message queue หรือซ็อกเก็ตเพื่อส่งคำร้องขอไปยังระบบย่อย	ไม่สนับสนุน
แจ้ง		
ระบบย่อย	นำไปปฏิบัติด้วยเมธอดที่จัดเตรียมไว้โดยระบบย่อย	นำไปปฏิบัติด้วยเมธอดที่จัดเตรียมไว้โดยระบบย่อย

การสื่อสารโดยใช้สัญญาณ

ประเภทของการสื่อสารแบบพื้นฐานระหว่างระบบย่อยกับ **srcmstr** daemon จะบรรลุเป้าหมายด้วยสัญญาณ เนื่องจากสัญญาณจะประกอบกันขึ้นเป็น scheme ของการสื่อสารแบบทางเดียว เฉพาะคำสั่ง SRC เท่านั้นที่จดจำสัญญาณของระบบย่อยคือคำร้องขอให้หยุดทำงาน ระบบย่อยที่ใช้สัญญาณไม่ได้จดจำสถานะแบบยาว รีเฟรช หรือคำร้องขอ trace หรือไม่ได้จดจำเซิร์ฟเวอร์ย่อย

สัญญาณของระบบย่อยต้องถูกนำไปปฏิบัติในรูทีนตัวจับสัญญาณ เช่น รูทีนย่อย **sigaction, sigvec** หรือ **signal** เพื่อจัดการกับคำร้องขอ **SIGNORM** และ **SIGFORCE**

สัญญาณของระบบย่อยจะถูกระบุในคลาสอ็อบเจ็กต์ของระบบย่อย SRC โดยออกสตริงคำสั่ง **mkssys -Snf** หรือโดยใช้รูทีน **defssys** และ **addssys**

Item	Descriptor
addssys	เพิ่มนิยามของระบบย่อยให้กับฐานข้อมูลคอนฟิกูเรชัน SRC
defssys	กำหนดค่าเริ่มต้นให้กับนิยามของระบบย่อยใหม่ด้วยค่าดีฟอลต์
mkssys	เพิ่มนิยามของระบบย่อยให้กับฐานข้อมูลคอนฟิกูเรชัน SRC

การสื่อสารซ็อกเก็ต

ด้วยการเพิ่มขึ้น อ็อพชันการสื่อสารของตัวเลือกสำหรับโปรแกรมเมอร์ระบบย่อยคือซ็อกเก็ต ซ็อกเก็ตยังเป็นประเภทการสื่อสารสำหรับ **srcmstr** daemon โปรดดู Sockets Overview*Communications Programming Concepts* สำหรับข้อมูลเพิ่มเติม

srcmstr daemon จะใช้ซ็อกเก็ตเพื่อรับคำร้องของงานจากกระบวนการของคำสั่ง เมื่อเลือกประเภทการสื่อสารนี้แล้ว **srcmstr** daemon จะสร้างซ็อกเก็ตของระบบย่อย ซึ่งระบบย่อยจะได้รับคำร้องขอ **srcmstr** daemon ซ็อกเก็ต UNIX sockets (**AF_UNIX**) จะถูกสร้างขึ้นสำหรับระบบย่อยแบบโลคัล ซ็อกเก็ตอินเตอร์เน็ต (**AF_INET**) จะถูกสร้างขึ้นสำหรับระบบย่อยแบบรีโมต ขั้นตอนต่อไปนี้อธิบายถึงลำดับของการประมวลผลคำสั่ง :

1. กระบวนการของคำสั่งจะยอมรับคำสั่งที่มาจากอุปกรณ์อินพุต สร้างขึ้นเป็นข้อความร้องของงาน และส่งคำร้องของงานดาตาแกรม UDP ไปยัง **srcmstr** daemon สำหรับพอร์ต SRC ที่รู้จัก **AF_INET** จะระบุอยู่ในไฟล์ **/etc/services**
2. **srcmstr** daemon จะ listen พอร์ต SRC ที่รู้จักสำหรับคำร้องของงาน สำหรับการรับคำร้องของงาน จะบอกให้ระบบกรอกข้อมูล **ซ็อกเก็ต** โครงสร้าง **sockaddr** ของรูทีนย่อย เพื่อขอรับแอดเดรสของระบบที่เป็นต้นกำเนิดและผนวกกับแอดเดรสและหมายเลขพอร์ต เพื่อร้องของงาน
3. **srcmstr** daemon ใช้ **srcrrqs** และรูทีนย่อย **srcsrpy** ซึ่งจะประมวลผลเฉพาะคำร้องขอเหล่านี้ ที่ไม่สามารถประเมินผลได้ จากนั้นส่งข้อมูลกลับไปยังการประมวลผลของคำสั่ง คำร้องขออื่นๆ จะถูกส่งต่อไปยังระบบย่อยที่เหมาะสม สำหรับพอร์ตที่ระบบย่อยได้ระบุไว้ในคำร้องของงาน
4. ระบบย่อยจะ listen พอร์ตที่ได้รับมาก่อนหน้านี้โดย **srcmstr** daemon สำหรับระบบย่อย (ระบบย่อยแต่ละระบบ จะสืบทอดพอร์ตเมื่อ **srcmstr** daemon เริ่มต้นระบบย่อย) ระบบย่อยจะประมวลผลคำร้องของงาน และส่งการตอบกลับกลับไปยัง การประมวลผลของคำสั่ง
5. การประมวลผลของคำสั่งจะ listen การตอบกลับ สำหรับพอร์ตที่ระบุ

สิทธิในการเข้าถึงไฟล์และแอดเดรสของซ็อกเก็ตจะถูกใช้โดย **srcmstr** daemon ที่คงไว้ในไดเร็กทอรีชั่วคราว **/dev/SRC** และ **/dev/.SRC-unix** temporary แม้ว่าความสามารถในการแสดงผลโดยใช้คำสั่ง **ls** ข้อมูลที่มีอยู่ในไดเร็กทอรีเหล่านี้จะเป็นการใช้ภายใน SRC เท่านั้น

message queues และซ็อกเก็ตจะมีการทำงานของระบบที่เท่ากัน

Item	Descriptor
srcrrqs	บันทึกแอดเดรสปลายทางของการตอบกลับของระบบของคุณลงใน แพ็กเก็ตที่ได้รับ (และดูเวอร์ชันที่ไม่มีผลกับการทำงานของเธรด **srcrrqs_r**)
srcsrpy	ส่งแพ็กเก็ตตอบกลับของระบบย่อยของคุณไปยังคำร้องขอที่ระบบย่อยของคุณได้รับ

การสื่อสารคิวข้อความ IPC

การทำงานของ IPC message queue จะคล้ายกับการทำงานของซ็อกเก็ต ประเภทของการสื่อสารทั้งสองประเภทสนับสนุนสภาวะแวดล้อม SRC แบบเต็มรูปแบบการทำงาน

เมื่อประเภทการสื่อสารคือ IPC message queue **srcmstr** daemon จะใช้ซ็อกเก็ตเพื่อรับคำร้องของงานจากการประมวลผลคำสั่ง จากนั้นใช้ IPC message queue ที่ระบบย่อยรับข้อความจาก SRC message queue จะถูกสร้างขึ้น เมื่อระบบย่อยเริ่มต้นขึ้น และถูกใช้ตั้งแต่นั้นเป็นต้นมา ระบบย่อย Message queue จะใช้ลำดับของการประมวลผลคำสั่งดังต่อไปนี้ เพื่อสื่อสารกับ **srcmstr** daemon:

1. **srcmstr** daemon จะได้รับ message queue ID จากอ็อบเจ็กต์ระบบย่อย SRC และส่งข้อความไปยังระบบย่อย
2. ระบบย่อยจะรอ message queue และออกคำสั่งรูทีนย่อย **msgrcv** เพื่อรับคำสั่งจาก message queue ในรูปแบบของโครงสร้าง **subreq** ที่ต้องการของคำร้องขอระบบย่อย
3. ระบบย่อยจะเรียกรูทีนย่อย **srcrrqs** เพื่อขอรับ tag ID ที่จะใช้ในการตอบกลับไปยังข้อความ
4. ระบบย่อยจะตีความและประมวลผลคำสั่งที่ได้รับ ซึ่งขึ้นอยู่กับคำสั่ง ระบบย่อยจะสร้างโครงสร้างข้อมูล **svrreply** หรือ **statcode** อย่างใดอย่างหนึ่ง เพื่อส่งคืนการตอบกลับไปยังการประมวลผลคำสั่ง โปรดอ้างถึงไฟล์ **/usr/include/spc.h** สำหรับข้อมูลเพิ่มเติมเกี่ยวกับโครงสร้างเหล่านี้
5. ระบบย่อยจะเรียกรูทีนย่อย **srcsrpy** เพื่อส่งบัฟเฟอร์การตอบกลับไปยังการประมวลผลคำสั่ง

การเขียนโปรแกรมการสื่อสารระบบย่อยด้วย SRC

คำสั่ง System Resource Controller (SRC) คือโปรแกรมเรียกทำงาน ซึ่งใช้อ็อพชันจากบรรทัดรับคำสั่ง

หลักจากที่ตรวจสอบไวยากรณ์คำสั่งแล้ว คำสั่งจะเรียกรูทีนย่อย SRC ณ รันไทม์เพื่อสร้างดาตาแกรม User Datagram Protocol (UDP) และส่งดาตาแกรมนั้นไปยัง **srcmstr** daemon

ส่วนต่อไปนี้แสดงข้อมูลเพิ่มเติมเกี่ยวกับรูทีนย่อย SRC และวิธีที่รูทีนย่อยเหล่านั้นสามารถนำมาใช้ได้โดยระบบย่อย เพื่อสื่อสารกับกระบวนการหลัก SRC:

การเขียนโปรแกรมระบบย่อยเพื่อรับคำร้องขอ SRC

ภารกิจโปรแกรมมิงที่เชื่อมโยงกับการรับคำร้องขอ SRC จะแตกต่างกันเนื่องจากชนิดของการสื่อสารที่ระบุไว้สำหรับระบบย่อย **srcmstr** daemon ใช้ซ็อกเก็ต เพื่อรับคำร้องของงานจากกระบวนการคำสั่ง และสร้างซ็อกเก็ตที่จำเป็นหรือคิวข้อความที่ส่งต่อคำร้องของงาน แต่ละระบบย่อยจำเป็นต้องตรวจสอบการสร้างซ็อกเกต หรือคิวข้อความของระบบ อ่านส่วนต่อไปนี้สำหรับข้อมูลเกี่ยวกับคำแนะนำ ที่ระบุเฉพาะกับประเภทการสื่อสารในการโปรแกรมมิงระบบย่อยของคุณ เพื่อขอรับแพ็กเกจคำร้องขอ

Note: ระบบย่อยทั้งหมด ซึ่งไม่พิจารณาถึงประเภทการสื่อสารต้องกำหนดรูทีนตัวจับสัญญาณ เพื่อจัดการกับคำร้องขอ **SIGTERM**

การรับสัญญาณ SRC

ระบบย่อยที่ใช้สัญญาณเป็นประเภทการสื่อสารของระบบ ต้องกำหนดรูทีนย่อยตัวจับสัญญาณไว้ เพื่อจับสัญญาณ **SIGNORM** และ **SIGFORCE** เมธอดการจับสัญญาณที่ใช้ เป็นระบบย่อยที่ต้องอาศัยการพึ่งพา ต่อไปนี้คือสองตัวอย่างของชนิดของรูทีนย่อย ซึ่งสามารถใช้สำหรับวัตถุประสงค์นี้

รูทีนย่อย	คำอธิบาย
รูทีน **sigaction, sigvec** หรือ **signal**	ระบุการดำเนินการที่ใช้ตามการส่งสัญญาณ
รูทีนย่อย **sigset, sighold, sigrelse** หรือ **sigignore**	พัฒนาตัวช่วยส่งสัญญาณ และจัดเตรียมการจัดการกับสัญญาณสำหรับการประมวลผลแอ็พพลิเคชัน

การรับแพ็กเก็ตคำร้องขอ SRC โดยใช้ซ็อกเก็ต

ใช้คำแนะนำต่อไปนี้เมื่อโปรแกรมมิงซ็อกเก็ตระบบย่อย ให้รับแพ็กเก็ตคำร้องขอ SRC:

- รวมโครงสร้างระบบย่อย SRC ในโค้ดระบบย่อยของคุณโดย การระบุไฟล์ **/usr/include/spc.h** ไฟล์นี้มีโครงสร้างระบบย่อยที่ใช้เพื่อตอบกลับคำสั่ง SRC นอกจากนี้ ไฟล์ **spc.h** จะสอดแทรกไฟล์ **srcerrno.h** ซึ่งไม่ต้องการถูกสอดแทรก แยกจากกัน ไฟล์ **srcerrno.h** จะมีนิยามของโค้ดระบุความผิดพลาดสำหรับส่วนสนับสนุน daemon
- เมื่อระบบย่อยซ็อกเก็ตเริ่มทำงาน ซ็อกเก็ตที่ระบบย่อย ได้รับแพ็กเก็ตคำร้องขอ SRC จะถูกตั้งค่าเป็น file descriptor 0 ระบบย่อย ควรตรวจสอบค่านี้โดยการเรียกใช้รูทีนย่อย **getsockname** ซึ่งจะส่งคืน แอดเดรสของซ็อกเก็ตของระบบย่อย ถ้า file descriptor 0 ไม่ใช่ซ็อกเก็ต ระบบย่อยควรบันทึกข้อผิดพลาด จากนั้นออก โปรดดูที่ "การอ่านดาตาแกรมโปรแกรมตัวอย่าง" ใน *Communications Programming Concepts* สำหรับข้อมูล เกี่ยวกับวิธีที่สามารถใช้รูทีนย่อย **getsockname** เพื่อส่งคืน แอดเดรสของซ็อกเก็ตของระบบย่อย
- ถ้าระบบย่อยสำรวจซ็อกเก็ตที่มากกว่าหนึ่ง ให้ใช้รูทีนย่อย **select** เพื่อกำหนดซ็อกเก็ตที่มีบางสิ่งที่ต้องอ่าน โปรดดูที่ "การตรวจสอบการเชื่อมต่อโปรแกรมตัวอย่างที่ค้างอยู่" ใน *Communications Programming Concepts* สำหรับข้อมูลเพิ่มเติม เกี่ยวกับวิธีที่สามารถใช้รูทีนย่อย **select** สำหรับจุดประสงค์นี้
- ใช้รูทีนย่อย **recvfrom** เพื่อขอรับแพ็กเก็ตคำร้องขอจากซ็อกเก็ต

  > **หมายเหตุ:** แอดเดรสที่ส่งคืนสำหรับ แพ็กเก็ตการตอบกลับระบบย่อยอยู่ในแพ็กเก็ตคำร้องขอ SRC ที่ได้รับ แอดเดรสนี้ไม่ความปะปนกับ แอดเดรสที่รูทีย่อย **recvfrom** ส่งคืนเป็นหนึ่งในพารามิเตอร์

  หลังจากที่รูทีนย่อย **recvfrom** เสร็จสิ้นแล้ว และได้รับแพ็กเก็ตแล้ว ให้ใช้รูทีนย่อย **srcrrqs** เพื่อส่งคืนตัวชี้ไปยังโครงสร้าง **srchdr** แบบสแตติก ตัวชี้นี้มีแอดเดรสที่ส่งคืนสำหรับการตอบกลับ ของระบบย่อย โครงสร้างนี้จะถูกเขียนทับแต่ละครั้งที่รูทีนย่อย **srcrrqs** ถูกเรียก ดังนั้น เนื้อหาควรถูกเก็บไว้ที่อื่น หากเนื้อหาเหล่านั้นจำเป็นต้องใช้หลังจากการเรียกรูทีนย่อย **srcrrqs** ถัดไป

การรับแพ็กเก็ตคำร้องขอ SRC โดยใช้คิวข้อความ

ใช้คำแนะนำต่อไปนี้เมื่อโปรแกรมมิงระบบย่อย programming message queue เพื่อรับแพ็กเก็ตคำร้องขอ SRC:

- สอดแทรกโครงสร้างระบบย่อย SRC ในโค้ดระบบย่อยของคุณโดยระบุไฟล์ **/usr/include/spc.h** ไฟล์นี้มีโครงสร้างระบบย่อย ที่ใช้เพื่อตอบกลับคำสั่ง SRC นอกจากนี้ ไฟล์ **spc.h** จะสอดแทรกไฟล์ **srcerrno.h** ซึ่งไม่ต้องการถูกสอดแทรก แยกจากกัน ไฟล์ **srcerrno.h** จะมีนิยามของโค้ดระบุความผิดพลาดสำหรับส่วนสนับสนุน daemon
- ระบุ **-DSRCBYQUEUE** เป็นอ็อพชันการคอมไพล์ ซึ่งจะวางฟิลด์ชนิดข้อความ (`mtype`) เป็นฟิลด์แรกในโครงสร้าง **srcreq** โครงสร้างนี้ควรถูกใช้ทุกครั้งที่ได้รับแพ็กเก็ต SRC
- เมื่อระบบย่อยได้เริ่มต้นขึ้น ให้ใช้รูทีนย่อย **msgget** เพื่อตรวจสอบว่า message queue ถูกสร้างที่การเริ่มต้นทำงานของระบบ ระบบย่อยควรบันทึกข้อผิดพลาด และออก หากไม่ได้สร้าง message queue ไว้
- ถ้าระบบย่อยสำรวจพบ message queue ที่มากกว่าหนึ่ง ให้ใช้รูทีนย่อย **select** เพื่อกำหนด message queue ที่มีบางสิ่งที่ต้องอ่าน โปรดดูที่ "การตรวจสอบการเชื่อมต่อโปรแกรมตัวอย่างที่ค้างอยู่" ใน *Communications Programming Concepts* สำหรับข้อมูล เพิ่มเติมเกี่ยวกับวิธีที่สามารถใช้รูทีนย่อย **select** สำหรับจุดประสงค์นี้

- ใช้รูทีนย่อย **msgrcv** หรือ **msgxrcv** เพื่อขอรับแพ็กเก็ตจาก message queue แอดเดรสที่ส่งคืนสำหรับแพ็กเก็ตการตอบกลับระบบย่อย จะอยู่ในแพ็กเก็ตที่ได้รับ
- เมื่อรูทีนย่อย **msgrcv** หรือ **msgxrcv** เสร็จสิ้นแล้ว และได้รับแพ็กเก็ตแล้ว ให้เรียกรูทีนย่อย **srcrrqs** เพื่อเสร็จสิ้นการประมวลผลที่รับไว้ รูทีนย่อย **srcrrqs** จะส่งคืนตัวชี้ไปยังโครงสร้าง **srchdr** แบบสแตติก ซึ่งถูกเขียนทับในแต่ละครั้งที่รูทีนย่อย **srcrrqs** ถูกเรียก ตัวชี้นี้มีแอดเดรสที่ส่งคืนสำหรับการตอบกลับของระบบย่อย

การเขียนโปรแกรมระบบย่อยเพื่อประมวลผลแพ็กเก็ตคำร้องขอ SRC

ระบบย่อยต้องมีความสามารถในการประมวลที่หยุดการร้องขอ หรือ ระบบย่อยอาจสนับสนุนการเริ่มต้น สถานะ ติดตาม และรีเฟรชคำร้องขอ

การประมวลผลแพ็กเก็ตการร้องขอที่เกี่ยวข้องกับการประมวลผลแบบสองขั้นตอน:

การอ่านแพ็กเก็ตคำร้องขอ SRC

แพ็กเก็ตการร้องขอ SRC จะได้รับโดยระบบย่อยในรูปแบบของโครงสร้าง **srcreq** ที่กำหนดในไฟล์ **/usr/include/spc.h** คำร้องขอระบบย่อยจะอยู่ในโครงสร้าง **subreq** ของโครงสร้าง **srcreq** :

```
struct subreq
      short object;         /*object to act on*/
      short action;         /*action START, STOP, STATUS, TRACE,\
                             REFRESH*/
      short parm1;          /*reserved for variables*/
      short parm2;          /*reserved for variables*/
      char objname;         /*object name*/
```

ฟิลด์ `object` ของโครงสร้าง **subreq** บ่งชี้ถึงอ็อบเจ็กต์ที่คำร้องขอใช้ เมื่อคำร้องขอใช้กับระบบย่อย ฟิลด์ `object` จะตั้งค่าเป็นค่าคงที่ SUBSYSTEM หรือ ฟิลด์ `object` จะตั้งค่าเป็นจุดโค้ดเซิร์ฟเวอร์ย่อย หรือฟิลด์ `objname` จะตั้งค่าเป็น PID ของเซิร์ฟเวอร์ย่อย ซึ่งเป็นสตริงอักขระ และเป็นความรับผิดชอบของระบบย่อย ในการกำหนดอ็อบเจ็กต์ที่คำร้องขอใช้

ฟิลด์ `action` ระบุการดำเนินการที่ร้องขอของระบบย่อย ระบบย่อยควรเข้าใจโค้ดการดำเนินการ START, STOP และ STATUS โค้ดการดำเนินการ TRACE และ REFRESH เป็นโค้ดเผื่อเลือก

ฟิลด์ `parm1` และ `parm2` จะถูกใช้ต่างกันตามการดำเนินการแต่ละตัว

แอ็คชัน	parm1	parm2
STOP	NORMAL หรือ FORCE	
STATUS	LONGSTAT หรือ SHORTSTAT	
TRACE	LONGTRACE หรือ SHORT-TRACE	TRACEON หรือ TRACEOFF

การดำเนินการ START เซิร์ฟเวอร์ย่อยและ REFRESH ไม่ได้ใช้ฟิลด์ `parm1` และ `parm2`

การเขียนโปรแกรมระบบย่อยเพื่อตอบสนองคำร้องขอ SRC

การดำเนินการกับระบบย่อยที่เหมาะสมที่สุดสำหรับการร้องขอ SRC คือการโปรแกรม เมื่ออ็อบเจ็กต์ระบบย่อยถูกกำหนดให้กับ SRC โครงสร้างที่ระบบย่อยใช้เพื่อตอบกลับไปยังคำร้องขอ SRC จะถูกกำหนดในไฟล์ **/usr/include/spc.h** ระบบย่อยอาจ

ใช้รูทีนย่อยแบบรันไทม์ของ SRC ต่อไปนี้เพื่อปฏิบัติตามข้อกำหนดการประมวลผลคำสั่ง:

รูทีนย่อย	คำอธิบาย
srcrrqs	อนุญาตให้ระบบย่อยเก็บส่วนหัวจากคำร้องขอ
srcsrpy	อนุญาตให้ระบบย่อยส่งการตอบกลับไปยังคำร้องขอ

การประมวลผลการร้องขอสถานะต้องการให้รวมภารกิจ และรูทีนย่อยเข้าด้วยกัน

เมื่อระบบย่อยได้รับคำร้องขอที่ไม่สามารถประมวลผลได้ หรือไม่ถูกต้อง ระบบย่อยเหล่านั้นต้องส่งแพ็กเก็ตข้อผิดพลาดด้วยโค้ดระบุความผิดพลาดของ **SRC_SUBICMD** ในการตอบกลับไปยังคำร้องขอที่ไม่รู้จักหรือไม่ถูกต้อง SRC จะสงวนโค้ดการดำเนินการ 0-255 สำหรับ SRC ใช้ภายใน ถ้าระบบย่อยของคุณได้รับคำร้องขอ ที่มีโค้ดการดำเนินการที่ไม่ถูกต้อง ระบบของคุณต้องส่งคืนโค้ดระบุความผิดพลาดของ **SRC_SUBICMD** โค้ดระบุความผิดพลาดที่ถูกต้องจะสนับสนุนโดย SRC ซึ่งถูกกำหนดในไฟล์ **spc.h** คุณยังสามารถกำหนดโค้ดการดำเนินการ ที่ระบุเฉพาะกับระบบย่อยได้ โค้ดการดำเนินการไม่ถูกต้องหากกำหนดโดย SRC หรือระบบย่อยของคุณ

> **หมายเหตุ:** โค้ดการดำเนินการ 0-255 ถูกสงวนไว้สำหรับให้ SRC ใช้

การประมวลผลคำร้องขอสถานะ SRC

ระบบย่อยอาจถูกร้องขอเพื่อให้จัดเตรียมรายงานชนิดของสถานะสามแบบ : สถานะของระบบย่อยแบบยาว สถานะของระบบย่อยแบบสั้น และสถานะของเซิร์ฟเวอร์ย่อยแบบยาว

> **หมายเหตุ:** การรายงานสถานะของระบบย่อยแบบสั้นจะถูกดำเนินการโดย **srcmstr** daemon ค่าคงที่ Statcode และสถานะการตอบกลับ สำหรับชนิดของรายงานนี้จะถูกกำหนดในไฟล์ **/usr/include/spc.h** ตาราง Status Value Constants ที่ต้องการและค่าที่แนะนำสำหรับโค้ดสถานะการตอบกลับ

โค้ดค่าสถานะการตอบกลับ

ค่า	ความหมาย	ระบบย่อย	เซิร์ฟเวอร์ย่อย
SRCWARN	รับคำร้องขอให้หยุดทำงาน จะหยุดทำงานภายใน 20 วินาที)	X	X
SRCACT	สตาร์ทและแอ็คทีฟ	X	X
SRCINAC	ไม่แอ็คทีฟ		
SRCINOP	ไม่ทำงาน	X	X
SRCLOSD	ปิด		
SRCLSPN	ในการประมวลผลที่ต้องการปิด		
SRCNOSTAT	เวลาที่ไม่มีการใช้งาน		
SRCOBIN	เปิด แต่ไม่แอ็คทีฟ		
SRCOPND	เปิด		
SRCOPPN	ในการประมวลผลที่ต้องการเปิด		
SRCSTAR	การเริ่มต้น		X

ค่า	ความหมาย	ระบบย่อย	เซิร์ฟเวอร์ย่อย
SRCSTPG	การหยุดทำงาน	X	X
SRCTST	TEST แอ็คทีฟ		
SRCTSTPN	TEST ค้างอยู่		

คำสั่ง SRC **lssrc** จะแสดงข้อมูลที่ได้รับเกี่ยวกับเอาต์พุตมาตรฐาน ข้อมูลจะถูกส่งคืนโดยระบบย่อยเพื่อตอบกลับไปยังคำร้องขอสถานะแบบยาว ที่อยู่ด้านซ้ายของการพิจารณาของระบบย่อย ระบบย่อยที่เป็นเจ้าของเซิร์ฟเวอร์ย่อย จะรับผิดชอบต่อการติดตามและรายงานสถานะที่เปลี่ยนแปลงของเซิร์ฟเวอร์ย่อย หากต้องการ ให้ใช้รูทีนย่อย **srcstathdr** เพื่อเรียกข้อมูลสถานะของส่วนหัวเพื่อส่งกลับไปที่จุดเริ่มต้น ของข้อมูลสถานะของคุณ

ขั้นตอนต่อไปนี้คือขั้นตอนที่แนะนำในการประมวลผลคำร้องขอสถานะ :

1. หากต้องการส่งคืนสถานะจากระบบย่อย (แบบสั้นหรือยาว) ให้จัดสรรอาร์เรย์ของโครงสร้าง **statcode** บวกกับโครงสร้าง **srchdr** โครงสร้าง **srchdr** ต้องเริ่มต้นบัฟเฟอร์ที่คุณต้องการส่งเพื่อตอบกลับไปยัง คำร้องขอสถานะ โครงสร้าง **statcode** จะถูกกำหนดไว้ในไฟล์ **/usr/include/spc.h**

   ```
   struct statcode
   {
      short objtype;
      short status;
      char objtext [65];
      char objname [30];
   };
   ```

2. กรอกข้อมูลในฟิลด์ `objtype` ด้วยค่าคงที่ SUBSYSTEM เพื่อบ่งชี้สถานะสำหรับระบบย่อย หรือด้วยโค้ดเซิร์ฟเวอร์ย่อยที่ชี้เพื่อบ่งชี้สถานะสำหรับเซิร์ฟเวอร์ย่อย
3. กรอกข้อมูลในฟิลด์ `status` ด้วยหนึ่งในค่าคงที่ของสถานะ SRC ที่กำหนดในไฟล์ **spc.h**
4. กรอกข้อมูลลงในฟิลด์ `objtext` ด้วยข้อความ NLS ที่คุณต้องการแสดงเป็นสถานะ ฟิลด์นี้ต้องมีค่าเป็นสตริงที่ถูกยกเลิก NULL
5. กรอกข้อมูลในฟิลด์ `objname` ด้วยชื่อของระบบย่อย หรือเซิร์ฟเวอร์ย่อยสำหรับฟิลด์ `objtext` ที่ใช้ ฟิลด์นี้ต้องมีค่าเป็นสตริงที่ถูกยกเลิก NULL

หมายเหตุ: ระบบบย่อยและผู้ร้องขอสามารถยอมรับการส่งข้อมูลที่กำหนดโดยระบบย่อยอื่น กลับไปยังผู้ร้องขอได้

การเขียนโปรแกรมระบบย่อยเพื่อส่งแพ็กเก็ตตอบกลับ

แพ็กเก็ตที่ระบบย่อยส่งคืนกลับไปยัง SRC ควรอยู่ในรูปแบบโครงสร้าง **srcrep** ตามที่กำหนดไว้ในไฟล์ **/usr/include/spc.h** โครงสร้ง **svrreply** ที่เป็นส่วนหนึ่งของโครงสร้าง **srcrep** จะมีการตอบกลับระบบย่อย:

```
struct svrreply
{
      short rtncode;        /*return code from the subsystem*/
      short objtype;        /*SUBSYSTEM or SUBSERVER*/
      char objtext[65];     /*object description*/
      char objname[20];     /*object name*/
      char rtnmsg[256];     /*returned message*/
};
```

ใช้รูทีนย่อย **srcsrpy** เพื่อส่งคืนแพ็กเก็ตไปยังผู้ร้องขอ

การสร้างการตอบกลับ

หากต้องการโปรแกรมการตอบกลับระบบย่อย ให้ใช้โพรซีเดอร์ต่อไปนี้:

1. กรอกข้อมูลลงในฟิลด์ `rtncode` ด้วยโค้ดระบุความผิดพลาดของ SRC ที่ใช้ ให้ใช้ **SRC_SUBMSG** เป็นฟิลด์ `rtncode` เพื่อส่งคืนข้อความ NLS กลับไปยังระบบย่อยที่ระบุเฉพาะ
2. กรอกข้อมูลในฟิลด์ `objtype` ด้วยค่าคงที่ SUBSYSTEM เพื่อบ่งชี้การตอบกลับสำหรับระบบย่อย หรือด้วยโค้ดเซิร์ฟเวอร์ย่อยที่ชี้เพื่อบ่งชี้สถานะสำหรับเซิร์ฟเวอร์ย่อย
3. กรอกข้อมูลในฟิลด์ `objname` ด้วยชื่อระบบย่อย ชนิดเซิร์ฟเวอร์ย่อย หรืออ็อบเจ็กต์เซิร์ฟเวอร์ย่อยที่ใช้เพื่อตอบกลับ
4. กรอกข้อมูลในฟิลด์ `rtnmsg` ด้วยข้อความ NLS ที่ระบุเฉพาะกับระบบย่อย
5. คีย์รายการที่เหมาะสมในพารามิเตอร์ **srcsrpy** *Continued* โปรดดู "แพ็กเก็ตความต่อเนื่อง srcsrpy " สำหรับข้อมูลเพิ่มเติม

> **หมายเหตุ:** แพ็กเก็ตท้ายสุดจากระบบย่อย ต้องมีคำว่า END ระบุอยู่ในพารามิเตอร์ *Continued* ในรูทีนย่อย **srcsrpy**

แพ็กเก็ตความต่อเนื่อง srcsrpy

ระบบย่อยที่ตอบกลับไปยังคำร้องขอ SRC จะถูกทำในรูปของ แพ็กเก็ตความต่อเนื่อง แพ็กเก็ตความต่อเนื่องมีอยู่ด้วยกันสองชนิด ซึ่งอาจถูกระบุ: ข้อความรายละเอียด และแพ็กเก็ตตอบกลับ

ข้อความรายละเอียดไม่ได้ส่งกลับไปยังไคลเอ็นต์ แต่จะพิมพ์ออกไปเป็นเอาต์พุตมาตรฐานของไคลเอ็นต์แทน ข้อความต้องประกอบด้วยข้อความ NLS พร้อมกับข้อความโทเค็นที่กรอกข้อมูลโดยระบบย่อยที่ส่ง หากต้องการส่งชนิดของแพ็กเก็ตความต่อเนื่องแบบนี้ ให้ระบุ CONTINUED ลงในรูทีนย่อย **srcsrpy** พารามิเตอร์ *Continued*

> **หมายเหตุ:** การดำเนินการระบบย่อย STOP ไม่อนุญาตให้ใช้ชนิดของความต่อเนื่องใดๆ อย่างไรก็ตาม การดำเนินการอื่นๆ ทั้งหมดที่ร้องขอซึ่งได้รับโดยระบบย่อย จาก SRC อาจส่งข้อความรายละเอียด

แพ็กเก็ตตอบกลับจะถูกส่งกลับไปยังไคลเอ็นต์ สำหรับการประมวลผลเพิ่มเติม ดังนั้น แพ็กเก็ตต้องยอมรับโดยระบบย่อยและผู้ร้องขอ ตัวอย่างหนึ่งขอชนิดของความต่อเนื่องนี้คือ การร้องขอสถานะ ขณะที่ตอบกลับไปยังการร้องขอสถานะของระบบย่อย ให้ระบุ STATCONTINUED ในพารามิเตอร์ **srcsrpy** *Continued* เมื่อการรายงานสถานะเสร็จสิ้นแล้ว หรือส่งแพ็กเกจการตอบกลับที่กำหนดโดยระบบย่อยทั้งหมดแล้ว ให้ระบุ END ในพารามิเตอร์ **srcsrpy** *Continued* แพ็กเก็ตจะส่งไปยังไคลเอ็นต์เพื่อบ่งชี้ถึงจุดสิ้นสุดของการตอบกลับ

การเขียนโปรแกรมระบบย่อยเพื่อส่งคืนแพ็กเก็ตข้อผิดพลาด SRC

ระบบย่อยต้องการส่งคืนแพ็กเก็ตข้อผิดพลาดสำหรับทั้งข้อผิดพลาดของ SRC และข้อผิดพลาดที่ไม่ใช่ SRC

ขณะที่ส่งคืนข้อผิดพลาด SRC แพ็กเก็ตการตอบกลับที่ระบบย่อยส่งคืน ควรอยู่ในรูปแบบของโครงสร้าง**svrreply** ของโครงสร้าง **srcrep** ที่มีฟิลด์ `objname` ถูกกรอกข้อมูลด้วยชื่อของระบบย่อย ชนิดเซิร์ฟเวอร์ย่อย หรืออ็อบเจ็กต์ของเซิร์ฟเวอร์ย่อยที่มีข้อผิดพลาด ถ้าข้อความ NLS ที่เชื่อมโยงกับหมายเลขข้อผิดพลาดของ SRC ไม่ได้สอดแทรกโทเค็นใดๆ ไว้ แพ็กเก็ตข้อผิดพลาดจะถูกส่งคืนในรูปแบบสั้นๆ นั่นหมายความว่า แพ็เก็ตข้อผิดพลาดมีหมายเลขข้อผิดพลาดของ SRC เท่านั้น อย่างไรก็ตาม ถ้าโทเค็นถูกเชื่อมโยงกับหมายเลขข้อผิดพลาด เนื้อความ NLS มาตรฐานจากแค็ตตาล็อกข้อความควรถูกส่งคืน

เมื่อส่งคืนข้อผิดพลาดที่ไม่ใช่ SRC แล้ว แพ็กเก็ตการตอบกลับควรมีโครงสร้าง **svrreply** พร้อมกับฟิลด์ `rtncode` ที่ตั้งค่าเป็นค่าคงที่ SRC_SUBMSG และฟิลด์ `rtnmsg` ที่ตั้งค่าเป็นข้อความ NLS ที่ระบุเฉพาะกับระบบย่อย ฟิลด์ `rtnmsg` จะถูกพิมพ์ไปยังเอาต์พุตมาตรฐานของไคลเอ็นต์

การตอบสนองต่อคำร้องขอการติดตาม

ส่วนสนับสนุนสำหรับคำสั่ง **traceson** และ **tracesoff** คือการพึ่งพาระบบย่อย ถ้าคุณเลือกที่จะสนับสนุนคำสั่งเหล่านี้ การดำเนินการติดตามสามารถระบุได้สำหรับระบบย่อยและเซิร์ฟเวอร์ย่อย

การร้องขอการติดตามระบบบ่อยจะได้รับในรูปแบบดังนี้: คำร้องขอการติดตามระบบย่อยจะมีฟิลด์ **subreq** `action` ที่ตั้งค่าเป็นค่าคงที่ TRACE และฟิลด์ **subreq** `object` ที่ตั้งค่าเป็นค่าคงที่ SUBSYSTEM การดำเนินการติดตามใช้ `parm1` เพื่อบ่งชี้ถึงการติดตาม LONGTRACE หรือ SHORTTRACE และใช้ `parm2` เพื่อบ่งชี้ TRACEON หรือ TRACEOFF

เมื่อระบบย่อยได้รับแพ็กเก็ตการติดตามระบบพร้อมกับ `parm1` ที่มีค่า SHORTTRACE และ `parm2` ที่มีค่า TRACEON ระบบย่อยควรเปิดการติดตามแบบสั้น ในทางกลับกัน เมื่อระบบย่อยได้รับแพ็กเก็ตการติดตามระบบย่อยพร้อมกับ `parm1` ที่มีค่า LONGTRACE และ `parm2` ที่มีค่า TRACEON ระบบย่อยควรเปิดการติดตามแบบยาว เมื่อระบบย่อยได้รับแพ็กเก็ตการติดตามระบบพร้อมกับ `parm2` ที่มีค่า TRACEOFF ระบบย่อยควรปิดการติดตามระบบย่อย

การร้องขอการติดตามเซิร์ฟเวอร์ย่อยจะได้รับในรูปแบบต่อไปนี้: คำร้องขอการติดตามเซิร์ฟเวอร์ย่อยจะมีฟิลด์ **subreq** `action` ที่ตั้งค่าเป็นค่าคงที่ TRACE และฟิลด์ **subreq** `object` ที่ตั้งค่าเป็นโค้ดเซิร์ฟเวอร์ย่อย ที่ชี้จุดของเซิร์ฟเวอร์ที่ส่งสถานะ การดำเนินการติดตามจะใช้ `parm1` เพื่อบ่งชี้ถึง LONGTRACE หรือ SHORTTRACE และ `parm2` เพื่อบ่งชี้ TRACEON หรือ TRACEOFF

เมื่อระบบย่อยได้รับแพ็กเก็ตการติดตามเซิร์ฟเวอร์ย่อยพร้อมกับ `parm1` ที่มีค่า SHORTTRACE และ `parm2` ที่มีค่า TRACEON ระบบย่อยควรเปิดการติดตามเซิร์ฟเวอร์ย่อยแบบสั้น ในทางกลับกัน เมื่อระบบย่อยได้รับแพ็กเก็ตการติดตามเซิร์ฟเวอร์ย่อยพร้อมกับ `parm1` ที่มีค่า LONGTRACE และ `parm2` ที่มีค่า TRACEON ระบบย่อยควรเปิดการติดตามเซิร์ฟเวอร์ย่อยแบบยาว เมื่อระบบย่อยได้รับแพ็กเก็ตการติดตามเซิร์ฟเวอร์ย่อยพร้อมกับ `parm2` ที่มีค่า TRACEOFF ระบบย่อยควรปิดการติดตามเซิร์ฟเวอร์ย่อย

การตอบสนองต่อคำร้องขอรีเฟรช

ส่วนสนับสนุนสำหรับการร้องขอให้รีเฟรชระบบย่อยเป็นการพึ่งพาระบบย่อย โปรแกรมเมอร์ระบบย่อยที่เลือกที่จะสนับสนุนคำสั่ง **refresh** ควรโปรแกรมระบบย่อยเพื่อโต้ตอบกับ SRC ด้วยวิธีต่อไปนี้:

- คำร้องขอให้รีเฟรชระบบย่อยจะมีโครงสร้าง **subreq** ฟิลด์ `action` ที่ตั้งค่าเป็นค่าคงที่ REFRESH และโครงสร้าง **subreq** ฟิลด์ `object` ที่ตั้งค่าเป็นค่าคงที่ SUBSYSTEM การดำเนินการรีเฟรชระบบย่อย ไม่ได้ใช้ `parm1` หรือ `parm2`
- เมื่อระบบย่อยได้รับคำร้องขอให้รีเฟรช ระบบย่อยจะปรับแต่งตัวระบบเอง

การกำหนดระบบย่อยของคุณให้กับ SRC

ระบบย่อยจะถูกนิยามให้กับคลาสอ็อบเจ็กต์ SRC ให้เป็นอ็อบเจ็กต์ระบบย่อย เซิร์ฟเวอร์ย่อยถูกกำหนดในฐานข้อมูลคอนฟิกูเรชัน SRC ให้เป็นอ็อบเจ็กต์ชนิดเซิร์ฟเวอร์ย่อย

โครงสร้างที่เชื่อมโยงกับแต่ละชนิดของอ็อบเจ็กต์ถูกกำหนดอยู่ในไฟล์ **sys/srcobj.h**

อ็อบเจ็กต์ของระบบจะถูกสร้างด้วยคำสั่ง **mkssys** หรือรูทีนย่อย **addssys** อ็อบเจ็กต์ชนิดเซิร์ฟเวอร์ย่อยจะถูกสร้างด้วยคำสั่ง **mkserver** คุณอาจไม่ต้องระบุอ็อพชันที่เป็นไปได้ทั้งหมด และพารามิเตอร์ที่ใช้คำสั่งและรูทีนย่อย configuration SRC นำเสนอค่าดีฟอลต์ที่ตั้งค่าไว้ล่วงหน้า ซึ่งคุณต้องระบุเฉพาะฟิลด์บังคับ และฟิลด์ใดๆ ที่คุณต้องการให้มีค่าบางค่า ที่ไม่ใช่ค่าดีฟอลต์

Descriptor สามารถถูกเพิ่มหรือแก้ไขได้ที่บรรทัดรับคำสั่งโดยเขียนสคริปต์ shell และยังสามารถเพิ่มหรือแก้ไขได้โดยใช้อินเตอร์เฟส C คำสั่งและรูทีนย่อยจะพร้อมใช้งานสำหรับการปรับแต่งและการแก้ไขอ็อบเจ็กต์ SRC

หมายเหตุ: ตัวเลือกของโปรแกรมมิงอินเตอร์เฟสจะถูกจัดเตรียมไว้สำหรับความสะดวกสบายเท่านั้น

ที่บรรทัดรับคำสั่ง ให้ใช้คำสั่งต่อไปนี้:

คำสั่ง	คำอธิบาย
mkssys	เพิ่มนิยามของระบบย่อยให้กับฐานข้อมูลคอนฟิกูเรชัน SRC
mkserver	เพิ่มนิยามของเซิร์ฟเวอร์ย่อยให้กับฐานข้อมูลคอนฟิกูเรชัน SRC
chssys	เปลี่ยนนิยามของระบบย่อยในฐานข้อมูลคอนฟิกูเรชัน SRC
chserver	เปลี่ยนนิยามเซิร์ฟเวอร์ย่อยในฐานข้อมูลคอนฟิกูเรชัน SRC
rmssys	ลบนิยามของระบบย่อยออกจากฐานข้อมูลคอนฟิกูเรชัน SRC
rmserver	ลบนิยามของเซิร์ฟเวอร์ย่อยออกจากฐานข้อมูลคอนฟิกูเรชัน SRC

ขณะที่ใช้อินเตอร์เฟส C ให้ใช้รูทีนย่อยต่อไปนี้:

รูทีนย่อย	คำอธิบาย
addssys	เพิ่มนิยามของระบบย่อยให้กับฐานข้อมูลคอนฟิกูเรชัน SRC
chssys	เปลี่ยนนิยามของระบบย่อยในฐานข้อมูลคอนฟิกูเรชัน SRC
defssys	กำหนดค่าเริ่มต้นให้กับนิยามของระบบย่อยใหม่ด้วยค่าดีฟอลต์
delssys	ลบนิยามของระบบย่อยที่มีอยู่ออกจากฐานข้อมูลคอนฟิกูเรชัน SRC **หมายเหตุ:** โค้ดอ็อบเจ็กต์ที่ทำงานด้วยรูทีนย่อย **chssys** ต้องรันด้วยระบบกลุ่ม
getssys	ขอรับนิยามของระบบย่อยจากฐานข้อมูลคอนฟิกูเรชัน SRC
getsubsvr	ขอรับนิยามของเซิร์ฟเวอร์ย่อยจากฐานข้อมูลคอนฟิกูเรชัน SRC

คำสั่ง **mkssys** และ **mkserver** จะเรียกรูทีนย่อย **defssys** ภายใน เพื่อกำหนดค่าดีฟอลต์ของระบบย่อยและเซิร์ฟเวอร์ย่อยก่อนที่จะเพิ่มหรือแก้ไขค่าใดๆ ที่ป้อนไว้ที่บรรทัดรับคำสั่ง

รูทีนย่อย **getssys** และ **getsubsvr** จะถูกนำมาใช้เมื่อโปรแกรมต้นแบบ SRC หรือโปรแกรมระบบย่อยต้องการดึงข้อมูลจากไฟล์คอนฟิกูเรชัน SRC

รายการของรูทีนย่อย SRC เพิ่มเติม

ใช้รูทีนย่อยต่อไปนี้เพื่อโปรแกรมการสื่อสารด้วย SRC และระบบย่อยที่ควบคุมโดย SRC:

รูทีนย่อย	คำอธิบาย
src_err_msg	ส่งคืนเนื้อความสำหรับข้อผิดพลาด SRC ที่พบโดยรูทีนไลบรารี SRC (โปรดดูเวอร์ชัน **src_err_msg_r** ที่ไม่มีผลกับการทำงาน)
srcsbuf	ร้องขอสถานะจากระบบย่อยในรูปแบบที่สามารถพิมพ์ได้ (และดูเวอร์ชัน **srcsbuf_r** ที่ไม่มีผลกับการทำงาน)
srcsrqt	ส่งข้อความหรือคำร้องขอไปยังระบบย่อย (และดูเวอร์ชัน **srcsrqt_r** ที่ไม่มีผลกับการทำงาน)
srcstat	ร้องขอสถานะของระบบย่อยแบบย่อ (และดูเวอร์ชัน **srcstat_r** ที่ไม่มีผลกับการทำงาน)
srcstathdr	ขอรับข้อความหัวเรื่องสำหรับสถานะ SRC
srcstattxt	ขอรับการแทนค่าข้อความสำหรับโค้ดรหัส SRC (และดูเวอร์ชัน **srcstattxt_r** ที่ไม่มีผลกับการทำงาน
srcstop	ร้องขอการยกเลิกระบบย่อย
srcstrt	ร้องขอสตาร์ทของระบบย่อย

ตัวช่วยการติดตาม

ตัวช่วยการติดตามจะช่วยคุณแยกปัญหาของระบบ โดยมอนิเตอร์เหตุการณ์ของระบบหรือการประมวลผลที่เลือกไว้ เหตุการณ์ที่สามารถมอนิเตอร์ได้คือ: การเข้าและการออกจากรูทีนย่อยที่เลือกไว้ รูทีนเคอร์เนล รูทีนส่วนขยายเคอร์เนล และ interrupt handler

การติดตามยังสามารถจำกัดการติดตาม ชุดของการประมวลผลหรือเธรดที่รันอยู่ หรือสามารถนำมาใช้เพื่อเริ่มต้น และติดตามโปรแกรม

เมื่อตัวช่วยการติดตามแอ็คทีฟอยู่ ข้อมูลจะถูกบันทึกไว้ในไฟล์บันทึกการติดตามของระบบ ตัวช่วยการติดตามจะสอดแทรกคำสั่งสำหรับการเรียกใช้งาน และการควบคุมการติดตาม รวมถึงการสร้างรายงานการติดตาม แอ็พพลิเคชันและส่วนขยายเคอร์เนล สามารถใช้รูทีนย่อยต่างๆ เพื่อบันทึกเหตุการณ์เพิ่มเติมได้

สำหรับข้อมูลเพิ่มเติมเกี่ยวกับตัวช่วยการติดตาม โปรดอ้างถึงหัวข้อต่อไปนี้:

ภาพรวมของตัวช่วยการติดตาม

ตัวช่วยการติดตามจะอยู่ในชุดไฟล์ **bos.sysmgt.trace** หากต้องการดูว่า คุณได้ติดตั้งชุดไฟล์นี้หรือไม่ ให้พิมพ์คำสั่งต่อไปนี้บนบรรทัดรับคำสั่ง:

```
lslpp -l | grep bos.sysmgt.trace
```

ถ้าผลลัพธ์ที่ได้มีบรรทัด **bos.sysmgt.trace** แสดงอยู่ นั่นหมายความว่า คุณได้ติดตั้งชุดไฟล์แล้ว มิฉะนั้น คุณต้องติดตั้งชุดไฟล์

ตัวช่วยการติดตามของระบบจะบันทึกเหตุการณ์การติดตามซึ่งสามารถจัดรูปแบบได้ในภายหลัง โดยใช้คำสั่งรายงานการติดตาม เหตุการณ์การติดตามจะถูกคอมไพล์ในโค้ดเคอร์เนลหรือโค้ดแอ็พพลิเคชัน แต่จะถูกติดตามก็ต่อเมื่อการติดตามนั้นแอ็คทีฟอยู่

การติดตามจะถูกเรียกใช้งานพร้อมกับคำสั่ง **trace** หรือรูทีนย่อย **trcstart** การติดตามจะหยุดทำงานด้วยคำสั่ง **trcstop** หรือรูทีนย่อย **trcstop** อย่างใดอย่างหนึ่ง ขณะที่แอ็คทีฟ การติดตามจะสามารถหยุดทำงานชั่วคราวหรือกลับสู่การทำงานต่อด้วยคำสั่ง **trcoff** และ **trcon** หรือรูทีนย่อย **trcoff** และ **trcon**

หากการติดตามได้ถูกหยุดทำงานด้วยคำสั่ง **trcstop** รายงานการติดตามสามารถสร้างขึ้นได้ด้วยคำสั่ง **trcrpt** คำสั่งนี้จะใช้ไฟล์เท็มเพลต **/etc/trcfmt** เพื่อให้ทราบถึงวิธีการจัดรูปแบบรายการ เท็มเพลตจะถูกติดตามด้วยคำสั่ง **trcupdate** สำหรับการกล่าวถึงเท็มเพลต โปรดดูคำสั่ง **trcupdate**

การควบคุมการติดตาม

คำสั่ง **trace** จะเริ่มต้นการติดตามเหตุการณ์ของระบบ และควบคุมบัฟเฟอร์การติดตามและขนาดของไฟล์บันทึกการทำงาน คำสั่งนี้จะถูกจัดทำเป็นเอกสารในหัวข้อ **trace** daemon ในการอ้างอิงของคำสั่ง

มีเมธอดของการเก็บรวบรวมข้อมูลการติดตามอยู่ด้วยกันสามวิธี

1. เมธอดดีฟอลต์คือ การใช้บัฟเฟอร์สองชุดเพื่อเก็บรวบรวมข้อมูลการติดตาม หนึ่งบัฟเฟอร์สำหรับการเขียน ขณะที่ข้อมูลจะถูกส่งไปยังอีกบัฟเฟอร์หนึ่ง ไฟล์บันทึกการทำงาน จะตัดคำ เมื่อไฟล์นั้นมีขนาดเต็มแล้ว
2. เมธอดแบบวนรอบจะเก็บรวบรวมข้อมูลการติดตามอย่างต่อเนื่อง แต่จะเขียนข้อมูลลงในไฟล์บันทึกการทำงานก็ต่อเมื่อการติดตามนั้นหยุดทำงานแล้ว เมธอดนี้มีประโยชน์โดยเฉพาะ สำหรับการดีบักปัญหาที่คุณทราบเมื่อเกิดปัญหาขึ้น และคุณเพียงแค่ต้องการดักจับข้อมูล ณ เวลานั้น คุณสามารถเริ่มต้นการติดตามได้ทุกเวลา และหยุดการติดตามหลังจากที่ปัญหาเกิดขึ้น และคุณจะมีเหตุการณ์ที่ถูกดักจับโดยรอบของปัญหา เมธอดนี้จะถูกเปิดใช้งานด้วยแฟล็ก **-l** trace daemon
3. อ็อพชันที่สามจะใช้หนึ่งบัฟเฟอร์การติดตาม และออกจากการติดตามเมื่อบัฟเฟอร์นั้นถูกเติม และเขียนบัฟเฟอร์ลงในไฟล์บันทึกการทำงาน การติดตามไม่ได้หยุดทำงานที่จุดนี้ แต่การติดตามจะถูกปิดเนื่องมาจากการออกคำสั่ง **trcoff** แทนที่จุดนี้ คุณจะมีความต้องการหยุดการติดตามด้วยคำสั่ง **trcstop** เสมอ บ่อยครั้งที่อ็อพชันนี้จะถูกนำมาใช้เพื่อเก็บรวบรวมข้อมูลประสิทธิภาพการทำงานที่เราไม่ต้องการให้การติดตามทำ i/o หรือการสลับบัฟเฟอร์จนกว่าข้อมูลจะถูกเก็บรวบรวม ใช้แฟล็ก **-f** เพื่อเปิดใช้งานอ็อพชันนี้

โดยปกติแล้ว คุณจะต้องการรันคำสั่ง trace แบบอะซิงโครนัส หรืออีกนัยหนึ่งคือ คุณต้องการป้อนคำสั่ง trace จากนั้นทำงานอื่นต่อไป หากต้องการรันการติดตามแบบอะซิงโคนัส ให้ใช้แฟล็ก **-a** หรือแฟล็ก **-x** ถ้าคุณใช้แฟล็ก **-a** คุณต้องหยุดการติดตามด้วยคำสั่ง **trcstop** ถ้าคุณใช้แฟล็ก **-x** การติดตามจะหยุดโดยอัตโนมัติ เมื่อโปรแกรมเสร็จสิ้นแล้ว

ซึ่งปกติแล้วจะออกแบบมาเพื่อจำกัดข้อมูลที่ถูกติดตาม ใช้แฟล็ก **-j events** หรือแฟล็ก **-k events** เพื่อระบุชุดของเหตุการณ์ที่ต้องการสอดแทรก (**-j**) หรือแยกออก (**-k**)

หมายเหตุ: เมื่อคุณจำกัดการติดตามกับการประมวลผลหรือเธรดโดยเฉพาะ คุณยังต้องจำกัดจำนวนของข้อมูลที่ติดตาม

หากต้องการแสดงชื่อโปรแกรมที่เชื่อมโยงกับการติดตาม hook ซึ่งต้องเปิดใช้งาน hook บางตัว และจะระบุได้โดยใช้กลุ่มเหตุการณ์การติดตาม **tidhk** ตัวอย่างเช่น คุณต้องการติดตาม **mbuf** hook, 254 และแสดงชื่อโปรแกรมด้วย คุณจำเป็นต้องรัน **trace** ดังต่อไปนี้:

```
trace -aJ tidhk -j 254
```

การติดตามจะเกิดขึ้น หากต้องการหยุดการติดตาม ให้พิมพ์คำสั่งต่อไปนี้บนบรรทัดรับคำสั่ง:

```
trcstop
trcrpt -O exec=on
```

อ็อพชัน **-O exec=on** trcrpt จะแสดงชื่อโปรแกรม โปรดดูคำสั่ง **trcrpt** สำหรับข้อมูลเพิ่มเติม

ซึ่งมีความสามารถในออกแบบมาเพื่อระบุขนาดบัฟเฟอร์และขนาดสูงสุดของไฟล์บันทึกการทำงาน บัฟเฟอร์การติดตามต้องการหน่วยความจำที่ใช้จริงซึ่งต้องพร้อมใช้งาน ดังนั้น จึงไม่มีการเพจที่จำเป็นต่อการบันทึกการติดตาม hook ไฟล์บันทึกการทำงานจะกรอกข้อมูลขนาดสูงสุดที่ระบุไว้ จากนั้นจะตัดคำโดยรอบ ละทิ้งข้อมูลการติดตามที่เก่าที่สุด แฟล็ก **-T size** and **-L size** ระบุขนาดของบัฟเฟอร์หน่วยความจำและขนาดสูงสุดของข้อมูลการติดตามที่อยู่ใน ไฟล์บันทึกการทำงานในหน่วยไบต์

หมายเหตุ: เนื่องจากตัวช่วยการติดตามตรึงบัฟเฟอร์การรวบรวมข้อมูลไว้ ซึ่งทำให้จำนวนของหน่วยความจำนี้ไม่พร้อมใช้งานกับส่วนที่เหลือของระบบ ตัวช่วยการติดตามอาจส่งผลกระทบกับผลการทำงานในสภาวะแวดล้อมที่มีข้อจำกัดด้านหน่วย

ความจำ ถ้าแอ็พพลิเคชันที่ถูกมอนิเตอร์ไม่มีข้อจำกัดด้านหน่วยความจำ หรือถ้าเปอร์เซ็นต์ของหน่วยความจำที่ใช้โดยรูทีนการติดตามมีขนาดเล็กเมื่อเทียบกับที่พร้อมใช้งานในระบบ ผลกระทบของการติดตามหน่วยความจำที่ถูก “ขโมย” ควรมีขนาดเล็ก ถ้าคุณไม่ได้ระบุค่า การติดตามจะใช้ขนาดที่เป็นค่าดีฟอลต์

การติดตามยังสามารถควบคุมได้จากแอ็พพลิเคชัน โปรดดูบทความ **trcstart** และ **trcstop**

การบันทึกข้อมูลเหตุการณ์การติดตาม

มีชนิดข้อมูลการติดตามอยู่ด้วยกันสองแบบ

ข้อมูลทั่วไป

ประกอบด้วยคำบรรจุข้อมูล บัฟเฟอร์ของข้อมูล opaque และความยาวของข้อมูล opaque ซึ่งจะมีประโยชน์สำหรับไอเท็มการติดตาม เช่น ชื่อพาธ โปรดดูบทความเรื่อง ช่องสัญญาณการติดตามทั่วไปได้ใน **ภาพรวมของตัวช่วยการติดตาม** ไม่ พบใน **ตัวช่วยการติดตาม**

หมายเหตุ: การติดตามของการประมวลผลหรือเธรดโดยเฉพาะ จะสนับสนุนสำหรับช่องสัญญาณ 0 และจะไม่สนับสนุนช่องสัญญาณการติดตามทั่วไป

ข้อมูลที่ไม่ใช่ข้อมูลทั่วไป

ข้อมูลนี้คือสิ่งที่ถูกติดตามโดยระบบปฏิบัติการ AIX entry ของชนิดนี้ประกอบด้วย hook word และมากสุด 5 คำของข้อมูลการติดตาม สำหรับแอ็พพลิเคชันแบบ 64 บิต คำเหล่านี้คือคำที่มีขนาด 8 ไบต์ โปรแกรมเมอร์ C ควรใช้แมโคร TRCHKL0 ผ่าน TRCHKL5 และ TRCHKL0T ผ่าน TRCHKL5T ซึ่งนิยามไว้ในไฟล์ **/usr/include/sys/trcmacros.h** เพื่อบันทึกข้อมูลที่ไม่ใช่ข้อมูลทั่วไป ถ้าแมโครเหล่านี้ไม่สามารถนำมาใช้ได้ โปรดดูบทความเรื่อง รูทีนย่อย **utrchook**

การสร้างรายงานการติดตาม

โปรดดูบทความเรื่อง คำสั่ง **trcrpt** สำหรับคำอธิบายโดยสมบูรณ์ของ **trcrpt** คำสั่งนี้จะถูกนำมาใช้ เพื่อสร้างรายการการติดตามที่สามารถอ่านได้จากไฟล์บันทึกการทำงานที่สร้างโดยคำสั่ง **trace** ตามค่าดีฟอลต์ คำสั่งจะจัดรูปแบบข้อมูลจากไฟล์บันทึกการทำงานที่เป็นค่าดีฟอลต์ **/var/adm/ras/trcfile** เอาต์พุต **trcrpt** จะถูกเขียนไปยังเอาต์พุตมาตรฐาน

หากต้องการสร้างรายงานการติดตามจากไฟล์บันทึกการทำงานที่เป็นค่าดีฟอลต์ และเขียนลงใน /tmp/rptout ให้ป้อน

```
trcrpt >/tmp/rptout
```

หากต้องการสร้างรายงานการติดตามจากไฟล์บันทึกการทำงาน /tmp/tlog to /tmp/rptout ซึ่งประกอบด้วยชื่อโปรแกรมและชื่อการเรียกระบบ ให้ใช้

```
trcrpt -O exec=on,svc=on /tmp/tlog >/tmp/rptout
```

การแตกข้อมูลการติดตามจากดัมพ์

ถ้าการติดตามแอ็คทีฟ เมื่อระบบใช้ดัมพ์ การติดตามสามารถเรียกข้อมูลได้ด้วยคำสั่ง **trcdead** หากต้องการหลีกเลี่ยงการเขียนไฟล์บันทึกการติดตามที่เป็นค่าดีฟอลต์บนระบบปัจจุบัน ให้ใช้อ็อพชัน **-o *output-file***

ตัวอย่างเช่น:

```
trcdead -o /tmp/tlog /var/adm/ras/vmcore.0
```

สร้างไฟล์บันทึกการติดตาม /tmp/tlog ซึ่งอาจจัดรูปแบบด้วยคำสั่งต่อไปนี้ :

```
trcrpt /tmp/tlog
```

คำสั่งตัวช่วยการติดตาม

คำสั่งต่อไปนี้เป็นส่วนหนึ่งของตัวช่วยการติดตาม:

คำสั่ง	ฟังก์ชัน
ติดตาม	เริ่มต้นการติดตามเหตุการณ์ของระบบ ด้วยคำสั่งนี้ คุณสามารถควบคุมขนาดและจัดการกับไฟล์บันทึกการติดตามและบัฟเฟอร์การติดตามภายใน ที่เก็บรวบรวมข้อมูลเหตุการณ์การติดตามไว้
trcdead	ดึงรายละเอียดการติดตามจากดัมพ์ของระบบ ถ้าระบบหยุดทำงาน ขณะที่ตัวช่วยการติดตามแอ็คทีฟอยู่ เนื้อหาของบัฟเฟอร์การติดตามภายในจะถูกดักจับไว้ คำสั่งนี้จะดึงข้อมูลเหตุการณ์การติดตามจากดัมพ์ และเขียนลงในไฟล์บันทึกการติดตาม
trcnm	สร้างรายชื่อเคอร์เนลที่ใช้โดยคำสั่ง **trcrpt** รายชื่อเคอร์เนลจะประกอบด้วยตารางสัญลักษณ์ และตารางสัญลักษณ์โหลดเดอร์ ของอ็อบเจ็กต์ไฟล์ คำสั่ง **trcrpt** ใช้ไฟล์รายชื่อเคอร์เนล เพื่อตีความแอดเดรสขณะที่จัดรูปแบบรายงานจากไฟล์บันทึกการติดตาม **หมายเหตุ:** ซึ่งขอแนะนำให้คุณใช้อ็อพชัน **–n trace** แทน **trcnm** ซึ่งจะใส่ข้อมูลรายชื่อ ลงในไฟล์บันทึกการติดตามแทนการแยกไฟล์ และสอดแทรกสัญลักษณ์จากส่วนขยายเคอร์เนล
trcrpt	จัดรูปแบบรายงานของข้อมูลเหตุการณ์การติดตามที่มีอยู่ในไฟล์บันทึกการติดตาม คุณสามารถระบุเหตุการณ์ที่จะสอดแทรก (หรือละเว้น) ในรายงาน และกำหนดการนำเสนอเอาต์พุตด้วยคำสั่งนี้ คำสั่ง **trcrpt** จะใช้เท็มเพลตการจัดรูปแบบการติดตามที่เก็บอยู่ในไฟล์ **/etc/trcfmt** เพื่อกำหนดวิธีการตีความข้อมูลที่บันทึกไว้สำหรับแต่ละเหตุการณ์
trcstop	หยุดการทำงานการติดตามเหตุการณ์ของระบบ
trcupdate	อัพเดตเท็มเพลตการจัดรูปแบบการติดตามที่เก็บอยู่ในไฟล์ **/etc/trcfmt** เมื่อคุณเพิ่มแอ็พพลิเคชันหรือส่วนขยายเคอร์เนลที่บันทึกเหตุการณ์การติดตามไว้ เท็มเพลตสำหรับเหตุการณ์เหล่านี้ต้องถูกเพิ่มลงในไฟล์ **/etc/trcfmt** คำสั่ง **trcrpt** จะใช้เท็มเพลตการจัดรูปแบบการติดตามเพื่อกำหนดวิธีการตีความข้อมูลที่บันทึกไว้ สำหรับแต่ละเหตุการณ์ ผลิตภัณฑ์ซอฟต์แวร์ที่บันทึกเหตุการณ์ไว้จะรันคำสั่ง **trcupdate** ซึ่งเป็นส่วนหนึ่งของกระบวนการติดตั้ง

รูทีนย่อยและการเรียกใช้ตัวช่วยการติดตาม

การเรียกและรูทีนย่อยต่อไปนี้ คือส่วนหนึ่งของตัวช่วยการติดตาม:

รูทีนย่อย	คำอธิบาย
trcgen, trcgent	บันทึกเหตุการณ์การติดตามของข้อมูลที่มีจำนวนคำมากกว่าห้าคำ รูทีนย่อย **trcgen** สามารถนำมาใช้เพื่อบันทึกเหตุการณ์ที่เป็นส่วนหนึ่งของการติดตามเหตุการณ์ของระบบ (การติดตามช่องสัญญาณ 0) หรือเพื่อบันทึกเหตุการณ์เกี่ยวกับช่องสัญญาณการติดตามทั่วไป (ช่องสัญญาณ 1 ถึง 7) ระบุหมายเลขช่องสัญญาณในพารามิเตอร์รูทีนย่อย เมื่อคุณบันทึกเหตุการณ์การติดตาม รูทีนย่อย **trcgent** จะผนวกการประทับเวลากับข้อมูลเหตุการณ์ ขณะที่ใช้ AIX 5L Version 5.3 with the 5300-05 Technology Level และสูงกว่า การประทับเวลาจะถูกผนวกเข้ากับข้อมูลเหตุการณ์โดยไม่พิจารณาถึงรูทีนย่อยที่ใช้ใช้ **trcgenk** และ **trcgenkt** ในเคอร์เนล โปรแกรมเมอร์ C ควรใช้แมโคร **TRCGEN** และ **TRCGENK** เสมอ

รูทีนย่อย	คำอธิบาย
utrchook, utrchook64	บันทึกเหตุการณ์การติดตามของข้อมูลที่มีจำนวนคำสูงสุดห้าคำ รูทีนย่อยเหล่านี้สามารถใช้เพื่อบันทึกเหตุการณ์ที่เป็นส่วนหนึ่งของการติดตามเหตุการณ์ของระบบ (การติดตามช่องสัญญาณ 0) โปรแกรมเมอร์เคอร์เนลสามารถใช้ **trchook** และ **trchook64** ได้ โปรแกรมเมอร์ C ควรใช้แมโคร **TRCHKL0** – **TRCHKL5** และ **TRCHKL0T** – **TRCHKL5T** เสมอ ถ้าคุณไม่ได้ใช้แมโครเหล่านี้ คุณจำเป็นต้องสร้างการติดตาม hook word ของคุณเอง รูปแบบจะจัดทำเป็นเอกสารด้วยไฟล์ **/etc/trcfmt** หมายเหตุ การติดตามแบบ 32 บิตและ 64 บิตจะมีรูปแบบ hook word ที่ต่างกัน
trcoff	หยุดการเก็บรวบรวมข้อมูลการติดตามชั่วคราวสำหรับช่องสัญญาณการติดตามเหตุการณ์ของระบบ (ช่องสัญญาณ 0) หรือการติดตามช่องสัญญาณทั่วไป (1 ถึง 7) ช่องสัญญาณการติดตามยังคงแอ็คทีฟ และการรวบรวมข้อมูลการติดตามสามารถกลับสู่การทำงานต่อได้โดยใช้รูทีนย่อย **trcon**
trcon	เริ่มต้นการเก็บรวบรวมข้อมูลการติดตามบนช่องสัญญาณการติดตาม ช่องสัญญาณสามารถเป็น ช่องสัญญาณการติดตามเหตุการณ์ของระบบ (0) หรือช่องสัญญาณทั่วไป (1 ถึง 7) อย่างไรก็ตาม ช่องสัญญาณการติดตามต้องถูกเรียกใช้งานก่อนหน้านี้โดยใช้คำสั่ง **trace** หรือรูทีนย่อย **trcstart** คุณสามารถหยุดการรวบรวมข้อมูลการติดตามชั่วคราวได้โดยใช้รูทีนย่อย **trcoff**
trcstart	จัดเตรียมอินเตอร์เฟสไลบรารีให้กับคำสั่งการติดตาม ซึ่งจะส่งคืน หมายเลขช่องสัญญาณของการติดตามที่เริ่มต้นขึ้น ถ้าช่องสัญญาณทั่วไปถูกร้องขอ หมายเลขช่องสัญญาณคือหนึ่งในหมายเลขต่อไปนี้: 1,2,3,4,5,6, 7 มิฉะนั้น หมายเลขช่องสัญญาณจะเป็น 0
trcstop	ทำให้ว่างและหยุดทำงานช่องสัญญาณการติดตามทั่วไป

ไฟล์ตัวช่วยการติดตาม

ไฟล์	คำอธิบาย
/etc/trcfmt	มีเท็มเพลตการจัดรูปแบบการติดตามที่ใช้โดยคำสั่ง **trcrpt** เพื่อกำหนดวิธีการตีความข้อมูลที่บันทึกไว้สำหรับแต่ละเหตุการณ์
/var/adm/ras/trcfile	มีไฟล์บันทึกการติดตามที่เป็นค่าดีฟอลต์ คำสั่ง **trace** อนุญาตให้คุณระบุไฟล์บันทึกการติดตามที่ต่างกัน
/usr/include/sys/trchkid.h	มีนิยามของ hook identifier การติดตาม
/usr/include/sys/trcmacros.h	มีการใช้แมโครทั่วไปสำหรับการบันทึกเหตุการณ์การติดตาม

ข้อมูลเหตุการณ์การติดตาม

โปรดดูไฟล์ **/etc/trcfmt** สำหรับการจัดรูปแบบของข้อมูลเหตุการณ์การติดตาม

ตัวบ่งชี้ hook การติดตาม

hook identifier การติดตามคือเลขฐานสิบหกที่มีสามหรือสี่หลัก ซึ่งระบุเหตุการณ์ที่กำลังติดตาม AIX 7.1 เวอร์ชันก่อนหน้าและบนแอ็พพลิเคชันแบบ 32 บิตที่รันอยู่บน AIX 7.1 และสูงกว่า จะมี hook identifier เพียงสามหลักเท่านั้นที่สามารถนำมาใช้ได้ ขณะที่ใช้แมโครการติดตาม เช่น **TRCHKL1** คุณระบุ hook การติดตามดังนี้:

```
hhh00000
```

โดยที่ hhh คือ ID hook

บนแอ็พพลิเคชัน 64 บิตและเคอร์เนลรูทีนที่รันบน AIX 7.1 และสูงกว่า สามารถใช้ตัวบ่งชี้ ที่มี สาม และสี่หลัก ขณะที่ใช้แมโครการติดตาม เช่น **TRCHKL1** คุณระบุ hook การติดตามดังนี้:

hhhh0000

โดยที่ hhhh คือ hook id

หมายเหตุ: ถ้าใช้ identifier ที่มีสี่หลักและ identifier มีขนาดน้อยกว่า 0x1000 หลักที่มีความสำคัญน้อยต้องมีค่าเป็น 0 (ในรูปของ 0x0hh0)

identifier ที่มีสามหลักจะมี 0 ในหลักที่มีความสำคัญน้อย ดังนี้น hook identifier แบบ 32 บิตจะมีค่าเทียบเท่ากับ hook แบบ 64 บิตของ hhh0

hook identifier ส่วนใหญ่จะถูกนิยามอยู่ในไฟล์ **/usr/include/sys/trchkid.h** ค่า 0x0100 ถึง 0x0FF0 จะพร้อมใช้งานสำหรับการใช้โดยแอ็พพลิเคชันผู้ใช้แบบ 64 บิต ค่า 0x010 ถึง 0x0FF จะพร้อมใช้งานสำหรับผู้ใช้แอ็พพลิเคชันแบบ 32 บิต ค่าอื่นๆ ทั้งหมดจะถูกสงวนไว้สำหรับให้ระบบใช้ identifiers hook การติดตามที่กำหนดไว้ในปัจจุบันจะสามารถแสดงรายการของการใช้คำสั่ง **trcrpt –j** ได้

แชนแนลการติดตามทั่วไปของตัวช่วยการติดตาม

ตัวช่วยการติดตามจะสนับสนุนเซสชันการติดตามที่แอ็คทีฟได้ทั้งหมดแปดเซสชัน ในแต่ละครั้ง แต่ละเซสชันการติดตามจะใช้ช่องสัญญาณของไฟล์พิเศษสำหรับการติดตามแบบมัลติแพล็ก **/dev/systrace** ช่องสัญญาณ 0 จะถูกนำมาใช้โดยตัวช่วยการติดตาม เพื่อบันทึกเหตุการณ์ของระบบ การติดตามเหตุการณ์ของระบบจะเริ่มต้นขึ้น และจะหยุดทำงานด้วยคำสั่ง **trace** และ **trcstop** ถ้าคุณติดตามการประมวลผลหรือเธรด หรือถ้าโปรแกรมถูกติดตาม เฉพาะช่องสัญญาณ 0 เท่านั้นที่สามารถใช้ได้ ช่องสัญญาณ 1 ถึง 7 จะอ้างถึงช่องสัญญาณการติดตามแบบทั่วไป และสามารถใช้โดยระบบย่อยสำหรับชนิดอื่นๆของการติดตาม เช่น การติดตามการเชื่อมต่อข้อมูล

หากต้องการนำการติดตามไปใช้โดยช่องสัญญาณการติดตามทั่วไป ของตัวช่วยการติดตาม ระบบย่อยจะเรียกรูทีนย่อย **trcstart** เพื่อเรียกทำงานช่องสัญญาณการติดตามและกำหนดหมายเลขช่องสัญญาณ โมดูลระบบย่อย สามารถบันทึกเหตุการณ์การติดตามได้โดยใช้แมโคร **TRCGEN** หรือ **TRCGENT** หรือถ้าจำเป็น **trcgen, trcgent, trcgenk** หรือ **trcgenkt** หมายเลขช่องสัญญาณที่ส่งคืนโดยรูทีนย่อย **trcstart** คือหนึ่งในพารามิเตอร์ ที่ต้องส่งผ่านไปยังรูทีนย่อยเหล่านี้ ระบบย่อยสามารถหยุดทำงานและกลับสู่การรวบรวมข้อมูลการติดตามได้โดยใช้รูทีนย่อย **trcoff** และ **trcon** และสามารถหยุดทำงานการติดตามช่องสัญญาณได้โดยใช้รูทีนย่อย **trcstop** ระบบย่อยต้องจัดเตรียมส่วนการติดต่อกับผู้ใช้เพื่อเรียกทำงาน และหยุดทำงานการ trace ระบบย่อย

สำหรับ ID hook การติดตาม ส่วนใหญ่จะเก็บอยู่ในไฟล์ **/usr/include/sys/trchkid.h** และเท็มเพลตการจัดรูปแบบการติดตาม ซึ่งถูกเก็บอยู่ในไฟล์ **/etc/trcfmt** จะแบ่งใช้โดยช่องสัญญาณการติดตามทั้งหมด

ข้อมูลที่เกี่ยวข้อง:

ติดตาม

trcdead

trcnm

trcrpt

trcstop

trcupdate

การเริ่มต้นตัวช่วยการติดตาม

ใช้โพรซีเดอร์ต่อไปนี้เพื่อปรับแต่ง และสตาร์ท trace ของระบบ:

การกำหนดค่าคำสั่ง trace

คำสั่ง **trace** สตาร์ทการ trace เหตุการณ์ของระบบ และควบคุมขนาดและจัดการกับไฟล์บันทึกการทำงาน **trace** เช่นเดียวกับบัฟเฟอร์ trace ภายในที่เก็บรวบรวมข้อมูลเหตุการณ์ของ trace สำหรับข้อมูลเพิ่มเติมเกี่ยวกับไวยากรณ์ของคำสั่งนี้ โปรดดู trace Daemon

การบันทึกข้อมูลเหตุการณ์การติดตาม

ข้อมูลที่บันทึกไว้สำหรับแต่ละเหตุการณ์ที่ได้รับการ trace แล้วประกอบด้วยคำที่มี trace hook identifier และชนิดของ hook ตามด้วยตัวแปรจำนวนค่ำของข้อมูล trace แล้วตามด้วยการประทับเวลาซึ่งเป็นตัวเลือก คำที่มี trace hook identifier และชนิดของ hook จะเรียกว่า hook word hook word ขนาดสองไบต์ที่เหลืออยู่จะเรียกว่า hook data และพร้อมใช้งานเพื่อบันทึกข้อมูลเหตุการณ์

ตัวบ่งชี้ hook การติดตาม

hook identifier การติดตามคือเลขฐานสิบหกที่มีสามหรือสี่หลัก ซึ่งระบุเหตุการณ์ที่กำลังติดตาม AIX 6.1 เวอร์ชันก่อนหน้าและบนแอ็พพลิเคชันแบบ 32 บิตที่รันอยู่บน AIX 6.1 และสูงกว่า จะมี hook identifier เพียงสามหลักเท่านั้นที่สามารถนำมาใช้ได้ ขณะที่ใช้แมโคร trace เช่น **TRCHKL1** trace hook จะถูกระบุด้วยรูปแบบดังนี้:

```
hhh0000
```

โดยที่ hhh คือ hook identifier

บนแอ็พพลิเคชัน 64 บิตและเคอร์เนลรูทีนที่รันบน AIX 6.1 และสูงกว่า สามารถใช้ตัวบ่งชี้ ที่มี สาม และสี่หลัก ขณะที่ใช้แมโคร trace เช่น **TRCHKL1** trace hook จะถูกระบุด้วยรูปแบบดังนี้:

```
hhhh0000
```

โดยที่ hhhh คือ hook identifier

หมายเหตุ: ถ้าใช้ identifier ที่มีสี่หลักและ identifier มีขนาดน้อยกว่า 0x1000 หลักที่มีความสำคัญน้อยต้องมีค่าเป็น 0 (ในรูปของ 0x0hh0)

identifier ที่มีสามหลักจะมี 0 อยู่ในหลักที่มีความสำคัญน้อย เช่น hook identifier แบบ 32 บิตของ hhh จะเทียบเท่ากับ hook identifier แบบ 64 บิตของ hhh0

hook identifier ส่วนใหญ่จะถูกนิยามอยู่ในไฟล์ **/usr/include/sys/trchkid.h** ค่า 0x0100 ถึง 0x0FF0 จะพร้อมใช้งานสำหรับแอ็พพลิเคชันแบบ 64 บิต ค่า 0x010 ถึง 0x0FF จะพร้อมใช้งานสำหรับผู้ใช้แอ็พพลิเคชันแบบ 32 บิต ค่าอื่นๆ ทั้งหมดจะถูกสงวนไว้สำหรับให้ระบบใช้ trace hook identifer ที่นิยามไว้ในปัจจุบันสามารถแสดงรายการได้ด้วยคำสั่ง **trcrpt –j**

การใช้แชนแนลการติดตามทั่วไป

ตัวช่วยการติดตามจะสนับสนุนเซสชันการติดตามที่แอ็คทีฟได้ทั้งหมดแปดเซสชัน ในแต่ละครั้ง แต่ละเซสชันการติดตามจะใช้ช่องสัญญาณของไฟล์พิเศษสำหรับการติดตามแบบมัลติแพล็ก **/dev/systrace** ช่องสัญญาณ 0 จะถูกนำมาใช้โดยตัวช่วยการติด

ตาม เพื่อบันทึกเหตุการณ์ของระบบ การติดตามเหตุการณ์ของระบบจะเริ่มต้นขึ้น และจะหยุดทำงานด้วยคำสั่ง **trace** และ **trcstop** ช่องสัญญาณ 1 ถึง 7 จะถูกอ้างอิงเป็น trace channel ทั่วไป และอาจถูกใช้โดยระบบย่อยสำหรับการ trace ชนิดอื่น เช่น การ trace ระบบเชื่อมโยงข้อมูล

หากต้องการนำการติดตามไปใช้โดยช่องสัญญาณการติดตามทั่วไป ของตัวช่วยการติดตาม ระบบย่อยจะเรียกรูทีนย่อย **trcstart** เพื่อเรียกทำงานช่องสัญญาณการติดตามและกำหนดหมายเลขช่องสัญญาณ โมดูลระบบย่อยสามารถบันทึกเหตุการณ์ trace สำหรับแมโคร **TRCGEN** หรือ **TRCGENT** หรือรูทีนย่อย **trcgen, trcgent, trcgenk** หรือ **trcgenkt** หมายเลขช่องสัญญาณที่ส่งคืนโดยรูทีนย่อย **trcstart** คือหนึ่งในพารามิเตอร์ ที่ต้องส่งผ่านไปยังรูทีนย่อยเหล่านี้ ระบบย่อยสามารถหยุดทำงานและกลับสู่การรวบรวมข้อมูลการติดตามได้โดยใช้รูทีนย่อย **trcoff** และ **trcon** และสามารถหยุดทำงานการติดตามช่องสัญญาณได้โดยใช้รูทีนย่อย **trcstop** เหตุการณ์ trace สำหรับแต่ละช่องสัญญาณจะถูกเขียนลงในไฟล์บันทึกการทำงาน **trace** ที่แยกออกจากกัน ซึ่งตามค่าดีฟอลต์แล้ว ไฟล์บันทึกการทำงานนั้นจะชื่อ **/var/adm/ras/trcfile.*n*** โดยที่ *n* คือหมายเลขช่องสัญญาณ ระบบย่อยต้องจัดเตรียมส่วนการติดต่อกับผู้ใช้เพื่อเรียกทำงาน และหยุดทำงานการติดตามระบบย่อย

การเริ่มต้นการติดตาม

ใช้หนึ่งในโพรซีเดอร์ต่อไปนี้เพื่อสตาร์ทตัวช่วย trace

- สตาร์ทตัวช่วย trace โดยใช้คำสั่ง **trace**

  สตาร์ท trace แบบอะซิงโครนัส ตัวอย่างเช่น:

  ```
  trace -a
  mycmd
  trcstop
  ```

  ขณะที่ใช้ตัวช่วย trace แบบอะซิงโครนัส ให้ใช้ asynchro**trace** daemon เพื่อติดตามเหตุการณ์ของระบบที่เลือกไว้ (เช่น คำสั่ง **mycmd**) จากนั้น ให้ใช้คำสั่ง **trcstop** เพื่อหยุด trace

  OR

  สตาร์ท trace แบบโต้ตอบ ตัวอย่างเช่น:

  ```
  trace
  -> !mycmd
  ->quit
  ```

  เมื่อใช้ตัวช่วยการติดตามแบบโต้ตอบ ให้เข้าสู่โหมดการโต้ตอบที่แสดงโดยพร้อมต์ -> และใช้คำสั่งย่อย **trace** (เช่น **!**) เพื่อ trace เหตุการณ์ของระบบที่เลือกไว้ ใช้คำสั่งย่อย **quit** เพื่อหยุด trace
- ใช้ **smit trace** และเลือก **Start Trace**

  ```
  smit trace
  ```

การหยุดการติดตาม

ใช้หนึ่งในโพรซีเดอร์ต่อไปนี้เพื่อหยุด trace ที่คุณเริ่มต้นไว้ก่อน

- ขณะที่ใช้ **trace** ที่บรรทัดรับคำสั่ง ให้ใช้คำสั่ง **trcstop**:

  ```
  trace -a
  mycmd
  trcstop
  ```

  ขณะที่ใช้ตัวช่วย trace แบบอะซิงโครนัส ให้ใช้ asynchro**trace** daemon เพื่อติดตามเหตุการณ์ของระบบที่เลือกไว้ (เช่น คำสั่ง **mycmd**) จากนั้น ให้ใช้คำสั่ง **trcstop** เพื่อหยุด trace

- ขณะที่ใช้ **trace** แบบโต้ตอบที่บรรทัดรับคำสั่ง ให้ใช้คำสั่งย่อย **quit** :

  ```
  trace
  -> !mycmd
  ->quit
  ```

  โหมดการโต้ตอบจะถูกแสดงโดยพร้อมต์ -> ใช้คำสั่งย่อย **trace** (เช่น **!**) เพื่อ trace เหตุการณ์ของระบบที่เลือกไว้ ใช้คำสั่งย่อย **quit** เพื่อหยุด trace
- ใช้ **smit trace** และเลือกอ็อพชัน **Stop Trace**:

  ```
  smit trace
  ```

การสร้างรายงานการติดตาม

ใช้โพรซีเดอร์อย่างใดอย่างหนึ่งต่อไปนี้ เพื่อสร้างรายงานของเหตุการณ์ที่ trace แล้ว

- ใช้คำสั่ง **trcrpt**:

  ```
  trcrpt>/tmp/NewFile
  ```

  ตัวอย่างก่อนหน้านี้จัดรูปแบบไฟล์บันทึกการทำงาน **trace** และส่งรายงานไปยัง **/tmp/newfile** คำสั่ง **trcrpt** จะอ่านไฟล์บันทึกการทำงาน **trace** จัดรูปแบบรายการ trace และเขียนรายงาน
- ใช้คำสั่ง **smit trcrpt**:

  ```
  smit trcrpt
  ```

การติดตามแอ็พพลิเคชันสำหรับผู้ใช้

หัวข้อนี้อธิบายถึงการ trace แอ็พพลิเคชันสำหรับผู้ใช้

การดำเนินการ trace ขึ้นอยู่กับกระบวนการสามแบบที่แตกต่างกัน คือ: กระบวนที่ trace แล้ว กระบวนการควบคุม และกระบวนการตัววิเคราะห์ กระบวนการสามารถเป็นกระบวนที่ trace แล้วอย่างพร้อมเพียงกัน กระบวนการควบคุม และกระบวนการตัววิเคราะห์ เมื่อกระบวนการที่ trace แล้วกำลังทำงานอยู่ และเข้าถึงจุด trace แล้ว เหตุการณ์สำหรับ trace จะถูกบันทึกลงในสตรีม trace ที่สร้างขึ้นสำหรับกระบวนการนั้นหากตัวระบุชนิดของเหตุการณ์ trace ที่เชื่อมโยงกับกระบวนการนี้ไม่ได้ถูกกรองออก

กระบวนการควบคุมจะควบคุมการบันทึกเหตุการณ์สำหรับ trace ลงในสตรีม trace กระบวนการควบคุมจะดำเนินการกับการดำเนินการต่อไปนี้สำหรับสตรีม trace ที่แอ็คทีฟ:

- กำหนดค่าเริ่มต้นแอ็ตทริบิวต์ของสตรีม trace
- สร้างสตรีม trace สำหรับกระบวนการที่ trace แล้วซึ่งระบุเฉพาะ โดยใช้แอ็ตทริบิวต์ที่กำหนดค่าเริ่มต้นแล้ว
- สตาร์ทและหยุดการ trace สำหรับสตรีม trace
- กรองชนิดของเหตุการณ์สำหรับ trace ที่ต้องการบันทึกไว้
- สิ้นสุดสตรีม trace

กระบวนการตัววิเคราะห์จะเรียกข้อมูลเหตุการณ์สำหรับ trace ณ รันไทม์ อย่างใดอย่างหนึ่ง เมื่อสตรีม trace แอ็คทีฟอยู่ และกำลังบันทึกเหตุการณ์ trace หรือ หลังจากการเปิดบันทึกการทำงาน trace ที่บันทึกและปิดไว้ก่อนหน้านี้

รูทีนย่อย **posix_trace_create, posix_trace_create_withlog** และ **posix_trace_open** จะสร้างตัวระบุสตรีม trace รูทีนย่อย **posix_trace_create** และ **posix_trace_create_withlog** จะถูกใช้โดยกระบวนการควบคุมเท่านั้น รูทีนย่อย **posix_trace_open** จะถูกใช้โดยกระบวนการวิเคราะห์เท่านั้น

กระบวนการที่ trace แล้วมีการแม็พชื่อเหตุการณ์สำหรับ trace กับตัวระบุชนิดเหตุการณ์สำหรับ trace ซึ่งได้นิยามไว้สำหรับกระบวนการ เหตุการณ์สำหรับ trace ที่แอ็คทีฟจะบันทึกชนิดเหตุการณ์สำหรับ trace ที่ได้ถูกกำหนดไว้ก่อนของระบบ เช่น POSIX_TRACE_START และชนิดเหตุการณ์สำหรับ trace ที่ได้นิยามไว้สำหรับกระบวนการที่ trace แล้วแต่ยังไม่ได้กรองออกโดยสตรีม trace หากต้องการนิยามการแม็พ ให้เรียกรูทีนย่อย **posix_trace_eventid_open** จากแอ็พพลิเคชันที่ติดตั้งเครื่องมือไว้ หรือเรียกรูทีนย่อย **posix_trace_trid_eventid_open** จากตัวควบคุมกระบวนการ สำหรับสตรีม trace ที่ถูกบันทึกไว้ก่อน รายการของชนิดเหตุการณ์สำหรับ trace จะได้รับมาจากบันทึกการทำงาน trace ที่ได้บันทึกไว้ก่อน

รูทีนย่อย tracing สามารถนำมาใช้เพื่อการดีบักโค้ดที่ติดตั้งเครื่องมือไว้ก่อน และวิเคราะห์ข้อบกพร่องที่เกิดขึ้น การดีบักโค้ดที่ติดตั้งเครื่องมือไว้ล่วงหน้า- อาจต้องการความสามารถในการกรองเพื่อหลีกเลี่ยงสตรีม trace ที่มีจำนวนมากและอนุญาตให้สนใจกับข้อมูลที่คาดการณ์ไว้ การวิเคราะห์ข้อบกพร่องจำเป็นต้องมีความสามารถในการ trace ซึ่งครอบคลุมเนื้อหา เพื่อให้สามารถบันทึกชนิดของข้อมูลทั้งหมดได้

เหตุการณ์ที่จำเป็นต้อง trace จะเป็นเหตุการณ์ของคลาสสองคลาสต่อไปนี้:

- เหตุการณ์สำหรับ trace ผู้ใช้ที่สร้างแอ็พพลิเคชันที่ได้ติดตั้งไว้
- เหตุการณ์สำหรับ trace ระบบที่ระบบปฏิบัติการสร้างขึ้น เพื่อให้สอดคล้องกับการดำเนินการควบคุม trace

ในไฟล์ที่เชื่อมโยงกับสตรีม trace ที่แอ็คทีฟ ฟิลด์ `st_ctime` และ `st_mtime` จะถูกทำเครื่องหมายสำหรับการอัพเดตทุกครั้งที่การดำเนินการ trace ใดๆ แก้ไขไฟล์นั้น

ในไฟล์ที่เชื่อมโยงกับสตรีม trace ฟิลด์ `st_atime` จะถูกทำเครื่องหมายสำหรับการอัพเดตทุกครั้งที่การดำเนินการ trace เป็นต้นเหตุทำให้ข้อมูลถูกอ่านจากไฟล์นั้น

ถ้าแอ็พพลิเคชันดำเนินการกับการดำเนินการใดๆ บนไฟล์ descriptor ที่เชื่อมโยงกับสตรีม trace ที่บันทึกไว้ล่วงหน้า ผลลัพธ์จะไม่ถูกกำหนดไว้จนกว่าจะเรียกรูทีนย่อย **posix_trace_shutdown** หรือ **posix_trace_close** สำหรับสตรีม trace นั้น

การติดตามโครงสร้างข้อมูล

ส่วนนี้อธิบายการติดตามโครงสร้างข้อมูล

ไฟล์ส่วนหัว <trace.h> จะกำหนดโครงสร้าง **posix_trace_status_info** และโครงสร้าง **posix_trace_event_info**

โครงสร้าง posix_trace_status_info

เพื่ออำนวยความสะดวกในการควบคุมสตรีมการติดตาม ให้เรียกรูทีนย่อย **posix_trace_get_status** เพื่อขอรับข้อมูลเกี่ยวกับสถานะปัจจุบันของสตรีมการติดตามที่แอ็คทีฟ

โครงสร้าง The **posix_trace_status_info** ที่กำหนดไว้ในไฟล์ <trace.h> จะมีสมาชิกต่อไปนี้:

สมาชิก	ชื่อของสมาชิก	คำอธิบาย
int	posix_stream_status	โหมดการดำเนินการของสตรีมการติดตาม
int	posix_stream_full_status	สถานะเต็มรูปแบบของสตรีมการติดตาม
int	posix_stream_overrun_status	บ่งชี้ว่า เหตุการณ์สำหรับการติดตามจะหายไปในสตรีมการติดตาม
int	posix_stream_flush_status	บ่งชี้ว่า การล้างข้อมูลกำลังดำเนินการอยู่
int	posix_stream_flush_error	บ่งชี้ว่ามีข้อผิดพลาดใดๆ เกิดขึ้น ในระหว่างการดำเนินการล้างข้อมูลล่าสุด
int	posix_log_overrun_status	บ่งชี้ว่าเหตุการณ์สำหรับการติดตามจะหายไปในบันทึกการติดตาม
int	posix_log_full_status	สถานะเต็มรูปแบบของบันทึกการทำงาน

สมาชิก **posix_stream_status** บ่งชี้ถึงโหมดการดำเนินงานของสตรีมการติดตาม ซึ่งสามารถมีค่าหนึ่งในค่าต่อไปนี้ ที่กำหนดโดยค่าคงที่ที่แสดงอยู่ในส่วนหัว <trace.h>:

POSIX_TRACE_RUNNING
การติดตามกำลังดำเนินการอยู่ สตรีมการติดตามกำลังยอมรับเหตุการณ์สำหรับการติดตาม

POSIX_TRACE_SUSPENDED
สตรีมการติดตามไม่ได้ยอมรับเหตุการณ์สำหรับการติดตาม การดำเนินการติดตามไม่ได้เริ่มต้นขึ้น หรือหยุดลง โดยปฏิบัติตามการเรียกรูทีนย่อย **posix_trace_stop** หรือ เนื่องจากรีซอร์สการติดตามถูกใช้จนหมด

สมาชิก **posix_stream_full_status** จะบ่งชี้ถึงสถานะเต็มรูปแบบของสตรีมการติดตาม ซึ่งสามารถมีค่าหนึ่งในค่าต่อไปนี้ ที่กำหนดโดยค่าคงที่ที่แสดงอยู่ในส่วนหัว <trace.h> :

POSIX_TRACE_FULL
พื้นที่ในสตรีมการติดตามสำหรับเหตุการณ์สำหรับการติดตามจะถูกใช้จนหมด

POSIX_TRACE_NOT_FULL
พื้นที่ในสตรีมการติดตามไม่เต็ม

การรวมกันของสมาชิก **posix_stream_status** และ **posix_stream_full_status** จะบ่งชี้ถึง สถานะจริงของสตรีม สถานะสามารถตีความได้ดังนี้:

POSIX_TRACE_RUNNING และ POSIX_TRACE_NOT_FULL
การติดตามกำลังดำเนินการอยู่ และพื้นที่จะพร้อมใช้งาน สำหรับการบันทึกเหตุการณ์สำหรับการติดตามเพิ่มเติม

POSIX_TRACE_RUNNING และ POSIX_TRACE_FULL
การติดตามกำลังดำเนินการอยู่ และสตรีมการติดตามของเหตุการณ์ สำหรับการติดตามเต็มแล้ว ถ้านโยบายเต็มรูปแบบของสตรีมถูกตั้งค่าเป็น POSIX_TRACE_LOOP สถานะนี้อาจเกิดขึ้นได้ สตรีมการติดตามมีเหตุการณ์สำหรับการติดตามที่บันทึกไว้ในระหว่างการย้ายหน้าต่างของเหตุการณ์สำหรับการติดตามก่อนหน้านี้ และเหตุการณ์สำหรับการติดตามก่อนหน้านี้ อาจถูกเขียนทับและหายไป

POSIX_TRACE_SUSPENDED และ POSIX_TRACE_NOT_FULL
การติดตามไม่ได้เริ่มต้นขึ้น รูทีนย่อย **posix_trace_stop** จะหยุดการติดตาม หรือรูทีนย่อย **posix_trace_clear** จะล้างข้อมูลการติดตาม

POSIX_TRACE_SUSPENDED และ POSIX_TRACE_FULL
การติดตามหยุดทำงาน ซึ่งหยุดทำงานเนื่องจากแอ็ตทริบิวต์นโยบายเต็มรูปแบบของสตรีม ตั้งค่าเป็น POSIX_TRACE_UNTIL_FULL และรีซอร์สการติดตามจะถูกใช้จนหมด หรือเนื่องจากการเรียกรูทีนย่อย **posix_trace_stop** เมื่อรีซอร์สการติดตามถูกใช้จนหมด

สมาชิก **posix_stream_overrun_status** บ่งชี้ว่า เหตุการณ์สำหรับการติดตามจะหายไปในสตรีมการติดตาม สมาชิกนี้สามารถมีค่าหนึ่งในค่าต่อไปนี้ ที่กำหนดโดยค่าคงที่ที่แสดงอยู่ในส่วนหั <trace.h> :

POSIX_TRACE_OVERRUN
เหตุการณ์สำหรับการติดตามอย่างน้อยหนึ่งเหตุการณ์จะหายไป และจะไม่บันทึกลงในสตรีมการติดตาม

POSIX_TRACE_NO_OVERRUN
ไม่มีเหตุการณ์สำหรับการติดตามที่หายไป

เมื่อสตรีมการติดตามที่สอดคล้องกันถูกสร้างขึ้น สมาชิก **posix_stream_overrun_status** จะถูกตั้งค่าเป็น POSIX_TRACE_NO_OVERRUN เมื่อการรันที่มากเกินไปเกิดขึ้น สถานะจะถูกตั้งค่าเป็น POSIX_TRACE_OVERRUN

การรันที่มีมากเกินไปเกิดขึ้นในสถานการณ์ต่อไปนี้:

- นโยบายคือ POSIX_TRACE_LOOP และเหตุการณ์สำหรับการติดตามที่บันทึกไว้ถูกเขียนทับ
- นโยบายคือ POSIX_TRACE_UNTIL_FULL และสตรีมการติดตามเต็ม เมื่อเหตุการณ์สำหรับการติดตามถูกสร้างขึ้น
- นโยบายคือ POSIX_TRACE_FLUSH และมีเหตุการณ์สำหรับการติดตามมากกว่าหนึ่งเหตุการณ์หายไป ขณะที่ล้างข้อมูลสตรีมการติดตามในบันทึกการติดตาม

สมาชิก **posix_stream_overrun_status** จะถูกรีเซ็ตให้มีค่าศูนย์ หลังจากที่อ่านค่านั้นแล้ว

สมาชิก **posix_stream_flush_status** ระบุว่า การดำเนินการล้างถูกดำเนินการและสามารถมีค่าหนึ่งในค่าต่อไปนี้ ที่กำหนดโดยค่าคงที่ที่แสดงอยู่ในไฟล์ส่วนหัว <trace.h> :

POSIX_TRACE_FLUSHING
สตรีมการติดตามกำลังถูกล้างข้อมูลในบันทึกการติดตาม

POSIX_TRACE_NOT_FLUSHING
สมาชิก **posix_stream_flush_status** จะถูกตั้งค่าเป็น **POSIX_TRACE_NOT_FLUSHING** ถ้าไม่มีการดำเนินการล้างข้อมูลกำลังดำเนินการอยู่

สมาชิก **posix_stream_flush_status** จะถูกตั้งค่าเป็น **POSIX_TRACE_FLUSHING** ในสถานการณ์ต่อไปนี้:

- การดำเนินการล้างข้อมูกำลังดำเนินการอยู่ เนื่องจากการเรียกรูทีนย่อย **posix_trace_flush**
- การดำเนินการล้างข้อมูลกำลังดำเนินการอยู่ เนื่องจากสตรีมการติดตามเต็มไปด้วยแอ็ตทริบิวต์นโยบายเต็มรูปแบบของสตรีม – – ตั้งค่าเป็น POSIX_TRACE_FLUSH

สมาชิก **posix_stream_flush_error** จะมีค่าศูนย์หากไม่มีข้อผิดพลาดเกิดขึ้นในการล้างข้อมูล ถ้ามีข้อผิดพลาดเกิดขึ้นในการดำเนินการล้างข้อมูล สมาชิก **posix_stream_flush_error** จะมีค่าเป็นค่าของข้อผิดพลาดแรกที่เกิดขึ้น ถ้ามีข้อผิดพลาดมากกว่าหนึ่งข้อเกิดขึ้นในการดำเนินการล้างข้อมูล ค่าของข้อผิดพลาดแรกจะถูกนำมาใช้ และค่าอื่นจะถูกละเว้น สมาชิก **posix_stream_flush_error** จะถูกรีเซ็ตให้มีค่าศูนย์ หลังจากที่อ่านค่าแล้ว

สมาชิก **posix_log_overrun_status** บ่งชี้ว่า เหตุการณ์สำหรับการติดตามจะหายไปในบันทึกการติดตาม สมาชิกนี้สามารถมีค่าหนึ่งในค่าต่อไปนี้ ที่กำหนดโดยค่าคงที่ที่แสดงอยู่ในการกำหนดส่วนหัว <trace.h> :

POSIX_TRACE_OVERRUN
เหตุการณ์สำหรับการติดตามอย่างน้อยที่สุดหนึ่งเหตุการณ์หายไป

POSIX_TRACE_NO_OVERRUN
ไม่มีเหตุการณ์สำหรับการติดตามใดๆ ที่หายไป

เมื่อการรันที่มากเกินไปเกิดขึ้น สมาชิก **posix_log_overrun_status** จะถูกตั้งค่าเป็น POSIX_TRACE_OVERRUN เมื่อสตรีมการติดตามที่สอดคล้องกันถูกสร้างขึ้น สมาชิก **posix_log_overrun_status** จะถูกตั้งค่าเป็น
POSIX_TRACE_NO_OVERRUN

สมาชิก **posix_log_overrun_status** จะถูกรีเซ็ตให้มีค่าศูนย์หลังจากที่อ่านค่าแล้ว

ถ้าสตรีมการติดตามที่แอ็คทีฟถูกสร้างขึ้นโดยรูทีนย่อย **posix_trace_create** และไม่มีบันทึกการทำงาน สมาชิก **posix_log_overrun_status** จะถูกตั้งค่าเป็น POSIX_TRACE_NO_OVERRUN

สมาชิก **posix_log_full_status** ระบุสถานะแบบเต็ม ของบันทึกการติดตามและสามารถมีค่าหนึ่งในค่าต่อไปนี้ ที่กำหนดโดยค่าคงที่ที่แสดงอยู่ในส่วนหัว <trace.h> :

POSIX_TRACE_FULL
พื้นที่ในบันทึกการติดตามเต็ม

POSIX_TRACE_NOT_FULL
พื้นที่ในบันทึกการติดตามไม่เต็ม

สมาชิก **posix_log_full_status** จะมีความหมายเท่านั้น เมื่อแอ็ตทริบิวต์นโยบายเต็มรูปแบบของบันทึกการทำงานคือ POSIX_TRACE_UNTIL_FULL หรือ POSIX_TRACE_LOOP อย่างใดอย่างหนึ่ง

ถ้าสตรีมการติดตามที่แอ็คทีฟถูกสร้างขึ้นโดยรูทีนย่อย **posix_trace_create** และไม่มีบันทึกการทำงาน สมาชิก **posix_log_full_status** จะถูกตั้งค่าเป็น POSIX_TRACE_NOT_FULL

โครงสร้าง posix_trace_event_info

โครงสร้างเหตุการณ์สำหรับการติดตาม **posix_trace_event_info** จะมีข้อมูลสำหรับเหตุการณ์สำหรับการติดตามที่บันทึกไว้ รูทีนย่อย **posix_trace_getnext_event, posix_trace_timedgetnext_event** และ **posix_trace_trygetnext_event** ส่งคืนรูทีนย่อยนี้

โครงสร้าง **posix_trace_event_info** ที่กำหนดไว้ใน ไฟล์ส่วนหัว <trace.h> ประกอบด้วยสมาชิกต่อไปนี้:

สมาชิก	ชื่อสมาชิก	คำอธิบาย
trace_event_id_t	posix_event_id	identification ชนิดของเหตุการณ์สำหรับการติดตาม
pid_t	posix_pid	ID การประมวลผลของการประมวลผลที่สร้างเหตุการณ์ สำหรับการติดตาม
void*	posix_prog_address	แอดเดรสที่เรียกใช้งานจุดสำหรับการติดตาม

สมาชิก	ชื่อสมาชิก	คำอธิบาย
int	posix_truncation_status	สถานะการตัดข้อมูลที่เชื่อมโยงกับเหตุการณ์สำหรับการติดตามนี้
struct timespec	posix_timestamp	เวลาที่เหตุการณ์สำหรับการติดตามถูกสร้างขึ้น
pthread_t	posix_thread_id	ID ของเธรดที่สร้างเหตุการณ์สำหรับการติดตาม

สมาชิก **posix_event_id** จะแสดง identification ของชนิดของเหตุการณ์สำหรับการติดตาม คุณไม่สามารถกำหนดค่าสมาชิกได้โดยตรง หากต้องการส่งคืน identification ให้รันหนึ่งในรูทีนย่อยต่อไปนี้:

- posix_trace_trid_eventid_open
- posix_trace_eventtypelist_getnext_id
- posix_trace_eventid_open

หากต้องการขอรับชื่อของชนิดของเหตุการณ์สำหรับการติดตาม ให้รันรูทีนย่อย **posix_trace_eventid_get_name**

posix_pid คือ identifier การประมวลผลของกระบวนการที่ติดตามแล้ว ซึ่งสร้างเหตุการณ์สำหรับการติดตาม ถ้าสมาชิก **posix_event_id** คือหนึ่งในเหตุการณ์สำหรับการติดตามระบบ และเหตุการณ์สำหรับการติดตามไม่ได้เชื่อมโยงกับกระบวนการใดๆ สมาชิก **posix_pid** จะถูกตั้งค่าเป็นศูนย์

ในเหตุการณ์สำหรับการติดตามผู้ใช้ สมาชิก **posix_prog_address** จะถูกแม็พกับแอดเดรสของกระบวนการ และจุดที่เชื่อมโยงกับการเรียกรูทีนย่อย **posix_trace_event** ที่ถูกสร้างขึ้น

สมาชิก **posix_truncation_status** จะนิยามสถานะการตัดของข้อมูลที่เชื่อมโยงกับเหตุการณ์สำหรับการติดตาม สมาชิก **posix_truncation_status** สามารถมีค่าหนึ่งในค่าต่อไปนี้ที่กำหนดโดยค่าคงที่ที่แสดงอยู่ในส่วนหัว <trace.h> :

POSIX_TRACE_NOT_TRUNCATED
ข้อมูลที่ติดตามทั้งหมดที่มีอยู่

POSIX_TRACE_TRUNCATED_RECORD
ข้อมูลที่ถูกตัดเมื่อเหตุการณ์สำหรับการติดตามถูกสร้างขึ้น

POSIX_TRACE_TRUNCATED_READ
ข้อมูลที่ถูกตัดเมื่อเหตุการณ์สำหรับการติดตามถูกอ่านจากสตรีมการติดตาม หรือบันทึกการติดตาม สถานะของการตัดนี้จะแทนทับสถานะ POSIX_TRACE_TRUNCATED_RECORD

สมาชิก **posix_timestamp** จะกำหนดเวลาเมื่อเหตุการณ์สำหรับการติดตามถูกสร้างขึ้น นาฬิกาที่ใช้คือ CLOCK_REALTIME หากต้องการเรียกคืนข้อมูลการแก้ปัญหาของนาฬิกานี้ ให้เรียกรูทีนย่อย **posix_trace_attr_getclockres**

สมาชิก **posix_thread_id** คือ identifier ของเธรดที่สร้างเหตุการณ์สำหรับการติดตาม ถ้าสมาชิก **posix_event_id** คือหนึ่งในเหตุการณ์สำหรับการติดตามระบบ และเหตุการณ์สำหรับการติดตามไม่ได้เชื่อมโยงกับเธรดใดๆ สมาชิก **posix_thread_id** จะถูกตั้งค่าให้เป็นศูนย์

แอ็ตทริบิวต์ Trace stream

แอ็ตทริบิวต์ trace stream ต่อไปนี้ประกอบด้วยอ็อบเจ็กต์แอ็ตทริบิวต์ trace stream **posix_trace_attr_t** :

- แอ็ตทริบิวต์ *genversion* จะระบุจุดเริ่มต้น และเวอร์ชันของระบบ trace
- แอ็ตทริบิวต์ *tracename* คือสตริงอักขระที่กำหนดโดยตัวควบคุม trace ซึ่งจะระบุ trace stream ไว้
- แอ็ตทริบิวต์ *creation-time* จะแสดงถึงเวลาที่สร้าง trace stream
- แอ็ตทริบิวต์ *clock-resolution* จะกำหนดการแก้ไขนาฬิกา ซึ่งใช้เพื่อสร้างการประทับเวลา
- แอ็ตทริบิวต์ *stream-min-size* จะกำหนดขนาดต่ำสุดในหน่วยไบต์ของ trace stream ขนาดจะถูกสงวนไว้สำหรับเหตุการณ์ trace อย่างเคร่งครัด
- แอ็ตทริบิวต์ *stream-full-policy* จะกำหนดนโยบายที่ต้องปฏิบัติตาม เมื่อ trace stream เต็ม ค่าของแอ็ตทริบิวต์นี้คือ POSIX_TRACE_LOOP, POSIX_TRACE_UNTIL_FULL หรือ POSIX_TRACE_FLUSH
- แอ็ตทริบิวต์ *max-data-size* จะกำหนดขนาดสูงสุดในหน่วยไบต์ของเร็กคอร์ดสำหรับเหตุการณ์ trace
- แอ็ตทริบิวต์ *inheritance* จะระบุระบบ trace ที่สร้างขึ้นใหม่ ซึ่งจะสืบทอดการ trace ใน trace stream กระบวนการหลัก ' ค่าของแอ็ตทริบิวต์นี้คือ POSIX_TRACE_INHERITED หรือ POSIX_TRACE_CLOSE_FOR_CHILD อย่างใดอย่างหนึ่ง
- แอ็ตทริบิวต์ *log-max-size* จะกำหนดขนาดสูงสุดในหน่วยไบต์ของบันทึกการทำงาน trace ที่เชื่อมโยงกับ trace stream ที่แอ็คทีฟ ข้อมูลสตรีมอื่นๆ จะไม่สอดแทรกอยู่ในขนาดนี้
- แอ็ตทริบิวต์ *log-full-policy* จะกำหนดนโยบายของบันทึกการทำงาน trace ที่เชื่อมโยงกับ trace stream ที่แอ็คทีฟซึ่งคือ POSIX_TRACE_LOOP, POSIX_TRACE_UNTIL_FULL หรือ POSIX_TRACE_APPEND

นิยามชนิดเหตุการณ์การติดตาม

ไฟล์ส่วนหัว <trace.h> จะกำหนดชนิดเหตุการณ์การติดตามระบบ และชนิดเหตุการณ์การติดตามผู้ใช้ หัวข้อนี้อธิบายถึงชนิดเหตุการณ์ trace และนิยาม

นิยามชนิดเหตุการณ์การติดตามระบบ

อธิบายถึงสตรีม trace หรือบันทึกการทำงาน trace การประมวลผลตัววิเคราะห์จำเป็นต้องมีข้อมูลเกี่ยวกับเหตุการณ์ trace และข้อมูลเกี่ยวกับเหตุการณ์ trace ของระบบที่รายงานการเรียกใช้งานการดำเนินการ trace

ชนิดเหตุการณ์ trace ของระบบต่อไปนี้จะติดตามการเรียกใช้งานของการดำเนินการ trace:

POSIX_TRACE_START
ติดตามการเรียกใช้งานของการดำเนินการสตาร์ท trace

POSIX_TRACE_STOP
ติดตามการเรียกใช้งานของการดำเนินการหยุด trace

POSIX_TRACE_FILTER
ติดตามการเรียกใช้งานของการดำเนินการเปลี่ยนชนิดเหตุการณ์ trace

ชนิดเหตุการณ์ trace ของระบบต่อไปนี้จะรายงานเหตุการณ์ trace ที่บันทึกไว้โดยรูทีนย่อย Posix Trace Library:

POSIX_TRACE_OVERFLOW
โอเวอร์โฟลว์ของ trace ที่เริ่มต้น

POSIX_TRACE_RESUME
โอเวอร์โฟลว์ของ trace ที่สิ้นสุดลง

POSIX_TRACE_FLUSH_START
การดำเนินการล้างข้อมูลที่เริ่มต้น

POSIX_TRACE_FLUSH_STOP
การดำเนินการล้างข้อมูลที่สิ้นสุดลง

POSIX_TRACE_ERROR
ข้อผิดพลาดของ trace ที่เกิดขึ้น

เหตุการณ์ trace สำหรับ POSIX_TRACE_START และ POSIX_TRACE_STOP จะระบุเวลาที่ใช้ไปซึ่งสตรีม trace กำลังทำงานอยู่

เหตุการณ์ trace สำหรับ POSIX_TRACE_STOP ที่มีค่าศูนย์ บ่งชี้ถึงการเรียกรูทีนย่อย **posix_trace_stop**

เหตุการณ์ trace สำหรับ POSIX_TRACE_STOP ที่มีค่าศูนย์ บ่งชี้ถึงการหยุดสตรีม trance แบบอัตโนมัติ สำหรับข้อมูลเพิ่มเติม โปรดดูรูทีนย่อย **posix_trace_attr_getstreamfullpolicy**

เหตุการณ์ trace สำหรับ POSIX_TRACE_FILTER บ่งชี้ถึงค่าตัวกรองของชนิดเหตุการณ์ trace ที่เปลี่ยนไป เมื่อสตรีม trace กำลังทำงาน

POSIX_TRACE_ERROR บ่งชี้ว่า มีข้อผิดพลาดภายในของระบบ trace เกิดขึ้น

เหตุการณ์ trace สำหรับ POSIX_TRACE_OVERFLOW ถูกรายงานด้วยการประทับเวลา ซึ่งเท่ากับการประทับเวลาของเหตุการณ์ trace ในครั้งแรกที่เขียนทับ เหตุการณ์ trace นี้บ่งชี้ว่า เหตุการณ์ trace บางตัวที่สร้างไว้หายไป

เหตุการณ์ trace สำหรับ POSIX_TRACE_RESUME บ่งชี้ว่า ระบบ trace กำลังบันทึกเหตุการณ์ trace หลังจากเกิดโอเวอร์โฟลว์

ค่าคงที่พร้อมกับชื่อของเหตุการณ์ trace และค่าคงที่ของ **trace_event_id_t** จะนิยามชนิดเหตุการณ์ trace ข้อมูลเหตุการณ์ trace ถูกเชื่อมโยงกับเหตุการณ์ trace เหล่านี้บางตัว

ตาราง เหตุการณ์ trance ของระบบอธิบายถึงเหตุการณ์ trace ของระบบที่ได้ถูกกำหนดไว้ก่อน

ตารางที่ 80. เหตุการณ์ trace ของระบบ

ชื่อเหตุการณ์	ค่าคงที่	ข้อมูลที่เชื่อมโยง	ชนิดข้อมูล
posix_trace_error	POSIX_TRACE_ERROR	error	**int**
posix_trace_start	POSIX_TRACE_START	event_filter	**trace_event_set_t**
posix_trace_stop	POSIX_TRACE_STOP	Auto	**int**
posix_trace_filter	POSIX_TRACE_FILTER	old_event_filter, new_event_filter	**trace_event_set_t**
posix_trace_overflow	POSIX_TRACE_OVERFLOW	*ไม่มี*	*none*
posix_trace_resume	POSIX_TRACE_RESUME	*none*	*none*

ตารางที่ 80. เหตุการณ์ trace ของระบบ (ต่อ)

ชื่อเหตุการณ์	ค่าคงที่	ข้อมูลที่เชื่อมโยง	ชนิดข้อมูล
posix_trace_flush_start	POSIX_TRACE_FLUSH_START	*none*	*none*
posix_trace_flush_stop	POSIX_TRACE_FLUSH_STOP	*none*	*none*

นิยามชนิดเหตุการณ์การติดตามผู้ใช้

ไฟล์ส่วนหัว <trace.h> จะกำหนดเหตุการณ์การติดตามผู้ใช้ POSIX_TRACE_UNNAMED_USEREVENT ถ้าเข้าใกล้ข้อจำกัดของ TRACE_USER_EVENT_MAX แล้ว เหตุการณ์ของผู้ใช้สำหรับ POSIX_TRACE_UNNAMED_USEREVENT จะส่งคืนค่า เมื่อแอ็พพลิเคชันลองลงทะเบียนเหตุการณ์เพิ่มเติม ที่มากกว่าจำนวนที่อนุญาตไว้ ไม่มีข้อมูลที่เชื่อมโยงกับเหตุการณ์ trace ของผู้ใช้

ค่าคงที่ POSIX_TRACE_UNNAMED_USEREVENT จะถูกกำหนดไว้สำหรับชื่อเหตุการณ์ trace
posix_trace_unnamed_userevent

รูทีนย่อย Trace

ส่วนนี้อธิบายถึงรูทีนย่อย trace และการ trace อื่นๆ ที่สนับสนุน

อินเตอร์เฟส trace จะถูกสร้างและจัดโครงสร้าง เพื่อปรับปรุงความสามารถในการพอร์ตโดยใช้ข้อมูล trace ชนิด opaque

หากต้องการตั้งค่าและปรับแต่งรีซอร์สสำหรับกระบวนการของตัวควบคุมการ trace เพื่อรันสตรีม trace ให้ใช้รูทีนย่อยในตารางต่อไปนี้

รูทีนย่อย	วัตถุประสงค์
posix_trace_attr_init	กำหนดค่าเริ่มต้นให้กับแอ็ตทริบิวต์
posix_trace_attr_destroy	ล้างค่าแอ็ตทริบิวต์
posix_trace_attr_getgenversion	ขอรับเวอร์ชันของสตรีม trace
posix_trace_attr_getname	ขอรับชื่อ trace
posix_trace_attr_setname	ตั้งค่าชื่อ trace
posix_trace_attr_getinherited	ขอรับนโยบายการสืบถอดของสตรีม trace
posix_trace_attr_setinherited	ตั้งค่านโยบายการสืบทอดของสตรีม trace
posix_trace_attr_setstreamfullpolicy	ตั้งค่านโยบายทั้งหมดของสตรีม
posix_trace_attr_getstreamfullpolicy	ขอรับนโยบายทั้งหมดของสตรีม
posix_trace_attr_setlogfullpolicy	ตั้งค่าบันทึกการทำงานของนโยบายทั้งหมดของสตรีม trace
posix_trace_attr_getlogfullpolicy	ขอรับบันทึกการทำงานของนโยบายทั้งหมดของสตรีม trace
posix_trace_attr_setlogsize	ตั้งค่าขนาดของบันทึกการทำงานของสตรีม trace
posix_trace_attr_setmaxdatasize	ตั้งค่าขนาดสูงสุดของข้อมูลเหตุการณ์สำหรับ trace ผู้ใช้
posix_trace_attr_setstreamsize	ตั้งค่าขนาดของสตรีม trace

รูทีนย่อย	วัตถุประสงค์
posix_trace_attr_getmaxusereventsize	ขอรับค่าความยาวสูงสุดของเหตุการณ์สำหรับผู้ใช้ ที่กำหนดไว้
posix_trace_create	สร้างสตรีม trace ที่แอ็คทีฟ
posix_trace_create_withlog	สร้างสตรีม trace ที่แอ็คทีฟ และเชื่อมโยงกับบันทึกการทำงาน trace
posix_trace_flush	คัดลอกเนื้อหาของสตรีม trace ลงในบันทึกการทำงาน trace ที่เชื่อมโยงของสตรีม trace
posix_trace_shutdown	ปิดระบบสตรีม trace
posix_trace_clear	ล้างข้อมูลสตรีม trace และบันทึกการทำงาน trace
posix_trace_trid_eventid_open	เชื่อมโยงตัวระบุชนิดเหตุการณ์สำหรับ trace กับชื่อเหตุการณ์สำหรับ trace ผู้ใช้
posix_trace_eventid_equal	เปรียบเทียบตัวระบุชนิดเหตุการณ์ trace สองตัว
posix_trace_eventid_get_name	ดึงชื่อเหตุการณ์สำหรับ trace จากตัวระบุชนิดเหตุการณ์สำหรับ trace
posix_trace_eventtypelist_getnext_id, posix_trace_eventtypelist_rewind	วนซ้ำการแม็พของชนิดเหตุการณ์สำหรับ trace
posix_trace_eventset_add	เพิ่มชนิดเหตุการณ์สำหรับ trace ในชนิดเหตุการณ์สำหรับ trace set
posix_trace_eventset_empty	ทำให้ชุดของชนิดเหตุการณ์สำหรับ trace ว่าง
posix_trace_eventset_del	ลบชนิดเหตุการณ์สำหรับ trace ออจากชุดของชนิดเหตุการณ์สำหรับ trace
posix_trace_eventset_fill	กรอกข้อมูลในชุดของชนิดเหตุการณ์สำหรับ trace
posix_trace_eventset_ismember	ทดสอบว่า ชนิดเหตุการณ์สำหรับ trace สอดแทรกอยู่ในชุดของชนิดเหตุการณ์สำหรับ trace
posix_trace_get_filter	ขอรับตัวกรองของสตรีม trace ที่กำหนดค่าเริ่มต้นแล้ว
posix_trace_set_filter	ตั้งค่าตัวกรองของสตรีม trace ที่กำหนดค่าเริ่มต้นแล้ว
posix_trace_start	เริ่มต้นสตรีม trace
posix_trace_stop	หยุดทำงานสตรีม trace
posix_trace_get_attr	อ่านข้อมูลสตรีม trace
posix_trace_get_status	ขอรับแอ็ตทริบิวต์ trace หรือสถานะ trace

หากต้องการตั้งค่าจุด trace สำหรับกระบวนการที่ trace แล้ว ให้ใช้รูทีนย่อย **posix_trace_event** และ **posix_trace_eventid_open** รูทีนย่อยเหล่านี้จะนิยามตัวระบุชนิดเหตุการณ์สำหรับ trace และแทรกจุด trace ไว้

ตารางต่อไปนี้แสดงรูทีนย่อยที่ดึงข้อมูลจากสตรีม trace และบันทึกการทำงาน trace สำหรับกระบวนการตัววิเคราะห์ trace

รูทีนย่อย	วัตถุประสงค์
posix_trace_attr_getname	ขอรับชื่อ trace
posix_trace_attr_getgenversion	อ่านข้อมูล identification
posix_trace_attr_getcreatetime	ขอรับเวลาที่สร้างสตรีม trace
posix_trace_attr_getinherited	ขอรับนโยบายการสืบถอดของสตรีม trace
posix_trace_attr_getstreamfullpolicy	ขอรับนโยบายทั้งหมดของสตรีม
posix_trace_attr_getlogfullpolicy	ขอรับบันทึกการทำงานของนโยบายทั้งหมดของสตรีม trace
posix_trace_attr_getmaxusereventsize	ขอรับค่าความยาวสูงสุดของเหตุการณ์สำหรับผู้ใช้ที่กำหนดไว้
posix_trace_attr_getmaxsystemeventsize	ขอรับขนาดสูงสุดของเหตุการณ์สำหรับ trace ของระบบ
posix_trace_attr_getlogsize	ขอรับขนาดของบันทึกการทำงานของสตรีม trace
posix_trace_attr_getmaxdatasize	ขอรับขนาดสูงสุดของข้อมูลเหตุการณ์สำหรับ trace ผู้ใช้
posix_trace_attr_getstreamsize	ขอรับขนาดของสตรีม trace
posix_trace_trid_eventid_open	เชื่อมโยงตัวระบุชนิดเหตุการณ์สำหรับ trace กับชื่อเหตุการณ์สำหรับ trace ผู้ใช้
posix_trace_eventid_equal	เปรียบเทียบตัวระบุชนิดเหตุการณ์ trace สองตัว
posix_trace_eventid_get_name	ขอรับชื่อเหตุการณ์สำหรับ trace จากตัวระบุชนิดเหตุการณ์สำหรับ trace
posix_trace_eventtypelist_getnext_id, posix_trace_eventtypelist_rewind	วนซ้ำการแม็พของชนิดเหตุการณ์สำหรับ trace
posix_trace_open	เปิดบันทึกการทำงาน trace
posix_trace_close	ปิดบันทึกการทำงาน trace
posix_trace_rewind	กำหนดค่าเริ่มต้นให้กับบันทึกการทำงาน trace สำหรับการอ่านอีกครั้ง
posix_trace_get_attr	อ่านข้อมูลของสตรีม trace
posix_trace_get_status	อ่านสถานะของสตรีม trace
posix_trace_getnext_event	อ่านเหตุการณ์สำหรับ trace
posix_trace_timedgetnext_event, posix_trace_trygetnext_event	ขอรับเหตุการณ์สำหรับ trace

ระบบย่อย tty

AIX คือระบบปฏิบัติการแบบผู้ใช้จำนวนมาก ซึ่งอนุญาตให้ผู้ใช้เข้าถึงได้จากอุปกรณ์ที่ต่อพ่วงแบบโลคัล หรือแบบรีโมต เลเยอร์การสื่อสารที่สนับสนุนฟังก์ชันนี้คือ ระบบย่อย tty

การสื่อสารระหว่างอุปกรณ์เทอร์มินัล และโปรแกรมที่อ่านและเขียนลงในอุปกรณ์เหล่านี้ที่ถูกควบคุมโดยอินเตอร์เฟส tty ตัวอย่างของอุปกรณ์ tty คือ:

- โมเด็ม
- เทอร์มินัล ASCII
- คอนโซลระบบ

- เครื่องพิมพ์แบบอนุกรม
- Xterms หรือ aixterms ภายใต้ X-Windows

ภาพรวมนี้จะแสดงข้อมูลเกี่ยวกับหัวข้อต่อไปนี้:

วัตถุประสงค์ของระบบย่อย TTY

ระบบย่อย tty จะรับผิดชอบ:
- การควบคุมการไหลของข้อมูลแบบฟิสิคัลบนสายสื่อสารแบบอะซิงโครนัส (ซึ่งรวมถึง ความเร็วในการส่งข้อมูล ขนาดตัวอักษร และสภาพพร้อมใช้งานของสายสื่อสาร)
- การตีความข้อมูลโดยการจดจำอักขระพิเศษ และปรับให้เป็นภาษาประจำชาติ
- การควบคุมงานและเทอร์มินัลที่เข้าถึงโดยใช้แนวคิดของการควบคุมเทอร์มินัล

การควบคุมข้อความเทอร์มินัล จะจัดการกับการดำเนินการอินพุตและเอาต์พุตของกลุ่มของการประมวลผล ไฟล์ชนิดพิเศษ **tty** จะสนับสนุนการควบคุมอินเตอร์เฟสของเทอร์มินัล ในทางปฏิบัติ ผู้ใช้โปรแกรมแทบจะไม่เปิดไฟล์เทอร์มินัล เช่น **dev/tty5** ไฟล์เหล่านี้จะถูกเปิดด้วยคำสั่ง **getty** or **rlogind** และกลายเป็นอุปกรณ์อินพุตและเอาต์พุตมาตรฐานของผู้ใช้

โปรดดู **ไฟล์ชนิดพิเศษ tty** in *Files Reference* สำหรับข้อมูลเพิ่มเติมเกี่ยวกับการควบคุมเทอร์มินัล

โมดูลระบบย่อย tty

หากต้องการดำเนินการกับภารกิจ ระบบย่อย tty จะประกอบด้วยโมดูลหรือระเบียบ โมดูลคือ ชุดของกฎของการประมวลผลที่ควบคุมอินเตอร์เฟสสำหรับการสื่อสารระหว่างคอมพิวพเตอร์ และอุปกรณ์แบบอะซิงโครนัส โมดูลอาจถูกเพิ่มหรือลบออกแบบไดนามิกสำหรับ tty แต่ละตัว

ระบบย่อย tty จะสนับสนุนชนิดของโมดูลหลักสามชนิด:

ไดรเวอร์ tty

ไดรเวอร์ tty หรือระเบียบฮาร์ดแวร์ จะควบคุมฮาร์ดแวร์โดยตรง (อุปกรร์ tty) หรือการเลียนแบบฮาร์ดแวร์ (อุปกรณ์ pty) ซึ่งจะดำเนินการอินพุตและเอาต์พุต ไปยังอะแด็ปเตอร์จริง โดยจัดเตรียมเซอร์วิสให้กับโมดูลที่อยู่สูงกว่า เซอร์วิสคือการควบคุมสายงานและซีแมนทิกส์พิเศษ เมื่อพอร์ตถูกเปิดขึ้น

ไดรเวอร์ tty ต่อไปนี้จะถูกจัดเตรียมไว้:

ไดรเวอร์	คำอธิบาย
cxma	ตัวควบคุม PCI แบบอะซิงโครนัส 128 พอร์ต
cxpa	ตัวควบคุม PCI แบบอะซิงโครนัส 8 พอร์ต
lft	เทอร์มินัลที่มีการทำงานต่ำ ชื่อ tty คือ **/dev/lft***Y* โดยที่ *Y* >= 0
pty	อุปกรณ์ไดรเวอร์ที่เลียนแบบเทอร์มินัล
sa	อะแด็ปเตอร์ PCI EIA-232 แบบอะซิงโครนัส 2 พอร์ต
sf	Universal asynchronous receiver/transceivers (UARTs) บน planar ของระบบ

ส่วนของ “ไดรเวอร์ TTY” ในหน้า 901 จะจัดเตรียมข้อมูลเพิ่มเติม

Line disciplines

ระเบียบของสายสื่อสารจะจัดเตรียมการแก้ไข การควบคุมงาน และการตีความหมายของอักขระพิเศษ ซึ่งจะดำเนินการแปลงสภาพที่เกิดขึ้นบน data stream ขาเข้าและขาออก ระเบียบของสายสื่อสารยังดำเนินการกับการจัดการข้อผิดพลาด และสถานะการมอนิเตอร์สำหรับไดรเวอร์ tty

ระเบียบสายสื่อสารต่อไปนี้จะถูกจัดเตรียมไว้:

รูทีนย่อย	**คำอธิบาย**
ldterm	อุปกรณ์เทอร์มินัล
sptr	เครื่องพิมพ์แบบอนุกรม (คำสั่ง **splp**)
slip	Serial Line Internet Protocol (คำสั่ง **slattach**)

โมดูลตัวแปลง

โมดูลตัวแปลง หรือ *ระเบียบการแม็พ* แปล หรือแม็พ อักขระอินพุตและเอาต์พุต

โมดูลตัวแปลงต่อไปนี้จะถูกจัดเตรียมไว้:

ตัวแปลง	**คำอธิบาย**
nls	สนับสนุนภาษาประจำชาติสำหรับการแม็พเทอร์มินัล ตัวแปลงนี้จะแปลอักขระขาเข้า และขาออกบน data stream โดยอ้างอิงตามอินพุตและเอาต์พุตแม็พ ที่กำหนดไว้สำหรับพอร์ต (โปรดดูคำสั่ง **setmaps**)
lc_sjis และ **uc_sjis**	ตัวแปลงให้สูงขึ้นและลดลงจะใช้เพื่อแปลอักขระแบบมัลติไบต์ ระหว่าง Shifted Japanese Industrial Standard (SJIS) และ Advanced Japanese EUC Code (AJEC) จะถูกจัดการโดยระเบียบสายสื่อสาร **ldterm**

“โมดูลตัวแปลง” ในหน้า 900 จัดเตรียมข้อมูลเพิ่มเติมเกี่ยวกับตัวแปลง

โครงสร้างระบบย่อย TTY

ระบบย่อย tty จะอ้างอิงตาม STREAMS ซึ่งโครงสร้างแบบอิง STREAMS จะจัดเตรียม modularity และความยืดหยุ่น และเปิดใช้งานคุณลักษณะต่อไปนี้:

- การปรับแต่งแบบง่ายๆ ผู้ใช้สามารถปรับแต่งสภาวะแวดล้อมของระบบย่อยเทอร์มินัล โดยเพิ่มหรือลบโมดูลของตัวเลือก
- โมดูลการนำกลับมาใช้ใหม่ ตัวอย่างเช่น โมดูลระเบียบของสายสื่อสารที่เหมือนกันสามารถนำมาใช้บนอุปกรณ์ tty จำนวนมาด้วยคอนฟิกูเรชันที่ต่างกัน
- การเพิ่มแบบง่ายๆ ของคุณลักษณะใหม่ในระบบย่อยเทอร์มินัล
- จัดเตรียมอินเตอร์เฟสที่เหมือนกับบนอุปกรณ์ต่างกัน

โครงสร้างของสตรีม tty จะสร้างขึ้นจากโมดูลต่อไปนี้ :

- ส่วนหัวของสตรีม ประมวลผลคำร้องของของผู้ใช้ ส่วนหัวของสตรีม จะเหมือนกับอุปกรณ์ tty ทั้งหมด โดยไม่พิจารณาถึงระเบียบของสายสื่อสาร หรือไดรเวอร์ tty ที่ใช้งานอยู่
- ตัวแปลงให้สูงขึ้นซึ่งเป็นตัวเลือก (ตัวอย่างเช่น **uc_sjis**) โมดูลตัวแปลงจะส่งระเบียบของสายสื่อสารข้างต้นเพื่อแปลงข้อมูลสตรีมขึ้น และสตรีมลง
- ระเบียบของสายสื่อสาร
- ตัวแปลงให้ต่ำกว่าซึ่งเป็นตัวเลือก (ตัวอย่างเช่น **lc_sjis**) โมดูลตัวแปลงจะส่งระเบียบของสายสื่อสารด้านล่างเพื่อแปลงข้อมูลสตรีมขึ้น และสตรีมลง

- โมดูลการแม็พอักขระที่เป็นตัวเลือก (**nls**) ตัวแปลงโมดูลที่ส่งไดรเวอร์ tty เพื่อสนับสนุนการแม็พอินพุตและเอาต์พุตเทอร์มินัล
- สตรีมสิ้นสุด: ไดรเวอร์ tty

เว้นเสียแต่จำเป็นต้องมี โมดูลนานาชาติที่ไม่ได้แสดงอยู่ในสตรีม tty

สำหรับเครื่องพิมพ์แบบอนุกรม โมดูลนานาชาติ จะไม่ได้แสดงอยู่บนสตรีม ดังนั้นโครงสร้างจึงเป็นแบบง่ายๆ

เซอร์วิสทั่วไป

ไฟล์ **/usr/include/sys/ioctl.h** และไฟล์ **/usr/include/termios.h** อธิบายถึงอินเตอรีเฟสไปยังเซอร์วิสทั่วไป ที่แสดงโดยระบบย่อย tty ไฟล์ **ioctl.h** ซึ่งถูกใช้โดยโมดูลทั้งหมดประกอบด้วยโครงสร้าง **winsize** พร้อมกับคำสั่ง ioctl ไฟล์ **termios.h** รวมถึงรูทีนย่อย POSIX ที่เข้ากันได้ และชนิดข้อมูล

เซอร์วิสที่จัดเตรียมไว้จะถูกจัดกลุ่มและกล่าวถึงที่นี่ ตามฟังก์ชันที่ระบุเฉพาะ

เซอร์วิสการควบคุมฮาร์ดแวร์

รูทีนย่อยต่อไปนี้ถูกจัดเตรียมไว้สำหรับการควบคุมฮาร์ดแวร์ :

รูทีนย่อย	คำอธิบาย
cfgetispeed	ขอรับอัตรา baud อินพุต
cfgetospeed	ขอรับอัตรา baud เอาต์พุต
cfsetispeed	ตั้งค่าอัตรา baud อินพุต
cfsetospeed	ตั้งค่าอัตรา baud เอาต์พุต
tcsendbreak	ส่งเส้นกั้นบนสายสื่อสารอนุกรมแบบอะซิงโครนัส

เซอร์วิสโฟลว์คอนโทรล

รูทีนย่อยต่อไปนี้จะถูกจัดเตรียมไว้สำหรับการควบคุมสายงาน:

รูทีนย่อย	คำอธิบาย
tcdrain	รอเอาต์พุตให้เสร็จสมบูรณ์
tcflow	ดำเนินการกับฟังก์ชันการควบคุมสายงาน
tcflush	ละทิ้งข้อมูลจากคิวที่ระบุ

ข้อมูลเทอร์มินัลและการควบคุม

รูทีนย่อยต่อไปนี้จะถูกจัดเตรียมไว้สำหรับ การจัดการกับกลุ่มและการประมวลผล:

รูทีนย่อย	คำอธิบาย
isatty	พิจารณาว่า อุปกรณ์คือเทอร์มินัลหรือไม่
setcsmap	อ่านชุดโค้ดของไฟล์แม็พและกำหนดให้กับอุปกรณ์อินพุตมาตรฐาน
tcgetattr	ขอรับสถานะของเทอร์มินัล
tcsetattr	ตั้งค่าสถานะของเทอร์มินัล
ttylock, ttywait, ttyunlock หรือ ttylocked	ควบคุมฟังก์ชันการล็อก tty
ttyname	ขอรับชื่อของเทอร์มินัล

เซอร์วิสขนาดหน้าต่างและเทอร์มินัล

เคอร์เนลจะเก็บโครงสร้าง **winsize** เพื่อจัดเตรียมอินเตอร์เฟสที่สอดคล้องกันสำหรับเทอร์มินัลปัจจุบันหรือขนาดหน้าต่าง โครงสร้าง **winsize** มีฟิลด์ต่อไปนี้:

ฟิลด์	คำอธิบาย
ws_row	บ่งชี้จำนวนแถว (ในหน่วยตัวอักษร) บนหน้าต่างหรือเทอร์มินัล
ws_col	บ่งชี้จำนวนของคอลัมน์ (ในหน่วยตัวอักษร) บนหน้าต่าง หรือเทอร์มินัล
ws_xpixel	บ่งชี้ถึงขนาดในแนวนอน (ในหน่วยพิกเซล) ของหน้าต่างหรือเทอร์มินัล
ws_ypixel	บ่งชี้ขนาดในแนวตั้ง (หน่วยพิกเซล) ของหน้าต่างหรือเทอร์มินัล

ตามระเบียบแล้ว ค่า 0 ที่อยู่ในฟิลด์โครงสร้าง **winsize** ทั้งหมดจะบ่งชี้ว่า ไม่ได้ตั้งค่าโครงสร้างไว้

รูทีนย่อย	คำอธิบาย
termdef	เคียวรีคุณสมบัติของเทอร์มินัล
TIOCGWINSZ	ขอรับขนาดของหน้าต่าง อาร์กิวเมนต์ในการดำเนินการ ioctl นี้คือ ตัวชี้ไปยังโครงสร้าง **winsize** ที่เทอร์มินัลปัจจุบัน หรือขนาดหน้าต่างที่ถูกวางตำแหน่ง
TIOCSWINSZ	ตั้งค่าขนาดของหน้าต่าง อาร์กิวเมนต์ในการดำเนินการ ioctl นี้ คือตัวชี้ไปยังโครงสร้าง **winsize** ซึ่งถูกใช้เพื่อตั้งค่าเทอร์มินัลปัจจุบัน หรือข้อมูลขนาดของหน้าต่าง ถ้าข้อมูลใหม่แตกต่างจากข้อมูลก่อนหน้านี้ สัญญาณ **SIGWINCH** จะถูกส่งไปยังกลุ่มของการประมวลผลเทอร์มินัล

เซอร์วิสการจัดการกลุ่มกระบวนการ

รูทีนย่อยต่อไปนี้จะถูกจัดเตรียมไว้สำหรับ การจัดการกับกลุ่มของการประมวลผล:

รูทีนย่อย	คำอธิบาย
tcgetpgrp	ขอรับ ID กลุ่มของการประมวลผลส่วนหน้า
tcsetpgrp	ตั้งค่า ID กลุ่มของการประมวลผลส่วนหน้า

การดำเนินการขนาดบัฟเฟอร์

การดำเนินการ ioctl ต่อไปนี้จะถูกใช้สำหรับการตั้งค่าขนาดของอินพุตเทอร์มินัล และเอาต์พุตบัฟเฟอร์ อาร์กิวเมนต์ของการดำเนินการเหล่านี้คือ ตัวชี้ไปยังตัวเลขที่ระบุขนาดของบัฟเฟอร์

การดำเนินการ	คำอธิบาย
TXSETIHOG	ตั้งค่าข้อจำกัด hog สำหรับจำนวนของอักขระอินพุตที่สามารถรับได้ และเก็บไว้ในบัฟเฟอร์ tty ภายในก่อนการประมวลผลจะอ่านอักขระอินพุตเหล่านั้น ข้อจำกัด hog ที่เป็นค่าดีฟอลต์คือ 8192 ตัวอักษร หากเข้าใกล้ข้อจำกัด hog บวกกับหนึ่งอักขระ ข้อผิดพลาดจะถูกบันทึกลงในบันทึกข้อผิดพลาด และอินพุตบัฟเฟอร์จะถูกล้างข้อมูล จำนวน hog ไม่ควรมีขนาดใหญ่เกินไป เนื่องจากบัฟเฟอร์ได้ถูกจัดสรรจากหน่วยความจำของระบบ
TXSETOHOG	ตั้งค่าข้อจำกัด hog สำหรับจำนวนของอักขระเอาต์พุตที่บัฟเฟอร์ในอินพุต echo ข้อจำกัด hog ที่เป็นค่าดีฟอลต์คือ 8192 ตัวอักษร หากเข้าใกล้ข้อจำกัดเอาต์พุต hog อักขระอินพุตไม่สามารถ echo ได้ จำนวน hog ไม่ควรมีขนาดใหญ่เกินไป เนื่องจากบัฟเฟอร์ได้ถูกจัดสรรจากหน่วยความจำของระบบ

การประสานเวลา

ระบบย่อย tty จะใช้ประโยชน์ของการประสานเวลาที่จัดเตรียมไว้โดย STREAMS โมดูลสตรีม tty จะถูกปรับแต่ง ด้วยการประสานเวลาในระดับของการจับคู่คิว การประสานเวลานี้อนุญาตให้ใช้การประมวลผลแบบขนาน สำหรับสตรีมที่ต่างกันสองตัว

ข้อมูลที่เกี่ยวข้อง:

rlogind

setmaps

stty

xdm

eucioctl.h

lft

pty

setmaps file

termios.h

tty file

โมดูล Line discipline (ldterm)

ldterm line discipline คือ line discipline ทั่วไปสำหรับเทอร์มินัล

line discipline นี้มีความเข้ากันได้กับ POSIX และยังทำให้แน่ใจว่ามีความเข้ากันได้กับอินเตอร์เฟส BSD line discipline ในภายหลัง สนับสนุนเฉพาะความเข้ากันได้กับแอ็พพลิเคชันเก่าเท่านั้น เพื่อเหตุผล ด้านการพกพา ขอแนะนำอย่างยิ่งให้คุณใช้อินเตอร์เฟส POSIX ใน แอ็พพลิเคชันใหม่

ส่วนนี้อธิบายคุณลักษณะที่จัดให้มีโดย **ldterm** line discipline สำหรับข้อมูลเพิ่มเติมเกี่ยวกับการควบคุม **ldterm** โปรดดูที่ ″termios.h File″ ใน *Files Reference*

พารามิเตอร์ Terminal

พารามิเตอร์ที่ควบคุมคุณสมบัติ I/O เทอร์มินัลเฉพาะ ที่ระบุในโครงสร้าง **termios** ดังที่กำหนดในไฟล์ **termios.h** โครงสร้าง **termios** ประกอบด้วย (แต่ไม่ได้จำกัด) สมาชิก ต่อไปนี้:

`tcflag_t c_iflag`
: โหมดอินพุต

`tcflag_t c_oflag`
: โหมดเอาต์พุต

`tcflag_t c_cflag`
: โหมดการควบคุม

`tcflag_t c_lflag`
: โหมดโลคัล

`cc_t c_cc[NCCS]`
: อักขระควบคุม

ประเภทจำนวนเต็มที่ไม่มีเครื่องหมาย `tcflag_t` และ `cc_t` ถูกกำหนดในไฟล์ **termios.h** สัญลักษณ์ **NCCS** ถูกกำหนดในไฟล์ **termios.h** เช่นกัน

การจัดการเซสชันกลุ่มกระบวนการ (job control)

เทอร์มินัลการควบคุมจะแยกกลุ่มการประมวลผลหนึ่ง ในเซสชัน ซึ่งจะถูกเชื่อมโยงเป็นกลุ่มการประมวลผล ส่วนหน้า กลุ่มการประมวลผลอื่นๆ ทั้งหมดในเซสชันถูกออกแบบเป็นกลุ่ม การประมวลผลส่วนหลัง กลุ่มการประมวลผลส่วนหน้าดำเนินบทบาทพิเศษในการจัดการ สัญญาณ

การประมวลผลของตัวแปลคำสั่งที่สนับสนุนการควบคุมงาน เช่น Korn shell (คำสั่ง **ksh**) และ C shell (คำสั่ง **csh**) สามารถจัดสรรเทอร์มินัลให้กับ *งาน* หรือกลุ่มการประมวลผลที่ต่างกัน โดยการรวมการประมวลผลที่เกี่ยวข้องกัน ในกลุ่มการประมวลผลเดียว หรือการเชื่อมโยงกลุ่มการประมวลผลนี้กับเทอร์มินัล การประมวลผลส่วนหน้าของเทอร์มินัลสามารถถูกตั้งค่า หรือตรวจสอบโดยการประมวลผล โดยถือว่ามีสิทธิตรงตามข้อกำหนดสิทธิที่ต้องการ เทอร์มินัลไดรเวอร์จะช่วยใน การจัดสรรงานโดยการจำกัดการเข้าถึงเทอร์มินัลของการประมวลผลที่ ไม่ได้อยู่ในกลุ่มการประมวลผลส่วนหน้า

การควบคุมการเข้าถึงเทอร์มินัล

ถ้าการประมวลผลที่ไม่อยู่ในกลุ่มการประมวลผลส่วนหน้า ของเทอร์มินัลการควบคุมของการประมวลผลพยายามอ่านข้อมูลจากเทอร์มินัลการควบคุม กลุ่มการประมวลผลของการประมวลผลนั้นจะถูกส่งสัญญาณ **SIGTTIN** อย่างไรก็ตาม ถ้ากระบวนการอ่านกำลังละเว้นหรือบล็อกสัญญาณ **SIGTTIN** หรือถ้ากลุ่มการประมวลผลของกระบวนการอ่านไม่สามารถดำเนินการได้ การร้องขอเพื่ออ่านจะส่งกลับค่า –1 ตั้งค่าตัวแปรโกลบอล **errno** เป็น **EIO** และไม่ส่งสัญญาณ

ถ้าการประมวลผลที่ไม่อยู่ในกลุ่มการประมวลผลส่วนหน้า ของเทอร์มินัลการควบคุมของการประมวลผลพยายามเขียนข้อมูลในเทอร์มินัลการควบคุม กลุ่มการประมวลผลของการประมวลผลนั้นจะถูกส่งสัญญาณ **SIGTTOU** อย่างไรก็ตาม การจัดการสัญญาณ **SIGTTOU** จะขึ้นกับ ค่าแฟล็ก **TOSTOP** ซึ่งถูกกำหนดในฟิลด์ `c_lflag` ของโครงสร้าง **termios** ถ้าแฟล็ก **TOSTOP** ไม่ถูกตั้งค่า หรือถ้าแฟล็ก **TOSTOP** ถูกตั้งค่าและการประมวลผลกำลังละเว้น หรือบล็อกสัญญาณ **SIGTTOU** การประมวลผลได้รับอนุญาตให้เขียนไปยังเทอร์มินัล และ สัญญาณ **SIGTTOU** ไม่ถูกส่ง ถ้าแฟล็ก **TOSTOP** ถูกตั้งค่า กลุ่มการประมวลผลของกระบวนการเขียนไม่สามารถดำเนินการได้ และกระบวนการเขียนไม่ได้ละเว้นหรือบล็อกสัญญาณ **SIGTTOU** ดังนั้นการร้องขอเพื่อเขียนจะส่งกลับค่า –1 ตั้งค่าตัวแปรโกลบอล **errno** เป็น **EIO** และ ไม่ส่งสัญญาณ

ฟังก์ชันบางฟังก์ชันที่ตั้งค่าพารามิเตอร์เทอร์มินัล (**tcsetattr, tcsendbreak, tcflow** และ **tcflush**) ถูกปฏิบัติในแนวทางเดียวกับการร้องขอเพื่อเขียน ยกเว้นว่าแฟล็ก **TOSTOP** ถูกละเว้น ผลที่ได้จะเหมือนกับการร้องขอเพื่อเขียนไปยังเทอร์มินัลเมื่อแฟล็ก **TOSTOP** ถูกตั้งค่า

การอ่านข้อมูลและการประมวลผลอินพุต

การประมวลผลอินพุตโดยทั่วไปมีสองประเภท ทั้งนี้ขึ้นกับว่าไฟล์อุปกรณ์เทอร์มินัลอยู่ในโหมด canonical หรือ noncanonical นอกจากนั้น อักขระอินพุตจะถูกประมวลผลตามค่าฟิลด์ `c_iflag` และ `c_lflag` ซึ่งการประมวลผล นั้นสามารถมี *echoing* หรือการส่งอักขระอินพุต กลับโดยทันทีไปยังเทอร์มินัลที่ส่งมา การสะท้อนมีประโยชน์ สำหรับเทอร์มินัลที่สามารถดำเนินงานในโหมด full-duplex

การร้องขอเพื่ออ่านสามารถถูกจัดการได้สองวิธีขึ้นอยู่กับ ว่าแฟล็ก **O_NONBLOCK** ถูกตั้งค่าโดยรูทีนย่อย **open** หรือ **fcntl** ถ้าแฟล็ก **O_NONBLOCK** ไม่ถูกตั้งค่า การร้องขอเพื่ออ่านจะถูกบล็อกจนกว่าข้อมูล พร้อมใช้งาน หรือจนกว่าได้รับสัญญาณ ถ้าแฟล็ก **O_NONBLOCK** ถูกตั้งค่า จะดำเนินการการร้องขอเพื่ออ่านเสร็จสมบูรณ์โดยไม่มีการบล็อก ในวิธี ใดวิธีหนึ่งในสามวิธี:

- ถ้ามีข้อมูลเพียงพอต่อความต้องการของ การร้องขอทั้งหมด การร้องขอเพื่ออ่านจะเสร็จสมบูรณ์ และส่งกลับ จำนวนไบต์ที่อ่าน
- ถ้าไม่มีข้อมูลเพียงพอต่อ ความต้องการของการร้องขอทั้งหมด การร้องขอเพื่ออ่านจะเสร็จสมบูรณ์โดย อ่านมากสุดเท่าที่จะกระทำได้ และส่งกลับจำนวนไบต์ที่สามารถ อ่านได้

- ถ้าไม่มีข้อมูลพร้อมใช้ การร้องขอเพื่ออ่าน จะส่งกลับค่า -1 และตั้งค่าตัวแปรโกลบอล **errno** เป็น **EAGAIN**

ความพร้อมใช้ของข้อมูลจะขึ้นอยู่กับโหมดการประมวลผล ข้อมูลว่าเป็น canonical หรือ noncanonical โหมด canonical หรือ noncanonical สามารถถูกตั้งค่าด้วยคำสั่ง **stty**

การประมวลผลอินพุตโหมด Canonical

ถ้าการประมวลผลอินพุตโหมด canonical (แฟล็ก **ICANON** ถูกตั้งค่าในฟิลด์ `c_lflag` ของโครงสร้าง **termios**) อินพุตของเทอร์มินัลจะถูกประมวลผลหน่วยเป็นจำนวนบรรทัด บรรทัด ถูกคั่นด้วยอักขระขึ้นบรรทัดใหม่ (ASCII LF) และอักขระสิ้นสุดไฟล์ (EOF) หรืออักขระสิ้นสุดบรรทัด (EOL) นี้หมายความว่าโปรแกรมที่กำลังพยายาม อ่านถูกบล็อกจนกว่าทั้งบรรทัดถูกพิมพ์หรือได้รับ สัญญาณ รวมทั้ง ไม่ว่าอักขระจะถูกระบุในการร้องขอเพื่ออ่านเป็นจำนวน เท่าใด ก็จะไม่มีการส่งกลับเกินกว่าหนึ่งบรรทัด อย่างไรก็ตาม ไม่จำเป็น ต้องอ่านทั้งบรรทัดในครั้งเดียว คุณสามารถระบุจำนวนอักขระเท่าใดก็ได้ ในการร้องขอเพื่ออ่านโดยไม่เกิดการสูญหายของข้อมูล ระหว่าง อินพุต มีการดำเนินการลบและคิลการประมวลผล

อักขระ ERASE
(Backspace ค่าดีฟอลต์) ลบอักขระล่าสุดที่พิมพ์

อักขระ WERASE
(ลำดับคีย์ Ctrl-W ค่าดีฟอลต์) ลบคำล่าสุดที่พิมพ์ใน บรรทัดปัจจุบัน แต่ไม่ลบช่องว่างหรือแท็บใดๆ ที่นำหน้า

(*word* ถูกกำหนดเป็นลำดับของอักขระที่ไม่ใช่ช่องว่าง แท็บถูกถือเป็นช่องว่าง) ไม่มีอักขระ ERASE หรือ WERASE ที่จะลบเกินจุดเริ่มต้นของบรรทัด

อักขระ KILL
(ลำดับ Ctrl-U ค่าดีฟอลต์) บนบรรทัดอินพุตทั้งบรรทัดและ เป็นทางเลือกจะเอาต์พุตอักขระขึ้นบรรทัดใหม่

อักขระทั้งหมดเหล่านี้ดำเนินการตามการเคาะคีย์บอร์ด โดยไม่เกี่ยวข้องกับ การกดถอยกลับหรือแท็บใดที่อาจเกิดขึ้น

อักขระ REPRINT
(ลำดับ Ctrl-R ค่าดีฟอลต์) พิมพ์บรรทัดใหม่ตามด้วยอักขระ จากบรรทัดก่อนหน้าที่ยังไม่ได้ถูกอ่าน

การพิมพ์ซ้ำยังเกิดขึ้นโดยอัตโนมัติถ้าอักขระที่โดยทั่วไปควร ถูกลบออกจากหน้าจอถูกแสดงผิดพลาดโดยเอาต์พุตของโปรแกรม อักขระ ถูกพิมพ์ซ้ำเหมือนกับถูกสะท้อน ดังนั้น ถ้าแฟล็ก **ECHO** ไม่ถูกตั้งค่าในฟิลด์ `c_lflag` ของโครงสร้าง **termios** อักขระจะไม่ถูกพิมพ์ อักขระ ERASE และ KILL สามารถถูกป้อนเป็นตัวอักษรโดยนำหน้าด้วย escape character \ (backslash) ซึ่ง escape character จะไม่ถูกอ่าน อักขระ ERASE, WERASE และ KILL สามารถเปลี่ยนแปลงได้

การประมวลผลอินพุตโหมด Noncanonical

ในการประมวลผลอินพุตโหมด noncanonical (แฟล็ก **-ICANON** ที่ตั้งค่าในฟิลด์ `c_lflag` ของโครงสร้าง **termios**) ไบต์อินพุตไม่ถูกรวมเป็นบรรทัด และลบและคิลการประมวลผลไม่เกิดขึ้น

MIN แทนจำนวนไบต์ต่ำสุดที่ควรได้รับเมื่อ ดำเนินการร้องขอเพื่ออ่านสำเร็จ

TIME ตัวจับเวลาหน่วย 0.1 วินาทีที่ถูกใช้เพื่อส่งการหมดเวลาใช้งานเป็นชุดอย่างเร็ว และการส่งข้อมูลระยะสั้น

ค่าของสมาชิก MIN และ TIME ของอาร์เรย์ `c_cc` ถูกใช้เพื่อพิจารณาวิธีประมวลผลไบต์ที่ได้รับ ค่า MIN และ TIME สามารถถูกตั้งค่าด้วยคำสั่ง **stty** MIN และ MAX มีค่าตั้งแต่ 0 ถึง 265 การผสมที่เป็นไปได้สี่แบบสำหรับ MIN และ TIME และการโต้ตอบถูกอธิบายในย่อหน้าต่อไป

Case A: MIN0, TIME0

ในกรณีนี้ TIME ทำหน้าที่เป็นตัวจับเวลาระหว่างไบต์ ซึ่ง ถูกเรียกทำงานหลังได้รับไบต์แรกและตั้งค่าใหม่ในแต่ละครั้งที่ได้รับหนึ่งไบต์ ถ้าได้รับ MIN ไบต์ก่อนตัวจับเวลาระหว่างไบต์หมดอายุ การร้องขอเพื่ออ่านจะเป็นตามต้องการ ถ้าตัวจับเวลาหมดอายุก่อนได้รับ MIN ไบต์ อักขระที่ได้รับณ จุดนั้นจะถูกส่งกลับไปยังผู้ใช้ ถ้า TIME หมดอายุ จะมีการส่งกลับอย่างน้อยหนึ่งไบต์ (ตัวจับเวลาจะไม่ถูกเปิดใช้งาน ยกเว้น ได้รับหนึ่งไบต์) การดำเนินการอ่านจะบล็อกจนกระทั่งกลไก MIN และ TIME ถูกเรียกทำงานโดยการได้รับไบต์แรก หรือจนกระทั่งได้รับสัญญาณ

Case B: MIN0, TIME = 0

ในกรณีนี้ สำคัญที่ค่า MIN เท่านั้น ตัวจับเวลา ไม่มีนัยสำคัญ (ค่าของ TIME เป็น 0) การร้องขอเพื่ออ่านที่ค้างอยู่จะไม่ถูกดำเนินตามต้องการ (บล็อก) จนกว่าได้รับ MIN ไบต์ หรือจนกว่าได้รับสัญญาณ โปรแกรม ที่ใช้กรณีนี้เพื่ออ่าน I/O เทอร์มินัลตามค่าเร็กคอร์ดสามารถบล็อกโดยไม่มีสิ้นสุด ในการดำเนินการอ่าน

Case C: MIN = 0, TIME0

ในกรณีนี้ เนื่องจากค่าของ MIN เป็น 0 ดังนั้น TIME ไม่ได้เป็นตัวจับเวลาระหว่างไบต์อีกต่อไป ในตอนนี้ทำหน้าที่เป็นตัวจับเวลาการอ่านที่ ถูกเรียกทำงานทันทีที่มีการประมวลผลการร้องขอเพื่ออ่าน การร้องขอเพื่ออ่านดำเนินตามต้องการ ทันที่ที่ได้รับหนึ่งไบต์ หรือเมื่อตัวจับเวลาการอ่านหมดอายุ โปรดทราบว่าถ้า ตัวจับเวลาหมดอายุ จะไม่มีไบต์ได้ถูกส่งกลับ ถ้าตัวจับเวลาไม่หมดอายุ การร้องขอเพื่ออ่านสามารถดำเนินตามต้องการต่อเมื่อได้รับหนึ่งไบต์ ในกรณีนี้ การดำเนินการอ่านไม่ได้บล็อกอย่างไม่มีสิ้นสุด เพื่อรอไบต์หนึ่งไบต์ ถ้าหลังจาก การร้องขอเพื่ออ่านถูกเริ่มต้น ไม่ได้รับไบต์จภายในระยะเวลาที่ระบุโดย TIME คูณด้วย 0.1 วินาที การร้องขอเพื่ออ่านจะส่งกลับค่า 0, โดยไม่มีการอ่านข้อมูล

กรณี D: MIN = 0, TIME = 0

ในกรณีนี้ ค่าต่ำสุดของจำนวนไบต์ ที่ร้องขอหรือจำนวนไบต์ที่พร้อมใช้งานถูกส่งกลับโดยไม่รอง การอินพุตเพิ่มอย่างน้อย 1 ไบต์ ถ้าไม่มีอักขระใดพร้อมใช้ได้ การร้องขอเพื่ออ่าน ส่งกลับค่า 0 โดยไม่อ่านข้อมูล

กรณี A และ B มีเพื่อจัดการกิจกรรมโหมดส่งเป็นชุดอย่างรวดเร็ว เช่นโปรแกรมโอนย้ายไฟล์ โดยที่โปรแกรมจำเป็นต้องประมวลผล อักขระอย่างน้อยที่สุดเป็นจำนวนตามที่ระบุโดย MIN ในหนึ่งครั้ง ในกรณี A ตัวจับเวลาระหว่างไบต์ถูกเรียกทำงานเป็นการวัดความปลอดภัย ในกรณี B ตัวจับเวลา ถูกปิดทำงาน

กรณี C และ D มีเพื่อจัดการอักขระเดียว เป็นการถ่ายโอน ที่จำกัด กรณีเหล่านี้พร้อมนำไปปรับใช้กับแอ็พพลิเคชันแบบหน้าจอที่จำเป็นต้องทราบมีอักขระแสดงในคิวอินพุตก่อนทำการรีเฟรช หน้าจอ ในกรณี C ตัวจับเวลาถูกเรียกทำงาน ในกรณี D ตัวจับเวลาถูกปิด ทำงาน กรณี D สามารถส่งผลต่อผลการทำงานถ้าใช้มากเกิน แต่ใช้ดีกว่า ดำเนินการร้องขอเพื่ออ่านด้วยการตั้งค่าแฟล็ก **O_NONBLOCK**

การเขียนข้อมูลและการประมวลผลเอาต์พุต

เมื่อมีอักขระอย่างน้อยหนึ่งอักขระถูกเขียน จะถูกส่งไปยัง เทอร์มินัลทันทีที่อักขระที่เขียนก่อนหน้านี้ถูกแสดง (อักขระ อินพุตถูกสะท้อนการแสดงโดยการวางอักขระในคิวเอาต์พุตเมื่อมาถึง) ถ้าการประมวลผลสร้างอักขระอย่างรวดเร็วมากกว่าที่จะสามารถแสดงได้ การประมวลผลจะหยุดทำงานชั่วคราวเมื่อคิวเอาต์พุตมีจำนวนเกินจำนวนที่จำกัด เมื่อคิว ลดจำนวนลงถึงค่าเส้นแบ่ง โปรแกรมจะทำงานต่อ

การจัดการโมเด็ม

ถ้าแฟล็ก **CLOCAL** ถูกตั้งค่า ในฟิลด์ c_cflag ของโครงสร้าง **termios** การเชื่อมต่อจะไม่ขึ้นกับสถานะของบรรทัดแสดงสถานะ โมเด็ม ถ้าแฟล็ก **CLOCAL** ถูกลบทั้งหมด บรรทัดแสดงสถานะ โมเด็มจะถูกมอนิเตอร์ในเหตุการณ์ปกติ ฟังก์ชัน **open** รอให้เสร็จสมบูรณ์ อย่างไรก็ตาม ถ้าแฟล็ก **O_NONBLOCK** หรือ **CLOCAL** ถูกตั้งค่า ฟังก์ชัน open จะส่งกลับค่าทันทีโดยไม่รอการเชื่อมต่อ

ถ้าแฟล็ก **CLOCAL** ไม่ถูกตั้งค่า ในฟิลด์ c_cflag ของโครงสร้าง **termios** และการยกเลิกการเชื่อมต่อโมเด็มถูกตรวจพบโดยอินเตอร์เฟส เทอร์มินัลสำหรับเทอร์มินัลการควบคุม สัญญาณ **SIGHUP** ถูกส่งไปที่กระบวนการควบคุมที่เชื่อมโยงกับเทอร์มินัล ยกเว้นมีการจัดการ เป็นอย่างอื่นไว้ สัญญาณนี้ทำให้การประมวลผลจบการทำงาน ถ้าสัญญาณ **SIGHUP** ถูกละเลย หรือพบ การร้องขอเพื่ออ่านที่มีในภายหลังใดๆ จะส่งกลับค่าที่บ่งบอกว่าสิ้นสุดไฟล์ จนกระทั่งเทอร์มินัลถูกปิด การร้องขอเพื่อเขียนที่มีในภายหลังใดๆ ไปยังเทอร์มินัลส่งกลับค่า -1 และตั้งค่า ตัวแปรโกลบอล **errno** เป็น **EIO** จนกระทั่ง อุปกรณ์ถูกปิดทำงาน

การปิดไฟล์อุปกรณ์เทอร์มินัล

การประมวลผลสุดท้ายที่ปิดไฟล์อุปกรณ์เทอร์มินัลเป็นสาเหตุให้ เอาต์พุตใดๆ ที่ถูกส่งไปยังอุปกรณ์และอินพุตใดๆ จะถูกละทิ้ง ดังนั้น ถ้า แฟล็ก **HUPCL** ถูกตั้งค่าในฟิลด์ c_cflag ของโครงสร้าง **termios** และ พอร์ตการสื่อสารสนับสนุนฟังก์ชันยกเลิกการเชื่อมต่อ อุปกรณ์เทอร์มินัลจะดำเนินการ ยกเลิกการเชื่อมต่อ

โมดูลตัวแปลง

โมดูล Converter เป็นโมดูลทางเลือก โมดูลถูกส่ง ไปบนสตรีม tty เมื่อจำเป็นเท่านั้น

โดยปกติโมดูลถูกจัดเตรียมสำหรับวัตถุประสงค์การทำให้เป็น สากลและดำเนินการแม็พอักขระต่างๆ

โมดูลการแปลงดังต่อไปนี้ถูกจัดส่ง:

- โมดูล **nls**
- โมดูล **uc_sjis** และ **lc_sjis**

โมดูล NLS

โมดูล **nls** คือโมดูลตัวแปลงระดับต่ำ ที่สามารถถูกส่งไปบนสตรีม tty ภายใต้ line discipline โมดูล **nls** ประกันการแม็พเทอร์มินัล: โมดูลทำการแม็พ อักขระอินพุตและเอาต์พุตสำหรับเทอร์มินัลที่ไม่ใช่แบบมาตรฐาน (นั่นคือสำหรับ เทอร์มินัลที่ไม่สนับสนุน codeset ISO 8859 พื้นฐานของระบบ)

กฎการแม็พถูกระบุในไฟล์แม็พสองไฟล์อยู่ใน ไดเร็กทอรี **/usr/lib/nls/termmap** ไฟล์ **.in** มีกฎการแม็พสำหรับอินพุตแป้นพิมพ์ไฟล์ **.out** มีกฎการแม็พสำหรับเอาต์พุตแสดงผล รูปแบบไฟล์ถูกระบุในรูปแบบไฟล์ **setmaps** "setmaps File Format" ใน *Files Reference*

โมดูล SJIS

โมดูล **uc_sjis** and **lc_sjis** คือโมดูลตัวแปลงที่สามารถผลักดันลงไปยังสตรีม tty ได้ โมดูลเหล่านี้รับประกันการจัดการกับชุดของโค้ด: โมดูลเหล่านี้เรียกทำงานการแปลงของอักขระแบบหลายไบต์ ระหว่างรูปแบบ shifted Japanese industrial standard (SJIS) และรูปแบบ advanced Japanese EUC code (AJEC) ซึ่งได้รับการสนับสนุนโดยระเบียบสายสื่อสาร โมดูลเหล่านี้จำเป็นต้องมีเมื่อผู้ใช้ ประมวลผลและเทอร์มินัลฮาร์ดแวร์ใช้ชุดรหัส IBM-943

AJEC เป็นการใช้ภาษาญี่ปุ่นของของวิธีการเข้ารหัส extended UNIX code (EUC) ซึ่งอนุญาตการรวมอักขระ ASCII, phonetic Kana, และ ideographic Kanji AJEC เป็นซูเปอร์เซ็ตของ UNIX Japanese industrial standard (UJIS), การนำภาษาญี่ปุ่นธรรมดามาใช้ของ EUC

ข้อมูล Japanese-encoded ประกอบด้วยอักขระจาก ชุดรหัสได้มากถึงสี่ชุด:

ชุดรหัส	อักขระที่มี
ASCII	อักษรโรมัน ตัวเลข เครื่องหมายวรรคตอนและอักขระควบคุม
JIS X0201	Phonetic Kana
JIS X0208	Ideographic Kanji
JIS X0212	Supplemental Kanji

AJEC ใช้ชุดรหัสทั้งสี่ชุด SJIS ใช้เฉพาะ ชุดรหัส ASCII, JIS X0201 และ JIS X0208 ดังนั้นโมดูล **uc_sjis** และ **lc_sjis** แปลง:

- อักขระ SJIS ทั้งหมดเป็นอักขระ AJEC
- อักขระ AJEC จากชุดรหัส ASCII, JIS X0201, และ JIS X0208 เป็นอักขระ SJIS
- อักขระ AJEC จากชุดรหัส JIS X0212 เป็น อักขระที่ไม่ได้กำหนด SJIS

โมดูล **uc_sjis** และ **lc_sjis** ถูกใช้ร่วมกันเสมอ ตัวแปลงระดับบน **uc_sjis** ถูกส่งไปบนสตรีม tty ที่อยู่เหนือ line discipline; ตัวแปลงระดับล่าง **lc_sjis** ถูกส่งไปบนสตรีม ใต้ line discipline โมดูล **uc_sjis** และ **lc_sjis** ถูกส่งโดยอัตโนมัติไปบนสตรีม tty โดยคำสั่ง **setmaps** และรูทีนย่อย **setcsmap** โมดูล ยังถูกควบคุมโดยการดำเนินการ EUC ioctl ตามที่อธิบายในไฟล์ **eucioctl.h** ใน *Files Reference*

ไดรเวอร์ TTY

ไดรเวอร์ tty คือไดรเวอร์ STREAMS ที่จัดการกับการเชื่อมต่อกับเทอร์มินัลของฮาร์ดแวร์ที่เกิดขึ้นจริง

ไดรเวอร์ tty มีอยู่ด้วยกันสามชนิด ซึ่งขึ้นอยู่กับการเชื่อมต่อ นั่นคือ: ไดรเวอร์ line แบบอะซิงโครนัส ไดรเวอร์ pty และไดรเวอร์ LFT

อะซิงโครนัสไลน์ไดรเวอร์

ไดรเวอร์ line แบบอะซิงโครนัสจัดเตรียมไว้เพื่อสนับสนุนอุปกรณ์ (ซึ่งปกติคือเทอร์มินัล ASCII) ที่เชื่อมต่อกับระบบโดยตรงผ่านสายสื่อสารแบบอะซิงโครนัส ซึ่งรวมถึงโมเด็มด้วยเช่นกัน

ไดรเวอร์ line แบบอะซิงโครนัสจัดเตรียมอินเตอร์เฟสสำหรับสายสื่อสารที่ควบคุมฮาร์ดแวร์:

- ไดรเวอร์ **cxma** สนับสนุนการ์ดอะแด็ปเตอร์ PCI 128 พอร์ต
- ไดรเวอร์ **cxpa** สนับสนุนการ์ดอะแด็ปเตอร์ PCI 8 พอร์ต
- ไดรเวอร์ **sf** สนับสนุนพอร์ตดั้งเดิมบน planar ของระบบ
- ไดรเวอร์ **sa** สนับสนุนการ์ดอะแด็ปเตอร์ PCI 2 พอร์ต

ไดรเวอร์ line แบบอะซิงโครนัสสำหรับการตั้งค่าพารามิเตอร์ เช่น อัตรา baud ขนาดตัวอักษร และการตรวจสอบพาริตี ผู้ใช้สามารถควบคุมพารามิเตอร์เหล่านี้ผ่านฟิลด์ `c_cflag` ของโครงสร้าง **termios**

ไดรเวอร์ line แบบอะซิงโครนัสมีลักษณะดังต่อไปนี้:

- ตัวควบคุมการทำงานของฮาร์ดแวร์และซอฟต์แวร์ หรือการปฏิบัติตามทำตามระเบียบจะระบุวิธีการจัดการกับการเชื่อมต่อเพื่อป้องกันบัฟเฟอร์โอเวอร์โฟลว์ ผู้ใช้สามารถควบคุมคุณลักษณะผ่านฟิลด์ `c_iflag` ของโครงสร้าง **termios** (ตัวควบคุมการทำงานของซอฟต์แวร์) และฟิลด์ `x_hflag` ของโครงสร้าง **termiox** (ตัวควบคุมการทำงานของฮาร์ดแวร์)
- การเปิดใช้ระเบียบข้อบังคับจะระบุวิธีการสร้างการเชื่อมต่อ คุณลักษณะนี้จะถูกควบคุมในเวลาที่ทำคอนฟิกูเรชันผ่านฟิลด์ `x_sflag` ของโครงสร้าง **termiox**

ไดรเวอร์ Pseudo-terminal

ไดรเวอร์ pseudo-terminal (pty) จัดเตรียมไว้เพื่อสนับสนุนเทอร์มินัลที่ต้องการประมวลผลพิเศษ เช่น เทอร์มินัล X หรือระบบรีโมตที่เชื่อมผ่านเน็ตเวิร์ก

ไดรเวอร์ pty จะส่งข้อมูลอินพุต และเอาต์พุตจากแอ็พพลิเคชันไปยังเซิร์ฟเวอร์ที่ประมวลผลผ่านสตรีมสำรอง การประมวลผลเซิร์ฟเวอร์ที่รันอยู่ในพื้นที่สำหรับผู้ใช้ คือ daemon เช่น **rlogind** daemon หรือ **xdm** daemon ซึ่งจะจัดการกับการสื่อสารที่เกิดขึ้นจริงด้วยเทอร์มินัล

โมดูลเผื่อเลือกอื่นๆ อาจถูกส่งไปยังผู้ใช้หรือเซิร์ฟเวอร์สตรีม อย่างใดอย่างหนึ่ง

โหลดเดอร์โดเมน

ในบางสภาวะแวดล้อมการโปรแกรม เป็นการดีกว่าที่จะ มีไลบรารีที่แบ่งใช้ถูกโหลดเข้ามาไว้ที่แอดเดรสเสมือนเดียวกันในแต่ละการประมวลผล

เนื่องจากลักษณะแบบไดนามิกของไลบรารีที่แบ่งใช้ถูกดูแลรักษาโดยโหลดเดอร์ระบบ AIX จึงไม่สามารถรับประกันเงื่อนไขนี้ได้ โหลดเดอร์โดเมนจัดให้มีวิธี การโหลดไลบรารีที่แบ่งใช้ไว้ที่แอดเดรสเสมือนเดียวกันในชุดของการประมวลผล

โหลดเดอร์ระบบได้โหลดไลบรารีที่แบ่งใช้เข้ามาในหลาย ขอบเขตของไลบรารีที่แบ่งใช้แบบโกลบอล ขอบเขตหนึ่งถูกเรียก ขอบเขตข้อความไลบรารี ที่แบ่งใช้ ซึ่งมีคำสั่งที่รันได้สำหรับไลบรารีที่แบ่งใช้ที่ถูกโหลดเข้ามา ขอบเขตข้อความไลบรารีที่แบ่งใช้ถูกแม็พกับแอดเดรสเสมือนเดียวกันในทุก การประมวลผล ขอบเขตอื่นคือขอบเขตข้อมูลไลบรารีที่แบ่งใช้ ขอบเขตนี้มี ข้อมูลสำหรับไลบรารีที่แบ่งใช้ เนื่องจากข้อมูลไลบรารีทีอ่าน/เขียนแบ่งใช้คือการอ่าน/การเขียน ในแต่ละการประมวลผลจะมีขอบเขตส่วนตัวของตนเองที่เป็นสำเนาของขอบเขตไลบรารีที่แบ่งใช้ แบบโกลบอล ขอบเขตส่วนตัวนี้ถูกแม็พไปที่แอดเดรสเสมือนเดียวกัน ในทุกการประมวลผล

ตั้งแต่ที่ขอบเขตไลบรารีที่แบ่งใช้แบบโกลบอลถูกแม็พ ไปที่แอดเดรสเสมือนเดียวกันในทุกการประมวลผล ไลบรารีที่แบ่งใช้ถูกโหลด ไปที่แอดเดรสเสมือนเดียวกันเป็นส่วนใหญ่ กรณีที่คำกล่าวนี้ไม่เป็นความจริง คือเมื่อมีมากกว่าหนึ่งเวอร์ชันของไลบรารีที่แบ่งใช้ถูกโหลดเข้ามาไว้ในระบบ นี้เกิดขึ้นเมื่อใดก็ตามที่ไลบรารีที่แบ่งใช้ที่มีการใช้งานอยู่ถูกแก้ไข หรือ มีไลบรารีที่แบ่งใช้ใดๆ ที่ต้องขึ้นกับไลบรารีนั้นถูกแก้ไข เมื่อเกิดเหตุการณ์นี้ขึ้น โหลดเดอร์ ต้องสร้างไลบรารีที่แบ่งใช้ที่ถูกแก้ไขเวอร์ชันใหม่ขึ้นรวมถึงไลบรารีที่แบ่งใช้อื่นๆ ทั้งหมด ที่ขึ้นอยู่กับไลบรารีที่แบ่งใช้ที่ถูกแก้ไขนั้น โปรดทราบว่าสุดท้ายแล้วไลบรารีที่แบ่งใช้ทั้งหมด จะขึ้นอยู่กับ *Kernel Name Space Kernel Name Space* มีการเรียกใช้ระบบทั้งหมด ที่กำหนดโดยเคอร์เนลและสามารถแก้ไขได้ตลอดเวลาที่ส่วนขยายเคอร์เนลถูกโหลดหรือถูก ยกเลิกการโหลดในลักษณะไดนามิก เมื่อโหลดเดอร์ระบบสร้างไลบรารีที่แบ่งใช้ เวอร์ชันใหม่ เวอร์ชันใหม่นั้นต้องถูกที่ตำแหน่งที่ต่างกันในเซ็กเมนต์ไลบรารีที่แบ่งใช้โกลบอล ดังนั้น การประมวลผลที่ใช้เวอร์ชันใหม่จะมี ไลบรารีที่แบ่งใช้ถูกโหลดมาที่แอดเดรสเสมือนที่ต่างกันกับการประมวลผลที่ ใช้เวอร์ชันก่อนหน้าของไลบรารีที่แบ่งใช้

โหลดเดอร์โดเมนคือเซ็ตย่อยของไลบรารีที่แบ่งใช้ทั้งหมด ที่ถูกโหลดมาระบบ ชุดของไลบรารีทีไลบรารีแบ่งใช้ที่โหลดเข้ามาใน ระบบถูกเรียกว่า *โกลบอลโหลดเดอร์โดเมน* โกลบอลโหลดเดอร์ โดเมนนี้สามารถแบ่งย่อยออกเป็นโหลดเดอร์โดเมนที่ผู้ใช้กำหนดเองที่มีขนาดเล็กลง โหลดเดอร์โดเมนที่ผู้ใช้กำหนดเองมีเวอร์ชันหนึ่งของไลบรารีที่แบ่งใช้เฉพาะ ส่วนใดๆ การประมวลผลสามารถระบุโหลดเดอร์โดเมน ถ้าการประมวลผลระบุโหลดเดอร์ โดเมน การประมวลผลจะใช้ไลบรารีที่แบ่งใช้ที่มีอยู่ในโหลดเดอร์โดเมนนั้น ถ้ามีมากกว่าหนึ่งการประมวลผลระบุที่โหลดเดอร์โดเมนเดียวกัน การประมวลผลเหล่านั้น จะใช้ชุดของไลบรารีที่แบ่งใช้ชุดเดียวกัน เนื่องจากโหลดเดอร์โดเมนมีเวอร์ชันหนึ่งของ ไลบรารีที่แบ่งใช้เฉพาะส่วนใดๆ การประมวลผลทั้งหมดที่ระบุโหลดเดอร์โดเมนเดียวกัน จะใช้เวอร์ชันของไลบรารีที่แบ่งใช้เดียวกันและมีไลบรารีที่แบ่งใช้ของตนถูกโหลดไว้ที่ แอดเดรสเสมือนเดียวกัน

การใช้โหลดเดอร์โดเมน

ถ้าการประมวลผลใช้โหลดเดอร์โดเมน ต้องถูกระบุไว้ ในเวลา exec โหลดเดอร์โดเมนที่ระบุจะมีผลได้ใช้ และใช้สำหรับทั้งหมดระหว่างการประมวลผล เมื่อการประมวลผลที่ระบุโหลดเดอร์โดเมนเรียกใช้ การเรียกใช้ระบบ **exec** โหลดเดอร์ระบบจะดำเนินการ ต่อไปนี้:

ค้นหา/สร้างโหลดเดอร์โดเมน
: สิทธิการเข้าถึงที่เชื่อมโยงกับโหลดเดอร์โดเมนถูกตรวจสอบ เพื่อพิจารณาว่าการประมวลผลนี้สามารถใช้โหลดเดอร์โดเมนได้หรือไม่ ถ้าการประมวลผลไม่มี สิทธิพิเศษที่เพียงพอต่อการเข้าถึง (อ่านหรือเขียน) โหลดเดอร์โดเมน จะไม่มีโดเมนใดถูกใช้งานการประมวลผลนี้ ถ้าการประมวลผลมีสิทธิพิเศษที่เพียงพอ รายการของโหลดเดอร์โดเมนที่ดูแลรักษาโดยโหลดเดอร์ระบบจะถูกค้นเพื่อหา โหลดเดอร์โดเมนที่ระบุโดยการประมวลผล ถ้าไม่พบโหลดเดอร์โดเมนที่ระบุ จะถูกสร้างขึ้นถ้าการประมวลผลมีสิทธิพิเศษที่เพียงพอ ถ้า การประมวลผลไม่มีสิทธิพิเศษเพียงพอที่จะสร้างโหลดเดอร์โดเมน การเรียกใช้ **exec** จะล้มเหลว และส่งกลับข้อผิดพลาด

ใช้โหลดเดอร์โดเมนเพื่อจำกัดการค้นหา
: ถ้าการประมวลผลต้องการใช้ไลบรารีที่แบ่งใช้ใดๆ ที่ถูกแสดงรายการไว้แล้ว ในโหลดเดอร์โดเมน เวอร์ชันของไลบรารีที่ระบุในโดเมนจะถูก ใช้ เวอร์ชันของไลบรารีที่แบ่งใช้ในโหลดเดอร์โดเมนถูกใช้ไม่ว่า จะมีเวอร์ชันอื่นของไลบรารีที่แบ่งใช้อยู่ในโกลบอลโกลบอล โดเมน

เพิ่มไลบรารีแบบแบ่งใช้เข้ากับโหลดเดอร์โดเมน
: ถ้าการประมวลผลต้องการใช้ไลบรารีที่ไม่อยู่ในโหลดเดอร์โดเมน โหลดเดอร์ จะโหลดไลบรารีนั้นเข้ามาไว้ในอิมเมจการประมวลผลโดยทำตามระเบียบวิธีโหลดเดอร์ ปกติของการโหลดเวอร์ชันล่าสุด ถ้าการประมวลผลมีสิทธิพิเศษเพียงพอ ไลบรารีเวอร์ชันนี้จะถูกเพิ่มในโหลดเดอร์โดเมนเช่นกัน ถ้าการประมวลผลไม่มีสิทธิพิเศษที่เพียงพอที่จะเพิ่มรายการ การเรียกใช้ **exec** จะล้มเหลว และส่งกลับข้อผิดพลาด

ไลบรารีที่แบ่งใช้ยังสามารถถูกโหลดด้วยการเรียกใช้ระบบ **load( )** อย่างชัดแจ้ง เมื่อไลบรารีที่แบ่งใช้ถูกโหลดอย่างชัดแจ้ง ข้อมูลสำหรับโมดูลเหล่านี้โดยปกติจะถูกวางไว้ที่ค่าที่หยุดปัจจุบันของการประมวลผล สำหรับการประมวลผล 32 บิต สำหรับการประมวลผล 64 บิต ข้อมูลสำหรับโมดูลถูกวางไว้ ในโมดูลที่ถูกโมดูลเป็นส่วนตัวของขอบเขต ถ้าการประมวลผลใช้โหลดเดอร์โดเมน โหลดเดอร์ระบบจะวางข้อมูลในขอบเขตแบ่งใช้ไลบรารีที่แบ่งใช้ แอดเดรสเสมือน ของโมดูลที่โหลดอย่างชัดแจ้งนี้จะเหมือนกันหมดทุกการประมวลผลที่ โหลดโมดูล ถ้าการประมวลผลมีสิทธิพิเศษเพียงพอ ไลบรารีที่แบ่งใช้ จะถูกเพิ่มในโหลดเดอร์โดเมน ถ้าการประมวลผลไม่มีสิทธิพิเศษเพียงพอ ที่จะเพิ่มรายการ การเรียกใช้ **load** จะล้มเหลว และส่งกลับ ข้อผิดพลาด

โหลดเดอร์โดเมนสามารถเชื่อมโยงกับไฟล์ปกติ ใดๆ เป็นสิ่งสำคัญที่ต้องทราบว่าโหลดเดอร์โดเมนเชื่อมโยงกับ ไฟล์ ไม่ใช่ชื่อพาธของไฟล์ โหมด (สิทธิการเข้าถึง) ของ ไฟล์จะพิจารณาการดำเนินการที่สามารถดำเนินการได้บนโหลดเดอร์โดเมน สิทธิการเข้าถึงบนไฟล์ที่เชื่อมโยงกับโหลดเดอร์โดเมน และการดำเนินการ ที่ได้รับอนุญาตบนโหลดเดอร์โดเมนมีดังนี้:

- ถ้าการประมวลผลสามารถอ่านไฟล์ได้ การประมวลผลจะสามารถระบุโหลดเดอร์โดเมนเพื่อจำกัดชุดของไลบรารีที่แบ่งใช้ที่ใช้ได้
- ถ้าการประมวลผลสามารถเขียนลงไฟล์ การประมวลผลจะสามารถเพิ่มไลบรารีที่แบ่งใช้ให้แก่โหลดเดอร์โดเมนและสร้างโหลดเดอร์โดเมนที่สัมพันธ์กับไฟล์นั้น

ถ้าการประมวลผลพยายามสร้างหรืออ่านรายการไปยังโหลดเดอร์โดเมนโดยไม่มีสิทธิพิเศษที่เหมาะสม การดำเนินการที่กำลังดำเนินอยู่ (**exec** หรือ **load**) จะล้มเหลว และส่งกลับ ข้อผิดพลาด

โหลดเดอร์โดเมนถูกระบุเป็นส่วนหนึ่งของข้อมูล **LIBPATH** ข้อมูล **LIBPATH** คือรายการที่คั่นด้วย โคลอน (:) ของชื่อพาธไดเร็กทอรีที่ใช้เพื่อกำหนดตำแหน่งไลบรารีที่แบ่งใช้ ข้อมูล **LIBPATH** สามารถมากจากตัวแปรสภาวะแวดล้อม **LIBPATH** หรือสตริง **LIBPATH** ที่ระบุในส่วนโหลดเดอร์ของไฟล์เรียกทำงาน ถ้าชื่อพาธแรก ในข้อมูล **LIBPATH** เป็นไฟล์ปกติ โหลดเดอร์โดเมนที่เชื่อมโยงกับไฟล์จะถูกระบุ ตัวอย่างเช่น:

- ถ้า `/etc/loader_domain/OOdomain_1` เป็นไฟล์ปกติ ดังนั้น ให้ตั้งค่าตัวแปรสภาวะแวดล้อม **LIBPATH** เป็นสตริง

  `/etc/loader_domain/OOdomain_1:/lib:/usr/lib`

  ทำให้การประมวลผลสร้างและใช้โหลดเดอร์โดเมนที่เชื่อมโยงกับ ไฟล์ `/etc/loader_domain/OOdomain_1`
- ถ้า `/etc/loader_domain/OOdomain_1` เป็นไฟล์ปกติ ดังนั้น โปรแกรม `ldom` จะสร้างโดยใช้คำสั่งต่อไปนี้:

  ```
  cc -o ldom ldom.c -L/etc/loader_domain/OOdomain_1
  ```

  ชื่อพาธ `/etc/loader_domain/OOdomain_1` ถูกแทรกเป็นรายการแรกในข้อมูล **LIBPATH** ของส่วนโหลดเดอร์สำหรับไฟล์ `ldom` เมื่อ `ldom` ถูกทำงาน จะสร้างและใช้โหลดเดอร์โดเมน ที่เชื่อมโยงกับไฟล์ `/etc/loader_domain/OOdomain_1`

การสร้าง/การลบโหลดเดอร์โดเมน

โหลดเดอร์โดเมนถูกสร้างเป็นครั้งแรกเมื่อการประมวลผล ที่มีสิทธิพิเศษที่เพียงพอพยายามใช้โดเมน การเข้าถึงโหลดเดอร์โดเมน ถูกควบคุมโดยการเข้าถึงไฟล์ปกติที่เชื่อมโยงกับโดเมน ผู้เขียน แอ็พพลิเคชันมีหน้าที่รับผิดชอบต่อการจัดการไฟล์ปกติที่เชื่อมโยงกับโหลดเดอร์โดเมนที่ใช้โดยแอ็พพลิเคชัน โหลดเดอร์โดเมนถูกเชื่อมโยงกับไฟล์ ปกติ ไม่ใช่ชื่อพาธของไฟล์ ตัวอย่างต่อไปนี้แสดง ให้เห็นในส่วนนี้:

- แอ็พพลิเคชัน `apl` ระบุโหลดเดอร์โดเมน `domain01` ที่อยู่ใน ข้อมูล **LIBPATH** ของตน จากนั้นแอ็พพลิเคชัน `apl` จะถูกเรียกใช้งาน ไดเร็กทอรีการทำงานปัจจุบันคือ `/home/user1` และมีไฟล์ปกติ `domain1` ที่สามารถเขียนได้โดย `apl` โหลดเดอร์โดเมนใหม่ ที่เชื่อมโยงกับไฟล์ `/home/user1/domain01` จะ ถูกสร้างขึ้น `apl` ถูกเรียกใช้งานอีกครั้ง ในครั้งนี้ `/home/user2` เป็นไดเร็กทอรีการทำงานปัจจุบัน และ ยังมีไฟล์ปกติ `domain01` ที่ สามารถเขียนได้โดย `apl` โหลดเดอร์โดเมนใหม่ ที่เชื่อมโยงกับไฟล์ `/home/user1/domain02` จะ ถูกสร้างขึ้น
- แอ็พพลิเคชัน `apl` ระบุโหลดเดอร์โดเมน `/etc/1_domain/domain01` ที่อยู่ใน ข้อมูล **LIBPATH** ของตน จากนั้น `apl` จะถูกเรียกใช้งาน `/etc/1_domain/domain01` คือ ไฟล์ปกติที่สามารถเขียนได้โดย `apl` โหลดเดอร์โดเมนใหม่ที่เชื่อมโยงกับไฟล์ `/etc/1_domain/domain01` ถูกสร้างขึ้น

  `/home/user1/my_domain` เป็นลิงก์สัญลักษณ์ไปยังไฟล์ `/etc/1_domain/domain01`

  แอ็พพลิเคชัน `ap2` มีโหลดเดอร์โดเมน `/home/user1/my_domain` ที่ระบุในข้อมูล **LIBPATH** ของตน จากนั้น `ap2` จะถูกเรียกใช้งาน โหลดเดอร์ระบบพบว่า `/home/user1/my_domain` อ้างถึงไฟล์เดียวกันกับ `/etc/1_domain/domain01` โหลดเดอร์โดเมนถูกเชื่อมโยงกับไฟล์ `/etc/1_domain/domain01` แล้วเรียบร้อย ดังนั้นโหลดเดอร์โดเมนนี้ จะถูกใช้โดยแอ็พพลิเคชัน `ap2`

- แอ็พพลิเคชัน apl ระบุโหลดเดอร์โดเมน /etc/1_domain/domain01 ที่อยู่ใน ข้อมูล **LIBPATH** ของตน จากนั้น apl จะถูกเรียกใช้งาน /etc/1_domain/domain01 คือ ไฟล์ปกติที่สามารถเขียนได้โดย apl โหลดเดอร์โดเมนใหม่ที่เชื่อมโยงกับไฟล์ /etc/1_domain/domain01 ถูกสร้างขึ้น

  ไฟล์ /etc/1_domain/domain01 ถูกลบและสร้างใหม่เป็นไฟล์ปกติ

  แอ็พพลิเคชัน apl ถูกเรียกใช้งานอีกครั้ง ไม่มีวิธีอื่นใดอีก ที่จะเข้าถึงไฟล์ปกติที่เชื่อมโยงกับโหลดเดอร์โดเมน /etc/1_domain/domain01 ต้นฉบับ ดังนั้น โหลดเดอร์โดเมนใหม่ที่เชื่อมโยงกับไฟล์ /etc/1_domain/domain01 จะถูกสร้าง

โหลดเดอร์โดเมนเป็นโครงสร้างไดนามิก ระหว่างช่วงอายุ ของโหลดเดอร์โดเมน ไลบรารีที่แบ่งใช้จะถูกเพิ่มและถูกลบออก ไลบรารีทีแบ่งใช้ ถูกเพิ่มในโหลดเดอร์โดเมนเมื่อการประมวลผลที่ระบุโหลดเดอร์โดเมน จำเป็นต้องใช้ไลบรารีที่แบ่งใช้ที่ยังไม่มีอยู่ในโดเมน แน่นอนที่ว่า นี้จะถือว่าการประมวลผลมีสิทธิพิเศษเพียงพอที่จะเพิ่มไลบรารีที่แบ่งใช้ ในโหลดเดอร์โดเมน

จำนวนนับการใช้แยกถูกเก็บไว้สำหรับแต่ละไลบรารีที่แบ่งใช้ ที่เป็นสมาชิกของโหลดเดอร์โดเมน จำนวนนับการใช้นี้จะเก็บค่าการติดตามว่ามีจำนวน กาโหลดเดอร์ประมวลผลที่มีโหลดเดอร์โดเมนเท่าไหร่ที่กำลังใช้ไลบรารีที่แบ่งใช้ เมื่อค่าจำนวนนับการใช้นี้ลดลงเหลือศูนย์ ไลบรารีที่แบ่งใช้จะถูกลบออกจากโหลดเดอร์โดเมน

Data management application programming interface

AIX จัดให้มี data management application programming interface (DMAPI) ซึ่ง เป็นการประยุกต์ใช้ของ ″System Management: Data Storage Management (XDSM) API″ มาตรฐาน X/Open ที่เผยแพร่โดย The Open Group

DMAPI อนุญาตให้คู่ค้าซอฟต์แวร์พัฒนาแอ็พพลิเคชัน การจัดการข้อมูลโดยใช้ชุดของฟังก์ชันและซีแมนทิกส์ที่ไม่ มีในระบบ POSIX-compliant ไม่มีการจัดเตรียมฟังก์ชันโดยตรง แก่ผู้ใช้ขั้นปลาย เอกสารสมบูรณ์ของ DMAPI อยู่ที่ส่วน Publications ของเว็บไซต์ The Open Group

DMAPI ที่จัดเตรียมโดย AIX เป็นการนำมาใช้ในแบบอเนกประสงค์ ระดับการสนับสนุนสำหรับอินเตอร์เฟสเป็นทางเลือกและการทำงาน ถูกกำหนดโดยการใช้ระบบไฟล์และถูกจัดทำเอกสาร ในส่วนแยกสำหรับระบบไฟล์เจาะจง

จุดประสงค์ของ DMAPI คือสนับสนุนผลิตภัณฑ์เดี่ยวบน ระบบไฟล์เดี่ยว DMAPI ไม่ห้ามในการที่ผลิตภัณฑ์ต่างกันจากผู้ค้าต่างกัน จะทำงานบนระบบไฟล์เดียวกัน แต่ไม่แนะนำ ผลิตภัณฑ์ต่างกันบนระบบไฟล์ต่างกันได้รับการสนับสนุนเต็มรูปแบบโดย DMAPI โดยพิจารณาถึงการจัดส่งเหตุการณ์ ภายใต้ข้อบังคับ ดังต่อไปนี้:

- เซสชันหลายเซสชันไม่สามารถรีจิสเตอร์การควบคุมเหตุการณ์เดียวกัน บนอ็อบเจ็กต์เดียวกัน
- ข้อความเหตุการณ์ถูกกำหนดเป้าหมายและจัดลำดับ ที่เซสชัน; ไม่มี การกำหนดเป้าหมายที่ชุดเจนของเหตุการณ์กับจำเพาะ กระบวนการ
- ถ้าไม่มีการรีจิสเตอร์เซสชันการควบคุมเหตุการณ์ ที่ไม่ใช่เหตุการณ์เชื่อมต่อ DMAPI จะไม่ สร้าง เหตุการณ์และอนุญาตให้กระบวนการดำเนินต่อไปเหมือนกับไม่มีการเปิดใช้งานเหตุการณ์ ถ้าไม่มีเซสชันถูกรีจิสเตอร์ การควบคุมสำหรับเหตุการณ์เชื่อมต่อ ซึ่งถูกเปิดใช้งานเสมอ DMAPI จะทำให้เหตุการณ์ล้มเหลวและไม่ อนุญาตให้ระบบไฟล์ถูกเชื่อมต่อ

DMAPI ถูกนำมาใช้ใน abstract เลเยอร์ภายใน AIX, อนุญาตให้ระบบไฟล์ทำการกำหนดแต่ละระดับ ของการสนับสนุนและการนำตัวเลือกมาใช้ ระบบไฟล์ journaled file system (JFS) ไม่มีการสนับสนุน DMAPI enhanced journaled file system (JFS2) behaviors สำหรับอ็อพชันการนำไปใช้งาน ข้อจำกัด และข้อมูลจำเพาะอื่น ที่อธิบายโดยมาตรฐาน X/Open และมีการสรุปไว้ใน สิ่งที่ต้องพิจารณาเกี่ยวกับ DMAPI สำหรับ Enhanced Journaled File System

ฟังก์ชัน **dm_init_service** ส่งกลับ 0 เมื่อ AIX DMAPI กำหนดค่าเริ่มต้นถูกต้อง และ -1 ถ้าการกำหนดค่าเริ่มต้นล้มเหลว การใช้ฟังก์ชัน DMAPI อื่นหลังจากการกำหนดค่าเริ่มต้นล้มเหลว จะล้มเหลวเช่นกัน

AIX DMAPI ไม่สนันสนุนฟังก์ชัน DMAPI ทางเลือกดังต่อไปนี้:

- **dm_downgrade_right**
- **dm_upgrade_right**
- **dm_obj_ref_*** family
- **dm_pending**

อินเตอร์เฟสทางเลือกอื่นอาจไม่ได้รับการสนับสนุนโดยระบบไฟล์ และถูกระบุในเอกสาร DMAPI สำหรับระบบไฟล์ที่เจาะจง

เมื่อแอ็พพลิเคชัน data management (DM) ระบุว่าต้องการ บล็อกจนกว่าสิทธิจะพร้อมใช้งาน แอ็พพลิเคชัน DM จะถูกบล็อกแบบไม่สามารถขัดจังหวะได้

AIX อนุญาตการมีหลายขอบเขตที่มีการจัดการ ขอบเขตที่มีการจัดการที่คงอยู่โดยไม่ซ้อนทับกัน เฉพาะไฟล์ปกติที่ได้รับอนุญาตให้มีขอบเขตที่มีการจัดการ ขอบเขตที่มีการจัดการ จะถูกจัดลำดับใหม่หรือถูกรวมหรือไม่ขึ้นอยู่กับ การนำระบบไฟล์มาใช้

ถ้าไม่มีเซสชันถูกรีจิสเตอร์เพื่อรับเหตุการณ์สำหรับ อ็อบเจ็กต์ซึ่งถูกเปิดใช้งานและกิจกรรมเกิดขึ้นซึ่งอาจทริกเกอร์ เหตุการณ์ AIX จะไม่สร้างเหตุการณ์และอนุญาต ให้กระบวนการดำเนินต่อ เหมือนกับไม่มีการเปิดใช้งานเหตุการณ์

การดำเนินการฟังก์ชัน **dm_set_eventlist** ทำให้รายการเหตุการณ์ คงที่ถูกเก็บพร้อมกับอ็อบเจ็กต์ ถ้าก่อนหน้านี้รายการเหตุการณ์ถูก ตั้งค่าสำหรับระบบไฟล์ทั้งหมด และรายการเหตุการณ์ต่อมาสำหรับ อ็อบเจ็กต์ในระบบไฟล์นั้นรวมเหตุการณ์ที่ถูกตั้งค่าสำหรับระบบไฟล์ เหตุการณ์จะดำเนินต่อเพื่อถูกสร้างจากรายการเหตุการณ์ สำหรับระบบไฟล์ จนถึงเวลาที่เหตุการณ์นั้นถูกปิดใช้งาน ในกรณีนี้รายการเหตุการณ์สำหรับอ็อบเจ็กต์จะเข้ามามีบทบาท

เมื่อกระบวนการที่สร้างเหตุการณ์ถูกบล็อก รอการตอบสนอง จากแอ็พพลิเคชัน DM, sleep สามารถถูกขัดจังหวะได้

AIX เลือกใช้โมเดลการส่งมอบข้อความอะซิงโครนัส ที่เชื่อถือได้อย่างสมเหตุสมผล จำนวนของข้อความอะซิงโครนัสที่ไม่ได้ส่งมอบ ถูกจำกัดโดยจำนวนของหน่วยความจำที่มี (จริงหรือเสมือน) ที่กำหนดค่า บนระบบ ถ้าจำนวนของข้อความเกินจำนวนของหน่วยความจำที่มี ข้อความอะซิงโครนัสที่ไม่ได้ส่งจะหายไป การส่งมอบ อะซิงโครนัสของข้อความเหตุการณ์ namespace ขึ้นอยู่กับ การนำระบบไฟล์มาใช้

สำหรับ AIX, DM_SESSION_INFO_LEN คือ 256, และ DM_ATTR_NAME_SIZE คือ 8

สำหรับอินเตอร์เฟส DMAPI ที่ส่งกลับข้อมูลไปที่บัฟเฟอร์ผู้ใช้ และให้ค่า กับตัวแปรผู้ใช้ด้วยขนาดผลลัพะของบัฟเฟอร์ เนื้อหา ทั้งสองของบัฟเฟอร์และตัวแปรขนาดของผู้ใช้ถูกยกเลิกการกำหนด เมื่ออินเตอร์เฟสล้มเหลวด้วยข้อผิดพลาดอื่นที่ไม่ใช่ E2BIG สำหรับข้อผิดพลาดเหล่านี้ ข้อผิดพลาดของบัฟเฟอร์ผู้ใช้ต้องถูกละเว้น เมื่ออินเตอร์เฟส ล้มเหลวและ errno คือ E2BIG เนื้อหาของตัวแปรขนาดผู้ใช้จะถูก ตั้งค่าเพื่อระบุขนาดที่ต้องการ ในกรณีนี้แอ็พพลิเคชัน สามารถกำหนดค่าอินเตอร์เฟสใหม่ด้วยบัฟเฟอร์ที่ปรับขนาด

สิ่งที่ต้องพิจารณาเกี่ยวกับ DMAPI สำหรับ enhanced journaled file system

หมายเหตุ: สแนปช็อตภายในไม่สามารถถูกใช้กับระบบไฟล์ DMAPI-managed

นอกจากการทำงานที่จัดเตรียมโดยการนำ AIX มาใช้โดยทั่วไปของ DMAPI การนำ JFS2 มาใช้จัดเตรียม การทำงานและข้อจำกัดดังต่อไปนี้

ฟังก์ชัน **dm_get_config** ส่งกลับค่าดังต่อไปนี้ สำหรับตัวเลือกและข้อจำกัด การนำ JFS2 มาใช้:

DM_CONFIG_BULKALL
: สนับสนุน

DM_CONFIG_LEGACY
: สนับสนุน

DM_CONFIG_PERS_ATTRIBUTES
: สนับสนุน

DM_CONFIG_PERS_EVENTS
: สนับสนุน

DM_CONFIG_PERS_INHERIT_ATTRIBS
: สนับสนุน

DM_CONFIG_PERS_MANAGED_REGIONS
: สนับสนุน

DM_CONFIG_PUNCH_HOLE
: สนับสนุน

DM_CONFIG_WILL_RETRY
: สนับสนุน

DM_CONFIG_CREATE_BY_HANDLE
: ไม่สนับสนุน

DM_CONFIG_LOCK_UPGRADE
: ไม่สนับสนุน

DM_CONFIG_OBJ_REF
: ไม่สนับสนุน

DM_CONFIG_PENDING
: ไม่สนับสนุน

DM_CONFIG_DTIME_OVERLOAD
: TRUE

DM_CONFIG_MAX_ATTR_ON_DESTROY
: 128

DM_CONFIG_MAX_ATTRIBUTE_SIZE
: 4072

DM_CONFIG_MAX_HANDLE_SIZE
: 32

DM_CONFIG_MAX_MANAGED_REGIONS
    167

DM_CONFIG_MAX_MESSAGE_DATA
    65536

DM_CONFIG_TOTAL_ATTRIBUTE_SPACE
    4072

ในการนำ JFS2 มาใช้ ค่าแอ็ตทริบิวต์ DM ทั้งหมดแบ่งใช้การจัดสรร เหมือนกัน ดังนั้น ขนาดของค่าแอ็ตทริบิวต์จะไม่เกิน DM_CONFIG_MAX_ATTRIBUTE_SIZE และมีข้อจำกัดเพิ่มเติมโดย ผลรวมของขนาดค่าของแอ็ตทริบิวต์ DM ทั้งหมดที่สัมพันธ์กับอ็อบเจ็กต์ ซึ่งถูกจำกัดด้วย DM_CONFIG_MAX_ATTRIBUTE_SIZE เช่นกัน

นอกจากอินเตอร์เฟสทางเลือกที่ไม่สนับสนุนโดย AIX, การนำ JFS2 มาใช้ไม่สนับสนุน เหตุการณ์ยกเลิกและเปิดตัว ของ DMAPI ที่เป็นทางเลือก ไม่สนับสนุนฟังก์ชัน **dm_getall_dmattr**, **dm_create_by_handle**, and **dm_symlink_by_handle** ที่เป็นทางเลือก

เนื่องจากการนำการสนับสนุนแอ็ตทริบิวต์ส่วนขยายของ JFS2 มาใช้ในปัจจุบัน ฟังก์ชัน **dm_set_region** ทำให้ ctime ของไฟล์ถูกปรับเปลี่ยน JFS2 ไม่พยายามที่จะจัดลำดับใหม่หรือรวมขอบเขต ที่มีการจัดการ

JFS2 สร้างข้อความเหตุการณ์อะซิงโครนัส namespace สำหรับ การดำเนินการทั้งหมดที่เกี่ยวข้อง ไม่ว่าสำเร็จหรือล้มเหลว

JFS2 จัดเตรียมอินเตอร์เฟสที่อนุญาตการจัดสรรก่อนและการควบคุมโดยตรง กับ metadata ภายในระบบไฟล์ การใช้อินเตอร์เฟสเหล่านี้กับ โหมด MM_ALLOC หรือ MM_RECORD สร้างเหตุการณ์เขียน DMAPI สำหรับออฟเซ็ตและความยาวที่ระบุ

ถ้าค่าไม่ได้ถูกระบุสำหรับมาสก์ให้กับฟังก์ชัน **dm_get_bulkall**, **dm_get_bulkattr**, **dm_get_dirattrs**, และ **dm_get_fileattr** (นั้นคือถูกตั้งค่าเป็นศูนย์) JFS2 จะส่งกลับฟิลด์ทั้งหมดในโครงสร้าง **dm_stat** ถ้า มาสก์ถูกตั้งค่าให้กับค่าเฉพาะ มีเพียงฟิลด์ที่ร้องขอโดย มาสก์ที่จะถูกส่งกลับ ค่าสำหรับฟิลด์ที่ไม่ได้ระบุโดย มาสก์จะเป็น undefined

JFS2 ไม่ใช้พารามิเตอร์ **respbufp** ของฟังก์ชัน **dm_respond_event** ถ้ามีการระบุ เนื้อหาของบัฟเฟอร์จะเป็น undefined เมื่อฟังก์ชัน ส่งกลับ

เนื่องจาก JFS2 โอเวอร์โหลด dm_ctime และ dm_dtime (นั้นคือ DM_CONFIG_DTIME_OVERLOAD เป็น true) พารามิเตอร์ **setdtime** ของฟังก์ชัน **dm_set_dmattr** จะถูกละเว้น

เมื่อไฟล์เป็นหน่วยความจำที่แม็พ (นั่นคือ เมื่อการเรียก **mmap(2)** ถูกดำเนินการ) ส่วนที่ไม่ใช่เรซิเดนต์ของไฟล์ต้องถูกทำให้เป็นเรซิเดนต์ โดยแอ็พพลิเคชัน DM เพื่อแจ้งแอ็พพลิเคชันถึงการแม็พ JFS2 จะสร้างเหตุการณ์ อ่านหรือเขียน ตรงกับโหมดและ ขอบเขตที่ถูกแม็พ

การเปิดใช้งาน DMAPI บนระบบไฟล์ JFS2

เพื่อเปิดใช้งาน DMAPI บนระบบไฟล์ JFS2 ให้พิมพ์ดังต่อไปนี้:

```
chfs -a managed=yes mountpoint
```

ถ้าระบบถูกเชื่อมต่อในขณะนี้เมื่อคำสั่ง **chfs** ถูกส่ง จะต้องมีแอ็พพลิเคชันที่เปิดใช้งาน DMAPI รอรับข้อมูลอยู่ และตอบสนองกับเหตุการณ์เชื่อมต่อเมื่อพารามิเตอร์ *managed* ถูกตั้งค่า ความสำเร็จของคำสั่ง **chfs** จะขึ้นอยู่กับวิธี ที่แอ็พพลิเคชันตอบสนองกับเหตุการณ์เชื่อมต่อ

เพื่อปิดใช้งาน DMAPI บนระบบไฟล์ JFS2 ให้พิมพ์ดังต่อไปนี้:

```
chfs -a managed=no mountpoint
```

ถ้าระบบถูกเชื่อมต่อในขณะนี้เมื่อคำสั่ง **chfs** ถูกส่ง จะต้องมีแอ็พพลิเคชันที่เปิดใช้งาน DMAPI รอรับข้อมูลอยู่ และตอบสนองกับเหตุการณ์ pre-unmount เมื่อพารามิเตอร์ *managed* ถูกตั้งค่า ความสำเร็จของคำสั่ง **chfs** จะขึ้นอยู่กับวิธี ที่แอ็พพลิเคชันตอบสนองกับเหตุการณ์ pre-unmount

การใช้ DMAPI บนระบบไฟล์ที่เข้ารหัส JFS2

เมื่อคุณดำเนินการ I/O ที่มองไม่เห็นในไฟล์ที่เข้ารหัสใน ระบบไฟล์ที่เข้ารหัส JFS2 ข้อจำกัดการจัดเรียงออฟเซ็ตและความยาว เดียวกันจะถูกนำมาใช้ เหมือนกับที่คุณทำกับ RAW mode I/O บนไฟล์โดย เฉพาะ ออฟเซ็ตและความยาวของ I/O ต้องถูก block-aligned ตามขนาดบล็อกของระบบไฟล์ ขนานของข้อมูลที่เข้ารหัส เป็นผลคูณของบล็อกระบบไฟล์เสมอ แม้เมื่อ ถูกถอดรหัสจะไม่เท่ากัน เมื่อขนาดไฟล์ไม่ได้ถูกจัดบล็อก ไฟล์ มีข้อมูลที่เข้ารหัสมากกว่าขนาดไฟล์

หมายเหตุ: รูทีนย่อย **stat** และอินเตอร์เฟส DMAPI เช่นฟังก์ชัน **dm_get_fileattr**, รายงานขนาดไฟล์ (ถูกถอดรหัส) ข้อความที่ชัดเจน ขณะที่รูทีนย่อย **statx** รายงานขนาดข้อมูลเข้ารหัส block-aligned เมื่อคุณส่งผ่าน **STX_EFSRAW** เป็นพารามิเตอร์ *command*

ฟังก์ชัน **dm_read_invis** และฟังก์ชัน **dm_write_invis** ต้องเป็นไปตาม ข้อกำหนดเพื่อทำให้การดำเนินการสำเร็จ:

dm_read_invis
สำหรับไฟล์ที่เข้ารหัส ทั้งพารามิเตอร์ *off* และ *len* ต้องเป็นขนาดบล็อกระบบไฟล์ ที่มีการจัดเรียง หรือไม่แล้วการดำเนินการจะล้มเหลวโดยมี โค้ดระบุความผิดพลาด **EINVAL**

dm_write_invis
สำหรับไฟล์ที่เข้ารหัส ทั้งพารามิเตอร์ *off* และ *len* ต้องเป็นขนาดบล็อกระบบไฟล์ ที่มีการจัดเรียงและการดำเนินการต้องไม่พยายามขยายไฟล์ หรือไม่แล้วการดำเนินการจะล้มเหลวโดยมีโค้ดระบุความผิดพลาด**EINVAL**

การใช้ DMAPI บนเวิร์กโหลดพาร์ติชัน AIX

คุณต้องเพิ่มสิทธิ์ **PV_FS_DMAPI** ให้กับชุดของสิทธิ์และกำหนดให้กับกระบวนการที่กำลังรันใน Workload Partition (WPAR) เพื่อรันแอ็พพลิเคชัน DMAPI ภายใน WPAR คุณสามารถเพิ่มและกำหนดชุดของสิทธิ์ให้กับ WPAR เมื่อ WPAR ถูกสร้างขึ้น หรือคุณแก้ไขชุดของสิทธิ์ภายหลัง

ตัวอย่าง

```
mkwpar -S privs+=PV_FS_DMAPI -n wparname
```

```
chwpar -S privs+=PV_FS_DMAPI wparname
```

โดยดีฟอลต์ เฉพาะกระบวนการ root จะได้รับสิทธิ์เพื่อรันใน WPAR ในระบบที่ปิดใช้งาน root หรือในการติดตั้ง Trusted AIX โดย root ถูกปิดใช้งานโดยดีฟอลต์ กระบวนการที่ไม่ใช่ root ได้รับสิทธิ์นี้โดยใช้ตาราง **privcmds** ในระบบโกลบอลหรือ WPAR สำหรับข้อมูลเพิ่มเติม โปรดดูที่ สิทธิ์ RBAC

การเขียนโปรแกรมหน่วยความจำสำหรับธุรกรรม AIX

หน่วยความจำสำหรับธุรกรรม (TM) เป็นโครงสร้างการซิงโครไนซ์หน่วยความจำ ที่แบ่งใช้ซึ่งช่วยให้เธรดกระบวนการสามารถทำการดำเนินการหน่วยเก็บข้อมูล ที่ละเอียดมากเมื่อเทียบกับเธรดกระบวนการหรือแอ็พพลิเคชันอื่น

ภาพรวม

TM เป็นโครงสร้างที่ช่วยให้สามารถดำเนินการ กับส่วนที่วิกฤตซึ่งใช้ล็อคของโค้ดได้โดยไม่ต้องใช้ล็อค ตัวประมวลผล IBM POWER 8 นับเป็น ตัวประมวลผลแรกที่นำการเขียนโปรแกรม TM ไปใช้

ใช้ตัวช่วย TM ในบางสถานการณ์จำลองต่อไปนี้:

- การดำเนินการที่เหมาะสมของแอ็พพลิเคชันซึ่งใช้ล็อค – TM สนับสนุน การดำเนินการที่คาดการณ์ของส่วนที่วิกฤตของโค้ด โดยไม่ต้องใช้ ล็อค เมธอดนี้มีข้อดีในการล็อคแอ็พพลิเคชันอย่างแน่นหนา โดยใช้ล็อคปัจจุบันที่ไม่มีการปรับสำหรับประสิทธิภาพ
- การเขียนโปรแกรมสำหรับธุรกรรมในภาษาระดับสูง – โมเดล การเขียนโปรแกรมสำหรับธุรกรรมเป็นมาตรฐานของอุตสาหกรรมที่กำลังเติบโต ซึ่งให้ผลผลิตที่ดีขึ้น และสัมพันธ์กับโปรแกรมหน่วยความจำที่แบ่งใช้ซึ่งใช้ ล็อค
- การใช้งานเช็คพอยต์/การย้อนกลับ – TM มีการใช้เป็นเช็คพอยต์ เพื่อเรียกคืนสภาพทางสถาปัตยกรรม เมธอดนี้ช่วยให้การใช้คอมไพลเลอร์ เหมาะสมที่สุดระหว่างการใช้โค้ดรันไทม์ หรือการสร้างและการจำลอง เช็คพอยต์

เพื่อใช้ตัวช่วย TM, เธรดกระบวนการจะทำเครื่องหมายการเริ่มต้น และการสิ้นสุดของลำดับของการเข้าถึงหน่วยเก็บข้อมูล หรือธุรกรรมโดยใช้คำสั่ง `tbegin.` และ `tend.` คำสั่ง `tbegin.` จะเริ่มต้นการดำเนินการสำหรับ ธุรกรรม ซึ่งในระหว่างนั้น โหลดและการจัดเก็บจะเกิดขึ้นในโครงสร้าง คำสั่ง `tend.` จะสิ้นสุดการดำเนินการสำหรับธุรกรรม

ถ้า ธุรกรรมถูกหยุดก่อนกำหนด การอัพเดตหน่วยเก็บข้อมูลที่ ทำหลังจากการดำเนินการคำสั่ง `tbegin.` จะถูก ย้อนกลับ นอกจากนี้ เนื้อหาของชุดย่อยของรีจิสเตอร์ ยังถูกย้อนกลับเป็นสภาพก่อนที่จะดำเนินการคำสั่ง `tbegin.` ด้วย เมื่อธุรกรรมถูกหยุดก่อนกำหนด จะมีการเริ่มต้น ตัวจัดการความล้มเหลวของซอฟต์แวร์ ความล้มเหลวสามารถเป็นแบบชั่วคราว หรือแบบถาวร ตัวจัดการความล้มเหลวสามารถลองธุรกรรมอีกครั้ง หรือเลือกที่จะใช้โครงสร้างการล็อคหรือพาธตรรกะอื่น โดย ขึ้นอยู่กับลักษณะของความล้มเหลว

ระบบปฏิบัติการ AIX สนับสนุนการใช้งาน TM รวมถึงการจัดการของการจัดการสถานะ TM เมื่อสลับบริบท และอินเตอร์รัปต์

สถานะเช็คพอยต์

เมื่อเริ่มต้นธุรกรรม ชุดของรีจิสเตอร์จะมีการบันทึกไว้ ซึ่งแสดงถึงสถานะเช็คพอยต์ของ ตัวประมวลผล ในกรณีที่ธุรกรรมล้มเหลว ชุดของรีจิสเตอร์ จะถูกเรียกคืนไปยังจุดก่อนหน้าการเริ่มต้นธุรกรรม สถานะ เช็คพอยต์ของตัวประมวลผลยังมีการเรียกอีกอย่างว่า สถานะก่อนหน้า ธุรกรรม สถานะเช็คพอยต์มีรีจิสเตอร์ที่เขียนได้และระบุปัญหา ยกเว้นสำหรับรีจิสเตอร์ `CR0`, `FXCC`, `EBBHR`, `EBBRR`, `BESCR`, รีจิสเตอร์การมอนิเตอร์ประสิทธิภาพ และ TM SPRs

หมายเหตุ: อย่างไรก็ตาม ไม่สามารถเข้าถึงสถานะเช็คพอยต์โดยตรงผ่านสถานะซุปเปอร์ไวเซอร์ หรือสถานะปัญหา สถานะเช็คพอยต์ถูกคัดลอกไปใน รีจิสเตอร์ตามลำดับหลังจากดำเนินการคำสั่ง `treclaim.` ใหม่ กระบวนการนี้ช่วยให้โค้ดที่มีสิทธิพิเศษสามารถบันทึกหรือปรับเปลี่ยน ค่าได้ สถานะเช็คพอยต์ถูกคัดลอกกลับไปในรีจิสเตอร์ที่เสี่ยง จากรีจิสเตอร์ซึ่งผู้ใช้สามารถเข้าถึงได้ตามลำดับ หลังจาก การดำเนินการคำสั่ง `trechkpt.` ใหม่

มีการเพิ่ม TM SPRs ต่อไปนี้ลงในสถานะเครื่องของตัวประมวลผล:

ชื่อ	หัวเรื่อง	คำอธิบาย	mtspr ที่มีสิทธิพิเศษ	mfspr ที่มีสิทธิพิเศษ	ขนาด (บิต)	SPR
FSCR	Facility Status and Control Register (สถานะตัวช่วยและรีจิสเตอร์ตัวควบคุม)	ควบคุมตัวช่วยที่พร้อมใช้งานในสถานะ ปัญหา และบ่งชี้สาเหตุของอินเตอร์รัปต์ซึ่งทำให้ตัวช่วยไม่พร้อมใช้งาน	ใช่	ใช่	64	153
TEXASR	Transaction Exception And Summary Register (ข้อยกเว้นธุรกรรมและรีจิสเตอร์สรุป)	มีข้อมูลระดับธุรกรรมและสรุป ที่ใช้โดยตัวจัดการความล้มเหลวของธุรกรรม บิต 0:31 มี สาเหตุของความล้มเหลว	ไม่	ไม่	64	130
TFHAR	Transaction Failure Handler Address Register (รีจิสเตอร์แอดเดรสของตัวจัดการความล้มเหลวของธุรกรรม)	บันทึก EA ของตัวจัดการความล้มเหลวของซอฟต์แวร์ รีจิสเตอร์ TFHAR มีการตั้งค่าเป็น NIA เสมอสำหรับคำสั่ง `tbegin.` ที่เริ่มต้นธุรกรรม	ไม่	no	64	128
TFIAR	Transaction Failure Instruction Address Register (รีจิสเตอร์แอดเดรสของคำสั่งความล้มเหลวของธุรกรรม)	ตั้งค่าเป็น EA ที่แน่นอนของคำสั่งซึ่ง ทำให้เกิดความล้มเหลว เมื่อเป็นไปได้ ความถูกต้องของรีจิสเตอร์ TFIAR มีการบันทึกไว้ในฟิลด์ Exact (บิต 37) ของรีจิสเตอร์ TEXASR	ไม่	no	64	129
TEXASRU	Transaction Exception and Summary Register (ข้อยกเว้นธุรกรรมและรีจิสเตอร์สรุป) (ครึ่งด้านบน)	ครึ่งลำดับด้านบนของรีจิสเตอร์ TEXASR	no	ไม่	32	131

รีจิสเตอร์ TEXASR ใหม่มีข้อมูลที่เกี่ยวข้อง กับสถานะของธุรกรรม และสาเหตุของความล้มเหลวของธุรกรรม ตารางต่อไปนี้อธิบายฟิลด์ที่มีอยู่ในรีจิสเตอร์ TEXASR:

ชื่อ	ฟิลด์	ค่า-ความหมาย	บิต
TEXASR	โค้ดระบุความล้มเหลว (หมายเหตุ: บิต 7 มีการอ้างอิงเป็น ฟิลด์ความล้มเหลวแบบถาวร)	โค้ดระบุความล้มเหลวของธุรกรรม	0:7
	ไม่อนุญาต	0b1 – ไม่อนุญาตคำสั่งชนิดการเข้าถึง	8
	Nesting Overflow	0b1 – เกินระดับธุรกรรมสูงสุด	9
	Footprint Overflow	0b1 – เกินขีดจำกัดการติดตามสำหรับการเข้าถึง หน่วยเก็บข้อมูลสำหรับธุรกรรม	10
	ข้อขัดแย้งที่เกิดขึ้นเพราะตนเอง	0b1 – มีข้อขัดแย้งที่เกิดขึ้นเพราะตนเองเกิดขึ้นในสถานะ ระงับไว้	11
	ข้อขัดแย้งที่ไม่เกี่ยวกับธุรกรรม	0b1 – มีข้อขัดแย้งเกิดขึ้นกับการเข้าถึงที่ไม่เกี่ยวกับ ธุรกรรมโดยตัวประมวลผลอื่น	12
	ข้อขัดแย้งธุรกรรม	0b1 – มีข้อขัดแย้งเกิดขึ้นกับธุรกรรมอื่น	13
	ข้อขัดแย้งการทำให้ธุรกรรมเป็นโมฆะ	0b1 – มีข้อขัดแย้งเกิดขึ้นกับการทำให้ TLB เป็นโมฆะ	14
	เฉพาะการนำไปใช้	0b1 – เงื่อนไขเฉพาะการนำไปใช้เป็นสาเหตุให้ ธุรกรรมล้มเหลว	15
	ข้อขัดแย้งการนำคำสั่งไปใช้	0b1 – การนำคำสั่งไปใช้โดยเธรดหรือเธรดอื่น ซึ่งทำจากบล็อกที่มีการเขียนธุรกรรม ก่อนหน้านี้	16
	สำรองไว้สำหรับกรณีความล้มเหลวในอนาคต		17:30
	ยกเลิก	0b1 – ยกเลิกเนื่องจากการดำเนินการของคำสั่ง TM เฉพาะ	31
	ระงับไว้	0b1 – ความล้มเหลวถูกบันทึกไว้ในสถานะ ระงับไว้	32
	สำรองไว้		33
	สิทธิพิเศษ	เธรดอยู่ในสถานะสิทธิพิเศษ (`[MSRHV||PR]`) ในเวลาที่บันทึกความล้มเหลว	34:35
	สรุปความล้มเหลว (FS)	0b1 – ตรวจพบและบันทึกความล้มเหลวไว้	36
	TFIAR Exact	0b0 – ค่าในฟิลด์ TFIAR เป็น ค่าโดยประมาณ 0b1 – ค่าในฟิลด์ TFIAR เป็น ค่าที่แน่นอน	37
	ROT	ตั้งค่าเป็น 0b0 เมื่อเรียกทำงานคำสั่ง `tbegin.` ที่ไม่ใช่ ROT ใหม่ ตั้งค่าเป็น 0b1 เมื่อเริ่มต้น ROT	38
	สำรองไว้		39:51
	ระดับธุรกรรม (TL)	ระดับธุรกรรม (ความลึกในการซ้อน + 1) สำหรับ ธุรกรรมที่แอ็คทีฟมีค่าดังต่อไปนี้: • 0 ถ้าธุรกรรมล่าสุดเสร็จเรียบร้อยแล้ว • ระดับธุรกรรมซึ่งธุรกรรมล่าสุดล้มเหลว ถ้าธุรกรรมไม่เสร็จสมบูรณ์ **หมายเหตุ:** ค่า 1 สอดคล้องกับ *ธุรกรรมด้านนอก* ค่าที่มากกว่า 1 สอดคล้องกับ *ธุรกรรมที่ซ้อน*	52:63

หมายเหตุ:

- บิต 1 พอดีของบิต 8–31 ของรีจิสเตอร์ TEXASR มีการตั้งค่าเมื่อบันทึกความล้มเหลวของธุรกรรม บิตเดียวที่ ตั้งค่าบ่งชี้ว่า คำสั่งหรือเหตุการณ์เฉพาะเป็นสาเหตุให้เกิดความล้มเหลว
- Rollback Only Transaction (ROT) คือลำดับของคำสั่ง ที่มีการดำเนินการเป็นหนึ่งยูนิต หรือไม่ได้ดำเนินการคำสั่ง โครงสร้างนี้ช่วยให้สามารถดำเนินการคำสั่งจำนวนมากที่เสี่ยง ด้วยต้นทุนน้อยที่สุด ROT ไม่มีลักษณะความละเอียดเต็มที่ เหมือนกับ ธุรกรรมปกติ หรือคุณสมบัติการซิงโครไนซ์และซีเรียลไลซ์ ดังนั้น ต้องไม่ใช้ ROTs เพื่อจัดวางข้อมูลที่แบ่งใช้

ตัวจัดการความล้มเหลวของซอฟต์แวร์

เมื่อธุรกรรม ล้มเหลว ฮาร์ดแวร์เครื่องจะเปลี่ยนทิศทางการควบคุมไปยังตัวจัดการความล้มเหลว ที่เชื่อมโยงกับธุรกรรมด้านนอกสุด เมื่อธุรกรรม ล้มเหลว การควบคุมจะมีการเปลี่ยนทิศทางไปยังคำสั่งที่ตามหลังคำสั่ง tbegin., CR0 มีการตั้งค่าเป็น

`0b101 || 0`

หรือ

`0b010 || 0`

ด้วยเหตุนี้ คำสั่งหลังจากคำสั่ง tbegin. จึงต้อง เป็นคำสั่งสาขาที่เพรดิเคตบนบิต 2 ของ CR0 ตัวอย่างเช่น หลังจาก ดำเนินการคำสั่ง tbegin. แล้ว คำสั่งสาขา beq มีการเพรดิเคตบนบิต 2 ของ CR0 ปลายทางของสาขาต้อง เป็นส่วนของโค้ดที่จะจัดการกับความล้มเหลวของธุรกรรม เมื่อรันคำสั่ง tbegin. เสร็จ เรียบร้อยแล้วเมื่อเริ่มต้นธุรกรรม, CR0 มีการตั้งค่า เป็น

`0b000 || 0 or 0b010 || 0`

หมายเหตุ: บิต 0:31 ของ TEXASR จะรายงานสาเหตุของความล้มเหลว ฟิลด์โค้ดระบุความล้มเหลว (FC) ในบิต 0–7 ใช้สำหรับสถานการณ์จำลอง ต่อไปนี้:

- โค้ดซุปเปอร์ไวเซอร์หรือไฮเปอร์ไวเซอร์ที่มีสิทธิพิเศษเป็นสาเหตุให้เกิดความล้มเหลว โดย การใช้คำสั่ง treclaim. ใหม่
- โค้ดสถานะของปัญหาเป็นสาเหตุทำให้เกิดความล้มเหลวโดยใช้แบบฟอร์มของ tabort. ใหม่

ค่า 1 ในบิต 7 ของ TEXASR บ่งชี้ว่า ความล้มเหลวเป็นแบบถาวร และธุรกรรมถูกโยงกับความล้มเหลวเมื่อ พยายามทำธุรกรรมอีกครั้ง โค้ดระบุความล้มเหลวที่สำรองไว้โดย ระบบปฏิบัติการ AIX บ่งชี้ สาเหตุของความล้มเหลวที่มีการกำหนดไว้ในไดเร็กทอรี /usr/include/sys/machine.h.

ธุรกรรมตัวอย่าง

ตัวอย่างโค้ด แอสเซมเบลอร์ต่อไปนี้แสดงธุรกรรมแบบง่าย ที่จะเขียนค่าใน GPR 5 ลงในแอดเดรสใน GPR 4 ซึ่งสมมติว่ามีการแบ่งใช้ระหว่างหลาย เธรดของการดำเนินการ ถ้าธุรกรรมล้มเหลวเนื่องจากสาเหตุแบบ ถาวร โค้ดจะย้อนกลับไปยังโค้ดพาธอื่นที่เลเบล lock_based_update ไม่มีการแสดงโค้ดสำหรับพาธอื่น

```
trans_entry:
  tbegin                          # Start transaction
  beq           failure_hdlr      # Handle transaction failure
  stw           r5, 0(r4)         # Write to memory pointed to by r4.
  tend.                           # End transaction
  b             trans_exit
failure_hdlr:                     # Handle transaction failures:
  mfspr         r4, TEXASRU       # Read high-order half of TEXASR
  andis.        r5, r4, 0x0100    # Is the failure persistent?
```

```
  bne          lock_based_update     # If persistent, acquire lock and
                                     # then perform the write.
  b            trans_entry           # If transient, try again.

lock_based_update:

trans_exit:
```

การกำหนดรันไทม์ของความสามารถ Transactional Memory

โปรแกรม สามารถกำหนดว่าระบบสนับสนุนหมวดหมู่ TM ของ POWER ISA หรือไม่โดยการอ่านตัวแปรระบบ SC_TM_VER โดยใช้รูทีนย่อย getsystemcfg แมโคร __power_tm() มีการจัดเตรียมไว้ในไฟล์ /usr/include/sys/systemcfg.h เพื่อกำหนดความสามารถ TM ภายในโปรแกรม แมโครนี้มีประโยชน์ สำหรับซอฟต์แวร์ที่ใช้ความสามารถ TM แบบมีเงื่อนไขเมื่อมีความสามารถ ดังกล่าว หรือใช้ฟังก์ชันที่เท่าเทียมกับโค้ดพาธซึ่งใช้ล็อค เมื่อไม่มีความสามารถ TM

โครงสร้างบริบทแบบขยาย

เวอร์ชันก่อนหน้า ของระบบปฏิบัติการ AIX แนะนำการสนับสนุนโครงสร้างบริบทแบบขยาย เพื่อสนับสนุน สถานะเวกเตอร์และคีย์ผู้ใช้ การสนับสนุนโครงสร้างบริบทแบบขยาย ที่มีอยู่มีการขยายเพิ่มขึ้นไปอีก เพื่อสนับสนุนสถานะเครื่องที่ต้องการโดย TM

บริบทแบบขยายมีการจัดสรรและพินสำหรับ เธรดกระบวนการด้านธุรกรรมแต่ละรายการ เมื่อเธรดใช้ TM เป็นครั้งแรก หากไม่สามารถจัดสรรและพิน พื้นที่บริบทแบบขยาย กระบวนการจะได้รับสัญญาณ SIGSEGV ที่ส่งผลให้เกิดการยุติ กระบวนการ

ข้อมูลบริบทเครื่องมีการรวมไว้ในโครงสร้าง sigcontext ที่จัดเตรียมให้กับตัวจัดการสัญญาณ เมื่อตัวจัดการสัญญาณกลับมา บริบทเครื่องที่มีอยู่ในโครงสร้าง sigcontext จะถูกเรียกใช้โดยแท้จริงแล้ว โครงสร้าง sigcontext คือ ชุดย่อยของโครงสร้าง ucontext ที่ใหญ่กว่า โครงสร้าง ทั้งสองเหมือนกันกับ sizeof(struct sigcontext) เมื่อระบบปฏิบัติการ AIX สร้างบริบทสัญญาณที่จะส่งผ่านไปยังตัวจัดการสัญญาณ โครงสร้าง ucontext จะมีการสร้างขึ้นบนสแต็กของตัวจัดการสัญญาณ ส่วนบริบทเครื่อง ของบริบทสัญญาณต้องมีสถานะเครื่องที่แอ็คทีฟทั้งหมด รวมถึงสถานะที่ลบเลือนได้และไม่ลบเลือน สำหรับบริบทที่ถูกอินเตอร์รัปต์โดยไม่ได้ตั้งใจ โครงสร้าง ucontext มี ตัวบ่งชี้เพื่อกำหนดว่าข้อมูลบริบทแบบขยาย พร้อมใช้งานหรือไม่

ฟิลด์ __extctx ในโครงสร้าง ucontext คือแอดเดรสของโครงสร้างบริบทแบบขยายที่กำหนดไว้ในไฟล์ /usr/include/sys/context.h ฟิลด์ __extctx_magic ในโครงสร้าง ucontext บ่งชี้ว่าข้อมูลบริบทแบบขยายถูกต้องหรือไม่ เมื่อ ค่าของฟิลด์ __extctx_magic เท่ากับ __EXTCTX_MAGIC สถานะเครื่องเพิ่มเติมสำหรับเธรดที่ใช้ความสามารถ TM มีการเรียกคืนและบันทึกไว้เป็นสมาชิกของส่วนขยายบริบท ในโครงสร้าง ucontext ซึ่งเป็นส่วนหนึ่งของการส่งสัญญาณและการส่งคืน

ถ้าแอ็พพลิเคชัน เลือกที่จะเปิดใช้งานการใช้ Transactional Memory อย่างชัดแจ้ง แอ็พพลิเคชันจะใช้โครงสร้าง ucontext ที่ขยายขนาด ซึ่งมีพื้นที่อยู่แล้ว สำหรับฟิลด์ __extctx ที่รวมไว้โดย คำนิยามโดยปริยายของ __EXTABI__ โดยคอมไพลเลอร์ โครงสร้าง ucontext ที่ขยายยังสามารถเลือกได้โดยคำนิยามที่ชัดแจ้งของ __AIXEXTABI

รูทีนย่อย getcontext(), setcontext(), makecontext() และ swapcontext() ของ libc ไม่ได้รับการสนับสนุน ขณะอยู่ในสถานะธุรกรรมหรือระงับไว้ เมื่อมีการเรียกรูทีนย่อย ภายในธุรกรรม รูทีนย่อย getcontext(), setcontext(), makecontext() ส่งผลให้เกิดความล้มเหลวของธุรกรรมแบบถาวรชนิด TM_LIBC ซึ่งมีการกำหนดไว้ในไฟล์ /usr/include/sys/machine.h

เมื่อมีการเรียก รูทีนย่อย swapcontext() ภายในธุรกรรม รูทีนย่อยจะส่งผลให้เกิดลักษณะการทำงานต่อไปนี้:

- เมื่อรูทีนย่อย swapcontext() อยู่ในสถานะ ธุรกรรม รูทีนย่อยจะส่งผลให้เกิดความล้มเหลวของธุรกรรมแบบถาวรชนิด TM_LIBC
- เมื่อรูทีนย่อย swapcontext() อยู่ในสถานะ ระงับไว้ รูทีนย่อยจะส่งผลให้ธุรกรรมสิ้นสุดลง โครงสร้าง ucontext ที่ระบุ swapped in และการดำเนินการของโปรแกรมมีการดำเนินต่อไปโดยโครงสร้าง ucontext ที่ระบุ สถานะที่เป็นผลลัพธ์และลักษณะการทำงานในลำดับต่อมาหลังจากรูทีนย่อย swapcontext() กลับมาไม่มีการกำหนดไว้

ถ้ารูทีนย่อย getcontext(), setcontext() และ swapcontext() มีการเรียกในสถานะที่ไม่ใช่ ธุรกรรม รูทีนย่อยจะไม่ดึงข้อมูลหรือเรียกคืนบริบท TM แบบขยายใดๆ ไปในหรือจากโครงสร้าง ucontext ซึ่งชี้โดยพารามิเตอร์ ucp หรือ oucp ไม่มีการบ่งชี้ข้อผิดพลาดเมื่อรูทีนย่อย setcontext() หรือ swapcontext() มีการเรียกโดยที่มีบริบท TM แบบขยายอยู่

โปรดดูไฟล์ส่วนหัว /usr/include/sys/context.h สำหรับข้อมูลรายละเอียดของบริบทแบบขยาย

การส่ง สัญญาณ

สัญญาณอะซิงโครนัสที่ ได้รับโดยแอ็พพลิเคชันขณะอยู่ในธุรกรรมมีการส่ง ในลักษณะที่ไม่ใช่ธุรกรรม เมื่ออยู่ในสถานะธุรกรรม ไม่อนุญาตการส่ง สัญญาณซิงโครนัส และส่งผลให้เกิดความล้มเหลว ของธุรกรรมแบบถาวรชนิด TM_SYNC_SIGNAL ตามที่กำหนดไว้ในไฟล์ /usr/include/sys/machine.h

อินเตอร์รัปต์การจัดแนวและอินเตอร์รัปต์โปรแกรม

ใน สถานะธุรกรรม อินเตอร์รัปต์การจัดแนวและอินเตอร์รัปต์โปรแกรม มีสาเหตุมาจากการดำเนินการที่ไม่ถูกต้อง หรือการดำเนินการที่ต้องใช้ การเลียนแบบ ซึ่งส่งผลให้เกิดความล้มเหลวของธุรกรรมแบบถาวรชนิด TM_ALIGN_INT หรือชนิด TM_INV_OP ตามที่กำหนดไว้ในไฟล์ /usr/include/sys/machine.h เมื่ออยู่ในสถานะระงับไว้ อินเตอร์รัปต์การจัดแนวและอินเตอร์รัปต์โปรแกรมมีการประมวลผล ตามปกติ โดยใช้ความหมายที่ไม่มีความเสี่ยง

การเรียกระบบ

ขอแนะนำว่า ไม่ควรเรียกใช้ การเรียกระบบภายในธุรกรรม การเรียกระบบได้รับการสนับสนุน ภายในธุรกรรมเฉพาะถ้าธุรกรรมถูกระงับไว้ผ่าน คำสั่ง tsuspend. ใหม่

เมื่อเรียกใช้การเรียกระบบ ขณะตัวประมวลผลหรือเธรดอยู่ในสถานะธุรกรรม และธุรกรรม ไม่ได้ถูกระงับไว้ การเรียกระบบจะไม่มีการเรียกใช้โดยเคอร์เนล AIX และธุรกรรมที่เชื่อมโยงจะล้มเหลว แบบถาวร เมื่อเกิดข้อผิดพลาดนี้ ฟิลด์ FC ของรีจิสเตอร์ TEXASR มีโค้ดระบุความล้มเหลว TM_ILL_SC ซึ่งมีการกำหนดไว้ในไฟล์ /usr/sys/include/machine.h

มีการ สมมติว่า การดำเนินการใดๆ ที่ทำภายใต้ธุรกรรมซึ่งระงับไว้ เมื่อแอ็พพลิเคชันโปรแกรมเมอร์ระงับธุรกรรมไว้อย่างชัดแจ้ง ถือว่าเป็นแบบถาวรโดยตั้งใจ การดำเนินการใดๆ ที่ทำโดย การเรียกระบบซึ่งเรียกใช้ขณะอยู่ในสถานะระงับไว้ไม่มีการย้อนกลับ แม้ว่าธุรกรรมจะล้มเหลว

ระบบปฏิบัติการ AIX ไม่สนับสนุนการเรียก ระบบที่จะทำขณะอยู่ในสถานะธุรกรรม เนื่องจากไม่มีวิธีการ ย้อนกลับการดำเนินการใดๆ รวมถึง I/O ซึ่งทำโดย AIX ภายใต้ การเรียกระบบ

รูทีนย่อย setjmp() และ longjmp()

รูทีนย่อย setjmp() และ longjmp() ของ libc ไม่ได้รับการสนับสนุนในสถานะธุรกรรมหรือระงับไว้ เนื่องจาก ผลกระทบของการตั้งค่าจัมพ์บัฟเฟอร์ และการกระโดดกลับไปยังบัฟเฟอร์ โปรดพิจารณาสถานการณ์จำลองต่อไปนี้:

1. ถ้าเรียกรูทีนย่อย setjmp() ภายใน ธุรกรรม และเรียกรูทีนย่อย longjmp() ที่สอดคล้องกัน หลังจากธุรกรรมสิ้นสุด การกระโดดจะอยู่ในสถานะเสี่ยง ซึ่งไม่ถูกต้องในตอนนี้
2. ถ้าเรียกรูทีนย่อย setjmp() ก่อนหน้า ธุรกรรม รูทีนย่อย longjmp() ที่สอดคล้องกัน จะอยู่ในสถานะก่อนธุรกรรมเริ่มต้น โดยไม่คำนึงว่าธุรกรรม สิ้นสุด ล้มเหลว หรือยกเลิกแล้ว
3. ถ้าเรียกรูทีนย่อย setjmp() ภายใน ธุรกรรม จากนั้น ธุรกรรมถูกยกเลิก อัพเดตที่ ทำในจัมพ์บัฟเฟอร์โดยรูทีนย่อย setjmp() จะไม่ปรากฏว่าเกิดขึ้นแล้ว

เมื่อเรียกรูทีนย่อย setjmp() ภายในธุรกรรม จะส่งผลให้เกิดความล้มเหลวของธุรกรรมแบบถาวร ชนิด TM_LIBC หรือชนิด TM_ILL_SC ที่มีการกำหนดไว้ในไฟล์ /usr/include/sys/machine.h

เมื่อมีการเรียก รูทีนย่อย longjmp() ภายในธุรกรรม รูทีนย่อยจะส่งผลให้เกิดลักษณะการทำงานต่อไปนี้:

- เมื่อรูทีนย่อย longjmp() อยู่ในสถานะ ธุรกรรม จะส่งผลให้เกิดความล้มเหลวของธุรกรรมแบบถาวรชนิด TM_LIBC หรือชนิด TM_ILL_SC ที่มีการกำหนดไว้ในไฟล์ /usr/include/sys/machine.h
- เมื่อรูทีนย่อย longjmp() อยู่ในสถานะ ระงับไว้ จะส่งผลให้ธุรกรรมสิ้นสุดลง มีการเรียกคืนจัมพ์บัฟเฟอร์ ที่ระบุ และการดำเนินการของโปรแกรมกลับไปยังรูทีนย่อย setjmp() ซึ่งสอดคล้องกัน สถานะที่เป็นผลลัพธ์และลักษณะการทำงานในลำดับต่อมาไม่มีการกำหนดไว้ หลังจากรูทีนย่อย setjmp() กลับมาจากรูทีนย่อย longjmp()

คอมไพลเลอร์

คอมไพลเลอร์ระบบปฏิบัติการ AIX ที่สนับสนุน Transactional Memory ต้องเป็นไปตาม AIX ABI เมื่อ TM เปิดใช้งาน คอมไพลเลอร์ C หรือ C++ ต้องกำหนด __EXTABI__ ไว้ล่วงหน้า โปรดอ้างถึงเอกสารคู่มือคอมไพลเลอร์สำหรับข้อมูลโดยละเอียด

แอสเซมเบลอร์

แอสเซมเบลอร์ระบบปฏิบัติการ AIX ใน /usr/ccs/bin/as สนับสนุน ชุดคำสั่งเพิ่มเติมที่กำหนดไว้สำหรับ TM ใน POWER ISA และ นำไปใช้โดยตัวประมวลผล POWER8 คุณสามารถใช้โหมดแอสเซมเบลอร์ **-m pwr8** หรือ **.machine pwr8** pseudo op ภายในซอร์สไฟล์เพื่อเปิดใช้งานแอสเซมเบลอร์ของคำสั่ง TM สำหรับข้อมูลเพิ่มเติม โปรดอ้างอิงข้อมูลอ้างอิงภาษาชุดประกอบ

ดีบักเกอร์

ดีบักเกอร์ /usr/ccs/bin/dbx สนับสนุนการดีบักระดับเครื่องของโปรแกรม TM การสนับสนุนนี้รวมถึง ความสามารถในการถอดแยกคำสั่ง Transactional Memory ใหม่ และการดู TM SPRs: รีจิสเตอร์ TEXASR, TEXASRU, TFIAR และ TFHAR

การตั้งค่าเบรกพอยต์ภายใน ของธุรกรรมเป็นสาเหตุให้ธุรกรรมล้มเหลวแบบไม่มีเงื่อนไข ในทุกครั้งที่ พบเบรกพอยต์ ด้วยเหตุนี้ แนวทางที่แนะนำคือ การดีบักธุรกรรมที่กำลังล้มเหลวเพื่อตั้งค่าเบรกพอยต์บน ตัวจัดการความล้มเหลวของธุรกรรม จากนั้น ดูรีจิสเตอร์ TEXASR และ TFIAR เมื่อพบเบรกพอยต์ เพื่อให้ทราบสาเหตุและตำแหน่ง ของความล้มเหลว

ใน dbx, รีจิสเตอร์ TEXASR, TFIAR และ TFHAR สามารถดูได้โดยใช้คำสั่งย่อย print พร้อมกับพารามิเตอร์ $texasr, $tfiar หรือ $tfhar บรรทัดของโค้ดที่เชื่อมโยง กับแอดเดรสซึ่งอยู่ในรีจิสเตอร์ TFIAR และ TFHAR สามารถดูได้ผ่านคำสั่งย่อย list ตัวอย่างเช่น:

```
(dbx) list at $tfiar
```

คำสั่งย่อย tm_status ของ dbx ใช้เพื่อดูและตีความ เนื้อหาของรีจิสเตอร์ TEXASR คำสั่งย่อยนี้ใช้เพื่อกำหนดลักษณะของความล้มเหลวของธุรกรรม

การเปิดใช้งาน ดีบักเกอร์ของบุคคลที่สามมีการจัดเตรียมให้ในรูปแบบของการดำเนินการ PTT_READ_TM ptrace ใหม่ สำหรับการอ่านสถานะ TM ของเธรด โปรดอ้างอิง เอกสารคู่มือ ptrace สำหรับรายละเอียด

การสนับสนุนการติดตาม

ตัวช่วยการติดตาม AIX มีการขยายเพื่อรวมชุด ของเหตุการณ์การติดตามสำหรับการดำเนินการ TM ที่ทำโดยระบบปฏิบัติการ AIX ซึ่งรวมถึงการจอง ที่เป็นสาเหตุให้ธุรกรรมล้มเหลว และการดำเนินการอื่นที่สามารถ ทำให้ธุรกรรมล้มเหลวได้ ตัวระบุเหตุการณ์การติดตาม 675 สามารถใช้ เป็นอินพุตในคำสั่ง **trace** และ **trcrpt** เพื่อดูเหตุการณ์การติดตามที่เกี่ยวข้องกับ TM

ไฟล์แกน

ระบบปฏิบัติการ AIX ยังสนับสนุนการรวมสถานะเครื่อง TM ไว้เป็นส่วนหนึ่งของ ไฟล์แกนสำหรับกระบวนการและเธรดที่ใช้ TM หากตัวประมวลผล หรือเธรดกำลังใช้หรือใช้ TM ไปแล้ว สถานะเครื่อง TM จะมีการรวมไว้ในอิมเมจหลักสำหรับเธรดนั้น

หมายเหตุ: สถานะ TM ได้รับการสนับสนุน เฉพาะในรูปแบบไฟล์แกนปัจจุบันสำหรับระบบปฏิบัติการ AIX คุณสามารถใช้คำสั่ง **dbx** เพื่ออ่านและ ดูสถานะเครื่อง TM ของไฟล์แกนที่เปิดใช้งาน TM

ไลบรารีเธรด AIX

การใช้ Transactional Memory ไม่ได้รับการสนับสนุน สำหรับแอ็พพลิเคชันที่ใช้เธรด M:N ลักษณะการทำงานที่ไม่ได้กำหนดไว้อาจเกิดขึ้น ในเธรดธุรกรรมในสภาวะแวดล้อม ซึ่งมากกว่าหนึ่งเธรด แบ่งใช้เคอร์เนลเธรดเดียว การใช้ Transactional Memory โดย แอ็พพลิเคชันที่ใช้เธรด M:N อาจนำไปสู่ความล้มเหลวของธุรกรรมแบบถาวร ซึ่งมีโค้ดระบุความล้มเหลว TM_PTH_PREEMPTED ที่ กำหนดไว้ในรีจิสเตอร์ TEXASR

คำประกาศ

ข้อมูลนี้พัฒนาขึ้นสำหรับผลิตภัณฑ์และบริการที่มีในประเทศสหรัฐอเมริกาเท่านั้น

IBM อาจไม่นำเสนอผลิตภัณฑ์ เซอร์วิส หรือคุณลักษณะที่อธิบายในเอกสารนี้ในประเทศอื่น โปรดปรึกษาตัวแทน IBM ในท้องถิ่นของคุณสำหรับข้อมูลเกี่ยวกับผลิตภัณฑ์และการบริการที่มีอยู่ในพื้นที่ของคุณในปัจจุบัน การอ้างอิงใดๆ ถึงผลิตภัณฑ์ โปรแกรม หรือการบริการของ IBM ไม่ได้มีวัตถุประสงค์ที่จะระบุหรือตีความว่าสามารถใช้ได้เฉพาะผลิตภัณฑ์ โปรแกรม หรือการบริการของ IBM เพียงอย่างเดียวเท่านั้น ผลิตภัณฑ์ โปรแกรม หรือการบริการใดๆ ที่สามารถทำงานได้เท่าเทียมกัน และไม่ละเมิดสิทธิทรัพย์สินทางปัญญาของ IBM สามารถนำมาใช้แทนได้ อย่างไรก็ตาม เป็นความรับผิดชอบของผู้ใช้ ที่จะประเมินและตรวจสอบการดำเนินการของผลิตภัณฑ์ โปรแกรม หรือการบริการใดๆ ที่ไม่ใช่ของ IBM

IBM อาจมีสิทธิบัตร หรืออยู่ระหว่างดำเนินการขอสิทธิบัตร ที่ครอบคลุมถึงหัวข้อซึ่งอธิบายในเอกสารนี้ การตกแต่งเอกสารนี้ไม่ได้ให้สิทธิใช้งานใดๆ ในสิทธิบัตรเหล่านี้แก่คุณ คุณสามารถส่งการสอบถามเกี่ยวกับใบอนุญาตเป็นลายลักษณ์อักษร ไปที่:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

หากมีคำถามเกี่ยวกับข้อมูลไบต์คู่ (DBCS) โปรดติดต่อแผนกทรัพย์สินทางปัญญาของ IBM ในประเทศของคุณ หรือส่งคำถามเป็นลายลักษณ์อักษรไปที่ :

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

INTERNATIONAL BUSINESS MACHINES CORPORATION จัดเตรียมเอกสาร "ตามสภาพที่เป็น" โดยไม่มีการรับประกันใดๆ ทั้งโดยชัดแจ้งหรือโดยนัย ซึ่งรวมถึง แต่ไม่จำกัดถึงการรับประกันโดยนัยที่ไม่ละเมิดความสามารถในการจัดจำหน่าย หรือตามความเหมาะสมสำหรับวัตถุประสงค์อย่างใดอย่างหนึ่ง ในบางรัฐไม่อนุญาตให้ ปฏิเสธการรับประกันทางตรงหรือทางอ้อมในธุรกรรมบางอย่าง ดังนั้น ข้อความนี้จึงอาจจะไม่ใช้กับคุณ

ข้อมูลนี้อาจมีความไม่ถูกต้องทางเทคนิคหรือความผิดพลาด ทางการพิมพ์ การเปลี่ยนแปลงข้อมูลในนี้จะมีเป็นระยะๆ ซึ่งจะสอดคล้องกับ การตีพิมพ์ในครั้งใหม่ IBM อาจปรับปรุงและ/หรือเปลี่ยนแปลงในผลิตภัณฑ์และ/หรือโปรแกรมที่อธิบายไว้ในสิ่งพิมพ์นี้ได้ตลอดเวลาโดยไม่ต้องแจ้งให้ทราบ

การอ้างอิงใดๆ ในข้อมูลนี้ถึงเว็บไซต์ที่ไม่ใช่ของ IBM มีการนำเสนอเพื่อความสะดวกเท่านั้น และไม่ได้เป็นการสนับสนุนเว็บไซต์ดังกล่าวในลักษณะใดๆ เนื้อหาที่อยู่ในเว็บไซต์เหล่านั้นไม่ได้เป็น ส่วนหนึ่งของเนื้อหาสำหรับผลิตภัณฑ์ของ IBM นี้ และการใช้เว็บไซต์ดังกล่าวถือเป็นความเสี่ยงของคุณเอง

IBM อาจใช้หรือแจกจ่ายข้อมูลใดๆ ที่คุณ ให้ในรูปแบบต่างๆ ซึ่ง IBM เชื่อว่ามีความเหมาะสมได้โดยไม่เกิดข้อผูกมัดใดๆ กับคุณ

ผู้รับใบอนุญาตของโปรแกรมนี้ที่ต้องการได้รับข้อมูลเกี่ยวกับโปรแกรมเพื่อเปิดใช้งาน: (i) การแลกเปลี่ยนข้อมูลระหว่างโปรแกรมที่สร้างขึ้นอย่างอิสระและโปรแกรมอื่นๆ (รวมถึงโปรแกรมนี้) และ (ii) การใช้ข้อมูลที่มีการแลกเปลี่ยนร่วมกัน ควรติดต่อ:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

ข้อมูลดังกล่าวอาจพร้อมใช้งานภายใต้ระยะเวลาและเงื่อนไขที่เหมาะสม โดยมีการชำระค่าธรรมเนียมในบางกรณี

โปรแกรมที่ได้รับอนุญาตซึ่งอธิบายไว้ในเอกสารนี้และเอกสารประกอบที่ได้รับอนุญาตทั้งหมดที่มีอยู่มีการนำเสนอโดย IBM ภายใต้ระยะเวลาของข้อตกลงกับลูกค้าของ IBM, ข้อตกลงเกี่ยวกับใบอนุญาตโปรแกรมระหว่างประเทศของ IBM หรือข้อตกลงที่เท่าเทียมกันใดๆ ระหว่างเรา

ข้อมูลประสิทธิภาพ และตัวอย่างลูกค้าที่ระบุมีการนำเสนอสำหรับวัตถุประสงค์การสาธิตเท่านั้น ผลลัพธ์ของประสิทธิภาพการทำงานจริงอาจขึ้นอยู่กับคอนฟิกูเรชันและเกณฑ์การทำงานที่ระบุเฉพาะ

ข้อมูลเกี่ยวกับผลิตภัณฑ์ที่ไม่ใช่ของ IBM ได้มาจากผู้จัดจำหน่ายของผลิตภัณฑ์เหล่านั้น คำประกาศที่เผยแพร่หรือแหล่งข้อมูลที่เปิดเผยต่อ สาธารณะ IBM ไม่ได้ทดสอบผลิตภัณฑ์ดังกล่าว และไม่สามารถยืนยันความถูกต้องของประสิทธิภาพ ความเข้ากันได้ หรือการเรียกร้องอื่นใดที่เกี่ยวข้องกับผลิตภัณฑ์ที่ไม่ใช่ของ IBM หากมีคำถามเกี่ยวกับความสามารถของผลิตภัณฑ์ที่ไม่ใช่ของ IBM ควรสอบถามกับ ผู้จัดจำหน่ายของผลิตภัณฑ์ดังกล่าว

ข้อความใดๆ ที่เกี่ยวข้องกับทิศทางในอนาคตและเจตจำนงค์ของ IBM อาจมีการเปลี่ยนแปลง หรือเพิกถอนได้โดยไม่ต้องแจ้งล่วงหน้า และนำเสนอเฉพาะเป้าหมาย และวัตถุประสงค์เท่านั้น

ราคาที่แสดงทั้งหมดของ IBM เป็นราคาขายปลีกที่แนะนำของ IBM ในปัจจุบัน และอาจเปลี่ยนแปลงได้โดยไม่ต้องแจ้งให้ทราบ ราคาของผู้แทนจำหน่ายอาจแตกต่างออกไป

ข้อมูลนี้ใช้สำหรับวัตถุประสงค์การวางแผนเท่านั้น ข้อมูลในเอกสารฉบับนี้อาจมีการเปลี่ยนแปลง ก่อนที่ผลิตภัณฑ์ที่กล่าวถึงจะมีจำหน่าย

ข้อมูลนี้ประกอบด้วยตัวอย่างข้อมูลและรายงานที่ใช้ในการดำเนินธุรกิจ ประจำวัน เพื่อแสดงให้เห็นอย่างสมบูรณ์ที่สุดเท่าที่จะเป็นไปได้ ตัวอย่างเหล่านี้จึงประกอบด้วย ชื่อของบุคคล บริษัท ตราสินค้า และผลิตภัณฑ์ ชื่อเหล่านี้ทั้งหมดเป็นชื่อสมมติ และความคล้ายคลึงใดๆ กับบุคคล หรือองค์กรธุรกิจที่มีอยู่จริง ถือเป็นเหตุบังเอิญ

ใบอนุญาตลิขสิทธิ์:

ข้อมูลนี้ประกอบด้วยโปรแกรมแอ็พพลิเคชันตัวอย่างในภาษาต้นฉบับ ซึ่งแสดงเทคนิคในการเขียนโปรแกรมบนแพล็ตฟอร์มปฏิบัติการที่หลากหลาย คุณสามารถคัดลอก ปรับเปลี่ยน และแจกจ่ายโปรแกรมตัวอย่างเหล่านี้ในรูปแบบต่างๆ ได้โดยไม่ต้องชำระเงินให้แก่ IBM เพื่อใช้สำหรับการพัฒนา การใช้งาน การตลาด หรือการแจกจ่ายโปรแกรมแอ็พพลิเคชันที่สอดคล้องกับ

อินเทอร์เฟซโปรแกรมแอ็พพลิเคชันของแพล็ตฟอร์มการดำเนินงานที่เขียนโปรแกรมตัวอย่าง ตัวอย่างเหล่านี้ยังไม่ได้ผ่านการทดสอบในทุกสภาพ ดังนั้น IBM จึงไม่สามารถรับประกันหรือแจ้งถึงความน่าเชื่อถือ การให้บริการได้ หรือฟังก์ชันของโปรแกรมเหล่านี้ได้ โปรแกรมตัวอย่างมีการนำเสนอ "ตาม สภาพ" โดยไม่มีการรับประกันประเภทใดๆ IBM ไม่ต้องรับผิดชอบต่อความเสียหายใดๆ ที่เกิดขึ้นจากการใช้โปรแกรมตัวอย่างของคุณ

แต่ละสำเนาหรือส่วนใดๆ ของโปรแกรมตัวอย่างเหล่านี้ หรืองานที่สืบเนื่องใดๆ ต้องมีคำประกาศ ลิขสิทธิ์ดังนี้:

© (ชื่อบริษัทของคุณ) (ปี)

ส่วนต่างๆ ของรหัสนี้ได้มาจากโปรแกรมตัวอย่างของ IBM Corp.

© ลิขสิทธิ์ IBM Corp. _ป้อนปี_

ข้อควรพิจารณาเกี่ยวกับนโยบายความเป็นส่วนตัว

IBM Software products, including software as a service solutions, (“Software Offerings”) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as the customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM’s Privacy Policy at http://www.ibm.com/privacy and IBM’s Online Privacy Statement at http://www.ibm.com/privacy/details the section entitled “Cookies, Web Beacons and Other Technologies” and the “IBM Software Products and Software-as-a-Service Privacy Statement” at http://www.ibm.com/software/info/product-privacy.

เครื่องหมายการค้า

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at Copyright and trademark information at www.ibm.com/legal/copytrade.shtml.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

ดัชนี

อักขระพิเศษ

ตัวเลข

A

B

C

D

E

F

H

I

J

L

M

O

P

R

S

T

V

W

ก

ข

ค

ง

จ

ช

ซ

ด

ต

ท

ธ

น

บ

ป

ผ

พ

ฟ

ภ

ม

ย

ร

ล

ว

ส

ห

อ

ฮ

พิมพ์ในสหรัฐอเมริกา