

Table of Contents

Table of Contents.....	1
About the Authors	3
Introduction	4
Tuning AIX: Situation before 5.2.....	5
Tuning AIX: Situation before 5.2 (continued)	6
Tuning AIX: Situation before 5.2 (continued)	7
Tuning AIX: 5.2 goals	8
AIX 5.2 Tunable Files	9
AIX 5.2 Tunable Files (continued).....	10
AIX 5.2 Tunable Files (continued).....	11
Tunable Parameter Types	12
AIX 5.2 New Tuning Commands	13
Example of New Tuning Commands	14
Common Flags of the Tuning Commands	15
SMITTY or Telnet Tuning.....	16
SMITTY Tuning: Global Manipulation of Tuning Parameters.....	17
SMITTY vmo Panel.....	18
SMITTY vmo Panel (continued)	19
Tuning AIX: WebSM VMM Table	21
Migration and Compatibility.....	22
Recovery Procedure	23
Compatibility Mode.....	24
Performance Tools	25
/proc Tools	26
Curt — CPU Usage Reporting Tool	27
Curt: Usage	28
Curt System Summary	29
Curt—Report Sections	30
Curt System Summary	31
Curt—Report Sections (continued)	32
Curt Application Summary Reports	33
Kproc and Application Pthread Summary	34
Curt System Calls Summary	35
Curt Flih Summary	36
Curt Slih Summary.....	37
Curt Thread Summary	38
Curt Process Summary	39
Curt—Trace Hooks	40
Collecting a Trace for Curt.....	41
Collecting a Trace for Curt (continued)	42
tprof Introduction.....	43
Tprof: introduction (continued)	44
tprof: 5.2 version.....	45
Tprof: usage	46

Tprof: new syntax	47
Tprof: profile output format	48
Tprof: microprofile output format.....	49
Tprof: format changes	50
tprof: modes of operation	51
Truss Update	52
Truss: tracing example	53
Truss: system call counting	54
The -d option	55
The -l option.....	56
The -f Option.....	57
Other Truss Flags	58
Perfstat API: introduction	59
Perfstat API: global interface example.....	60
Perfstat API: component interface example.....	61
Perfstat API: AIX 5.2	62
Conclusion	63
References	64
Trademarks	65

About the Authors

Nam Keung
Senior programmer

Nam Keung is a senior programmer for IBM in Austin, Texas. He has worked in the area of AIX ISDN communication, AIX SOM/DSOM development, AIX Multimedia development, NT Clustering technology, and Java performance. His current assignment involves helping ISVs in porting, deploying applications, performance tuning, and education for the pSeries platform.

Luc Smolders
Senior software engineer

Luc Smolders is a senior software engineer currently working in the UNIX Systems Development organization. He has worked with UNIX-based systems of various sizes since 1982. In 1989, he joined the IBM AIX Technical support organization in Belgium. In 1992, he moved to Austin, Texas, where he is still located. He has been working in the AIX Performance department since 1993, specializing in tools. He is currently the architect in charge of the performance tools shipping with AIX, the Performance Toolbox LPP, and tools developed for internal usage. In the past, his responsibilities also included the IA64 platform, and for several years, Linux PPC. He regularly teaches AIX Performance Analysis and Tuning classes. Luc holds a Master's degree in Computer Science Engineering from the Catholic University of Louvain in Belgium.

AIX 5.2 Performance Tools update



- ▶ **Nam Keung**
Senior Technical Consultant, IBM
 - ▶ **Luc Smolders**
Senior Software Engineer
- Copyright IBM, 2004, all rights reserved.
-

Introduction

Hello and welcome to this “AIX 5.2 Performance Tools Update” online course.

The release of AIX® 5.2 is a major and exciting revamp of the AIX family of performance tools. We have replaced some of the tuning commands and added new commands. And now, all tuning commands use the same syntax and provide consistent behavior.

During this course, you will learn the elements of the new AIX 5.2 tuning framework and the detailed descriptions of each of the new or modified kernel tuning commands.

Before we look at what is new in AIX 5.2, let's review the earlier release and its tuning limitations.



Tuning AIX: situation before 5.2

● AIX only provided a set of inconsistent commands

- **vmtune** (sample program)
- **schedtune** (sample program. not consistent with vmtune)
- **no** and **nfso** (somewhat consistent but different from vmtune/schedtune)

Tuning AIX: Situation before 5.2

Prior to AIX 5.2, the performance tools commands were inconsistent and presented inherent limitations. The 'range' and 'parameter' values were not available, there was no validation checking for the new values, the command parameters were inconsistent, and there was no clean way to make persistent changes in files that are required to make changes in persistent boots. We will talk about these points more on the next chart.

But first, let's understand some fundamental tuning issues prior to 5.2.

The **vmtune** command is used to modify the Virtual Memory Management (VMM) parameters that control the behavior of the memory-management subsystem. The vmtune command resides in the "samples" directory because it is VMM-implementation dependent. The vmtune code that accompanies each release of the operating system is tailored specifically to the VMM in that release. Running vmtune from one release on a server with a different VMM release can result in an operating server failure.

The **schedtune** (Scheduler Tunable) command is used to modify CPU scheduler and VMM processing parameters. The sample program provided for schedtune was not consistent with vmtune.

The **no** command (which manages network tuning parameters) and the **nfso** (Network File System Option tunable) command were somewhat consistent between themselves; but were not consistent with vmtune and schedtune.

All of these inconsistencies could make tuning quite a challenge—and if some of these are used incorrectly (for example, the no command), the server can be rendered inoperable.



Tuning AIX: situation before 5.2 (Continued)

- **Commands limitations**

- **Not all commands allow parameter defaults to be reset**
- **Ranges & parameter units not available online**
- **Do not always check for valid new values
(may accept changes impossible to make & report bogus values); e.g.:**
 - **no -o arptab_bsize=value** is accepted at any time
(only works before loading inet kernel extension; but it reports the new value!)
 - **vmtune -y 1 (to set memory_affinity)** is accepted unconditionally
(if bosboot is not called, vmtune does not change anything, but DOES report the new value!)
 - **nfso -o nfs_v2_vm_bufs=value** accepts any value in range
(sometimes seems not to work because it reports old value; but, in fact, this parameter value cannot be decreased)

Tuning AIX: Situation before 5.2 (continued)

The earlier releases of AIX had commands limitations. Not all commands had options to reset parameters back to their default values. And, the ranges and parameter units were not available online.

Additionally, the earlier releases did not always check for the validity of new parameter values. Worse, they would often indicate that changes were actually made, when in fact, they were impossible to make. The result was that bogus parameter values would occasionally be reported. Look briefly at the three examples shown in red on this chart and the explanations for why they could be troublesome.



Tuning AIX: situation before 5.2 (Continued)

- **There is no clean way to make persistent changes**
 - ▶ only workaround is to modify /etc/rc* files to call commands
 - ▶ makes it very hard to predict settings after reboot

Tuning AIX: Situation before 5.2 (continued)

Making persistent changes in earlier releases was awkward, requiring “tricks” that would ultimately produce those changes. The only real workaround, in fact, was to modify the contents of the /etc/rc* files, which are installation-specific, and which are used to perform normal startup initialization for the AIX server. That limitation made it difficult to accurately predict settings after rebooting.



Tuning AIX : 5.2 goals

- **Main goals of 5.2 tuning framework**
 - Cleanly support "permanent" & "reboot" values
 - Command consistency
- **Many usability improvements**
 - New set of SMIT menus & WebSM to manipulate tunables
 - Use only supported commands, via common syntax
 - Set/reset 'current' or 'after reboot' value from SMIT, WebSM, or cmd line
 - Parameter types, range, units & help are available
 - Dependencies between parameters is listed & enforced
- **Eliminates need to add calls to tune commands anywhere**
 - Reboot values applied automatically
 - Watch for 'Setting tunable parameters...complete' during boot sequence
 - If bosboot is needed, it is called automatically

Tuning AIX: 5.2 goals

The main goals of the AIX release 5.2 tuning framework have been accomplished. The new release cleanly supports permanent and reboot values. It also provides command consistency. Additionally, the new release offers many usability improvements, including a new set of System Management Interface Tool (SMIT) menus and a Web-based System Manager (WebSM) that supports manipulation of tunables. *[NOTE: SMIT is a graphical interface that must remote into the AIX host or, alternatively, SMIT must be a terminal on the host. WebSM is written in Java and is a graphical user interface.]*

All supported commands now use a common syntax. You may set or reset any "current" value, or "after reboot" value from SMIT, WebSM, or from the command line. Parameter types, range, units, and help are also available, and dependencies between parameters are listed and enforced.

You will also appreciate the convenience of having reboot values applied automatically, thus removing the need to ever add calls to tuning commands. You will see "Setting tunable parameters – complete" during the boot sequence, which lets you know that the process of tuning a command has been successfully accomplished. Furthermore, when a Base Operating System boot (bosboot) is needed, it will be initiated automatically.

[NOTE: Another online course, entitled "IBM eServer pSeries Logical Partitioning: Installation & Configuration," explains the process of installing the WebSM client in Windows. You can link to this alternate course from the "Hotlinks" section.]



AIX 5.2 Tunable Files

- All tunable parameters in the **/etc/tunables/nextboot** ASCII file
- **/etc/tunables/lastboot** is automatically generated with all the values
- The log file for any changes made is restored in **/etc/tunables/lastboot.log**
- Use SMIT panels (smitty tuning) or WebSm to manipulate current and reboot values for all tuning parameters

AIX 5.2 Tunable Files

In regard to AIX 5.2 tunable files, all tunable parameters are found in the **/etc/tunables/nextboot** ASCII file. The nextboot file is automatically applied at next reboot. The **/etc/tunables/lastboot** file contains all the tunable parameters that were set at the last machine reboot, while the log file for any changes that have been made is restored in **/etc/tunables/lastboot.log**.

You may now use System Management Interface Tool (SMIT) panels (SMITTY tuning) or WebSM to manipulate current values and reboot values for all tuning parameters.



AIX 5.2 Tunable Files (Continued)

- **5 new commands to modify tunables files**
 - tunsave
 - tunrestore
 - tuncheck
 - tundefault
 - tunchange

AIX 5.2 Tunable Files (continued)

Beginning with AIX 5.2, you can make permanent kernel-tuning changes without having to edit any **rc** files. This is achieved by centralizing the reboot values for all tunable parameters in the **/etc/tunables/nextboot** stanza file. When a server is rebooted, the values in the **/etc/tunables/nextboot** file are automatically applied.

The following commands are used to manipulate the **nextboot** file and other files containing a set of tunable parameter values:

- **tunsave** — Saves all current values to a file, which can include the nextboot file
- **tunrestore** — Applies values from a file, either immediately, or at the next reboot (through the use of the **-r** flag). When the **-r** flag is used, the tunrestore file is validated and is then copied over the current nextboot file.
- **tuncheck** — Validates a tunables file that has been created manually.
- **tundefault** — Resets the tunable parameters to their default values.
- **tunchange** — Updates stanzas in the tunables files.

The preceding commands work on both current and reboot values.



AIX 5.2 Tunable Files (Continued)

- Other files can be stored in **/etc/tunables**, but only nextboot file will be applied at boot time
- Files can be copied from another machine, created & manipulated from SMIT, WebSm, or vi.

AIX 5.2 Tunable Files (continued)

Note that other files can be stored in /etc/ tunables, but only the nextboot file will be applied at boot time. Furthermore, files can be copied from another server. Or, they can be created and manipulated from SMIT, WebSM, or with the vi editor.

[NOTE: the vi editor is a full-screen editor that allows you to edit text on a screen-by-screen basis.]



Tunable Parameters Type

- **Classified in 7 categories of parameters:**

- **Dynamic:** Can be changed at any time
- **Static:** Can never be changed (read-only parameter)
- **Reboot:** Can only be changed during reboot
- **Bosboot:** Can only be changed by running bosboot & then rebooting
- **Connect:** Changes only apply to future socket connections.
 - Changing one or more parameters of this type automatically restarts inetd.
- **Mount:** Are only effective for future file system or directory mounts
- **Incremental:** Can only be incremented (not decremented), except at boot time

Tunable Parameter Types

The tunable parameters are classified into seven categories.

- The **Dynamic** parameter can be changed at any time.
- However, the **Static** parameter is a “read-only” parameter that can never be changed.
- The **Reboot** parameter can only be changed during reboot.
- The **Bosboot** parameter can only be changed by running bosboot, and must be followed with a server reboot.
- Changes to the **Connect** parameter only apply to future socket connections. Changing one or more parameters of this type automatically restarts inetd. *[NOTE: The inetd daemon starts by default each time you start your server and runs as a background process to provide Internet service management functions for a network.]*
- **Mount** parameters are only effective for future file system mounts or for directory mounts.
- **Incremental** parameters can only be incremented. They can only be decremented at boot time.



AIX 5.2 New Tuning Commands

• 5 tuning commands manipulate tunable parameter values

Pre-AIX 5.2	AIX 5.2
no	no
vmtune	vmo
vmtune	ioo
schedtune	schedo
nfso	nfso

• vmtune & schedtune

- Replaced by shell scripts calling vmo, ioo & schedo
- Accept all existing flags
- Flags changing Bosboot type parameters
 - Can't be used in scripts because prompt to run bosboot was added
 - User must call vmo directly
 - Flags are accepted, but message is displayed indicating to use vmo directly

AIX 5.2 New Tuning Commands

Five tuning commands are available with AIX 5.2 to manipulate the tunable parameter values. Some of these commands are new, while others have been expanded to provide greater and more reliable tuning functionality.

The vmtune and schedtune commands have been replaced by shell scripts that call the new vmo, ioo, and schedo commands which are now used to more easily manage memory, I/O, and CPU.

- The **vmo** command, which specifically replaces part of the functionality of vmtune, manages virtual memory manager tunable parameters.
- The **ioo** command, which also replaces parts of the previous function provided by vmtune, manages the I/O tunable parameters.
- The **schedo** command manages the CPU scheduler tunable parameters.

These new commands accept all existing flags that have been set by vmtune and schedtune, so there is no need to “retune” server dynamics that are already working well. The exception to this is that any flags that change bosboot parameter types are not supported by the new commands. This is because a prompt to run bosboot was added to these new commands. Therefore, any flags that change bosboot type parameters will trigger this prompt. Since this is not desirable, if you wish to change a bosboot parameter, you must call the vmo command directly. As a side note, the bosboot flags are technically “accepted” by the vmo shell script, but you will be given a message indicating that you should call vmo directly.



Example of New Tuning Commands

To change parameters, use -r option.

vmtune options-	parameter name	new command
-g n1 -L n2	large page size # of large page to reserve	vmo -r -o lpg_size=n1 -o lpg_regions=n2
-m n	memory pools	vmo -r -o mempools=n
-v n	# of frames per second	vmo -r -o framesets=n
-y 0 1	p690 memory affinity	vmo -r -o memory_affinity=1

Example of New Tuning Commands

To change tuning parameters, you must use the -r option. (Remember, the -r option signifies that the parameter change is to apply to reboot values.)

Here are some examples of tuning parameters using the new commands.



Common flags of the tuning command

- vmo, ioo, schedo, nfso, & no commands have common flags & use same syntax

command [-p|-r] -o parameter[=value]

command [-p|-r] -d parameter

command [-p|-r] -D

command [-p|-r] -a

command -h [parameter]

command -L [parameter]

command -x [parameter]

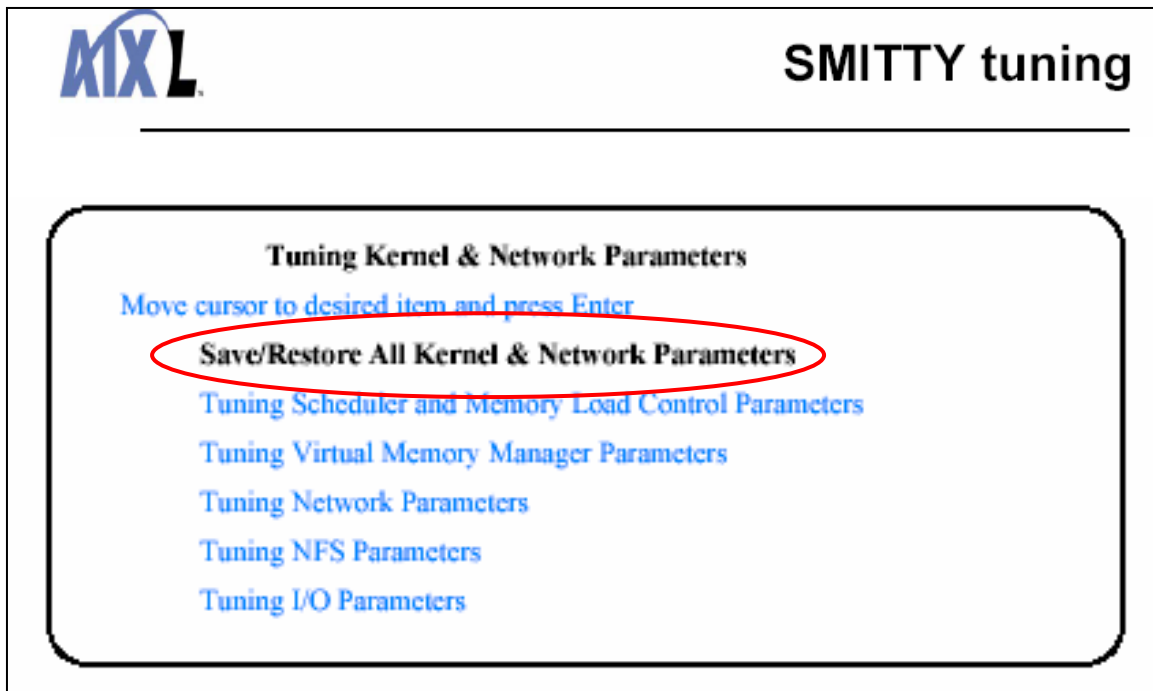
Flag	Description
-a	Displays values for all tunable parameters
-h	Displays command help or displays help about tunables
-d	Resets tunables to default value
-D	Resets all tunables to their default value
-o	Tunable = value, sets tunable to specific value
-p	Makes changes apply to both current and reboot values
-r	Make changes apply to reboot value only.
-L	Displays parameter characteristics
-x	Displays parameter characteristics in spreadsheet format

Common Flags of the Tuning Commands

All five tuning commands provided with AIX 5.2 (vmo, ioo, schedo, nfso, and no) have common flags, and use the same syntax. They are available to directly manipulate the tunable parameter values. This common syntax makes you more productive as you come up-to-speed on these new commands.

Please review these commands. You will soon be comfortable using them.

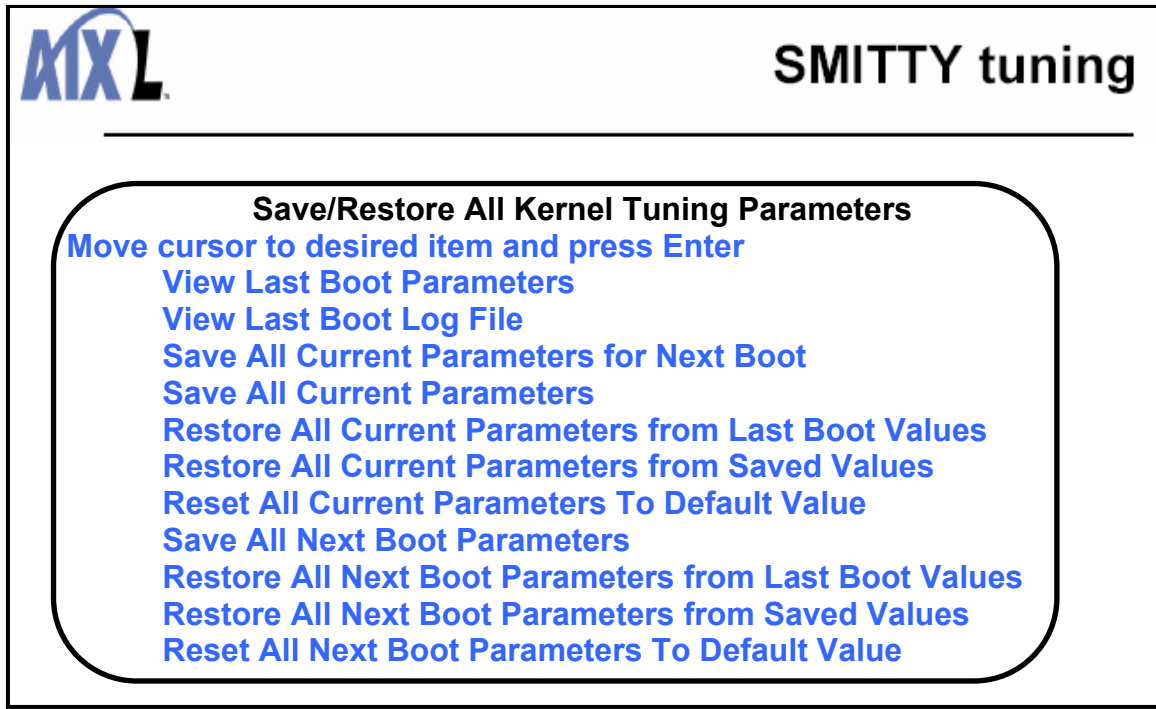
- **-a** displays the values for all tunable parameters.
- **-h** displays command help or displays help about tunables parameters.
- **-d** resets tunable to its default value.
- **-D** resets all tunables parameters to their default values.
- **-o** sets a tunable parameter to a specific value. The format for doing this is "Tunable = value."
- **-p** makes changes apply to both current and reboot tunable parameter values.
- **-r** makes changes apply to reboot values only. (We have mentioned this earlier in this course.)
- **-L** displays the tunables parameter characteristics so you can examine them.
- **-x** displays the tunables parameter characteristics in spreadsheet format, again, so that you can examine them, or document them for later reference.



SMITTY or Telnet Tuning

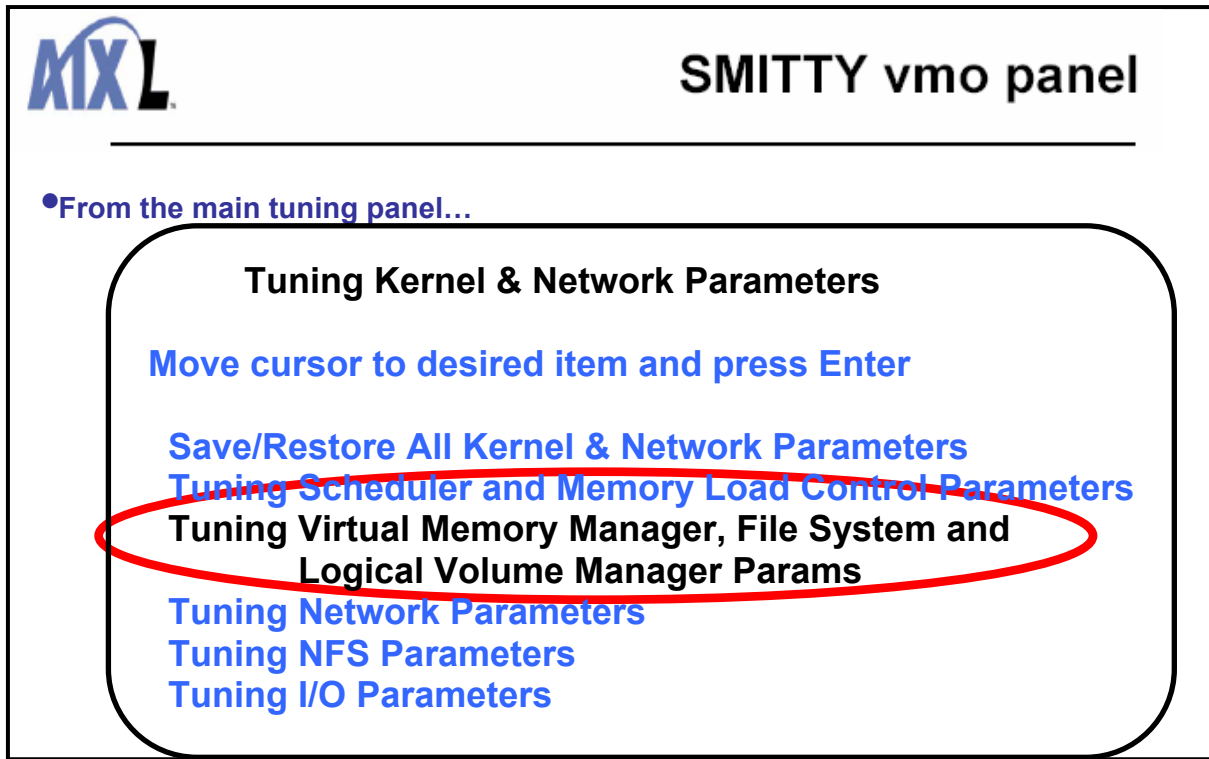
System Management Interface Tool (SMIT) panels and Web-based System Manager are also available to manipulate current and reboot values for all tuning parameters, as well as the files in the **/etc/tunables** directory.

To start the SMIT panels that manage AIX kernel tuning parameters, use the SMIT fast path **smitty tuning**. This graph is a view of the tuning panel. Select **Save/Restore All Kernel & Network Parameters** to manipulate all tuning parameter values at the same time. To individually change tuning parameters managed by one of the tuning commands, select any of the other lines.




SMITTY Tuning: Global Manipulation of Tuning Parameters

The main panel to manipulate all tunable parameters by sets looks similar to the chart shown here.



SMITTY vmo Panel

From the main tuning panel, let's select Tuning Virtual Memory Manager, File System and Logical Volume Manager Params. (All the panels for all five commands behave the same way.)



SMITTY vmo panel

•From the “Tuning Virtual Memory Manager, File System and Logical Volume Manager Params” panel...

Tuning Virtual Memory Manager , File System and Logical Volume Manager Params

Move cursor to desired item and press Enter

List All Characteristics of Current Parameters
Change / Show Current Parameters
Change / Show Parameters for Next Boot
Save Current Parameters for Next Boot
Reset Current Parameters to Default value
Reset Next Boot Parameters To Default Value


SMITTY vmo Panel (continued)

Here is the main panel used to manipulate parameters managed by the vmo command. Let's look at the interaction possibilities between parameter types and the different SMIT sub-panels shown on this screen capture.

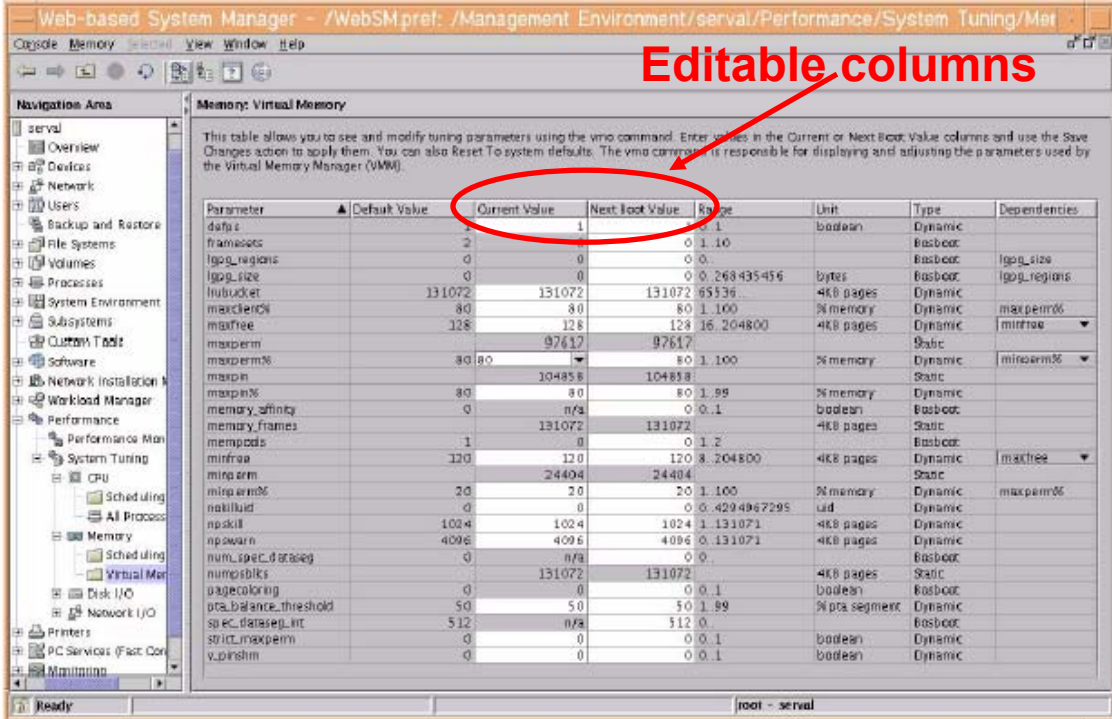
List all characteristics of current parameters	Lists current, default, reboot, limit values, unit, type and dependencies. This is the output of a tuning command called with the vmo -L option.
Change/show current parameters	Displays/changes current parameter value, except for parameter types Static, bosboot, & Reboot, which are displayed with no surrounding square brackets to indicate they cannot be changed.
Change/show parameters for next boot	Displays values from and rewrites updated values to nextboot file. If needed, bosboot is proposed. Only parameter type Static cannot be changed (no brackets around their value).
Save current parameters for next boot	Writes current parameters in nextboot file. Bosboot will be proposed if parameter of type bosboot were changed.
Reset current param's to default value	Resets current parameters to default values, unless they need a bosboot plus reboot, or reboot (bosboot and reboot type).
Reset nextboot parameters to default value	Clears values in the nextboot file, and proposes bosboot if any parameter of type bosboot was different from its default value.

Here are some additional smit subpanels...

Change reboot value	When used in combination with -o, -d or -D, vmo -r makes changes apply to reboot values, for example, turns on the updating of the /etc/tunables/nextboot file. If any parameter of type Bosboot is changed, the user will be prompted to run bosboot. When used with -a or -o without specifying a new value, next boot values for tunables are displayed instead of current values.
Save tunable parameters in a file	The tunsave command saves the current state of tunable parameters in a file. The -F Filename flag specifies the name of the file where the tunable parameters are saved (e.g., "nextboot"). The -a flag specifies that even tunable parameters that are set to their default value are saved. The -t vmo option specifies that the vmo command is to be used to update the parameter(s).
Reset all tunable parameter to their default value	The tundefault command can be used to reset all AIX tunable parameters to their default value, except for parameters of type Bosboot and Reboot , and parameters of type Incremental that are set at values bigger than their default value. The -t vmo option specifies that the vmo command is to be used to update the parameters. The -r flag specifies that the reset does not occur until the next reboot.



Tuning AIX: WebSm VMM table



Editable columns

Parameter	Default Value	Current Value	Next Boot Value	Range	Unit	Type	Dependencies
defsys	1	1	1	1-1	boolean	Dynamic	
framesize	2	2	2	0-1.10		basboot	
lpgs_regions	0	0	0	0-0		basboot	lpgs_size
lpgs_size	0	0	0	0-268435456	bytes	basboot	lpgs_regions
lsubdct	131072	131072	131072	65536	4KB pages	Dynamic	
maxclm6	80	80	80	1-100	% memory	Dynamic	maxperm6
maxfree	128	128	128	16-204800	4KB pages	Dynamic	minfree
maxperm	80	80	80	1-100	% memory	Dynamic	minperm
maxperm6	80	80	80	1-100	% memory	Dynamic	maxperm
maxperm8	80	80	80	1-100	% memory	Dynamic	
memory_affinity	0	0	0	0-1	boolean	basboot	
memory_frames	131072	131072	131072		4KB pages	Static	
memppds	1	1	1	0-1.2		basboot	
minfree	120	120	120	8-204800	4KB pages	Dynamic	maxfree
minperm	24404	24404	24404			Static	
minperm6	20	20	20	1-100	% memory	Dynamic	maxperm6
nokilld	0	0	0	0-4294967295	ud	Dynamic	
npool	1024	1024	1024	1-131071	4KB pages	Dynamic	
npooln	4096	4096	4096	0-131071	4KB pages	Dynamic	
num_spec_databeg	0	0	0	0-0		basboot	
numstbks	131072	131072	131072		4KB pages	Static	
pagecoloring	0	0	0	0-1	boolean	basboot	
pta_balance_threshold	50	50	50	1-89	% pta segment	Dynamic	
spec_databeg_int	512	512	512	0		basboot	
strict_maxperm	0	0	0	0-1	boolean	Dynamic	
v_pinshm	0	0	0	0-1	boolean	Dynamic	

Tuning AIX: WebSM VMM Table

There is a tuning table that allows all the characteristics of the tunable parameters to be viewed at a glance. The table has two editable columns, **Current Value** and **Next Boot Value**. Each cell in these two columns is an editable combobox, with only one predefined value of **Default**, for the capture of “new value” for a parameter. Data entered in these columns is validated when pressing **ENTER**.



Migration and Compatibility

- **When migrating to AIX 5.2 from previous AIX releases...**
 - Tuning commands are automatically set to run in Compatibility mode.
- **When a system is initially installed with AIX 5.2...**
 - It is automatically set to run in Tuning mode.
- **Mode is controlled by 'sys0' attribute called 'pre520tune'**
 - Set to run in Compatibility mode, or
 - Disable to run in AIX 5.2 mode.
- **To retrieve current 'pre520tune' attribute setting:**
 - **`lsattr -E -l sys0`**
- **To change current 'pre520tune' attribute setting:**
 - **`chdev -l sys0 -a pre520tune=enable`**
 - Or, use SMIT or WebSM to change current settings

Migration and Compatibility

Let's take a look at some points to be considered in migrating from a prior release of AIX performance tools to AIX 5.2.

- When servers are migrated to AIX 5.2 from previous releases, the tuning commands are automatically set to run in Compatibility mode.
- When a server is initially installed with AIX 5.2, it is automatically set to run in AIX 5.2 tuning mode.
- The mode is controlled by the 'sys0' attribute called 'pre520tune,' which can be set to enable running in Compatibility mode, and can be disabled to run in AIX 5.2 mode.
- To retrieve the current setting of the 'pre520tune' attribute, run the "List Attribute Characteristics" command shown in red on this chart. *[NOTE: lsattr displays attribute characteristics and possible values of attributes for devices in the server, while the -l parameter specifies the device logical name.]*
- To change the current setting of the 'pre520tune' attribute, run the "Change Device characteristics" command shown in green on this chart. *[NOTE: chdev changes the characteristics of a device in the server, while the -l parameter specifies the device logical name.]*
 - Or, you can use SMIT or Web-based System Manager to change the current settings of the 'pre520tune' attribute.



Recovery Procedure

- **If the system become unstable with a given nextboot file:**

- Put server into maintenance mode.
- Make sure pre520tune 'sys0' attribute is set to 'disable.'
- Delete nextboot file.
- Run Bosboot command.
- Move tunable line to be last in **/etc/inittab** file, & reboot.


- **These actions will set server back to Default mode.**

Recovery Procedure

The recovery procedure to reset to Default Mode is straightforward.

If the server becomes unstable with a given nextboot file, put it into Maintenance mode, make sure the pre520tune "sys0" attribute is set to disable, delete the nextboot file, run the Bosboot command, move the tunable line to be last in the **/etc/ inittab** file, and reboot.

These actions should guarantee that all tunables are set back to Default mode.



compatibility mode

- **Compatibility issue:**
 - Can only set *Reboot* & *Bosboot* parameters with **-r**
 - Including from rc.net
- **Solution**
 - Pre-5.2 compatibility mode
 - Set automatically when migrating from 5.1
 - Controlled by new `pre520tune sys0` attribute
 - Retrieve current setting:
 - **lsattr -E -l sys0 | grep pre520tune**
 - Change setting:
 - **chdev -l sys0 -apre520tune=disable|enable**
 - Also available from *smitty chgsys* or *WebSM*
- **Two modes**
 - `pre520tune=disable` : default mode for new 5.2 installation
 - `pre520tune=enable` : default mode after migration to AIX 5.2

Compatibility Mode

The *Reboot* and *Bosboot* parameters can no longer be set without using **-r** (which is the flag that refers to reboot values). This new stipulation is true even when specifying parameters from the rc.net file.

Here is the detailed solution to overcoming the compatibility issue. The pre-AIX 5.2 compatibility mode is set automatically, when migrating from 5.1, and it is controlled by the new 'pre520tune sys0' attribute.

To retrieve the current setting, use the "List Attribute Characteristics" command, as shown in red on this chart. To change setting, use the "Change Device Characteristics" command, also shown in green on this chart. *[NOTE: As mentioned earlier in this course, you can also retrieve and make changes to these settings via SMITTY chgsys or WebSM.]*

Note that you have two modes in which to solve the compatibility issues. You can disable the default mode for the new 5.2 installation, or you can enable the default mode after migration to AIX 5.2.



Performance tools

•For AIX 5.2

```
•lslpp -lI perfagent.tools bos.perf.proctools bos.perf.tools bos.perf.tune
bos.perf.diag_tool bos.perf.perfstat bos.sysmgt.trace
```

•For AIX 5.1

```
•lslpp -lI perfagent.tools bos.sysmgt.trace bos.acct bos.perf.tools bos.adt.samples
```

•Performance Reporting and Analysis Commands

- curt** Reports CPU utilization for each kernel thread
- truss** Trace a process's system calls
- tprof** Shows CPU usage by every AIX process ID & name

Performance Tools

Performance tools for the system environment fall into two general categories:

- Those that tell you what is occurring.
- Those that let you do something about it.
- And, there are a few tools that do both.

The performance-related commands are packaged as part of the AIX 5.2 perfagent.tools, bos.perf.proctools, bos.perf.tools, bos.perf.tune, bos.perf.diag_tool, bos.perf.perfstat, and bos.sysmgt.trace filesets that are shipped with the Base Operating System. You can determine whether all the performance tools have been installed by running one of the “List LPP” (lslpp) commands shown in this chart.

Performance reporting and analysis commands give you information on the performance of one or more aspects of the server, or on one or more of the parameters that affect performance. Let's talk about a few of these commands you will certainly be using:

- **curt** reports CPU utilization for each kernel thread (starting with AIX 5.2).
- **truss** traces a server's calls in one or more processes. *[NOTE: In AIX 5.2, all base system call parameter types are now recognized. In AIX 5.1, only about 40 system calls were recognized.]*
- **tprof** uses the trace facility to report the CPU consumption of kernel services, library subroutines, application-program modules, and individual lines of source code in the application program.



/proc tools

- **12 simple utilities in AIX**

- Similar to tools provided on Solaris
- Useful for debugging & analyzing process behavior

- **Display information on process using /proc data**

- Data is all-binary, so not easily accessible

proctree	procstack	procmap
procldd	procflags	procsig
proccred	procfiles	procwdx
procstop	procrun	procwait

- **/proc is also enhanced with fd subdirectory & cwd file**

/proc Tools

There are 12 simple utilities, similar to tools provided on Sun™ Solaris™, that are very useful for debugging and analyzing process behavior. The /proc file system gives access to information about the current state of processes and threads, but is provided in binary form—so it is not easily accessible. The /proc tools commands provide ASCII reports based on some of the available information, but in a much more readable format.

The /proc tools are utilities that exercise features of the /proc file system. The /proc files contain data that presents the state of processes and threads in the server, even as that state is constantly changing.

- **proctree** displays the process tree containing the specified process IDs or users.
- **procstack** displays the hexadecimal addresses and symbolic names for each of the stack frames of the current thread in process.
- **procmap** displays a process address map.
- **procldd** displays a list of libraries loaded by a process.
- **procflags** displays process tracing flags, and pending and holding signals.
- **procsig** lists the signal actions for a process.
- **proccred** prints a process credentials.
- **procfiles** prints a list of open file descriptors.
- **procwdx** prints the current working directory for a process.
- **procstop** stops a process.
- **procrun** restarts a process.
- **procwait** waits for all of the specified processes to terminate.

The /proc files have been enhanced further in AIX 5.2 with the ability to access the file descriptor (fd) subdirectory and Current Working Directory (cwd) file information.



Curt-cpu usage reporting tool

- Curt provides detailed CPU usage information on system, processor, processes, and kernel threads—based on a trace and a name file
- Report sections include:
 - System-level and per-processor summary
 - Time is sorted into application, system calls, kproc, flihs, slihs, dispatch, and idle
 - Thread, process, and program execution summary
 - kproc execution summary
 - System call summary
 - flih & slih reports (count, total, min, max & average CPU time)
 - process and thread detailed reports
 - System call options
 - Elapsed time
 - Errors

Curt — CPU Usage Reporting Tool

CURT, a trace post-processing tool, summarizes system utilization. It provides detailed CPU usage information on server, processor, processes, and kernel threads based on a trace and a name file. Its input is a binary AIX 5 system trace file. The output consists of CPU and elapsed-time reporting on processes and threads, and time spent in first- and second-level interrupt handlers.

CURT is contained in the bos.perf.tools fileset. It allocates all the time for each CPU in a trace to one of six states:

- Application — This state is for user space execution and includes kprocs other than wait.
- Kernel — This state is for system call (SVC) execution events.
- Flih — This state involves first-level, interrupt-handler (flih) execution.
- Slih — This state includes second-level interrupt-handler (slih) execution.
- Dispatch — This state involves dispatch logic execution.
- Wait — This state includes waitproc execution.



Curt : usage

Syntax:

```
curt -i inputfile [-o outputfile] [-n gennamesfile] [-m trcnmfile]
[-a pidnamefile] [-f|-l timestamp] [-ehpst]
```

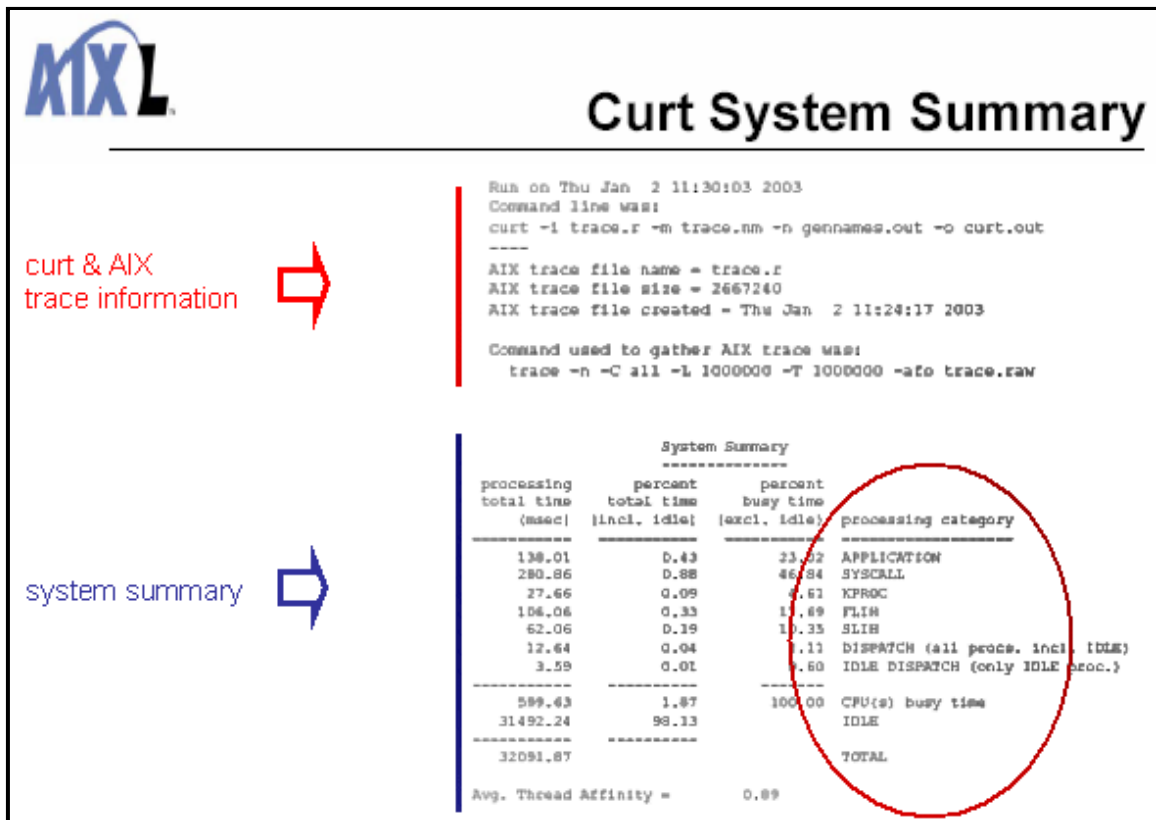
Flags:

-i inputfile	Specify input AIXTrace file to be analyzed.
[-o outputfile]	Specify output file (default is stdout).
[-n gennamesfile]	Specify a names file produced by gennames.
[-m trcnmfile]	Specify a names file produced by trcnm.
[-p]	Output detailed process information.
[-t]	Output detailed thread information.
[-P]	Output detailed pthread information (52B and up)
[-e]	Output elapsed time info. for system calls.
[-s]	Output info. about errors returned by system calls.
[-f timestamp]	Start processing trace at "timestamp" seconds.
[-l timestamp]	Stop processing trace at "timestamp" seconds.
[-a pidnamefile]	Specify a pid->process name mapping file.
[-h]	Display usage text (this information).

Curt: Usage

The syntax for correctly using the curt command is shown in red on this chart.

As many as 13 different flags are available with the Curt command, providing a great deal of flexibility in meeting your need to better understand exactly how your server environment is behaving. The flags related to this command are explained in this chart.




Curt System Summary

The time and date when this particular `curt` command was run is shown here at the top of this chart. It includes a display of the syntax of the `curt` command line that produced the report.

The lower part of this chart shows a system summary. This section prints out the times and percentage of total time the server (all CPUs) spent in various states. The times are application (user) mode time, kernel (supervisor) mode time, time spent in Flihs, time spent in Slihs, time spent running the dispatcher code, time spent in the dispatcher while dispatching the idle process, total busy time, and total idle (waitproc) time.

The processing categories circled in red on the right of this chart will be discussed on the next page.



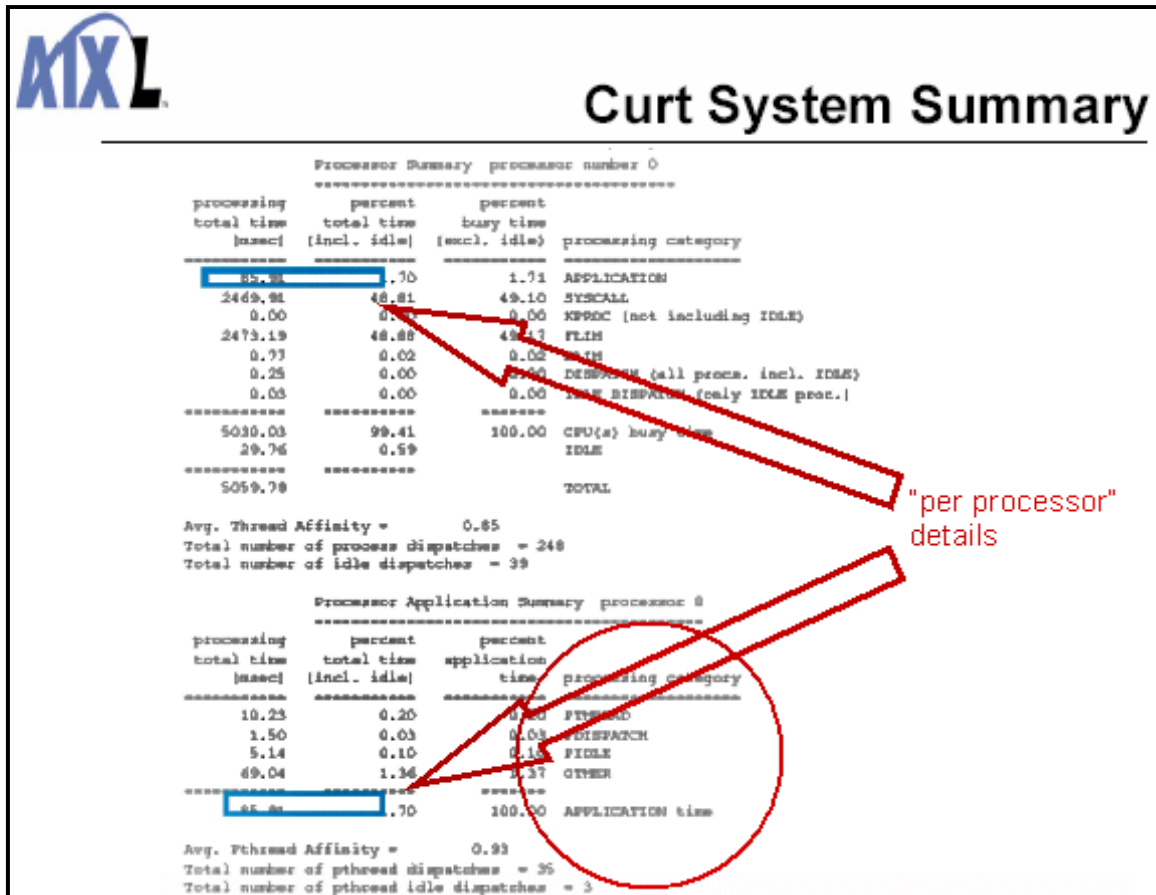
Curt - report sections

- **System-level and per-processor summaries**
 - **Processing categories in System and Processor Summary sections**
 - **Processor Summary sections**
 - application ● system calls
 - kproc ● flihs
 - slihs ● dispatch
 - Idle ● Total

Curt—Report Sections

The possible execution modes or processing categories in the System and Processor Summary section of the Curt report are interpreted as follows:


APPLICATION	Sum of times spent by all processors in User (that is, non-privileged) mode.
SYSCALL	Sum of times spent by all processors doing System Calls. This is the portion of time a processor spends executing in the kernel code providing services directly requested by a user process.
KPROC	Sum of times spent by all processors executing kernel processes other than the IDLE process. This is the portion of time that a processor spends executing specially created dispatchable processes that only execute kernel code.
FLIH	Sum of times spent by all processors executing FLIHs.
SLIH	Sum of times spent by all processors executing SLIHs.
DISPATCH	Sum of times spent by all processors executing the AIX dispatch code. This sum includes the time spent dispatching all threads (that is, it includes dispatches of the IDLE process).
IDLE DISPATCH	Sum of times spent by all processors executing AIX dispatch code where the dispatched process was IDLE. Because the DISPATCH category includes the IDLE DISPATCH category's time, the IDLE DISPATCH category's time is not separately added to calculate CPU(s) busy time or TOTAL (see below).
CPU(s) busy time	Sum of times spent by all processors executing in APPLICATION, SYSCALL, KPROC, FLIH, SLIH, and DISPATCH modes.
IDLE	Sum of times spent by all processors executing IDLE process.
TOTAL	Sum of CPU(s) busy time and IDLE.



Curt System Summary

A per-processor summary follows the System Summary, and is essentially the same information, but it is broken down on a processor-by-processor basis.

The processing categories circled in red on the right of this chart will be discussed on the next page.




Curt - report sections

- **System-level and per-processor summaries (continued)**
 - **Processing categories in System & Processor Application Summary sections**
 - Pthread
 - Pidle
 - other
 - pdispatch
 - application

Curt—Report Sections (continued)

The possible execution modes or processing categories in System and Processor Application Summary sections of the Curt report are interpreted as follows:

PTHREAD	Sum of times spent by all pthreads on all processors in traced pthread library calls.
PDISPATCH	Sum of times spent by all pthreads on all processors executing the libpthreads dispatch code.
PIDLE	Sum of times spent by all kernel threads on all processors executing the libpthreads vp_sleep code.
OTHER	Sum of times spent by all pthreads on all processors in non-traced user mode.
APPLICATION	Sum of times spent by all processors in User (that is, non-privileged) mode.



Curt : Application Summary Reports

Application Summary (by tid)

=====

== processing total [msec] ==			== percent of total processing time ==			
combined	application	syscall	combined	application	syscall	name (tid tid)
354.9568	354.9568	0.0000	1.7542	1.7542	0.0000	{21210 20017}
301.8945	0.0000	301.8945	1.4920	0.0000	1.4920	shmid(20798 21000)
267.2969	4.6100	262.6870	1.3207	0.0229	1.2978	shmid(21220 20027)
267.0185	8.7290	258.2895	1.3196	0.0333	1.2864	shmid(17860 18561)
266.9688	4.1496	262.8192	1.3189	0.0205	1.2984	shmid(17864 18563)
266.5486	6.0502	260.4984	1.3179	0.0299	1.2874	shmid(18928 24287)
266.9774	8.6241	258.3533	1.3140	0.0327	1.2813	shmid(21220 20029)
266.7922	4.8346	261.9577	1.3136	0.0238	1.2897	shmid(21222 20021)
266.7670	3.9613	262.7683	1.3132	0.0196	1.2937	shmid(17860 18559)

Application Summary (by tid)

=====


== processing total [msec] ==			== percent of total processing time ==			
combined	application	syscall	combined	application	syscall	name (tid) (thread count)
354.9568	354.9568	0.0000	1.7542	1.7542	0.0000	{21210} (1)
301.8945	0.0000	301.8945	1.4920	0.0000	1.4920	shmid(20798) (1)
267.2969	4.6100	262.6870	1.3207	0.0229	1.2978	shmid(21220) (1)
267.0185	8.7290	258.2895	1.3196	0.0333	1.2864	shmid(17860) (1)
266.9688	4.1496	262.8192	1.3189	0.0205	1.2984	shmid(17864) (1)
266.5486	6.0502	260.4984	1.3179	0.0299	1.2874	shmid(18928) (1)
266.9774	8.6241	258.3533	1.3140	0.0327	1.2813	shmid(21220) (1)

Application Summary (by process type)

=====

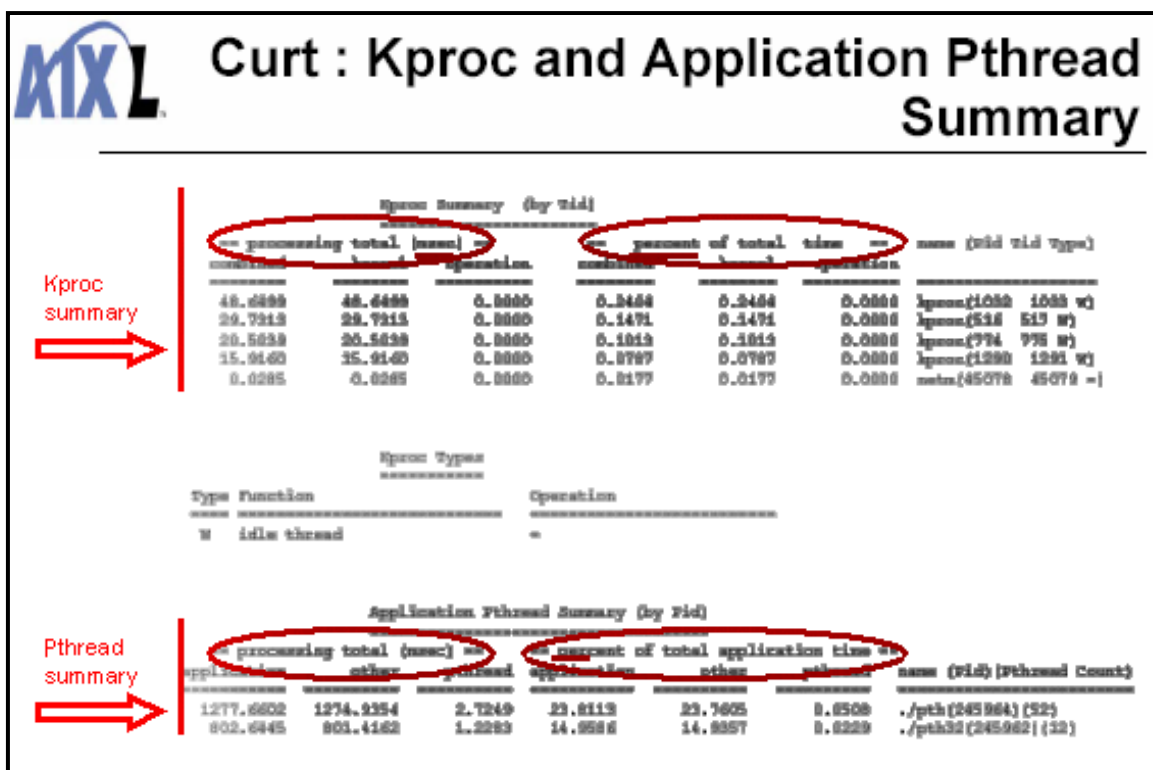
== processing total [msec] ==			== percent of total processing time ==			
combined	application	syscall	combined	application	syscall	name (thread count)
9013.0030	227.9010	9085.1020	48.4987	1.3263	47.1724	shmid(45)
564.5198	564.5198	0.0000	2.7899	2.7899	0.0000	{2}
153.8918	131.3246	22.5672	0.7805	0.6498	0.1315	perl(2)
11.8000	6.7442	5.0558	0.0583	0.0333	0.0250	topas(1)
4.3729	2.8852	1.4877	0.0216	0.0139	0.0077	dtgrent(1)

Application summaries



Curt Application Summary Reports

The `curt` command will generate information about the amount of CPU time spent in application and system call (syscall) mode. This time is expressed both in milliseconds and as a percentage of total CPU time—by thread, process, and process type. Also included in these summary reports are the number of threads per-process and per-process-type.




Kproc and Application Pthread Summary

The Kproc Summary (by TID, or Thread ID) generates information about the amount of CPU time spent executing each kernel process, including the idle process. CPU time is expressed in both milliseconds and as a percentage of the total CPU time. This summary also shows an output of all kernel process threads (and their CPU consumption) that were running on the server during the time of trace collection.

The **Application Pthread Summary (by PID, or Process ID)** shows an output of all the multi-threaded processes that were running on the server during trace collection and their CPU consumption, and that have spent time making pthread calls. The process that consumed the most CPU time during the trace collection is at the beginning of the list.

This information is also expressed in milliseconds and as a percentage of total CPU time.

[NOTE: The total processing time is split into two categories, "operation" and "kernel," which loosely correspond to "syscall" and "application" for a process which always runs in kernel code. Each kproc thread is identified by name, PID, TID, and type of kproc, if known.]



Curt System Calls Summary

System Calls Summary						
Count	Total Time (secs)	% sys time	Avg Time (secs)	Min Time (secs)	Max Time (secs)	SVC (Address)
40	9470.9112	46.81%	236.7728	0.1803	297.1308	_exit(13cfd0)
43	45.5119	0.22%	1.0584	0.5038	1.2273	kfork(13bc00)
1553	16.2352	0.08%	0.0305	0.0063	0.1183	__disclaim64(15b7ae0)
116	9.2561	0.05%	0.0798	0.0052	0.3418	hwrite(1b0080)
442	6.6149	0.03%	0.0150	0.0019	0.0723	hread(1b00ac)
3	3.4469	0.02%	1.1490	1.1259	1.1667	getprocs(22b0d0)
3	2.8527	0.01%	0.9509	0.4882	1.2370	move(13bc14)
41	2.8983	0.01%	0.0654	0.0263	0.1269	__disclaim64(15b8334)
42	2.2641	0.01%	0.0539	0.0306	0.0905	connect(13d71c)
168	2.8774	0.01%	0.0124	0.0004	0.0876	close(1b001c)
...						

Errors Returned by System Calls

Error#	errno	count	description	returned for System Call:	__disclaim64(15b7ae0)
4	41	1	*Interrupted system call*		
Error#	errno	count	description	returned for System Call:	close(1b001c)
9	42	1	*Bad file number*		
Error#	errno	count	description	returned for System Call:	ioctl(1b22a6a0)
64	168	1	*Operation not supported on socket*		
25	168	1	*Not a typewriter*		
Error#	errno	count	description	returned for System Call:	__disclaim64(15b8334)
11	42	1	*Resource temporarily unavailable*...		

Pending System Calls Summary

Accumulated Time (secs)	SVC (Address)	Procname (Pid Tid)
0.0123	thread_wait(13d248)	tftpd(7224 3671)
0.0085	waitpid(13cfd0)	cron(9624 7243)
0.0177	semop_64(15a040)	X(13188 13971)
0.0175	hread(1b00ac)	net(16862 16869)

Curt System Calls Summary

This report shows the completed system calls. It includes the name and address of the system call, the number of times the system call was executed, and the total CPU time (expressed in milliseconds and as a percentage). It also shows the average, minimum, and maximum number of times the system call was running.

You can also see the errors that were returned by the system calls and a summary of the pending system calls.

Curt Flih Summary

Global Flih Summary

Count	Total Time (nsec)	Avg Time (nsec)	Min Time (nsec)	Max Time (nsec)	Flih Type
32	0.6690	0.0207	0.0070	0.0265	5 [IO_INTR]
579	13.1338	0.0227	0.0022	0.0982	32 [QUEUED_INTR]
2082	30.4754	0.0146	0.0015	0.2881	31 [DECR_INTR]
129	0.2867	0.0016	0.0007	0.0064	4 [INSTR_PG_FLT]
9545	9493.9504	0.9915	0.0037	194.2942	3 [DATA_ACC_PG_FLT]

Per CPU Flih Summary

CPU Number 0:

Count	Total Time (nsec)	Avg Time (nsec)	Min Time (nsec)	Max Time (nsec)	Flih Type
8	0.1526	0.0191	0.0099	0.0220	5 [IO_INTR]
39	0.8492	0.0015	0.0007	0.0040	4 [INSTR_PG_FLT]
127	2.8236	0.0230	0.0022	0.0973	32 [QUEUED_INTR]
530	8.4741	0.0179	0.0070	0.0493	31 [DECR_INTR]
2289	2468.5887	1.0750	0.0037	194.2942	3 [DATA_ACC_PG_FLT]

CPU Number 1:

Count	Total Time (nsec)	Avg Time (nsec)	Min Time (nsec)	Max Time (nsec)	Flih Type
8	0.1747	0.0218	0.0070	0.0265	5 [IO_INTR]
529	7.5819	0.0144	0.0015	0.2881	31 [DECR_INTR]
141	3.7492	0.0239	0.0028	0.0969	32 [QUEUED_INTR]
27	0.8841	0.0019	0.0007	0.0060	4 [INSTR_PG_FLT]
2100	1948.8783	0.9241	0.0062	135.8694	3 [DATA_ACC_PG_FLT]

Pending Flih Summary

Accumulated Time(nsec)	Flih Type
0.8353	3 [DATA_ACC_PG_FLT]

Curt Flih Summary

The First level interrupt handler (FLIH) summary lists all first-level interrupt handlers that were called during the monitoring period, as seen in the example shown here.

Curt Slih Summary

Global Slih Summary

Count	Total Time (nsec)	Avg Time (nsec)	Min Time (nsec)	Max Time (nsec)	Slih Name (Address)
1	0.0278	0.0278	0.0278	0.0278	s_scxiddpin(1464210)
31	3.8027	0.1227	0.0527	0.2039	phxentdd(1585cb4)

Per CPU Slih Summary

CPU Number 0:


Count	Total Time (nsec)	Avg Time (nsec)	Min Time (nsec)	Max Time (nsec)	Slih Name (Address)
1	0.0278	0.0278	0.0278	0.0278	s_scxiddpin(1464210)
7	0.7376	0.1054	0.0527	0.1897	phxentdd(1585cb4)

CPU Number 1:

Count	Total Time (nsec)	Avg Time (nsec)	Min Time (nsec)	Max Time (nsec)	Slih Name (Address)
8	1.0346	0.1293	0.0757	0.2039	phxentdd(1585cb4)

Curt Slih Summary

The second level interrupt handler (Slih) summary lists all second-level interrupt handlers that were called during the monitoring period, as shown in this chart.



Curt Thread Summary

```

Report for Thread Id: 21698 (hex 54c3) Pid: 20798 (hex 513a)
Process Name: whoisd
-----
Total Application Time (ms): 0.000000
Total System Call Time (ms): 301.894276

Thread System Call Data
Count  Total Time  Avg Time  Min Time  Max Time  SVC (Address)
      (ms)      (ms)      (ms)      (ms)
-----
1      297.1300    297.1300    297.1300    297.1300    _exit(13cfd0)
33      0.3102      0.0094      0.0065      0.0381    _disclaim64(15b7aa0)
1       0.0306      0.0306      0.0306      0.0386    _disclaim64(15b833e)

Errors (errno : count : description) returned for System Calls: _disclaim64(0x15b8310)
76 :      1 : "Socket is not connected"
Errors (errno : count : description) returned for System Calls: _disclaim64(0x15b7c5c)
13 :      1 : "Permission denied"
19 :      1 : "No such device"

Pending System Calls Summary
-----
Accumulated  SVC (Address)
Time (ms)
-----
4.2385      klock-chld(1)

Processor affinity: 1 000000

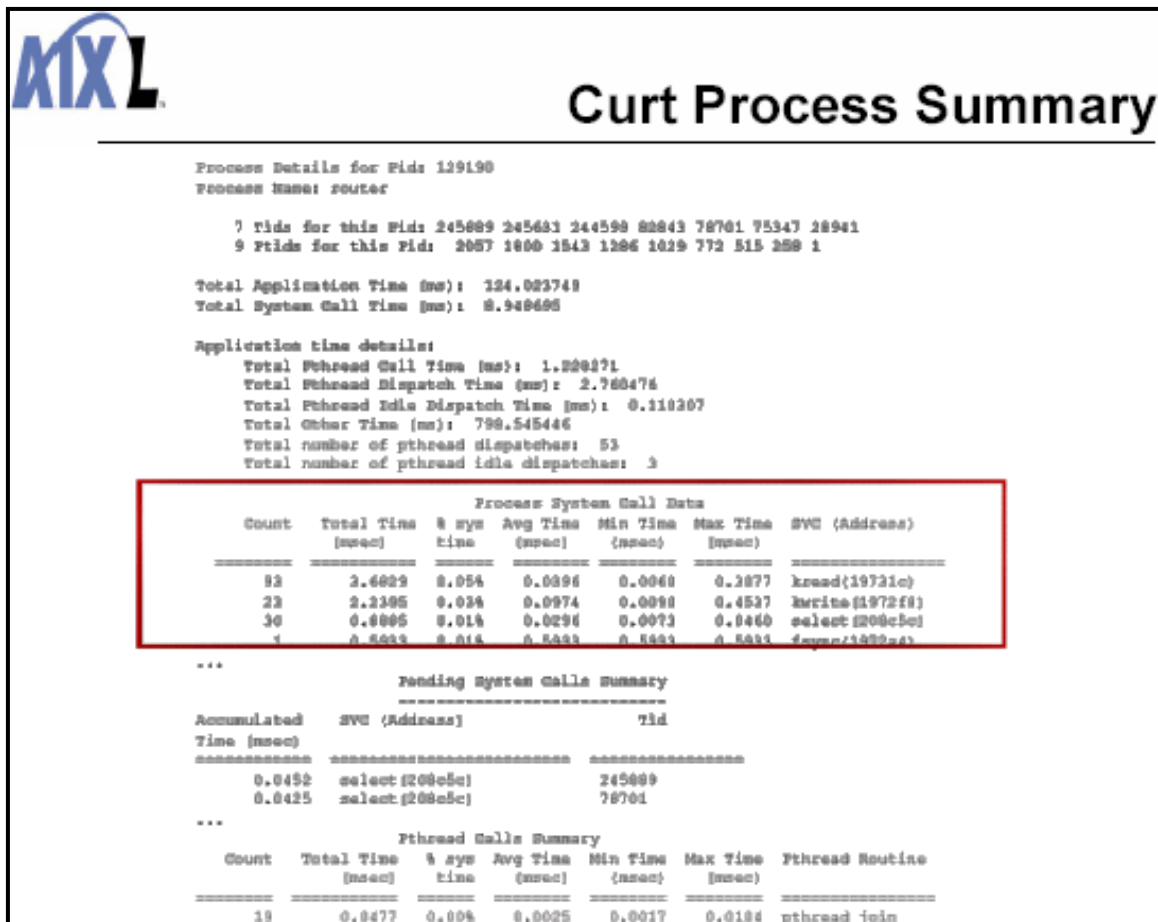
Dispatch Histogram for thread (CPUID : times_dispatched).
CPU 1 : 7
total number of dispatches: 7
avg. dispatch wait time (ms): 0.149033

Data on Interrupts that Occured while Thread was Running
Type of Interrupt  Count
-----
Data Access Page Faults (DSI): 174
Instr. Fetch Page Faults (ISI): 3
Align. Error Interrupts: 0

```

Curt Thread Summary

If you specify the `-t` flag in the CURT command, you will get a detailed report on thread status. This report will include: the amount of time the thread was in application and kernel mode, what system calls the thread made, processor affinity, the number of times the thread was dispatched, and the CPU to which it was dispatched.



Curt Process Summary

If you specify the `-p` flag in the CURT command, you will be able to view detailed process information. This report will include the count for the number of times the process ran and the total amount of time (milliseconds and percentage) that was utilized to run all the instances of that process. You will further be able to see the average, minimum, and maximum time committed to running each instance of the process. You can even see the server address space where the process was executed.



Curt -Trace Hooks

- **trace hooks necessary**
 - **100,101,102,103,104,106,10C,119,134,135,139,200,210,38F,465**
 - **for kernel monitoring**
 - **+605,609**
 - **pthread monitoring, also need LIBPATH=/usr/lib/cc/perf**
 - **52B and up**
 - **trace -J curt (trace hookids group) can also be used**
- **name mapping options**
 - **trcnm output**
 - **gennames output**

Curt—Trace Hooks

A raw, or unformatted, system trace from AIX Version 4 or AIX Version 5 is read by the `curt` command to produce summaries, as well as first- and second-level interrupt handlers. This chart shows the minimum trace hooks required for the `curt` command. Using only these trace hooks will limit the size of the trace file. However, other events on the server may not be captured if you only use these trace hooks. This is significant if you intend to analyze the trace in more detail. Of particular interest here are trace hooks 119 and 135, which are used to report on the time spent in the `exit` system call; and trace hooks 134, 139, 210, and 465, which are used to keep track of TIDs, PIDs, and process names.

Trace hooks 605 and 609 are used to report on the time spent in the pthreads library.

Trace `-J curt` (trace hookids group) can also be used.

You can collect the kernel names by using the `trcnm` command or the `gennames` command. This is not required for the `curt` command to run successfully. However, if you provide one or both of these files, the `curt` command will output names for system calls and interrupt handlers instead of merely indicating their addresses. The output of `trcnm` is typically much smaller than the output of `gennames` since it maps only the kernel names and not the kernel extensions or shared libraries. Curt does not use the extra information and will run somewhat faster without it.



Collecting a Trace for Curt

- Default report created by curt command:

```
# curt -i trace_report -m trace.nm -n gennames.out -o curt.out
```

- Example for collecting a trace for curt:

- Start trace daemon with:

```
#trace -a -d -f -C all -T Size -L Size -J curt -o trace_file
```

- -a runs trace daemon asynchronously.
- -d defers data collection until trcon is called.
- -f stops data collection when in-memory buffer is filled.
- -C all (if SMP system), creates set of trace buffers for every CPU. Else, you may overfill a single set of trace buffers or force CPU contention for access to shared buffers.
- -T & -L override default trace buffer & trace log file sizes. By default, all trace events are collected, which can quickly fill trace buffer & trace log file with unneeded data.
- -J curt captures events processed by curt.

- Start & stop trace data collection with:

```
#trcon; program; trcoff
```

- trcon starts trace data collection; program runs; then, trace data collection is stopped with trcoff—always trace one contiguous time-interval for curt.

Collecting a Trace for Curt

The trace daemon collects data for a curt report. The default report created by the curt command is shown at the top of this chart in red. An example of how you might collect a trace as input to the curt command is also shown here.

Let's review the trace options you can invoke. The **-a** option runs the trace daemon asynchronously. The **-d** option defers data collection until trcon is called. The **-f** option causes data collection to stop when the in-memory buffer is filled. If this is an SMP system, the **-C all** option creates trace buffers for every CPU. Otherwise you risk overfilling a single set of trace buffers, or at least force the CPUs to contend for access to the shared buffers. The **-T** and **-L** options override the default trace buffer and trace log file sizes. By default, all trace events are collected, which can quickly fill up the trace buffer (and trace log file) with data that is not needed. Use the **-J curt** option to capture the events processed by curt.

You start and stop trace data collection using "trcon; program; trcoff." The trcon command starts the trace data collection; the program is run, after which the trace data collection is stopped with trcoff. You always trace a single contiguous time interval for curt.



Collecting a Trace for Curt (cont'd)

- Collect kernel names with:

trcnm > trcnm.out or **gennames > gennames.out**

- trcnm output is typically much smaller than gennames output, since it maps only kernel names (not kernel extensions or shared libraries). curt doesn't use the extra information & runs faster without it.

- Stop trace with:

trcstop

- Recombine individual trace files (if **-C all** was used) with:

trcrpt -r -C all trace_file > trace_report

Collecting a Trace for Curt (continued)

You will collect kernel names with the *trcnm* command shown here in red. The output of *trcnm* is typically much smaller than the output of *gennames* since it maps only the kernel names and not the kernel extensions or shared libraries. *curt* does not use the extra information and will run somewhat faster without it.

You can stop the trace with *trcstop*, and recombine the individual trace files (if the **-C all** option has been used) with *trcrpt*.



Tprof: introduction

•Introduction

- program counter sampling-based profiler
- uses AIX trace, even in offline mode (i.e., as trace file post-processor)
- works on any executable, including stripped binaries
- produces flat profile of hot threads, object files and routines
- supports profiling of applications, libraries, & AIX kernel & extensions
- annotates source code when available

tprof Introduction

The tprof command is a standard UNIX® performance tool that also provides a detailed profile of CPU usage by process ID and name. It further profiles at the application level, routine level, and even to the source statement level. It delivers both a global view and a detailed view. The tprof command can profile kernel extensions, stripped executable programs, and stripped libraries. It will do subroutine-level profiling for most executable programs on which the stripnm command will produce a symbol table.

tprof profiles only CPU activity; it does not profile other system resources, such as memory or disks. It uses the system trace facility. Only one user at a time can execute the trace facility. Therefore, only one tprof command can be executing at one time.

tprof can charge CPU time to object files, processes, threads, subroutines (user mode, kernel mode, and shared library), and even to source lines of programs or individual instructions. Charging CPU time to subroutines is called profiling, and charging CPU time to source program lines is called microprofiling.

For subroutine-level profiling, the tprof command can be run without modifying executable programs. (No recompilation with special compiler flags is necessary). This is still true if the executables have been stripped, unless the back tables have also been removed. However, recompilation is required to get a microprofile, unless a listing file is already available.



Tprof: introduction

- **Usability issues with current version**

- creates files with cryptic names
- re-postprocessing mode not really supported (captures live data)
- only supports profiling
 - whole system: results go into __prof.all
 - one process or thread: results go into __program.all
- summary report only has ticks, user must calculate percentages
- detailed profiling only available when source available
- to generate user-mode profile, must also select a process or thread
- no whole-system user-mode profile available

Tprof: introduction (continued)

As you can see, there were many usability issues with the AIX 5.1 version of this tool. But instead of dwelling on these deficiencies, let's look at what has changed with the AIX 5.2 release.



Tprof: 5.2 version

•5.2 Enhancements


- support for multiple program profiling in one pass**
 - including micro-profiling
- full support for threads**
 - reports can include list of threads, all threads in one or more processes or the system
- new optional listing file annotation (instruction-level annotation)**
 - can use instead of source annotation
 - can use in combination with source annotation
- new optional detailed profiling report (address-level report)**
- new front-end options to collect trace & name mapping information**
- fully functional re-postprocessing mode supports online & offline data collection modes transparently**
 - optional cooking produces processed trace & symbol files
- new symbol mapping file format replaces gennames format**
 - uses new gensyms command offline
 - realtime mode generates same format when "cooking" is selected
- speed & other usability improvements**

tprof: 5.2 version

The tprof command has been completely rewritten for AIX 5.2; it is much faster and provides more function. This command is a useful tool for anyone with a Java™, C, C++, or FORTRAN program that might be CPU-bound and who wants to know which sections of the program are most heavily using the CPU.

Here's a brief list of the enhancements with AIX 5.2:

- Support for multiple program profiling in one pass, including microprofiling.
- Full support for threads. Reports can include a list of threads, all threads within one or more processes on the server.
- New optional listing file annotation (instruction-level annotation).
- New optional detailed profiling report (address-level report).
- New front-end options to collect trace and name mapping information.
- Fully functional re-postprocessing mode supporting online and offline data collection modes transparently.
- New symbol mapping file format replaces gennames format.
- Increased speed of various other usability improvements.



Tprof: usage

Usage:

```
tprof [-c] [-C {all | cpuidlist}] [-d] [-D] [-e] [-F]
      [-j] [-J profilehook] [-k] [-l] [-m objectlist]
      [-M sourcepathlist] [-p processlist] [-P {all | pidlist}]
      [-s] [-S searchpathlist] [-t] [-T buffersize] [-u] [-v]
      [-V verbosefilename] [-z] {{-r rootstring}} [{-A {all | cpuidlist}}] [-r rootstring] -x command }}
```

- A turns on Automated Offline mode. No argument turns off multi-CPU tracing, all or cpuidlist turns on multi-CPU tracing.
- c turns on Cooking.
- F turns on overwrite mode (to overwrite cooked files, if they exist). If used without -x option, -F forces manual offline mode.
- e turns on Kernel Extension profiling.
- j turns on Java profiling.
- k turns on kernel profiling.
- s turns on shared library profiling.
- u turns on user application profiling.
- D turns on detailed profiling (which displays CPU usage by instruction offset).
- m turns on microprofiling. (You supply a list of objects for source level profiling.
- p turns on process level details—only for those processes whose names are supplied in the process list.
- P turns on process level details for PIDs supplied in pids list. For details of all processes running, specify 'all' for pidlist.
- t turns on thread level details.
- r specifies the rootstring. tprof input and report files all have names in the form of rootstring.suffix.
- x is used to specify the command that needs to be executed by tprof. '-x' should always be placed after all the other flags.
- C turns on the multi-CPU mode, either 'all' or a list of cpuids must be provided.
- d turns on Deferred Mode trace.
- J is used to specify kernel profile trace hook.
- M is used to specify source path list—used to find the source during microprofiling.
- S is used to specify search path list—used to find any binaries (executable, library or kernel extension) for profiling.
- T is used to specify Trace Buffer size (in Realtime or Automated offline mode).
- v turns on verbose mode.
- V specifies verbose file.
- l turns on long listing (none of the names will be truncated).
- z turns on the ticks report.

Tprof: usage

The text shown in red on this chart indicates the syntax for the tprof command. The parameters for this command are shown on this chart, along with an explanation of each of them.

A few additional points to remember are that all the list type inputs are separated by commas except for pathlist. Pathlists are separated by a colon. Multi-CPU profiling mode will be automatically disabled while running in realtime mode. Microprofiling will be automatically disabled if multi-CPU profiling is turned on. Specified Log Buffer size will be omitted while running in realtime mode. If the -x option is specified without the -A option, then tprof runs in realtime mode.

[NOTE: If the -x option is specified with -A option, the tprof runs in Automated offline mode. If the -x option is omitted, then tprof runs in post-processing mode or manual offline mode.]



Tprof: new syntax

- **Uses rootstring concept to name input/output files**
 - *rootstring.trc*: raw
 - *rootstring.syms*
 - *rootstring.prof*
 - *rootstring.sourcefilename.mprof*
 - *rootstring.ctrc*
 - *rootstring.csyms*
 - default rootstring is program name when -x is used
 - any file created is always announced
- **Specify search paths**
 - for object files (-S)
 - for source files (-M)
 - default is \$PATH for both
- **Annotation more flexible**
 - -g (no listing or source) : annotates list of line numbers
 - -g (only listing): annotates line numbers & instructions
 - -g or -qlst (source is available) annotates source lines & instructions

Tprof: new syntax

The new tprof syntax uses the rootstring concept to name all input and output files. Here are the possible files that will be created, each is always announced:

- rootstring.trc is a raw trace file.
- rootstring.syms is a symbol file.
- rootstring.prof is a profile report file.
- rootstring.sourcefilename.mprof is a microprofiling report file for the source file name.
- rootstring.ctrc is the cooked trace file and is the equivalent to binary __trc_rpt2.
- rootstring.csyms is a cooked symbol file that is an ASCII file which only needs symbols.
- default rootstring is the program name file—when -x is used.

You specify search paths for object and source files (-S and -M, respectively). The default search path is “both” and is represented by the \$PATH parameter.

With this new syntax, you will notice more flexibility. For instance, if the -g flag is specified, and neither the listing or source is available, tprof will still produce annotations as a list of line numbers. If a listing is available, but the source is lacking, tprof will annotate line numbers and instructions. If the -g or -qlst flag is used and the source is available, tprof will annotate the source lines and instructions.



Tprof: profile output format

- File name: rootstring.prof
- Global summary section (always present)
 - summary by process name
 - summary by threads (tid)
- Optional global detailed report sections
 - pertains to execution of all processes on system
 - user mode routines profile (-u)
 - kernel routines profile (-k)
 - summary for kernel extensions (-e)
 - each kernel extension's routines profile (-e)
 - summary for shared libraries (-s)
 - each shared library's routines profile (-s)
 - summary for JAVA classes (-j)
 - JAVA methods of each JAVA class profile (-j)
- Optional process & threads report(s)
 - one report for each process or thread
 - same subsections as above
 - controlled by same flags

Tprof: profile output format

The output format for the results of the tprof command includes the file name—which is rootstring.prof. Next, there is a global summary section, which is always present. It contains a summary of CPU usage by process name and by threads (tid).

Some optional global detailed report sections pertain to the execution of all processes on the server. Each of these are listed on this chart.

Finally, there are also some optional process and threads report(s). Basically, there is one report for each process or thread—each containing the same subsections as in the global detailed report. Similarly, the printing of the subsections is controlled by the same flags that are listed on this chart for the global detailed report sections.



Tprof: microprofile output format

- File name: *rootstring.sourcefilename.mprof*
- Full path name of source file
- Hot lines list
 - sorted list of source line numbers with samples
 - Time % associated by line
 - if process (-p or -P) or thread (-t) level detail was selected
 - ▶ additional breakdown lines
 - ▶ lines include Pid &, optionally, Tid
- Annotated source listings
 - sorted functions list
 - ordered by relative hotness
 - Includes only functions with samples
 - Annotations at 2 levels
 - ▶ source line (even if source file is unavailable) if compiled with -g and/or -qlst
 - ▶ instruction line (if listing file is available)

Tprof: microprofile output format

The output format for microprofile starts with the file name, which as mentioned two charts previously, *rootstring.sourcefilename.mprof*. The full path name of the source file is provided, as is a hot lines list. The hot lines list is a sorted list of source line numbers, along with samples. A percentage of time is associated with each line. If process-level or thread-level detail was selected, additional lines will be provided, along with a breakdown for each thread or process. These additional lines will also include Pid and, optionally, Tid.

The microprofile output will show annotated source listings of functions. This listing is ordered by relative hotness. Further, only those functions with samples are listed. Two levels of annotations are provided. The first is source line annotation—even if the source file is unavailable. Second, there is instruction line annotation—if the listing file is available.

We'll talk more about these annotations on the next chart...



Tprof: format changes

- **Tprof output**

- ♦ **summary format: “%” replaces tick counts**
- ♦ **routine reports: Ticks, Address & Bytes columns are gone**
- ♦ **-z turns on old format**

- **Symbol file format**

- ♦ **gennames output used in 5.1 (decimal & hexadecimal numbers)**

```

/unix | 00000000| 00000000| 00429ad8| | |initialize
Symbols from /unix
low.s | | |file| | | |
.low | | |0|unamex| | | |.text
pin_obj_start | | |0|extern| | | |.text
start | | |36|extern| | | |.text
.svc1 | | |4128|extern| | | |.text
nonpriv_page | | |12288|extern| | | |.text
g_copyr | | |12288|extern| | | |.text

```

- ♦ **gensyms output used in 5.2 (hexadecimal), format close to nm**

```

Kernel: 00000000 00429ad8 00000000 /unix
Src: low.s
00000000 t .low
00000000 T pin_obj_start
00000024 T start
00001020 T .svc1
00003000 T nonpriv_page
00003000 T g_copyr

```

- ♦ **default 5.2 stripnm format is gensyms format**

- ♦ **-z turns on old gennames format**

Tprof: format changes

The tprof output, as mentioned, is provided in a summary manner where tick counts are replaced by percentages. This gives you a better perspective of the CPU resource being used by a particular process. On routinely formatted reports, what is no longer included are the “Ticks,” “Address,” and “Bytes” columns. However, if you need this information, you can invoke the `-z` flag to return to the old tprof reporting format.

Notice the first segment of tprof output shown here... it comes from the AIX 5.1 version of tprof. The gennames parameter was used, which includes a mix of decimal and hexadecimal numbers. The second segment shown here was produced by the AIX 5.2 tprof command using the gensyms parameter. It is cleaner, all in hexadecimal—in fact, the format resembles nm.

One final note, the default AIX 5.2 stripnm format is the gensyms format. Again, the `-z` flag is used to turn on the older gennames format.



Tprof: modes of operation

- **Realtime: collect data while executing a program**

- `tprof -x` program arguments
 - generates file program.prof with only summary section

- **Manual offline: post-process trace & symbol files**

- `tprof -r` rootstring -juske
 - input is rootstring.trc (from trace) & rootstring.syms (from gensyms)
 - output is rootstring.prof with all sections

- **Automated offline: starts trace, program, collects symbols & trace**

- `tprof -A -x` program arguments
 - output is program.trc, program.syms & program.prof (with only summary section)

- **Post-processing mode : generates new report from cooked files**

- `cooking`
 - `tprof -c -x` program arguments [manual online]
 - `tprof -A -c -x` program arguments [automated offline]
 - `tprof -c -r` program [manual offline]
- all produce program.prof, program.ctr & program.csyms

tprof: modes of operation

In AIX 5.2, tprof can run in the following four modes.

In **realtime** mode, tprof collects data while executing a program.

- `tprof -x` generates a program.prof file that contains only a summary section.
- `tprof -uske -x` generates the file program.prof with all sections (summary, along with use, shared libraries, kernel, and extension).

In **Manual offline** mode, tprof is used for the post-processing trace and symbol file. It reads already generated AIX trace data and produces tprof output.

- For `tprof -r` rootstring -juske, the input is rootstring.trc (from trace) and rootstring.syms (from gensyms). The output is rootstring.prof with all sections.

In **automated offline** mode, tprof starts an AIX trace in the background and logs data to a file. When completed, tprof uses the trace data to produce tprof output.

- `tprof -A -x` generates the output program.trc, program.syms, and program.prof (with only the summary section). The program.syms file contains name mapping information for all the loaded programs, libraries, and kernel extension. The program.trc file is simply a trace output file.

In **post-processing** mode, tprof can post-process data already collected to produce different reports, such as:

- Generate a new report from cooked files.
- Postprocess the cooked files into the AIX 5.1 old format
- Postprocess the cooked files with more details, including subsection details.



Truss update


- **/proc based debugging tool**
 - executes & traces a command or existing process
 - prints names of all system calls made with their arguments & return code
 - system call parameters are displayed symbolically
 - prints information about all signals received by process
- **All base system call parameter types are now recognized.**
 - In AIX 5.1, only 40 system calls were recognized.
- **5.2 version of truss supports library calls tracing**
 - for each call, prints parameters & return code
 - a subset of libraries, and/or routines in a given library, can be selected
 - timestamps on each line
 - all new features to be more compatible with Solaris version

Truss Update

AIX 5L™ supports the **truss** command, which allows you to trace system calls executed by a process, as well as record the received signals and the occurrence of server faults.

The application that is to be traced is either specified on the command line of the **truss** command or **truss** can be attached to one or more already running processes by using the -p flag with a list of process IDs.

With AIX 5.2, the **truss** command can now add timestamps on each output file. It can also trace library calls. For each call, it prints parameters and returns code values. A subset of libraries and/or routines can be selected or excluded from tracing.



Truss: tracing example

•truss -u libc.a::malloc ls

```


execve("/usr/bin/ls", 0x2FF22B98, 0x2FF22B9C)   argc: 1
shlk(0x00000000)                               = 0x200002A8
getuid(4)                                       = 0
loadx(0x01000090, 0x2FF1B970, 0x00000000, 0x2FF22B9C, 0x00000000) = 0x00076130
->libc.a:malloc(0x0)
<-libc.a:malloc() = 20001058      0.000000
->libc.a:malloc(0x180)
<-libc.a:malloc() = 20001078      0.000000
loadx(0x01000100, 0x2FF1B940, 0x00000000, 0xF02A194, 0xF02A0C4) = 0x200112F8
loadx(0x07000000, 0xF02A164, 0xFFFFFFFF, 0x200112F8, 0x00000000) = 0x2001214C
...
->libc.a:malloc(0x28)
<-libc.a:malloc() = 200012a8      0.000000
->libc.a:malloc(0x14)
<-libc.a:malloc() = 200012d8      0.000000
...
_getpid()                                       = 11234
getuid(2)                                       = 0
ioctl(1, 22826, 0x00000000, 0x00000000)       = 0
ioctl(1, 1074295912, 0x2FF22B10, 0x00000000)  = 0
shlk(0x00000000)                               = 0x2001F900
statx(".", 0x2FF22B10, 128, 011)               = 0
fcntl(1, F_GETFL, 0xF02A7370)                  = 2
fcntl(2, F_GETFL, 0x00000000)                  = 67108865
statx(".", 0x2FF22B40, 76, 0)                   = 0
open(".", O_RDONLY)                            = 3
->libc.a:malloc(0x0)
<-libc.a:malloc() = 20002328      0.000000
->libc.a:malloc(0x1001)
<-libc.a:malloc() = 20002428      0.000000
_exit(0)

```

Truss: tracing example

You can look at this sample trace that has been created with the truss command. The `-u` flag indicates that the truss command should dynamically trace loaded user-level function calls. Here, we want to trace the `malloc()` function call in the `libc.a` library while running the List (`ls`) command.

Notice the output isolated in the red blocks on this graphic.



Truss: system call counting

•truss -c ls

syscall	seconds	calls	errors
__mmap	.00	1	
__loadx	.00	17	
__exit	.00	1	
close	.00	2	
write	.02	78	
lseek	.00	1	
__getpid	.00	1	
getuidx	.00	19	
getdirent	.00	3	
ioctl	.00	3	
open	.00	1	
statx	.00	2	
getgidx	.00	18	
statx	.00	4	
access	.00	1	
fcntl	.00	6	
<hr/>			
sys totals:	.04	158	0
usr time:	.00		
elapsed:	.04		

Truss: system call counting

This sample truss command contains a `-c` flag, which indicates that the command should count traced system calls, faults, and signals... rather than displaying the trace results line by line. A summary report is produced after the traced command terminates or when truss is interrupted.

[NOTE: If the `-f` flag has also been used, the counts would have included all traced Syscalls, Faults, and Signals for child processes.]



The -d option

truss -d ls

```
0.0028:  execve("/usr/bin/ls", 0x2FF22980, 0x2FF22988)  argc: 1
0.0388:  sbrk(0x00000000)          = 0x20000EC8
0.0399:  sbrk(0x00000008)          = 0x20000EC8
0.0410:  sbrk(0x00010010)          = 0x20000ED0
0.0421:  getuidx(4)                  = 0
0.0429:  getuidx(2)                  = 0
0.0437:  getuidx(1)                  = 0
0.0445:  getgidx(4)                  = 0
0.0454:  getgidx(2)                  = 0
0.0462:  getgidx(1)                  = 0
0.0472:  __loadx(0x01000080, 0x2FF1E740, 0x00003E80, 0x2FF226D0, 7130
0.0486:  __loadx(0x01000180, 0x2FF1E730, 0x00003E80, 0xF0335458, 1398
0.0516:  __loadx(0x07080000, 0xF0335428, 0xFFFFFFFF, 0x20011398, 2238
0.0523:  __loadx(0x07080000, 0xF0335368, 0xFFFFFFFF, 0x20011398, 2244
.....
```

The -d option specifies display of a timestamp in seconds relative with each line of truss output.

The -d option

Here is yet another sample truss command. The -d option shown here specifies that a timestamp will be included with each line of output. The time displayed is in seconds relative to the beginning of the trace. The first line of the trace output will show the base time from which the individual time stamps are measured. By default, timestamps are not displayed.



The -l option

truss -l ls

```
22483: execve("/usr/bin/ls", 0x2FF22980, 0x2FF22988) argc: 1
22483: sbrk(0x00000000)          = 0x20000EC8
22483: sbrk(0x00000008)          = 0x20000EC8
22483: sbrk(0x00010010)          = 0x20000ED0
22483: getuidx(4)                  = 0
22483: getuidx(2)                  = 0
22483: getuidx(1)                  = 0
22483: __loadx(0x01000080, 0x2FF1E740, 0x00003E80, 0x2FF226D0, 0x00000000) = 0xD0077130
22483: __loadx(0x01000180, 0x2FF1E730, 0x00003E80, 0xF0335458, 0xF0335388) = 0x20011398
22483: __loadx(0x07080000, 0xF0335428, 0xFFFFFFFF, 0x20011398, 0x00000000) = 0x20012238
22483: __loadx(0x07080000, 0xF0335368, 0xFFFFFFFF, 0x20011398, 0x00000000) = 0x20012244
22483: getuidx(4)                  = 0
22483: getuidx(2)                  = 0
.....
```

The -l option specifies that the thread ID for the responsible thread is to be included in the truss output for each event reported.

The -l option

This sample truss command uses the -l (lowercase “L”) option. This option is invoked in order to cause the display of the id (i.e., the thread id) of the responsible Lightweight Processing (LWP) process along with the truss output. By default, the LWP id is not displayed in the output.

[NOTE: The lightweight processes mentioned in truss are actually kernel threads.]



The -f option

truss -f ls

```
19854: execve("/usr/bin/ls", 0x2FF22980, 0x2FF22988)  argc: 1
19854: sbrk(0x00000000)                                = 0x20000EC8
19854: sbrk(0x00000008)                                = 0x20000EC8
19854: sbrk(0x00010010)                                = 0x20000ED0
19854: getuidx(4)                                       = 0
....
19854: kwrite(1, 0xF0377D18, 22)                      = 22
19854: kwrite(1, 0xF0377D18, 1)                      = 1
19854: kwrite(1, 0xF0377D18, 6)                      = 6
19854: kwrite(1, 0xF0377D18, 19)                     = 19
19854: kwrite(1, 0xF0377D18, 24)                     = 24
19854: kwrite(1, 0xF0377D18, 15)                     = 15
19854: kwrite(1, 0xF0377D18, 1)                      = 1
19854: kfcntl(1, F_GETFL, 0x00000001)                 = 2
19854: close(1)                                       = 0
19854: kfcntl(2, F_GETFL, 0x00000000)                 = 67108865
.....
```

The -f option causes truss to follow all children of the specified process or command (including signal, faults, and system calls). By default, truss only reports on actions of first level commands or processes. The process id is included with each line of truss output to show which process executed the system call, received the signal, or incurred the fault

The -f Option

This sample truss command includes the -f option. This option causes truss to follow all children created by the **fork** system call and includes their signals, faults, and system calls in the trace output. Normally, however, only the first-level command or process is traced. When the -f flag is specified, the process id is included with each line of trace output to show which process executed the system call or received the signal.



Other Truss Flags

- i** Keeps interruptible sleeping system calls from being reported
- m** Specifies the machine faults to include in or exclude from the report
The default is **-m all**
- s** Specifies signals to include or exclude from the report. The default is **-s all**.
- t** Specifies system calls to include in or exclude from the report.
The default is **-t all**
- x** Selects raw(usually hexadecimal) display of parametric data from specified system calls

Other Truss Flags

There are five other truss flags that provide valuable additional functionality.

The **-i** option keeps interruptible sleeping system calls from being displayed. Certain system calls on terminal devices or pipes, such as `open` and `kread`, can sleep for indefinite periods and are interruptible. Normally, truss reports on such sleeping system calls if they remain asleep for more than one second. The system call is then reported a second time when it completes. The **-i** flag causes such system calls to be reported only once, upon completion.

The **-m** option traces the machine faults that occur in the process. The machine faults that are to be traced must be separated from each other by a comma. If the list begins with the `!"` symbol, the specified faults are excluded from being traced and are not displayed with the trace output. The default is: `m all -m!fltpage`

The **-s** option permits listing the *signals* that are to be traced or excluded. Those signals specified in a list (separated by a comma) are traced. The trace output reports the receipt of each specified signal, even if the signal is being ignored—but not blocked—by the process. Blocked signals are not received until the process releases them. If the list begins with the `!"` symbol, the listed signals are excluded from being displayed with the trace output. The default is: `-s all`.

The **-t** option includes or excludes system calls from the trace process. System calls that are to be traced must be specified in a list and separated by commas. If the list begins with an `!"` symbol, the specified system calls are excluded from the trace output. The default is: `-t all`.

The **-x** option displays data from the specified parameters of traced system calls in raw format, usually hexadecimal, rather than symbolically. The default is: `x!all`.



Perfstat API: introduction

- Library that provides set of interfaces to easily collect many AIX performance metrics without root authority.
- Lets ISVs easily write portable OS monitoring utilities (from AIX release-to-release).
- 5.1 version provides access to set of basic metrics equivalent to vmstat, iostat, sar & some netstat -D output.
 - system-level processor metrics (sysinfo structure)
 - utilization, syscalls, forks, context switch, ...
 - per-processor metrics (cpuinfo structure)
 - utilization, syscalls, forks, reads, write...
 - system-level memory metrics
 - size, utilization, page faults, ...
 - system-level & per-disk metrics
 - similar to iostat plus size & utilization
 - system-level & per network interface metrics
- Several interfaces include configuration information retrieved from ODM.


Perfstat API: introduction

The **perfstat** application programming interface (API) is a collection of C programming language subroutines that execute in user space and that use the **perfstat** kernel extension to extract various AIX performance metrics. System component information is also retrieved from the Object Data Manager (ODM) and returned with the performance metrics.

The **perfstat** API is both a 32-bit and a 64-bit API, is thread-safe, and does not require root authority. This API also supports extensions so binary compatibility is maintained across all releases of AIX.

Two types of APIs are available. Global types return global metrics related to a set of components, while individual types return metrics related to individual components. Both types of interfaces have similar signatures, but slightly different behavior.

All the interfaces (of both types) return raw data; that is, values of running counters. Multiple calls must be made at regular intervals to calculate rates. Several interfaces return data retrieved from the ODM (object data manager) database.



Perfstat API: global interface example

```

#include <stdio.h>
#include <libperfstat.h>

int main(int argc, char* argv[]) {
    perfstat_memory_total_t minfo;

    perfstat_memory_total(NULL, &minfo, sizeof(perfstat_memory_total_t), 1);

    printf("Memory statistics\n");
    printf("-----\n");
    printf("real memory size           : %llu MB\n",minfo.real_total*4096/1024/1024);
    printf("reserved paging space      : %llu MB\n",minfo.pgsp_rsvd);
    printf("virtual memory size        : %llu MB\n",minfo.virt_total*4096/1024/1024);
    printf("number of free pages       : %llu\n",   minfo.real_free);
    printf("number of pinned pages     : %llu\n",   minfo.real_pinned);
    printf("number of pages in file cache : %llu\n",   minfo.numperm);
    printf("total paging space pages    : %llu\n",   minfo.pgsp_total);
    printf("free paging space pages     : %llu\n",   minfo.pgsp_free);
    printf("used paging space          : %3.2f%%\n",
        (float) (minfo.pgsp_total-minfo.pgsp_free)*100.0/(float)
minfo.pgsp_total);
    printf("number of paging space page ins : %llu\n",   minfo.pgspins);
    printf("number of paging space page outs : %llu\n",   minfo.pgspouts);
    printf("number of page ins          : %llu\n",   minfo.pgins);
    printf("number of page outs         : %llu\n",   minfo.pgouts);
}

```

Perfstat API: global interface example

As just mentioned, there are global API types. These interfaces return global metrics related to a set of components on a server (such as processors, disks, or memory).

All the following AIX 5.2 interfaces use the naming convention **perfstat_subsystem_total**, and use a common signature:

perfstat_cpu_total	Retrieves global CPU usage metrics
perfstat_memory_total	Retrieves global memory usage metrics
perfstat_disk_total	Retrieves global disk usage metrics
perfstat_netinterface_total	Retrieves global network interfaces metrics

Thus, the example you are looking at here is retrieving global memory usage metrics.



Perfstat API: component interface example

```

#include <stdio.h>
#include <stdlib.h>
#include <libperfstat.h>

int main(int argc, char *argv[]) {
    int i, retcode, cputotal;
    perfstat_id_t firstcpu;
    perfstat_cpu_t *statp;

    /* check how many perfstat_cpu_t structures are available */
    cputotal = perfstat_cpu(NULL, NULL, sizeof(perfstat_cpu_t), 0);

    /* allocate enough memory for all the structures */
    statp = calloc(cputotal, sizeof(perfstat_cpu_t));

    /* set name to first cpu */
    strcpy(firstcpu.name, FIRST_CPU);

    /* ask to get all the structures available in one call */
    retcode = perfstat_cpu(&firstcpu, statp, sizeof(perfstat_cpu_t), cputotal);

    for (i = 0; i < retcode; i++) {
        printf("\nStatistics for CPU : %s\n", statp[i].name);
        printf("-----\n");
        printf("CPU user time (raw ticks) : %llu\n", statp[i].user);
        printf("CPU sys time (raw ticks) : %llu\n", statp[i].sys);
        printf("CPU idle time (raw ticks) : %llu\n", statp[i].idle);
        printf("CPU wait time (raw ticks) : %llu\n", statp[i].wait);
        printf("number of syscalls : %llu\n", statp[i].syscall);
        printf("number of readings : %llu\n", statp[i].sysread);
        printf("number of writings : %llu\n", statp[i].syswrite);
        printf("number of forks : %llu\n", statp[i].sysfork);
        printf("number of execs : %llu\n", statp[i].sysexec);
        printf("number of char read : %llu\n", statp[i].readch);
        printf("number of char written : %llu\n", statp[i].writech);
    }
}

```

Perfstat API: component interface example

Component-specific interfaces report metrics related to individual components on a server (such as a processor, disk, network interface, or paging space). All of the following AIX interfaces use the naming convention **perfstat_subsystem**, and use a common signature:

perfstat_cpu	Retrieves individual CPU usage metrics	
perfstat_disk	Retrieves individual disk usage metrics	
perfstat_diskpath	Retrieves individual disk path metrics	New in AIX 5.2
perfstat_diskadapter	Retrieves individual disk adapter metrics	
perfstat_netinterface	Retrieves individual network interfaces metrics	
perfstat_protocol	Retrieves individual network protocol-related metrics	New in AIX 5.2
perfstat_netbuffer	Retrieves individual network buffer allocation metrics	
perfstat_pagingspace	Retrieves individual paging space metrics	

Thus, the example you see here is retrieving individual CPU usage metrics.



Perfstat API: 5.2

- **New metrics**

- Processor metrics now include all cpuinfo & sysinfo fields

- **New interfaces**

- gather paging spaces metrics
- report on network memory allocations (similar to netstat -m)
- report disk adapter metrics (same metrics as disks)
- collect all network protocol metrics
- new perfstat_diskpath interface
- perform updates to perfstat_disk & perfstat_disk_adapter

- **See <libperfstat.h> for complete list of interfaces & metrics**

Perfstat API: AIX 5.2

Perfstat has been enhanced in AIX 5.2 to deliver new processor metrics that now include all cpuinfo and sysinfo fields.

There are new interfaces, too, as is listed on this graphic.

The perfstat API subroutines reside in the libperfstat.a library, where you can also find the interface declarations and type definitions of the data structures to use when calling the interfaces. Detailed information for the individual interfaces and the data structures used can be found in the libperfstat.h file in the *AIX 5L Version 5.2 Files Reference* manual.



5.2 Performance Tools—Summary

- Prior to 5.2...**
 - Performance commands that were inconsistent & limited in functionality
- Now with 5.2...**
 - Command consistency
 - Usability improvements
 - Greater functionality
 - Support for permanent & reboot values (tunable parameters)

Conclusion

Before the release of AIX Version 5.2, IBM® eServer® pSeries™ systems administrators and developers did not have an easy job of tuning the server for optimal performance. The performance commands were limited in functionality, they were inconsistent in their syntax, or were even non-existent.

During the development of the latest AIX release, however, a strong commitment was made toward providing pSeries devotees with much more useful and powerful tuning mechanisms. This course has spent all of its focus on helping you better understand the new wealth of performance tuning devices at your fingertips.

There is greater consistency in both the use of command parameters and their effects. There is also a significant increase in the high-level functionality of the performance tools. And, importantly, the tunable parameters you set can be specified as permanent reboot values... so you do not have to retune the server after each restart.

There is a great deal of additional information that is available on the IBM Web site regarding everything discussed in this course. Please look at the references section of this course for a jump-start on where you can gain an even greater understanding of the new power and flexibility you have to tune your pSeries server. Optimal performance may be just a few “commands” away.

References

- pSeries Information Center
publib16.boulder.ibm.com/pseries/en_US/infocenter/base/
 - AIX 5L Version 5.2 Performance Tools Guide and Reference
 - AIX 5L Version 5.2 Performance Management Guide
 - AIX 5L Version 5.2 Commands Reference, Volume 1-6
- AIX 5.2 Performance Tools Update, part 1
ibm.com/developerworks/eserver/articles/Keung_AIXPerf.html
- AIX 5.2 Performance Tools Update, part 2
ibm.com/developerworks/eserver/articles/AIX5.2PerfTools.html
- AIX 5.2 Performance Tools Update, part 3
ibm.com/developerworks/eserver/articles/AIX5.2_performance_toolsupdatepart3.html
- An online course, entitled “IBM eServer pSeries Logical Partitioning: Installation & Configuration,” explains the process of installing the WebSM client in Windows
ibm.com/servers/enable/site/peducation/abstracts/abs_1a8a.html

Trademarks

IBM, eServer, AIX, pSeries, and AIX5L are registered trademarks of IBM Corporation in the United States, other countries, or both.

Java, Sun, Solaris, and all Java-based trademarks are trademarks or service marks of Sun Microsystems in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

IBM makes no commitment to make available any products referred to herein.

All other registered trademarks and trademarks are properties of their respective owners.

References in this publication to IBM products or services do not imply that IBM intends to make them available in every country in which IBM operates.