AIX версия 7.2

Настройка производительности

AIX версия 7.2

Настройка производительности

Примечание

Перед началом работы с этим изданием и описанным в нем продуктом ознакомьтесь с информацией, приведенной в разделе “Примечания” на стр. 463.

Данное издание относится к AIX версии 7.2, а также ко всем последующим выпускам и модификациям, если в соответствующих изданиях не будет оговорено обратное.

Содержание

Об этом документе

В данном наборе разделов приведены сведения для программистов, инженеров разработчика, системных инженеров, системных администраторов, технических специалистов, конечных пользователей, а также системных программистов, о настройке производительности процессоров, файловых систем, памяти, ввода-вывода дисков, NFS, Java и ввода-вывода линий связи. В этом наборе разделов описано также эффективное проектирование систем и приложений, включая их реализацию. Данный набор разделов можно найти и на компакт-диске документации, который поставляется вместе с операционной системой.

Специальные обозначения

В этом документе применяются следующие способы выделения текста:

Полужирный	Полужирным шрифтом выделены команды, процедуры, ключевые слова, имена файлов, структуры, каталоги и прочие объекты системы с предопределенными именами. Кроме того, полужирным шрифтом выделены графические объекты: кнопки, метки и значки, которые могут быть выбраны пользователем.
Курсив	Курсивом выделены параметры, фактические имена и значения, которые указываются пользователем.
`Непропорциональный`	Непропорциональным шрифтом выделены примеры данных, текст, появляющийся на экране компьютера, примеры программного кода, системные сообщения и вводимая пользователем информация.

Учет регистра символов в AIX

В операционной системе AIX учитывается регистр символов, т.е. различаются прописные и строчные буквы. Например, с помощью команды **ls** можно просмотреть список файлов. При вводе команды `LS` отобразится сообщение `Команда не найдена`. Точно так же, **FILEA**, **FiLea**, и **filea** - это имена трех различных файлов, даже если эти файлы находятся в одном каталоге. Для достижения ожидаемых результатов всегда указывайте строковые значения с учетом регистра символов.

ISO 9000

При разработке и производстве данного продукта использовались зарегистрированные системы ISO 9000.

Руководство по настройке производительности

В данном разделе приведены сведения для программистов, инженеров разработчика, системных инженеров, системных администраторов, технических специалистов, конечных пользователей, а также системных программистов о настройке производительности процессоров, файловых систем, памяти, дисков, ввода-вывода, NFS, Java и документации. Также данный набор разделов адресован опытным системным разработчикам и разработчикам приложений. Данный раздел можно найти и на компакт-диске документации, который поставляется вместе с операционной системой.

Примечание: Показатели, отслеживаемые инструментами сбора статистики, такими как **lparstat**, **vmstat**, **iostat** и **mpstat**, включая приложения на основе API Perfstat и API SPMI, могут отличаться в случае параллельного выполнения с одним и тем же интервалом выборки в конкретный момент времени.

Новое в книге Управление производительностью

Описание существенных изменений в разделах, посвященных управлению производительностью.

Как узнать об изменениях и добавлениях

В этом файле PDF дополнения и изменения могут обозначаться символами вертикальной черты (|) в левом поле.

Апрель 2017 года

- Добавлена информация о параметре *Ограничение ожидающих обработки запросов Fibre Channel* в раздел “Настраиваемые параметры диска и адаптера” на стр. 449.

Декабрь 2016 года

- Обновлена информация о переменной среды *AIXTHREAD_SCOPE* в разделе “Переменные среды нити” на стр. 72.
- Удалена информация о параметре `mempools`, поскольку он не поддерживается в AIX 6.1 и выше.

Октябрь 2016 года

Ниже приведено краткое описание изменений, внесенных в разделы, посвященные тонкой настройке производительности:

- Добавлена информация о разделе “Выгрузка прерванных нитей VMM” на стр. 178.

Основы настройки производительности

Для оценки системной производительности нужно понимать динамику выполнение программы.

Рабочая схема

Для оценки и настройки производительности системы важно правильно понимать, что вкладывается в понятие рабочей схемы.

Изменение рабочей схемы зачастую оказывает намного большее влияние на производительность системы, чем изменение быстродействия процессора или размера оперативной памяти. Понятие рабочей схемы включает в себя не только тип и число запросов к системе, но и набор установленных пакетов программного обеспечения, а также набор локальных прикладных программ.

Важно учитывать работу системы в фоновом режиме. Например, если некоторые файловые системы могут удаленно монтироваться с помощью NFS, и к ним часто обращаются пользователи других систем, то значительная часть ресурсов системы может затрачиваться на обработку запросов к этим файловым системам, даже если эта система не считается сервером.

Рабочая схема, позволяющая сравнивать производительность различных систем называется *Тестом*. Однако реальные нагрузки редко повторяют алгоритмы и требования стандартного теста. Даже те тесты, которые были созданы на основе реальных приложений, часто упрощаются и стандартизируются таким образом, чтобы они могли выполняться на различных аппаратных платформах. Единственное применение стандартных тестов производительности - это грубый отбор систем для дальнейшего тщательного тестирования. Таким образом, в процессе анализа рабочей схемы и производительности системы не следует полностью полагаться на результаты тестов.

Рабочие схемы можно разделить на следующие категории:

Многопользовательская среда
: Рабочая схема, в которой несколько пользователей работают с разными терминалами. Как правило, критерием производительности является объем выполняемой работы при фиксированном минимальном времени отклика или время отклика для стандартного объема работы.

сервера
: Рабочая схема, в которой система отвечает на запросы других систем. Например, работа файлового сервера в основном состоит из выполнения запросов на чтение с диска и запись на диск. Файловый сервер отвечает за дисковый ввод-вывод в многопользовательской среде (помимо него могут применяться NFS и другие средства ввода-вывода), поэтому к нему также применим критерий в виде объема выполняемой работы при фиксированном времени отклика. В других рабочих схемах сервер может применяться для выполнения программ, требующих сложных вычислений, обработки транзакций, заданий печати и так далее.

Рабочая станция
: Рабочая схема, в которой пользователь вводит запросы с помощью клавиатуры, а результаты выполнения запросов выводятся на экран монитора. Как правило, главным критерием производительности для такой рабочей схемы является время отклика на запросы пользователя.

Требования к производительности

После определения рабочей схемы, в которой будут исследоваться системы, вы можете выбрать критерий, по которому системы будут оцениваться, и установить цели отбора на основе этого критерия.

Основными критериями оценки производительности является время ответа и эффективность работы.

Время ответа - это время, через которое выдается ответ на отправленный запрос. Например:

- Время, необходимое для выполнения запроса к базе данных
- Время, необходимое для отображения символов на терминале
- Время, необходимое для доступа к Web-станице

Производительность определяется объемом работы, выполняемой за единицу времени. Например:

- Число транзакций базы данных, выполняемых в минуту
- Скорость передачи файла (КБ/с)
- Скорость чтения или записи файла (КБ/с)
- Число обращений к Web-серверу в минуту

Связи между этими показателями достаточно сложны. Довольно часто можно добиться высокой производительности системы за счет увеличения времени ответа, либо уменьшения времени ответа за счет снижения производительности. В других случаях определенные изменения могут улучшить обе величины. Оптимальной считается настройка, при которой значения обоих показателей приемлемы.

При подготовке к настройке системы определите время ответа и производительности системы, которых необходимо добиться при обработке рабочей схемы. В противном случае вы рискуете потратить время и деньги на оптимизацию несущественных аспектов быстродействия системы.

Модель выполнения программы

Для того чтобы получить правильное представление о параметрах рабочей схемы, необходимо использовать динамическую, а не статическую модель выполнения программы. Она показана на следующем рисунке.

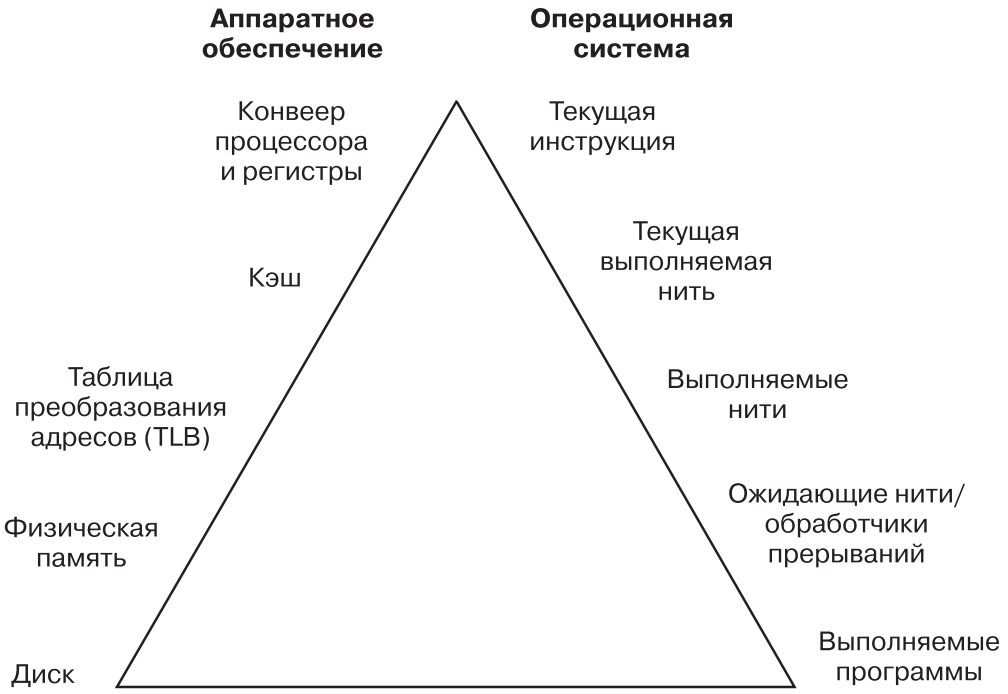


Рисунок 1. Иерархия выполнения программы. Схема представляет собой треугольник. С левой стороны перечислены компоненты аппаратного обеспечения, соответствующие компонентам операционной системы, показанным с правой стороны. Программа переходит с нижнего уровня, на котором она представляет собой набор данных, хранящихся на диске, на верхний уровень, на котором инструкции программы выполняются процессором. Например, если перечислить все уровни, начиная с нижнего, то на диске хранятся исполняемые программы, в оперативной памяти хранятся ожидающие нити операционной системы и обработчики прерываний, в буфере таблицы преобразования адресов хранятся нити, готовые к выполнению, в кэше хранятся нити, переданные на выполнение, а в регистрах и конвейере процессора находятся выполняемые в данный момент инструкции.

Для выполнения программа должна параллельно пройти все этапы аппаратной и программной иерархии. Каждый элемент в аппаратной иерархии имеет большую ценность, чем элемент, расположенный ниже него. Программа конкурирует с другими программами за каждый ресурс, а переход с одного уровня на другой занимает некоторое время. Для того чтобы понять, в чем заключается процесс выполнения программы, нужно иметь представление о каждом из уровней иерархии.

Аппаратная иерархия

Обычно время, необходимое для перехода с одного аппаратного уровня на другой, зависит от времени ожидания на нижнем уровне. Под временем ожидания понимается время, проходящее с момента отправки запроса до получения первых данных.

Жесткие диски

При выполнении программы больше всего времени затрачивается на считывание исходного кода и данных программы с жесткого диска. Это время тратиться на выполнение следующих действий:

- Контроллеру накопителя должен быть передан запрос на обращение к определенным блокам диска (время помещения в очередь).
- Считывающие головки диска необходимо подвести к правильному цилиндру (время поиска).
- Необходимо дождаться, пока нужный блок появится под считывающими головками (время поворота цилиндра).
- Данные должны быть переданы контроллеру (время передачи) и отправлены приложению (время обработки прерывания).

Низкая производительность обращений к диску может быть вызвана не только явными запросами прикладной программы. Иногда настройка системы может привести к выполнению ненужных операций дискового ввода-вывода.

Физическая память

Обращения к оперативной памяти (RAM) обрабатываются намного быстрее, чем обращения к диску, однако оперативная память значительно дороже дисковой. Операционные системы стараются хранить в оперативной памяти только те инструкции и данные, которые используются в текущий момент, выгружая ненужную информацию на диск или вообще не загружая ее в память.

Скорость работы оперативной памяти не всегда соответствует скорости процессора. Задержка между обращением аппаратного обеспечения к оперативной памяти и получением данных или инструкций процессором может составлять порядка десяти тактов процессора.

При обращении к странице виртуальной памяти, выгруженной на диск или еще не загруженной в оперативную память, возникает страничная ошибка, и выполнение программы прерывается до тех пор, пока страница не будет считана с диска.

Таблица преобразования адресов (TLB)

Одним из способов освобождения программ от физических ограничений системы является использование виртуальной памяти. Вы можете разработать и закодировать приложение таким образом, как если бы объем памяти был очень большим, а система, в свою очередь, будет отвечать за преобразование виртуальных адресов инструкций и данных в адреса физической памяти. Для того чтобы сократить время, затрачиваемое на преобразование адресов, физические адреса тех страниц виртуальной памяти, к которым обращались в последнее время, хранятся в кэше, который называется таблицей преобразования адресов (TLB).

До тех пор, пока программа обращается к небольшому числу страниц виртуальной памяти, полное преобразование виртуальных адресов не производится. Если программа обращается к странице, для которой в таблице TLB отсутствует запись (такая ситуация называется *промах TLB*), то на преобразование адреса затрачивается несколько десятков тактов процессора (*время обработки промаха TLB*).

Кэш

Для минимизации задержек, связанных с обращением к оперативной памяти, в системе предусмотрен кэш для команд и данных. Если необходимые команды или данные находятся в кэше, то в результате попадания они будут переданы процессору уже на следующем такте без дополнительной задержки. В случае промаха возникнет задержка.

В некоторых системах кэши расположены на двух или трех уровнях. Обычно они называются кэшами L1, L2 и L3. Если необходимые данные не будут найдены в кэше L1, то система попытается найти их в кэше L2. В случае промаха в кэше L2 система переходит на уровень L3, если он существует, или к оперативной памяти.

Размер и структура кэша зависит от модели системы, однако принцип его работы одинаковый.

Конвейер и регистры

При соблюдении некоторых условий конвейерная архитектура дает возможность одновременно обрабатывать несколько инструкций. Большой набор регистров общего назначения и регистров для чисел с плавающей точкой позволяет хранить значительную часть данных программы в регистрах, без необходимости постоянного обращения к памяти.

При оптимизации программы компилятор старается задействовать максимальное число регистров. При компиляции программ всегда следует включать функцию оптимизации, независимо от размера программы. Более подробная информация об эффективном использовании компиляторов приведена в книге *Optimization and Tuning Guide for XL Fortran, XL C and XL C++*.

Программная иерархия

Во время выполнения программа проходит ряд уровней в программной иерархии.

Исполняемые программы

При запуске программы операционная система выполняет ряд действий для преобразования кода и данных этой программы, хранящихся на диске, в работающую программу.

Сначала выполняется поиск требуемой копии программы в соответствии с текущей переменной среды *PATH*. Затем загрузчик программы (не путайте с редактором связей - командой **ld**) преобразует внешние ссылки программы на общие библиотеки.

В ответ на получение запроса операционная система создает процесс, или набор ресурсов, таких как частный сегмент виртуальной памяти, необходимых для выполнения программы.

Операционная система дополнительно создает нить процесса. *Нить* определяет текущее состояние выполнения отдельного экземпляра программы. В операционной системе AIX время процессора и другие ресурсы выделяются отдельным нитям, а не всему процессу. Прикладная программа может создавать несколько нитей внутри одного процесса. Такие нити совместно используют ресурсы процесса, в котором они запущены.

Наконец, система выполняет переход на точку входа программы. Если страница, содержащая точку входа, еще не загружена в память (это может произойти в случае, если программа недавно компилировалась, запускалась или копировалась), происходит страничная ошибка, и страница загружается в память с диска.

Обработчики прерываний

Для оповещения операционной системы о каком-либо внешнем событии применяется механизм прерываний, который позволяет приостановить выполнение текущей нити и передать управление обработчику прерываний.

Перед запуском обработчика прерываний сохраняется информация о состоянии аппаратных компонентов, чтобы после обработки прерывания система могла восстановить среду выполнения нити. На работе обработчика прерываний сказываются те же задержки, связанные с перемещением по аппаратной иерархии, что и на работе обычных программ (исключение составляют страничные ошибки). Если обработчик прерываний вызывался достаточно давно (и программа не является крайне экономной по отношению к ресурсам), то маловероятно, что его код или данные сохранятся в TLB или кэш-памяти.

Когда прерванная нить снова передается на выполнение, ее контекст (в частности, содержимое регистров) восстанавливается, поэтому она может продолжить свою работу. При этом содержание TLB и кэш-памяти формируется на основе последующих запросов программы. Таким образом, и в обработчике прерываний, и в прерванной программе будут возникать серьезные задержки, связанные с промахами при обращении к кэшу и TLB.

Ожидающие нити
Если запрос программы нельзя обработать немедленно, (например, операцию синхронного ввода-вывода нельзя выполнить из-за страничной ошибки), то до окончания обработки запроса нить переходит в состояние ожидания.

Как правило, это приводит к дополнительным задержкам при работе с TLB и кэшем, не считая самого времени обработки запроса.

Нити, готовые к выполнению
Нить, готовая к выполнению, но не выполняющаяся в конкретный момент, фактически простаивает. Кроме того, для выполнения текущих нитей в кэше могут быть удалены данные нити, ожидающей передачи на выполнение, а из оперативной памяти могут быть выгружены ее страницы данных. Следовательно, когда нить будет передана на выполнение, возникнут дополнительные задержки при обращении к кэшу и оперативной памяти.

Нити, выполняемые в данный момент
Планировщик выбирает для выполнения нить, которая имеет максимальные потребности по использованию ресурсов процессора.

Алгоритм выбора нитей для их передачи на выполнение описан в разделе “Производительность планировщика процессора” на стр. 39. Когда нить передается на выполнение, восстанавливается то состояние процессора, которое было в тот момент, когда ее выполнение было в последний раз прервано.

Текущие машинные инструкции
Если при обращении к TLB или кэшу не возникают промахи, то большинство машинных инструкций выполняются за один такт процессора.

Однако, если часто выполняется переход к различным частям программы, а данные считываются из различных областей памяти, то при обращении к TLB и кэшу возникает большое число промахов. В результате число тактов процессора, затрачиваемых на выполнение одной инструкции (CPI), может быть намного больше единицы. В этом случае говорят, что в программе ссылки расположены не компактно. Даже если программа содержит небольшое число инструкций, на их выполнение может быть затрачено довольно много тактов процессора. Это одна из причин, по которой нельзя оценить время выполнения программы, подсчитав число инструкций. Как правило, чем короче программа, тем быстрее она выполняется, однако размер программы не прямо пропорционален времени ее выполнения.

Компилятор оптимизирует код программы таким образом, чтобы минимизировать число тактов процессора, необходимых на выполнение программы. Для того чтобы добиться максимальной производительности программы, нужно предоставить компилятору полную информацию, необходимую для эффективной оптимизации кода, а не строить предположения о том, какой способ оптимизации будет выбран компилятором (за дополнительной информацией обратитесь к разделу Повышение эффективности работы препроцессоров и компиляторов). Настоящим показателем производительности может служить только скорость обработки конкретной рабочей схемы.

Настройка системы
После эффективной реализации прикладных программ дальнейшая оптимизация производительности связана с настройкой системы.

Ниже приведен список компонентов, которые настраиваются на системном уровне:

Средства связи
В зависимости от характера рабочей схемы и типа канала связи, может потребоваться настроить один или несколько из следующих драйверов устройств связи: службы TCP/IP и NFS.

Жесткие диски
Администратор логических томов (LVM) управляет размещением файловых систем и пространства

подкачки на диске. От выбранного способа размещения в значительной степени зависит время обработки обращений к диску. Драйверы дисков определяют порядок обработки запросов на чтение и запись данных.

Физическая память
Администратор виртуальной памяти (VMM) отвечает за пополнение пула свободных страниц оперативной памяти и определяет, какие страницы можно принудительно освободить для пополнения пула.

Выполняемая нить
Следующая нить, которой должно быть передано управление, выбирается планировщиком. В AIX управление ресурсами осуществляется на уровне нитей. См. раздел “Поддержка нитей” на стр. 39.

Настройка производительности

Выполнение настройки производительности системы и рабочей схемы очень важно.

Введение в настройку производительности

Настройка производительности системы в основном заключается в перераспределении ресурсов и настройке параметров системы.

Изменение рабочей схемы и конфигурации системы для повышения эффективности использования ресурсов можно разбить на следующие этапы:

1. Определение рабочей схемы системы
2. Определение целей:
   a. Определение способов измерения параметров
   b. Оценка и выбор наиболее приоритетных объектов для оптимизации
3. Определение наиболее важных ресурсов, ограничивающих производительность системы
4. Снижение потребности в ресурсах, ограничивающих производительность:
   a. Использование оптимального ресурса, если есть выбор
   b. Снижение потребности отдельных программ и функций системы в ресурсах, ограничивающих производительность
   c. Обеспечение параллельного использования ресурсов
5. Изменение правил выделения ресурсов с учетом приоритетов
   a. Изменение приоритетов и ограничений на использование ресурсов отдельными программами
   b. Изменение параметров управления ресурсами системы
6. Повторение действий 3- 5 до тех пор, пока не будут достигнуты поставленные цели (или не будут исчерпаны возможности настройки)
7. При необходимости, увеличение объема ресурсов

Для выполнения описанных действий по настройке производительности системы предусмотрены специальные инструменты (дополнительная информация приведена в разделе “Отслеживание и настройка производительности, команды и функции” на стр. 405). Некоторые из этих инструментов выпускаются фирмой IBM®. На приведенном ниже рисунке проиллюстрированы различные этапы настройки производительности системы, подключенной к сети LAN.

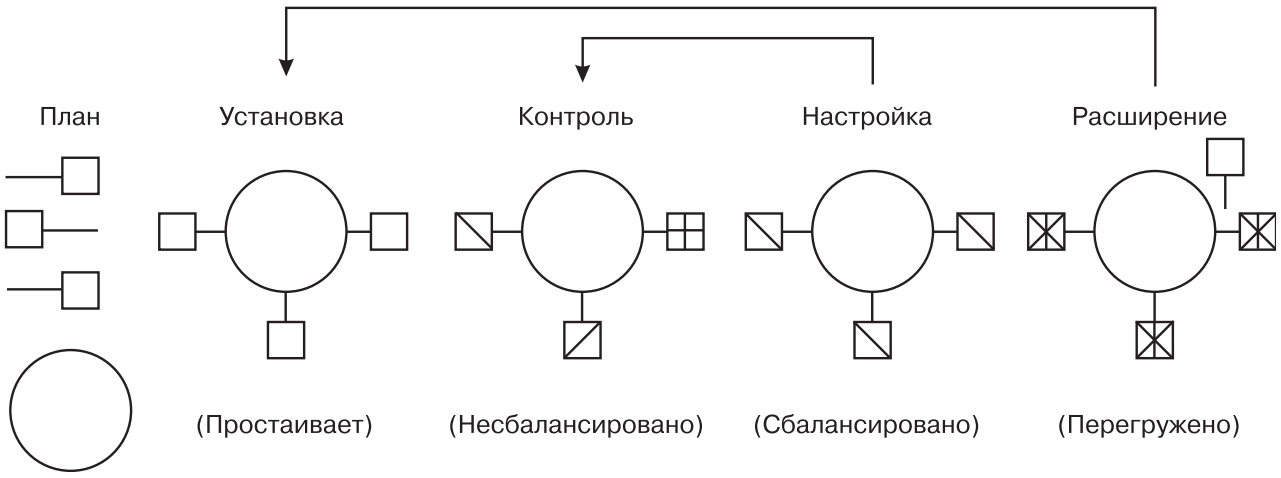


Рисунок 2. Этапы настройки производительности. На этом рисунке этапы настройки производительности системы изображены в виде пяти окружностей: планирование, установка, отслеживание, настройка и наращивание ресурсов. Каждая окружность представляет различные состояния системы: простаивает, несбалансирована, сбалансирована и перегружена. Обратите внимание, что если система перегружена, то требуется увеличить объем ее ресурсов, если нужно сбалансировать нагрузку, то требуется настроить параметры системы, если система несбалансирована, то требуется собрать информацию о ее производительности, а если нужно увеличить мощность системы, то требуется установить дополнительные ресурсы.

Определение рабочих нагрузок

Необходимо учесть всю работу, которая выполняется системой. В системах, подключенных к локальной сети, может возникнуть крайне сложная структура смонтированных каталогов, созданная исключительно на основе устных договоренностей пользователей. Такие файловые системы также необходимо учесть при настройке производительности.

В многопользовательских средах необходимо определить как среднюю, так и максимальную нагрузку. Кроме того, нужно максимально точно оценить время, в течение которого пользователь вводит данные и ожидает вывода результата.

При определении рабочей схемы в первую очередь нужно решить, в какой системе будут проводиться измерения и настраиваться параметры: в рабочей системе или в тестовой системе с аналогичной рабочей схемой. Аналитик должен оценить, что важнее - точные результаты измерений в рабочей системе или гибкость среды, в которой можно проводить рискованные эксперименты.

Важность постановки задач

Хотя в некоторых случаях поставленные перед собой задачи можно описать количественно, часто цель настройки можно оценить только субъективно, например, если требуется получить приемлемое время ответа. При этом не следует изменять только те показатели, которые легко измерить, вместо того, что действительно важно. Если требуется улучшить те аспекты работы системы, которые нельзя измерить с помощью средств, предусмотренных в системе, то подобные измерения нужно выполнить самостоятельно.

Самым главным аспектом определения целей является не выбор оптимальных значений, а выбор приоритетов различных целей оптимизации. Пока эти приоритеты не будут определены и согласованы со всеми заинтересованными лицами, аналитик не сможет выбрать правильную стратегию настройки, поскольку ему придется непрерывно вести консультации с другими людьми. Часто пользователи и администраторы бывают неприятно удивлены тем, что некоторые аспекты производительности были проигнорированы. Если настройка системы затрагивает не только интересы сотрудников организации, то может потребоваться заключить письменное соглашение между поставщиками услуг и пользователями, в котором будут описаны все цели настройки производительности и расставлены приоритеты.

Определение наиболее важных ресурсов

Как правило, эффективность обработки рабочей схемы зависит от объема и быстродействия одного или двух наиболее важных ресурсов системы. Аналитик должен точно выявить такие ресурсы, иначе ему придется действовать методом проб и ошибок.

В системах имеются физические, логические и, возможно, виртуальные ресурсы. Среди физических ресурсов проще выделить те, которые влияют на производительность системы, поскольку в системе предусмотрено много средств для оценки использования физических ресурсов. Чаще всего производительность системы зависит от следующих физических ресурсов:

- Быстродействие процессора
- Оперативная память
- Шина ввода-вывода
- Различные адаптеры
- Дисковое пространство
- Сеть

Определить важные логические ресурсы сложнее. Логические ресурсы обычно представляют собой абстракции, позволяющие разделить физические ресурсы. Такое разделение упрощает задачи администрирования и применение общих ресурсов.

Можно использовать виртуальные ресурсы в системах IBM System p, работающих на базе POWER5, включая микроразделение, виртуальный последовательный адаптер, виртуальный интерфейс SCSI и виртуальный Ethernet.

Ниже приведены некоторые примеры логических и виртуальных ресурсов, созданных на основе физических ресурсов:

процессор

- Квант процессорного времени
- Разделение прав на использование CPU или микроразделение
- Виртуальный Ethernet

Оперативная память

- Страницы памяти
- Стеки
- Буферы
- Очереди
- Таблицы
- Блокировки и семафоры

Дисковое пространство

- Логические тома
- Файловые системы
- Файлы
- Логические разделы
- Виртуальный интерфейс SCSI

Сеть

- Сеансы
- Пакеты

- Каналы
- Общий Ethernet

Логические и виртуальные ресурсы необходимо учитывать наравне с физическими. Работа нитей с равной вероятностью может быть заблокирована как из-за отсутствия логического ресурса, так и из-за нехватки физического ресурса. Увеличение объема физического ресурса, на основе которого создан логический ресурс, не гарантирует, что будут созданы дополнительные логические ресурсы. Например, для обработки каждого ожидающего удаленного запроса NFS на ввод-вывод на сервере применяется отдельный демон сервера NFS (демон **nfsd**). В связи с этим, число демонов **nfsd** ограничивает число операций ввода-вывода NFS, выполняемых системой параллельно. С помощью системных функций вы можете проверить, что когда в системе нет свободных демонов **nfsd**, различные физические ресурсы (например, процессор) загружены весьма незначительно. Может возникнуть ложное впечатление, что система не полностью загружена, хотя на самом деле в системе не хватает демонов **nfsd**, что не дает возможности использовать другие ресурсы системы. Для работы демона **nfsd** необходимы ресурсы процессора и оперативная память, однако в описанной ситуации проблему нельзя решить путем увеличения объема физической памяти или замены процессора на более мощный. Вместо этого нужно создать дополнительные логические ресурсы ,демоны **nfsd**.

При разработке приложений могут случайно создаваться логические ресурсы или возникать узкие места. Например, логический ресурс может быть связан со способом передачи данных или управления устройством. Для таких логических ресурсов обычно отсутствует способ получения информации об их использовании и интерфейсы для управления за их выделением. Такие ресурсы могут недооцениваться до тех пор, пока они не станут причиной снижения производительности системы.

Снижение потребности в ресурсах, ограничивающих производительность

Снижение потребности в ресурсах, ограничивающих производительность, можно разбить на три этапа.

Использование подходящих ресурсов:

Решение о выборе ресурса должно приниматься сознательно, с учетом поставленных целей.

Например, при разработке приложения иногда приходится делать выбор между увеличением объема используемой памяти и увеличением нагрузки на процессор. При настройке сети часто приходится принимать решение о том, будут ли файлы храниться централизованно на удаленном сервере или локально на каждой рабочей станции.

Снижение потребности в ресурсах:

Для оптимизации собственных приложений можно повысить эффективность используемых функций или удалить ненужные функции.

Менее приоритетные рабочие схемы, на обработку которых затрачивается часть ресурсов, можно перенести в другие системы, выполнять в другое время или настроить с помощью приложения WLM.

Параллельное использование ресурсов:

Поскольку для выполнения рабочей схемы требуется несколько ресурсов, и эти ресурсы не зависят друг от друга, работа с ними может вестись параллельно.

Например, если программа последовательно считывает данные из файла, то включается алгоритм упреждающего чтения, который считывает из файла дополнительные данные, в то время как приложение обрабатывает текущие данные. Параллелизм может применяться и на уровне управления системой. Например, если приложение одновременно работает с несколькими файлами, то для повышения эффективности дискового ввода-вывода можно добавить еще один диск и разместить файлы приложения на разных дисках.

Приоритет выделения ресурсов

В операционной системе предусмотрено много способов для задания приоритета операций.

Некоторые из них, например, ограничение ввода-вывода, применяются на уровне системы. Другие способы, такие как установка приоритетов процессов, могут использоваться отдельными пользователями для того, чтобы указать важность той или иной задачи.

Повторение этапов настройки

Процесс анализа производительности бесконечен, поскольку с течением времени в системе всегда появляется другой ресурс, ограничивающий производительность. Снижение потребности в одном ресурсе означает, что теперь пропускная способность и время отклика будут ограничиваться другим ресурсом.

Предположим, что в системе ресурсы используются следующим образом:

Процессор: 90% Диск: 70% Память: 60%

Работа такой системы зависит от скорости процессора. Если вам удастся настроить рабочую схему таким образом, что уровень использования процессора сократится с 90 до 45 процентов, то вы вправе ожидать, что производительность возрастет в два раза. Однако на деле получится, что после этого производительность системы будет ограничена скоростью дискового ввода-вывода, а показатели использования ресурсов станут примерно следующими:

Процессор: 45% Диск: 90% Память: 60%

Снижение нагрузки на процессор позволяет программам быстрее передавать запросы на общение к диску, в результате чего может быть достигнуто ограничение на использование диска. В результате производительность возрастет всего лишь на 30 процентов, а не на 100.

Ресурсы, которые ограничивают производительность, есть всегда. Важно лишь то, достигнуты ли цели оптимизации с помощью имеющихся в наличии ресурсов.

Внимание: Неправильная настройка системы с помощью команд **vmo**, **ioo**, **schedo**, **no** и **nfso** может привести к непредсказуемым результатам - в частности, к снижению скорости выполнения программ и зависанию системы. Изменять эти параметры следует только в случае, если возникшее узкое место точно локализовано.

Примечание: Общих рекомендаций по применению параметров настройки производительности нет.

Увеличение объема ресурсов

Если, испробовав все описанные выше способы, вы не смогли добиться высокой производительности системы, необходимо увеличить объем ресурсов, ограничивающих производительность.

Если производительность системы ограничена логическим ресурсом, то вы можете увеличить объем этого ресурса без дополнительных затрат, при условии, что объем базового физического ресурса достаточен. Если производительность ограничена физическим ресурсом, аналитик должен ответить на следующие вопросы:

- Насколько требуется расширить ресурс?
- Что произойдет - производительность системы повысится или она станет зависеть от объема другого ресурса?
- Если производительность станет зависеть от объема другого ресурса, то что выгоднее - увеличить объем всех ресурсов, ограничивающих производительность, либо перенести часть задач, выполняемых в текущей системе, в другую систему?

Тесты для оценки производительности

Сравнение скорости работы некоторого компонента программы в различных средах часто приводит к возникновению различного рода ошибок. В этом разделе приведена информация о том, как избежать таких ошибок. Информация о различных способах вычисления времени работы программы приведена в других разделах.

При измерении времени обработки системного вызова пользователь получает значение, учитывающее следующие компоненты:

- Собственно время выполнения инструкций, необходимых для обработки вызова.
- Задержки, связанные с получением инструкций и данных из оперативной памяти (то есть задержки, связанные с промахами при обращении к кэшу и TLB)
- Время, необходимое для считывания показаний системных часов в начале и конце вызова
- Время, затраченное на обработку стандартных событий, таких как прерывания по таймеру системы
- Время, затраченное на обработку случайных событий, таких как прерывания ввода-вывода

Для того чтобы избежать случайных погрешностей, измерения обычно выполняются несколько раз. Поскольку к фактическому времени обработки системного вызова добавляется еще несколько величин, зависящих от внешних факторов, обычно по результатам измерений строят кривую, пример которой приведен на следующем рисунке.

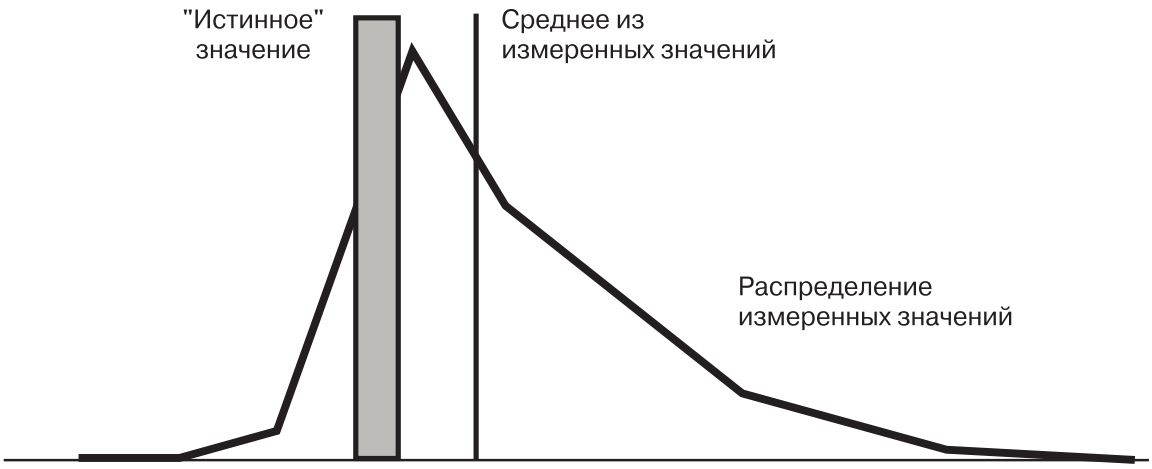


Рисунок 3. Кривая на основе набора измерений.

Крайне низкий уровень может соответствовать маловероятной ситуации оптимального кэширования или являться следствием ошибки округления.

Если во время обработки системного вызова регулярно возникали внешние события, то у полученной кривой может быть два максимальных значения, как показано на следующем рисунке.

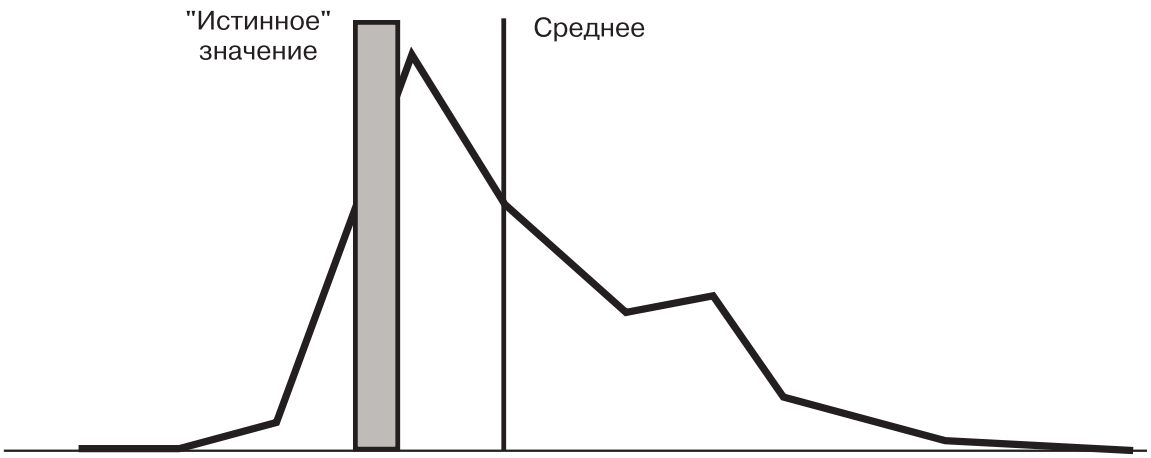


Рисунок 4. Кривая с двумя вершинами

Если во время обработки системного вызова возникло несколько прерываний, на обработку которых было затрачено значительное время, то кривая может исказиться еще больше, как показано на следующем рисунке:

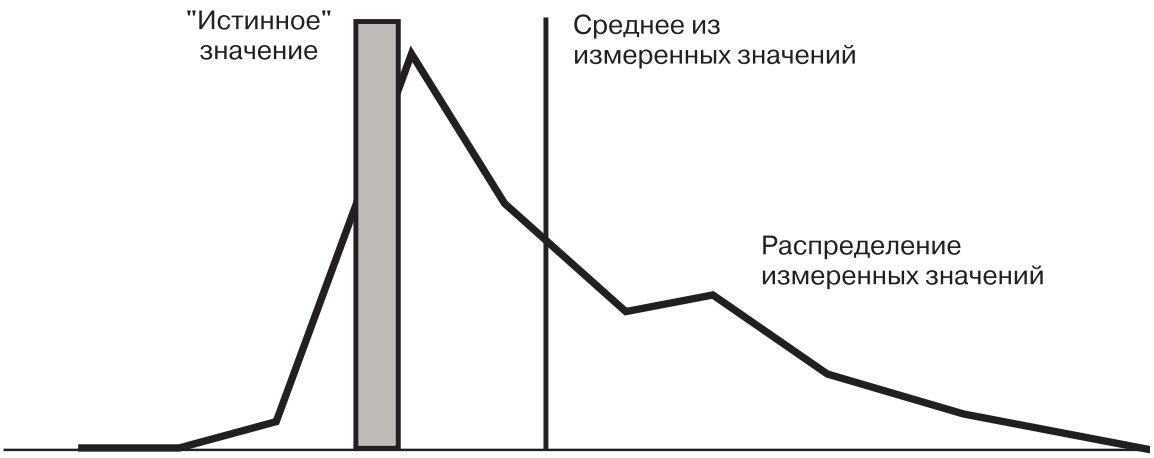


Рисунок 5. Искаженная кривая

Распределение результатов измерений относительно фактического значения нельзя считать случайным, поэтому стандартные тесты статистики, основанной на логическом выводе, должны применяться с крайней осторожностью. Кроме того, в зависимости от цели измерения, для описания производительности может не подходить ни среднее, ни фактическое значение.

Отслеживание производительности системы

В AIX поддерживаются различные инструменты и приемы для мониторинга производительности системы.

Преимущества непрерывного отслеживания производительности

Описаны преимущества непрерывного отслеживания производительности.

Ниже перечислены основные преимущества непрерывного отслеживания производительности:

- Обнаружение некоторых ошибок до того, как они негативно скажутся на работе системы
- Обнаружение неполадок, снижающих эффективность работы пользователей

- Сбор данных при первом возникновении неполадки
- Создание начального набора данных для сравнения

Отслеживание производительности включает в себя следующие задачи:
- Периодический сбор информации о производительности средствами операционной системы
- Сохранение информации для последующего использования при диагностике неполадок
- Просмотр и анализ информации системным администратором
- Отслеживание ситуаций, требующих сбора дополнительной информации, а также сбор информации по указанию системного администратора
- Сбор и сохранение требуемой подробной информации
- Отслеживание изменений, вносимых в систему и приложения

Непрерывное отслеживание производительности системы с помощью различных команд

Команды **vmstat**, **iostat**, **netstat** и **sar** позволяют создать механизм отслеживания производительности системы.

Вы можете написать сценарии, которые сокращают объем вывода команд, а при снижении производительности либо отправляют предупреждение, либо сохраняют информацию о состоянии системы. Например, сценарий может сравнивать время простоя процессора с нулем и запускать другой сценарий, если процессор полностью загружен. Следующий сценарий покажет список из 15 процессов, которые не принадлежат владельцу сценария, и на выполнение которых было затрачено максимальное количество процессорного времени:

```
# ps -ef | egrep -v "STIME|$LOGNAME" | sort +3 -r | head -n 15
```

Непрерывное отслеживание производительности системы командой vmstat

Команда **vmstat** позволяет получить общий обзор использования CPU, пространства подкачки и оперативной памяти.

Ниже приведен пример отчета команды **vmstat**:

```
# vmstat 5 2
нити     память             страница          ошибки        cpu
----- ----------- ----------------------- ------------ -----------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
 1  1 197167 477552   0   0   0   7   21   0 106 1114 451  0  0 99  0
 0  0 197178 477541   0   0   0   0    0   0 443 1123 442  0  0 99  0
```

Обратите внимание, что первый отчет команды **vmstat** содержит информацию о работе системы, собранную с момента последней загрузки системы. Во втором отчете учитывается работа за первые 5 секунд.

Подробные сведения о команде **vmstat** приведены в разделах “Команда vmstat” на стр. 104, “Оценка интенсивности использования памяти с помощью команды vmstat” на стр. 129 и “Оценка производительности дисков командой vmstat” на стр. 182.

Непрерывное отслеживание производительности системы командой iostat

Команда **iostat** позволяет получить информацию об использовании дисков и CPU.

Операционная система AIX поддерживает ведение хронологии работы диска. В следйющем примере хронология дисковых операций ввода-вывода выключена, так как выдается следующее сообщение:

```
Отсутствует хронология работы диска с момента загрузки.

Это не влияет на интервальную статистику дисковых операций ввода-вывода.
```

Для включения хронологии дисковых операций ввода-вывода введите в командной строке `smit chgsys`, а затем выберите **true** в поле **Постоянное ведение хронологии ввода-вывода диска**.

В следующем примере отчета показано, когда запускалась команда **iostat**:

```
# iostat 5 2

tty:      tin         tout   avg-cpu:  % user    % sys     % idle    %iowait
          0.1        102.3               0.5      0.2       99.3       0.1

               Отсутствует хронология работы диска с момента загрузки.

               Это не влияет на интервальную статистику дисковых операций ввода-вывода.

tty:      tin         tout   avg-cpu:  % user    % sys     % idle    %iowait
          0.2        79594.4               0.6      6.6       73.7      19.2

Диски :       % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk1           0.0       0.0       0.0          0         0
hdisk0          78.2     1129.6     282.4       5648         0
cd1              0.0       0.0       0.0          0         0
```

Первый отчет команды **iostat** содержит информацию о работе системы, собранную с момента последнего сброса счетчиков работы диска. Во втором отчете учитывается работа за первые 5 секунд.

Понятия, связанные с данным:

“Команда iostat” на стр. 107
Команда **iostat** позволяет быстро обнаружить неполадки дискового ввода-вывода, приводящие к снижению производительности.

Задачи, связанные с данной:

“Оценка производительности дисков командой iostat” на стр. 179
Начните оценку с запуска команды **iostat** во время максимальной нагрузки на систему или работы важного приложения, задержки ввода-вывода для которого необходимо свести к минимуму.

Непрерывное отслеживание производительности системы командой netstat

Команда **netstat** позволяет определить число отправленных и принятых пакетов.

Ниже приведен пример отчета команды **netstat**:

```
# netstat -I en0 5
    ввод     (en0)     вывод            ввод    (всего)    вывод
 пакетов ошибок пакетов ошибок конфл. пакетов ошибок пакетов ошибок конфл.
 8305067     0  7784711     0     0 20731867     0 20211853     0     0
       3     0        1     0     0        7     0        5     0     0
      24     0      127     0     0       28     0      131     0     0
CTRL C
```

Обратите внимание, что первый отчет команды **netstat** содержит информацию о работе системы, собранную с момента последней загрузки системы. Во втором отчете учитывается работа за первые 5 секунд.

Кроме того, команду **netstat** можно выполнить с опциями **-s** и **-v**. Дополнительная информация приведена в разделе “Команда netstat” на стр. 300.

Непрерывное отслеживание производительности системы командой sar

Команда **sar** позволяет получить информацию об использовании CPU.

Ниже приведен пример отчета команды **sar**:

```
# sar -P ALL 5 2

AIX aixhost 2 5 00040B0F4C00    01/29/04

10:23:15 cpu    %usr    %sys    %wio   %idle
```

```
10:23:20  0        0       0       1      99
          1        0       0       0     100
          2        0       1       0      99
          3        0       0       0     100
          -        0       0       0      99
10:23:25  0        4       0       0      96
          1        0       0       0     100
          2        0       0       0     100
          3        3       0       0      97
          -        2       0       0      98

Average   0        2       0       0      98
          1        0       0       0     100
          2        0       0       0      99
          3        1       0       0      99
          -        1       0       0      99
```

Команда **sar** не создает отчет с информацией, собранной с момента загрузки системы.

Подробные сведения о команде **sar** приведены в разделах “Команда sar” на стр. 107 и “Оценка производительности дисков командой sar” на стр. 183.

Непрерывное отслеживание производительности системы с помощью команды topas

Программа **topas** собирает наиболее важную статистическую информацию о работе локальной системы, такую как объем памяти и число системных вызовов записи.

Команда **topas** работает с библиотекой curses. Вывод ее - текстовый на терминал 80x25 символов или графический в окно не меньше указанного размера. По умолчанию **topas** собирает и выводит статистическую информацию с периодом в две секунды. В **topas** предусмотрены следующие меню:

- Общая статистика системы
- Список наиболее загруженных процессов
- Статистика WLM
- Список активных физических дисков
- Вывод логических разделов
- Представление разделов

Для работы **topas** в системе должен быть установлен набор файлов `bos.perf.tools` и `perfagent.tools`.

Дополнительная информация о **topas** приведена в разделе команда topas в *Справочник по командам, том 5*.

Список общей статистики системы

Список общей статистики системы разделен на две части - фиксированную и переменную.

Две верхние строки, расположенные слева, содержат имя системы, в которой запущена программа **topas**, дату и время сбора данных и интервал сбора данных. Ниже в левой части экрана показаны следующие списки:

- Использование CPU
- Сетевые интерфейсы
- Физические диски
- Классы WLM
- Процессы

Справа от этой области показаны следующие фиксированные списки:

- СОБЫТИЯ/ОЧЕРЕДИ

- ФАЙЛЫ/TTY
- ПОДКАЧКА
- ПАМЯТЬ
- ПР. ПОДКАЧК.
- NFS

Ниже приведен пример списка общей статистики системы:

```
Монитор Topas для хоста:     aixhost              СОБЫТИЯ/ОЧЕРЕДИ ФАЙЛЫ/TTY
среда 4 февраля 2004 11.23.41  Интервал: 2      Cswitch      53  Readch     6323
                                                 Syscall     152  Writech     431
Kernel    0.0   |                            |  Reads         3  Rawin         0
Польз.    0.9   |                            |  Writes        0  Ttyout        0
Ожид.     0.0   |                            |  Forks         0  Igets         0
Простой  99.0   |############################|  Execs         0  Namei        10
                                                 Runqueue    0.0  Dirblk        0
Сеть     KBPS   I-Pack  O-Pack   KB-In  KB-Out  Waitqueue   0.0
en0       0.8     0.4     0.9     0.0     0.8
lo0        0.0     0.0     0.0     0.0     0.0  ПОДКАЧКА         ПАМЯТЬ
                                                 Faults        2  Real,MB    4095
Диск    Busy%     KBPS     TPS KB-Read KB-Writ  Steals        0  % Comp      8.0
hdisk0    0.0      0.0     0.0     0.0     0.0  PgspIn        0  % Noncomp  15.8
hdisk1    0.0      0.0     0.0     0.0     0.0  PgspOut       0  % Client   14.7
                                                 PageIn        0
Класс WLM (Активн.)    CPU%    Mem%  Disk-I/O%  PageOut       0  ПР. ПОДКАЧК.
Система                   0       0          0  Sios          0  Size,MB     512
Общий                     0       0          0                   % Used      1.2
По умолчанию              0       0          0  NFS (calls/sec)  % Free     98.7
Имя             PID CPU% PgSp Class         0  ServerV2       0
topas         10442  3.0  0.8 System          ClientV2       0   Нажмите:
ksh           13438  0.0  0.4 System          ServerV3       0   "h" - справка
gil            1548  0.0  0.0 System          ClientV3       0   "q" - выход
```

Все списки, за исключением переменного списка Процессы, можно отсортировать по любому столбцу, переместив курсор на заголовок необходимого столбца. Все переменные списки, кроме списка Процессы, предусматривают следующие представления:

- Список процессов, наиболее активно использующих ресурсы
- Отчет с общим обзором работы системы

Например, в представлении отчета может быть представлена только суммарная производительность дисков или сети.

В списке с информацией о процессорах пользователь может просмотреть либо список наиболее активно используемых процессоров, либо итоговое значение использования процессоров, как показано в примере, приведенном выше.

Получение списка наиболее активных процессов с помощью монитора topas

Для того чтобы просмотреть список наиболее активных процессов, укажите в команде **topas** флаг **-P**.

Этот список аналогичен списку Процессы общей статистики системы, но в отличие от него содержит дополнительную информацию. Список можно отсортировать по любому столбцу, переместив курсор на заголовок необходимого столбца. Ниже приведен пример списка наиболее активных процессов:

```
Монитор Topas     хост:    aixhost     Интервал:   2    среда 4 февраля 2004 11.23.41

                              DATA  TEXT  PAGE              PGFAULTS
USER       PID  PPID PRI NI   RES   RES SPACE    TIME CPU%  I/O  OTH COMMAND
root         1       0  60 20   202     9   202    0:04  0.0  111 1277 init
root       774       0  17 41     4     0     4    0:00  0.0    0    2 reaper
root      1032       0  60 41     4     0     4    0:00  0.0    0    2 xmgc
root      1290       0  36 41     4     0     4    0:01  0.0    0  530 netm
```

```
root      1548       0  37 41    17     0    17    1:24  0.0    0   23 gil
root      1806       0  16 41     4     0     4    0:00  0.0    0   12 wlmsched
root      2494       0  60 20     4     0     4    0:00  0.0    0    6 rtcmd
root      2676       1  60 20    91    10    91    0:00  0.0   20 6946 cron
root      2940       1  60 20   171    22   171    0:00  0.0   15  129 errdemon
root      3186       0  60 20     4     0     4    0:00  0.0    0  125 kbiod
root      3406       1  60 20   139     2   139    1:23  0.0   1542187 syncd
root      3886       0  50 41     4     0     4    0:00  0.0    0    2 jfsz
root      4404       0  60 20     4     0     4    0:00  0.0    0    2 lvmbb
root      4648       1  60 20    17     1    17    0:00  0.0    1   24 sa_daemon
root      4980       1  60 20    97    13    97    0:00  0.0   37  375 srcmstr
root      5440       1  60 20    15     2    15    0:00  0.0    7   28 shlap
root      5762       1  60 20     4     0     4    0:00  0.0    0    2 random
root      5962    4980  60 20    73    10    73    0:00  0.0   22  242 syslogd
root      6374    4980  60 20    63     2    63    0:00  0.0    2  188 rpc.lockd
root      6458    4980  60 20   117    12   117    0:00  0.0   54  287 portmap
```

Список статистики WLM монитора topas

Для того чтобы просмотреть список статистики WLM, укажите в команде **topas** флаг **-W**.

Показанное меню будет разделено следующим образом:

- В верхней области экрана показан список классов WLM, потребляющих больше всего ресурсов. Аналогичный список содержится в списке общей статистики системы. Этот список можно отсортировать по любому столбцу.
- В нижней области экрана показан список активных процессов класса WLM, которые можно выбрать с помощью клавиш со стрелками или клавиши *f*.

Ниже приведен пример списка статистики WLM:

```
Монитор Topas   хост:  aixhost   Интервал:  2  срд 4 фев 2004 11.24.29
Класс WLM (Активн.)             CPU%      Mem%     Disk-I/O%
Системный                        0         0            0
Общий                            0         0            0
По умолчанию                     0         0            0
Неуправляемый                    0         0            0
Неклассифицированный             0         0            0
```

```
==============================================================================
                              DATA  TEXT  PAGE               PGFAULTS
USER       PID  PPID PRI NI   RES   RES SPACE    TIME CPU%  I/O  OTH COMMAND
root         1       0  60 20   202     9   202    0:04  0.0    0    0 init
root       774       0  17 41     4     0     4    0:00  0.0    0    0 reaper
root      1032       0  60 41     4     0     4    0:00  0.0    0    0 xmgc
root      1290       0  36 41     4     0     4    0:01  0.0    0    0 netm
root      1548       0  37 41    17     0    17    1:24  0.0    0    0 gil
root      1806       0  16 41     4     0     4    0:00  0.0    0    0 wlmsched
root      2494       0  60 20     4     0     4    0:00  0.0    0    0 rtcmd
root      2676       1  60 20    91    10    91    0:00  0.0    0    0 cron
root      2940       1  60 20   171    22   171    0:00  0.0    0    0 errdemon
root      3186       0  60 20     4     0     4    0:00  0.0    0    0 kbiod
```

Просмотр экрана физических дисков

Для просмотра экрана со списком оперативных физических дисков используйте команду **topas** с флагом **-D**.

Наибольшее число отображаемых физических дисков равно числу отслеживаемых оперативных физических дисков, заданному флагом **-d**. Список оперативных физических дисков хранится в поле KBPS.

Ниже приведен пример отчета, созданного командой **topas -D**:

```
Topas Monitor for host:    aixcomm    Interval:   2    Fri Jan 13 18:00:16 XXXX
================================================================================
Disk     Busy%  KBPS     TPS   KB-R   ART   MRT   KB-W   AWT   MWT   AQW    AQD
hdisk0     3.0  56.0     3.5   0.0    0.0   5.4   56.0   5.8   33.2  0.0    0.0
cd0        0.0   0.0     0.0   0.0   0.0    0.0   0.0    0.0    0.0  0.0    0.0
```

Дополнительную информацию о команде **topas-D** можно найти в разделе команда topas в книге *Справочник по командам, том 5*.

Просмотр панели обмена данными между разделами

Для просмотра статистики об обмене данными между разделами в **topas** используйте команду **topas** с флагом **-C** или нажмите клавишу **C** в любой другой панели.

Экран разделен на следующие части:

- В верхней части экрана показаны комплексные данные группы разделов, отражающие работу общего раздела, памяти и процессора. С помощью клавиши **G** в этом разделе можно переключаться с краткого списка на подробный или скрыть списки.
- В нижней части экрана, которая разделена на две части: общие разделы и выделенные разделы, показана статистика по каждому разделу. С помощью клавиши **S** можно скрыть общие разделы. С помощью клавиши **D** можно скрыть выделенные разделы.

Ниже приведен пример вывода команды **topas -C**:

```
Topas CEC Monitor            Interval:  10           Wed Mar  6 14:30:10 XXXX
Partitions     Memory (GB)              Processors
Shr:  4        Mon:  24  InUse:  14     Mon:  8  PSz:  4  Shr_PhysB:  1.7
Ded:  4        Avl:  24                 Avl:  8  APP:  4  Ded_PhysB:  4.1

Host         OS  M Mem InU Lp  Us Sy Wa Id  PhysB  Ent  %EntC Vcsw PhI
-------------------------------shared-------------------------------------------
ptools1      A53 u 1.1 0.4  4  15  3  0 82   1.30  0.50  22.0  200   5
ptools5      A53 U  12  10  1  12  3  0 85   0.20  0.25   0.3  121   3
ptools3      A53 C 5.0 2.6  1  10  1  0 89   0.15  0.25   0.3   52   2
ptools7      A53 c 2.0 0.4  1   0  1  0 99   0.05  0.10   0.3  112   2
------------------------------dedicated-----------------------------------------
ptools4      A53 S 0.6 0.3  2  12  3  0 85   0.60
ptools6      A52   1.1 0.1  1  11  7  0 82   0.50
ptools8      A52   1.1 0.1  1  11  7  0 82   0.50
ptools2      A52   1.1 0.1  1  11  7  0 82   0.50
```

Разделы можно упорядочить по любому столбцу, кроме столбцов Host, OS и M, переместив курсор на заголовок необходимого столбца.

Дополнительную информацию о команде **topas -C** можно найти в разделе команда topas в книге *Справочник по командам, том 5*.

Просмотр информации об уровне локального логического раздела

Для просмотра информации об уровне раздела и показателей производительности для каждого логического процессора используйте команду **topas** с флагом **-L** или нажмите клавишу **L** в любой другой панели.

Экран разделен на две части:

- В верхней части экрана приведена информация о уровне разбиения на разделы.
- В нижней части экрана отображен упорядоченный список показателей логических процессоров.

Ниже приведен пример вывода команды **topas -L**:

```
    Interval:   2           Logical Partition: aix       Sat Mar 13 09:44:48 XXXX
    Poolsize: 3.0                Shared SMT ON           Online Memory:   8192.0
    Entitlement: 2.5              Mode: Capped           Online Logical CPUs:  4
                                                         Online Virtual CPUs:  2
    %user  %sys  %wait  %idle physc  %entc %lbusy   app   vcsw  phint %hypv hcalls
```

```
 47.5  32.5    7.0   13.0   2.0   80.0  100.0   1.0    240    150   5.0   1500
--------------------------------------------------------------------------------
logcpu minpf majpf  intr   csw  icsw runq lpa  scalls usr sys  wt idl  pc lcsw
cpu0    1135   145   134    78    60    2  95   12345  10  65  15  10 0.6  120
cpu1     998   120   104    92    45    1  89    4561   8  67  25   0 0.4  120
cpu2    2246   219   167   128    72    3  92   76300  20  50  20  10 0.5  120
cpu3    2167   198   127    62    43    2  94    1238  18  45  15  22 0.5  120
```

Дополнительную информацию о команде **topas-D** можно найти в разделе команда topas в книге *Справочник по командам, том 5*.

Панели SMIT для topas/topasout/topasrec

Панели программы SMIT служат для более удобной настройки функции записи **topas** и создания отчетов.

Для того чтобы перейти в панель **topas** программы SMIT введите `smitty performance` (или `smitty topas`) и выберите **Настроить опции Topas** .

Появится меню Настроить опции Topas.

```
                      Настроить опции Topas
Переместите курсор к нужному пункту и нажмите клавишу Enter.

 Добавление хоста к файлу поиска внешней подсети topas (Rsi.hosts)
 Перечислить хосты в файле поиска внешней подсети topas (Rsi.hosts)
 Показать активные записи
 Запустить запись
 Остановить запись
 Показать завершенные записи
 Создать отчет
 Установка службы Управление производительностью
```

Дополнительная информация приведена в разделе команда topas in *Справочник по командам, том 5*.

Добавление хоста к файлу поиска внешней подсети topas (Rsi.hosts):

Клиенты PTX и команда **topas –C|topasrec –C** ограничены способом идентификации удаленных хостов API интерфейса удаленной статистики (Rsi).

При запуске клиента он передает запрос на порт **xmquery**, являющийся зарегистрированной службой демона **inetd**. Удаленные хосты видят этот запрос, после чего производится настройка файла `inetd.conf`, запуск демонов **xmservd** или **xmtopas** и ответ на запрос клиента. Существующая архитектура ограничивает вызов **xmquery** для обработки хостами в одной подсети.

Для решения этого вопроса PTX поддерживает настраиваемый список хостов, которые могут быть вне подсети. RSi использует имена хостов или IP из этого списка (файл `RSi.hosts`). Файл `RSi.hosts` можно изменять. По умолчанию RSi использует следующий порядок поиска:

1. `$HOME/Rsi.hosts`
2. `/etc/perf/Rsi.hosts`
3. `/usr/lpp/perfmgr/Rsi.hosts`

Каждый хост занимает отдельную строчку в файле. Имя имеет формат IP-адреса или следующий формат полного имени:

```
ptoolsl1.austin.ibm.com
9.3.41.206
...
```

Выберите опцию **Добавление хоста к файлу поиска внешней подсети topas (Rsi.hosts)** для добавления хостов к файлу **Rsi.hosts**. Выберите опцию **Перечислить хосты в файле поиска внешней подсети topas (Rsi.hosts)** для просмотра списка хостов в файле `Rsi.hosts`.

Запустить запись:

Команда **Запустить запись** позволяет запустить локальную запись или запись CEC с сохранением или без сохранения состояния в соответствии с данными, указанными пользователем. Для запуска локальной записи и записи CEC применяются разные меню.

Постоянная запись:

Постоянная запись запускается с помощью SMIT с опцией разбиения на файлы и сохранения. Можно указать число дней для регистрации в одном файле записи, а также время хранения записи. Одновременно в системе может выполняться не более одной операции **постоянной записи** одного типа (CEC или локальная). При запуске **постоянной записи** вызывается команда записи с параметрами, указанными пользователем. Опции командной строки, применяемые для запуска постоянной записи, добавляются в записи **inittab**. Такой подход обеспечивает автоматический перезапуск записи в случае перезагрузки или перезапуска системы.

Рассмотрим систему, в которой уже запущена локальная постоянная запись (двоичная или nmon). Для запуска новой операции локальной постоянной записи необходимо предварительно остановить текущую операцию постоянной записи с помощью опции **Остановить постоянную запись**, доступной в меню **Остановить запись**. Затем можно запустить новую операцию **локальной постоянной записи** с помощью опции **Запустить локальную постоянную запись**. Запустить постоянную запись нельзя, если в системе уже выполняется операция постоянной записи такого же формата. Поскольку постоянная запись предусматривает добавление записей **inittab**, ее могут запустить только привилегированные пользователи.

Например, если в параметре числа дней в файле указано значение **n**, то отдельный файл будет содержать не более **n** дней записи. По истечении **n** дней создается новый файл для сохранения новых данных записи. Если указано нулевое число дней, то данные записи сохраняются только в одном файле. Если срок хранения составляет **m** дней, то система будет сохранять файл записи в течение **m** дней. Файлы записи, созданные одним и тем же экземпляром команды **topasrec**, удаляются через **m** дней.

Число дней записи в файле по умолчанию: **1**.

Срок хранения файла по умолчанию: **7**.

Появится меню Опции SMIT для запуска записи.

```
                  Опции SMIT для
запуска записи

                       Запустить запись
Поместите курсор на нужный элемент и нажмите Enter.

Запустить постоянную локальную запись
Запустить постоянную запись CEC
Запустить локальную запись
Запустить запись CEC
```

Запустить постоянную локальную запись:

Пользователь может указать один из следующих типов постоянной локальной записи: двоичная и nmon.

Для запуска постоянной записи в меню Тип постоянной записи следует выбрать одно из следующих значений:

```
                Тип постоянной записи

Поместите курсор на нужный элемент и нажмите Enter.

   двоичная
   nmon
```

```
F1=Справка          F2=Обновить         F3=Отменить
```

Если выбран двоичный тип, запись будет отображена следующим образом:

```
   Введите или выберите значения в полях ввода.
   После внесения изменений нажмите клавишу Enter.

    Тип записи                                        двоичная
    Длина записи                                      постоянная
 Интервал между записями (в секундах)                [300]
    * Период записи в один файл (в днях)              [1]
    * Кол-во полных дней                              [7]
 Путь вывода                                        []
 * Перезаписать существующий файл записи            no
 * Активировать WLE                                 no
 * Включить базовые диск. показ.                    [Yes/No]
 * Включить служебн. показ. времени                  [Yes/No]
 * Включить базов. показ. диск. адаптера            [yes/No]
 * Включить служебн. показ. времени диск. адапт.    [Yes/No]
```

Интервал записи (в секундах) должен быть кратен 60 секундам. В случае двоичной локальной записи пользователь может разрешить создание отчетов IBM Workload Estimator (WLE) в меню SMIT. Инструмент оценки нагрузки создается только в воскресенье в 00:45 a.m. Для непрерывности данных в отчете необходимо, чтобы была включена двоичная локальная запись. Данные будут правильными только при наличии локальной записи.

Отчет инструмента оценки нагрузки сохраняются в файле**/etc/perf/<hostname>_aixwle_weekly.xml**. Например, если имя хоста **ptoolsl1**, еженедельный отчет записывается в файл **/etc/perf/ ptoolsl1_aixwle_weekly.xml**.

Дополнительная информация приведена в следующих разделах:

- “Постоянная запись” на стр. 21
- доступные фильтры nmon

Запустить постоянную запись CEC:

Команда **Запустить постоянную запись CEC** позволяет запустить постоянную запись CEC и настроить ее параметры. Начальная панель ввода содержит значения по умолчанию.

Интервал записи (в секундах) должен быть кратен 60 секундам.

Дополнительная информация приведена в разделе “Постоянная запись” на стр. 21.

Запустить локальную запись:

Команда **Запустить локальную запись** позволяет запустить локальную запись в соответствии параметрами, указанными пользователем. Пользователь может выбрать тип записи (двоичная или nmon) и продолжительность записи (день, час или другое значение).

```
                          Тип записи

   Поместите курсор на нужный элемент и нажмите Enter.

     двоичная
     nmon

   F1=Справка          F2=Обновить         F3=Отменить
```

После выбора типа записи в следующем меню необходимо указать продолжительность: день, час или другое значение.

```
                          Продолжительность записи

Поместите курсор на нужный элемент и нажмите Enter.

  день
  час
  другое значение

F1=Справка          F2=Обновить         F3=Отменить
F8=Образ                F10=Выход               Enter=Выполнить
```

Если выбрана опция "день" или "час", то такие параметры, как интервал записи и число операций выборки изменить нельзя. В случае настраиваемой записи можно изменить интервал записи и число операций выборки. Интервал записи должен быть кратен 60 секундам. Настраиваемая запись предназначена для выполнения конкретного числа операций выборки с указанным интервалом и завершения записи. Если указано нулевое число операций выборки, то запись будет выполняться непрерывно до тех пор, пока не будет остановлена.

На панелях отображаются значения по умолчанию.

Дополнительная информация приведена в разделе “Запись NMON”.

Запустить запись CEC:

Команда **Запустить запись CEC** позволяет запустить запись CEC.

Пользователь должен указать продолжительность записи (день, час или другое значение).

```
                          Продолжительность записи

Поместите курсор на нужный элемент и нажмите Enter.

  день
  час
  другое значение

F1=Справка          F2=Обновить         F3=Отменить
```

Если выбрана опция "день" или "час", то такие параметры, как интервал записи и число операций выборки изменить нельзя.

Если выбрана опция "другое значение", то пользователь может указать произвольное число операций выборки; в качестве интервала записи можно указать число, кратное 60. Настраиваемая запись предназначена для выполнения конкретного числа операций выборки с указанным интервалом и завершения записи. Если указано нулевое число операций выборки, то запись будет выполняться непрерывно до тех пор, пока не будет остановлена.

Запись NMON:

Для NMON предусмотрены вспомогательные фильтры записи, применяемые в ходе настройки **записи NMON**. Можно выбрать или отменить выбор следующих разделов:

- JFS
- Ядро RAW и LPAR
- группа томов
- пространство подкачки
- MEMPAGES

- NFS
- WLM
- Большая страница
- Процесс Shared Ethernet (для VIOS)
- Большая страница, асинхронный режим

Примечание: Опции Число дисков в строке, Файл группы дисков и Предпочитаемые диски допустимы только в том случае, если запись включает в себя раздел конфигурации дисков. Фильтр процессов и пороговое значение процессов доступны только в том случае, если в запись добавлен список процессов.

Фильтры процессов и дисков настраиваются автоматически в соответствии с последними параметрами записи, настроенными пользователем. В параметре Внешний сборщик данных - Начальная или конечная программа можно указать внешнюю команду для выполнения на начальном или заключительном этапе записи NMON. В параметре Внешний сборщик данных - Программа создания моментальной копии можно указать внешнюю команду для периодической записи показателей. Дополнительная информация о применении внешних команд для записи nmon приведена в разделе Команда nmon книги *Справочник по командам, том 4*.

Соглашение об именах:

Записанные файлы сохраняются в файлах, указанных в следующей таблице:

- Имя файла вывода для отдельной записи создается на основе имени каталога и префикса имени исходного файла следующим образом:

Стиль	**файлы**
Локальная Nmon:	<имя-файла>_ГГММДД_ЧЧММ.nmon
Локальная Nmon:	<имя-файла>_ГГММДД_ЧЧММ.topas
CEC Topas:	<имя-файла>_ГГММДД_ЧЧММ.topas

- Имя файла вывода для нескольких файлов записи создается на основе имени каталога и префикса имени исходного файла следующим образом:

Стиль	**файлы**
Локальная Nmon:	<имя-файла>_ГГММДД.nmon
Локальная Nmon:	<имя-файла>_ГГММДД.topas
CEC Topas:	<имя-файла>_CEC_ГГММДД.topas

- Имя файла вывода для отдельной записи создается на основе имени каталога и без префикса имени исходного файла следующим образом:

Стиль	**файлы**
Локальная Nmon:	<имя-файла/имя-хоста>_ГГММДД_ЧЧММ.nmon
Локальная Nmon:	<имя-файла/имя-хоста>_ГГММДД_ЧЧММ.topas
CEC Topas:	<имя-файла/имя-хоста>_CEC_ГГММДД_ЧЧММ.topas

- Имена файлов вывода для нескольких файлов записи создаются на основе имени каталога и без префикса имени исходного файла следующим образом:

Стиль	файлы
Локальная Nmon:	<имя-файла/имя-хоста>_ГГММДД.nmon
Локальная Nmon:	<имя-файла/имя-хоста>_ГГММДД.topas
CEC Topas:	<имя-файла/имя-хоста>_CEC_ГГММДД.topas

Две операции записи одного и того же формата с одинаковыми значениями параметра **имя-файла** нельзя запустить одновременно, поскольку процессы записи будут обращаться к одному и тому же файлу.

Примеры:

1. Предположим, что пользователь запускает локальную двоичную запись отдельного дня, указав каталог вывода **/home/test/sample_bin**. Имя файла записи, созданного 10 марта 2008 года в 12:05, для хоста с именем ses15: **/home/test/sample_bin/ses15_080310_1205.topas**.
2. Предположим, что пользователь запускает постоянную запись CEC со следующими параметрами: число дней в файле - 2, каталог вывода - **/home/test/sample**. Имя файла записи, созданного 10 марта 2008 года в 12:05, для хоста с именем ses15: /home/test/sample_bin/ses15_cec_080310.topas. После сохранения двух дней записи будет создан файл записи **/home/test/sample_bin/ses15_cec_080312.topas** (создается 12 марта для сохранения данных за 12 и 13 марта).

Остановить запись:

Команда Остановить запись позволяет остановить текущую операцию записи. Пользователь может выбрать конкретную операцию записи в списке.

В меню необходимо выбрать тип записи, которую требуется остановить. После выбора типа записи в меню будет показан список активных операций записи, в котором можно выбрать нужную операцию записи.

Ниже показана панель выбора типа записи:

```
                    Остановить запись

Остановить постоянную запись
Остановить двоичную запись
Остановить запись nmon
Остановить запись CEC
```

Примечание: Для остановки записей необходимы права на остановку процесса записи.

Показать активные записи:

Команда **Показать активные записи** позволяет просмотреть список активных операций записи в указанном каталоге.

Для просмотра списка активных записей выполните следующие действия:

1. Введите путь записи.
2. Выберите тип записи.

```
                        Тип записи
Поместите курсор на нужный элемент и нажмите Enter.

постоянная
двоичная
nmon
cec
все

F1=Справка              F2=Обновить             F3=Отменить
F8=Образ                F10=Выход               Enter=Выполнить
/=Найти                 n=Найти далее
```

В списке будут показаны такие свойства активных операций записи, как формат, начальное время и каталог вывода.

Каталоги вывода постоянных операций записи обозначаются префиксом (*). Каталоги вывода постоянных двоичных локальных операций записи обозначаются префиксом (#).

Показать завершенные записи:

Команда **Показать завершенные записи** позволяет просмотреть список завершенных записей из указанного каталога. На основе завершенных записей можно создать файлы отчетов с помощью меню **Создать отчет**.

Для **просмотра списка завершенных записей** выполните следующие действия:

1. Введите путь к записи (путь к файлу записи).
2. Выберите тип записи.

```
persistent
binary
nmon
cec
all
```

Будет показан список завершенных записей из указанного пути со следующими столбцами **Тип записи, Начальное время** и **Конечное время**.

Создание отчетов на основании существующих файлов записи:

Опция **Создать отчет** позволяет создавать отчеты на основе существующих файлов записи из указанных каталогов.

Если выбран каталог с файлом постоянной записи, то выполняются следующие условия:

1. Для создания отчетов выбирается активная операция постоянной записи.
2. Если постоянная запись не выполняется, то создания отчетов выбирается последняя завершенная операция постоянной записи.

С помощью опции **Создать отчет** пользователь задает значения файла записи, формат отчета, начальное время, конечное время, интервал и имя файла или принтера.

Для создания отчета выполните следующие шаги:

1. Выберите имя файла или принтера для отправки отчета:

```
                Вывести отчет в файл/на принтер

Поместите курсор на нужный элемент и нажмите Enter.

1 Имя файла
2 Принтер
```

2. Укажите путь к файлу записи:

```
 Путь к файлу записи                              [] +
```

3. Выберите формат отчета (на основании типа записи):

```
* Формат отчета                                  []     +
```

Ниже приведен пример типа отчета *через запятую*:

```
* Тип записи
[]
* Формат отчета                                  []
    Тип                                        [средний]    +
  Имя файла записи                                          []
* Файл вывода                                    []
```

Ниже приведен пример типа *nmon* файла отчета:

```
* Тип записи                                  []
* Формат отчета                          []
  Имя файла записи                            []
* Файл вывода                            []
```

Примечание: Поле **Файл вывода** - обязательное для типов *через запятую* и *nmon* и необязательное для остальных форматов. Записи **topas** поддерживают только средний тип для форматов отчета *через запятую* и *электронная таблица*.

Ниже приведен пример типа отчета *сводка/диск сводка/подробный/сеть сводка*:

```
* Тип записи                                  []
* Формат отчета                          []
  Начальное время (ГГММДДЧЧММ)                []
  Конечное время (ГГММДДЧЧММ)                 []
  Интервал                                    []
  Имя файла записи                            []
  Файл вывода (по умолчанию - stdout)            []
```

Во всех примерах, приведенных выше, первые два поля изменять нельзя, и они заполняются значениями на основании предыдущих выборов.

Если для вывода отчета выбран принтер, поле **Файл вывода** заменяется обязательным полем **Имя принтера**, в котором перечислены доступные принтеры:

```
* Имя принтера                               [ ] +
```

Дополнительная информация о форматах отчетов для двоичных локальных записей и записей CEC приведена в разделе Команда topas книги *Справочник по командам, том 5*.

Установка службы Управление производительностью:

Это меню используется для установки и настройки службы Управление производительностью.

```
  Установка службы Управление производительностью

Поместите курсор на нужный элемент и нажмите Enter.

  Включить передачу данных PM
  Выключить передачу данных PM
  Повторно передать записанные данные
  Изменить/Показать сведения о клиенте
  Изменить/Показать период хранения
  Изменить/Показать тенденции и планирование
```

- **Включить передачу данных PM**

  Опция **Включить передачу данных PM** используется для включения передачи данных в IBM из Electronic Service Agent (ESA) или Консоли аппаратного обеспечения (HMC).

- **Выключить передачу данных PM**

  Опция **Выключить передачу данных PM** используется для выключения передачи статистических данных в IBM.

- **Повторно передать записанные данные**

  Опция **Повторно передать записанные данные** используется для повторной передачи записанных ранее статистических данных.

```
  Повторно
  передать записанные данные

  Введите или выберите значения в полях ввода.
  После внесения изменений нажмите клавишу Enter.
```

```
                                                    [Поля ввода]
* Введите дату [ГГГГММДД]                          []  #
```

  – Если вы хотите повторно передать данные PM датированные 12-м февраля 2009, введите `20090212` в текстовое поле.
  – Введите `0` для передачи всех доступных записанных файлов данных PM.
- После ввода данных будут показаны следующие действия, которые нужно выполнить вручную для отправки данных IBM с помощью ESA или HMC:

```
Действия для отправки вручную из Electronic Service Agent в HMC

1.  Войдите в HMC
2.  Выберите 'Управление службой'
3.  Выберите 'Передать служебную информацию'
4.  Нажмите кнопку 'Отправить' с меткой 'Для немедленной передачи
информации об управлении производительностью
    нажмите Отправить'.  (вторая кнопка Отправить на странице)
5.  Проверьте результаты в протоколе События консоли
```

```
Действия для отправки вручную из Electronic
Service Agent для AIX

1. Войдите в ESA с помощью web-интерфейса (https://имя-хоста:порт/esa)
2. Перейти в раздел 'Служебная информация'
3. Выберите действие 'Сбор данных'
4. Выберите переключатель Управление производительностью
5. Нажмите OK
6. Проверьте результаты в протоколе операций"
```

- **Изменить/Показать сведения о клиенте**

  Опция **Изменить/Показать сведения о клиенте** используется для отображения или обновления сведений о клиенте. Сведения о клиенте будут отправлены в IBM, если включена опция Передача данных PM
- **Изменить/Показать период хранения**

  Опция **Изменить/Показать период хранения** используется для отображения или изменения периода хранения данных. Период хранения определяет, как долго старые данные будут храниться в каталоге Данные перед их удалением.
- **Изменить/Показать тенденции и планирование**

  Опция **Изменить/Показать тенденции и планирование** используется для отображения или изменения тенденций и планирования.

Установка Workload Estimator:

Это меню используется для установки и настройки службы Workload Estimator.

```
  WLE

Введите или выберите значения в полях ввода.
После внесения изменений нажмите клавишу Enter.

  Набор WLE
  Тип входных данных WLE
```

- **Набор WLE**

  Используйте **Набор WLE** для включения или выключения опции создания отчетов, используемых в качестве входных данных WLE. В данном поле по умолчанию указано текущее значение параметра **Набор WLE**. Отключите данный параметр перед остановкой записи.
- **Тип входных данных WLE**

  Используйте **Тип входных данных WLE** для указания типа входных данных WLE (активные двоичные записи или запись **nmon**). Данный параметр используется только для постоянных записей.

Непрерывное отслеживание производительности системы с помощью службы Управление производительностью (PM)

Служба Управление производительностью (PM) автоматизирует сбор, архивацию и анализ статистических данных и возвращает отчеты, помогающие пользователю управлять ресурсами системы. Собираются данные об использовании системы, информация о производительности и информация о конфигурации аппаратного обеспечения.

Собранные данные службы Управление производительностью (PM) отправляются в IBM посредством Electronic Service Agent (ESA) или Консоли аппаратного обеспечения (HMC). IBM сохраняет данные для клиента и предоставляет их в серии отчетов и графиков, которые показывают производительность сервера и прирост. Клиенты могут получить доступ к своим отчетам в электронном виде с помощью обычного браузера.

При использовании с IBM Systems Workload Estimator это предложение позволяет клиентам лучше понимать тенденции своего бизнеса и планировать время требуемого обновления аппаратного обеспечения, такого как Центральный процессор (CPU) или диск. IBM Systems Workload Estimator может определять размер консолидации систем или оценивать необходимость обновления системы, разбитой на логические разделы, имея PM IBM Power Systems для отправки данных для нескольких систем или разделов IBM Systems Workload Estimator.

Служба Управление производительностью использует постоянную двоичную запись **topasrec** для сбора статистических данных. Таким образом, постоянная двоичная запись **topasrec** должна быть всегда включена для службы PM, чтобы производился сбор статистических данных.

Примечания:

1. С включением службы PM вы соглашаетесь с тем, что IBM может использовать данные, собранные PM для серверов предприятия IBM IBM Power Systems без ограничений, в том числе в целях выявления неполадок, содействия в планировании ресурсов и повышения производительности, поддержки деловых отношений с IBM, уведомления о существующих и предполагаемых ограничениях ресурсов и совершенствования продуктов IBM. Вы также соглашаетесь с тем, что ваши данные могут быть переданы в любую страну, независимо от того, принадлежит ли она Европейскому союзу.
2. Вы можете предоставить IBM право совместно использовать ваши данные с другими фирмами, включая одного или несколько поставщиков решений и бизнес-партнеров, с целью осведомления их о ваших потребностях в производительности и мощности, и позволяя им предоставить вам более высокий уровень обслуживания. Предоставление этого права совершается при просмотре электронных графиков.

Дополнительная информация о службах Управление производительностью находится в файле `README.perf.tools`.

Первичная диагностика производительности

При анализе производительности системы необходимо принять во внимание различные типы сообщений о неполадках.

Типы отчетов о проблемах производительности

При снижении производительности рекомендуется оценить возникшую ситуацию и сузить круг возможных причин.

Медленно работает отдельная программа

Б⌂стродействие программ⌂ может упасть по нескольким причинам.

Следующие вопросы могут показаться тривиальными, но ответы на них очень важны:

- Программа всегда работала медленно?

Если программа работает медленно лишь с недавних пор, причиной снижения производительности могут быть последние изменения.

- Возможно, был изменен исходный код или установлена новая версия?

  Если это так, обратитесь за помощью к создателю программы или ее поставщику.

- Изменилась ли среда выполнения программы?

  Если был перемещен один из файлов, с которыми работает программа (в том числе и сам исполняемый файл), то задержки могут быть вызваны необходимостью передачи данных по сети. Другой вариант - конфликт между двумя файлами, которые ранее были расположены на разных дисках.

  Если администратор изменил параметры конфигурации системы, на выполнение программы могли быть наложены дополнительные ограничения. Например, если администратор изменил способ определения приоритетов, то быстродействие приложений, выполняемых в фоновом режиме, могло понизиться, а быстродействие интерактивных приложений - повыситься.

- Возможно, программа написана на языке **perl**, **awk**, **csh** или на другом интерпретируемом языке?

  К сожалению, компилятор не оптимизирует код на интерпретируемом языке. Кроме того, на языке, подобном **perl** или **awk**, очень просто составить программу длиной в несколько строк, выполнение которой потребует огромного количества операций ввода-вывода и значительных ресурсов процессора. Часто в таких случаях можно ограничиться визуальной проверкой исходного текста программы, подсчитав число неявных итераций, выполняемых для каждого оператора.

- Программа всегда выполняется с такой скоростью или в некоторых случаях выполняется быстрее?

  Файловая система резервирует часть системной памяти для хранения страниц файлов, к которым возможны последующие обращения. Если дважды с небольшим интервалом запустить одну и ту же программу, которая только обменивается данными с диском, то, как правило, во второй раз она будет выполняться быстрее. Это относится и к программам, применяющим NFS. Аналогичный эффект может наблюдаться и при работе больших программ, таких, как компиляторы. Даже если программа не обращается к диску, во второй раз она может выполняться быстрее за счет того, что часть исполняемого кода программы осталась в памяти и не нуждается в загрузке.

- Если программа всегда работала медленно, либо ее производительность снизилась без видимых причин, то проверьте, достаточно ли в системе ресурсов для ее выполнения.

  Инструкции по выполнению этой задачи приведены в разделе Определение ресурсов, ограничивающих производительность.

В определенное время дня медленно работают все программы

Снижение производительности системы в определённое время суток может наблюдаться по нескольким причинам.

Многие сталкиваются с падением производительности в "часы пик", то есть когда большое количество пользователей в организации интенсивно работают с системой. Это явление не всегда вызвано только повышенной нагрузкой. Иногда снижение производительности в "часы пик" может говорить о несбалансированности системы, которая проявляется только при высокой нагрузке. Следует рассмотреть и другие причины повторяющейся неполадки.

- Если проанализировать работу системы в "часы пик" с помощью утилит **iostat** и **netstat**, либо путем обработки данных, предварительно собранных монитором системы, то повышается ли в это время нагрузка на некоторые диски? Не падает ли время простоя CPU практически до нуля? Возможно, резко возрастает число переданных или принятых пакетов?
  - Если диски несбалансированные, обратитесь к “Производительность логических томов и дискового ввода-вывода” на стр. 178.
  - Если вы обнаружите, что процессор перегружен, то вызовите команду **ps** или **topas** и узнайте, какие программы выполнялись в этот период. Сценарий из раздела “Непрерывное отслеживание производительности системы с помощью различных команд” на стр. 14 упрощает поиск программ, наиболее активно использующих процессор.
  - Если снижение производительности заметно невооруженным глазом, например, если работа системы парализована в течение часа, то проверьте, не запустил ли кто-нибудь программу, потребляющую все

ресурсы системы (например, графический пакет **xlock** или мощную игру). Известно, что некоторые версии программы **xlock** при выводе графических шаблонов на пустой экран практически монопольно работают с процессором. Возможно также, что кто-то работает с программой, потребляющей почти все ресурсы CPU.

- Если файл `/var/adm/cron/cron.allow` не пуст, рекомендуется проверить, какой объем ресурсов потребляют операции из каталога `/var/adm/cron/crontab`.

Если вы обнаружили, что причина падения производительности заключается в конфликте между интерактивными процессами и фоновыми программами, рассчитанными на длительное выполнение и расходующими большой объем ресурсов CPU, то для повышения приоритета интерактивных процессов рекомендуется изменить способ вычисления приоритетов с помощью команды **schedo**. См. раздел “Вычисление значения приоритета нити” на стр. 125.

Время от времени медленно работают все программы

В этом случае рекомендуется воспользоваться программой обнаружения перегрузок, например, демоном **filtd** (компонентом PTX).

При возникновении перегрузки демон **filtd** может запускать указанный сценарий оболочки или собирать заданную информацию. Вы можете самостоятельно создать аналогичный инструмент с помощью сценариев оболочки, содержащих команды **vmstat**, **iostat**, **netstat**, **sar** и **ps**.

Если снижение производительности отмечено только в одной системе распределенной среды, можно предположить, что ресурсы захвачены одной программой, или что случайным образом конфликтуют два процесса.

У отдельного пользователя медленно работают все программы

Иногда кажется, что снижение производительности системы связано с заданиями, выполняемыми отдельным пользователем.

- Рекомендуется провести количественный анализ проблемы. Выясните у пользователя, какие команды он выполняет чаще всего, и соберите информацию об их выполнении с помощью команды **time**, как показано ниже:

  ```
  # time cp .profile testjunk
  фактическое 0м0,08с
  польз.      0м0,00с
  сист.       0м0,01с
  ```

  Затем выполните те же команды под управлением идентификатора другого пользователя, программы которого работают нормально. Будет ли разница в `фактическом` времени?
- Процессорное время (`пользовательское + системное`) не должно существенно измениться, но фактическое время может возрасти вследствие увеличения объема или снижения производительности ввода-вывода. Причина задержки может заключаться в том, что файлы пользователя находятся в смонтированном каталоге NFS или на диске, который интенсивно используется другими процессами.
- Проверьте, все ли компоненты поля *$PATH* файла `.profile` действительно необходимы. Например, если сначала всегда выполняется поиск в нескольких каталогах NFS, а только потом поиск в каталоге `/usr/bin`, то все программы будут работать медленно.

Одновременное снижение производительности нескольких систем, подключенных к локальной сети

При переходе от автономных систем к распределенным системам часто возникают одни и те же проблемы.

Их причиной, как правило, является стремление пользователей настроить отдельную систему согласно своим требованиям, или использование функций, потребляющих много ресурсов. Помимо настройки максимального размера блока передачи (MTU) и размера mbuf в конфигурации локальной сети, администратор должен убедиться, что сетевые ресурсы используются эффективно.

- С помощью статистической информации о работе сети убедитесь, что отсутствуют неполадки на уровне физической сети. Просмотрев вывод команд **netstat -v**, **entstat**, **tokstat**, **atmstat** или **fddistat** убедитесь, что число ошибок адаптера и конфликтов не превышает допустимый предел.
- Некоторые ошибки в программах или программно-аппаратных средствах могут эпизодически затруднять передачу по сети пакетов оповещения и других пакетов.

  При всплеске сетевой активности может существенно снизиться производительность всех систем сети, даже тех, которые почти не работают с сетью. Это происходит вследствие резкого увеличения количества прерываний и расхода ресурсов CPU на прием и обработку пакетов. Отслеживать и анализировать подобные ситуации лучше с помощью анализаторов сети, а не с помощью обычных инструментов анализа производительности.
- Возможно, две локальные сети объединены через систему?

  При использовании системы в качестве маршрутизатора значительный объем ресурсов CPU будет тратиться на обработку и копирование пакетов. Кроме того, между процессами, обрабатывающими пакеты, и остальными процессами будут возникать конфликты. Рекомендуется применять специальные аппаратные маршрутизаторы и мосты: они дешевле и надежнее.
- Есть ли серьезные причины для применения NFS в каждом отдельном случае?

  На некоторых этапах создания распределенной системы файловая система NFS применяется для того, чтобы пользователи в новых системах сохраняли доступ к своим домашним каталогам, расположенным в старых системах. Это упрощает переход к распределенной системе, однако требует передачи больших объемов данных по сети. Если пользователи из системы A работают преимущественно с данными, хранящимися в системе B, то это должно быть известно системному администратору.

  При доступе к файлам через NFS значительно увеличивается нагрузка на сеть, расход процессорного времени на клиенте и на сервере, а также время отклика конечного пользователя. Всегда следует стремиться к тому, чтобы пользователь работал в той системе, в которой хранятся его данные. Исключение составляют ситуации, когда работа с данными в удаленной системе оправдана. Например, если централизованное хранение данных необходимо для повышения надежности и управления резервным копированием, или если необходимо гарантировать, что все пользователи работают с самой последней версией программы.

  Если по этим или другим соображениям необходимо сохранить определенный уровень обмена данными между клиентом и сервером NFS, лучше выделить для этих целей отдельную систему-сервер, чтобы несколько систем не совмещали в себе функции клиента и сервера.
- Правильно ли настроены программы для работы с демонами вызова удаленных процедур (RPC) и нужны ли они вообще?

  Простейший способ переноса программы в распределенную среду - замена вызовов программ демонами RPC в соотношении 1:1. Однако при замене вызова локальной программы на RPC производительность может снизиться даже больше, чем при замене операций ввода-вывода с локальным диском на операции ввода-вывода в NFS. Если демоны RPC действительно необходимы, их следует по мере возможности объединять в пакетные вызовы.

Иногда снижается производительность всех программ, работающих с определенной службой или устройством

Производительность всех программ, работающих с определенной службой или устройством, может периодически снижаться по нескольким причинам.

Если время от времени снижается производительность программ, использующих некоторое устройство или службу, ознакомьтесь с разделом, посвященным этому устройству или службе:

- “Производительность процессора” на стр. 104
- “Быстродействие памяти” на стр. 128
- “Производительность логических томов и дискового ввода-вывода” на стр. 178
- “Производительность файловой системы” на стр. 235
- “Анализ производительности сети” на стр. 297
- “Отслеживание и настройка производительности NFS” на стр. 335

При удаленном подключении все программы работают медленно

Механизмы локальной и удаленной идентификации в системе могут работать по-разному. По умолчанию, когда пользователь указывает свой идентификатор при входе в систему, первыми просматриваются файлы локальной идентификации. При этом время ответа существенно меньше, чем при использовании служб сетевой идентификации.

Если при входе пользователя в систему применяется один из механизмов сетевой идентификации, то необходимо определить, какой именно механизм должен применяться первым при поиске идентификаторов пользователей. Это относится к любой команде, которая выполняет поиск идентификаторов пользователей, а также:

- **ps -ef**
- **ls -l**
- **ipcs -a**

Программы идентификации определены в файле `/usr/lib/security/methods.cfg`. По умолчанию применяется метод локальной идентификации `compat`. Для того чтобы получить информацию о применяемом для конкретного пользователя методе, войдите в систему от имени этого пользователя b введите в командной строке

```
# echo $AUTHSTATE
```

Для того чтобы первым действительно использовался механизм локальной идентификации, и лишь затем - механизм сетевой идентификации (например, DCE), введите в командной строке

```
# export AUTHSTATE="compat,DCE"
```

Определение ресурсов, ограничивающих производительность

Для оценки показателей использования ресурсов в многопользовательской системе рекомендуется применять команду **vmstat**.

Команда **vmstat** предоставляет информацию о загруженности процессора, интенсивности операций дискового ввода-вывода и использовании оперативной памяти. Ниже приведен пример команды **vmstat**, которая каждые пять секунд выводит на экран строку отчета:

```
# vmstat 5
```

В приведенном примере после значения интервала не указано число выводимых отчетов, поэтому команда будет выполняться до тех пор, пока вы ее не отмените.

Приведенный ниже отчет создан командой **vmstat** в системе, в которой запущены AIXwindows и несколько смешанных приложений (из отчета удалено несколько интервалов с относительно низкой нагрузкой):

```
нити     память            страница          ошибки        cpu
----- ----------- ---------------------- ------------ -----------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
 0  0  8793    81   0   0   0   1    7   0 125   42  30  1  2 95  2
 0  0  8793    80   0   0   0   0    0   0 155  113  79 14  8 78  0
 0  0  8793    57   0   3   0   0    0   0 178   28  69  1 12 81  6
 0  0  9192    66   0   0  16  81  167   0 151   32  34  1  6 77 16
 0  0  9193    65   0   0   0   0    0   0 117   29  26  1  3 96  0
 0  0  9193    65   0   0   0   0    0   0 120   30  31  1  3 95  0
 0  0  9693    69   0   0  53 100  216   0 168   27  57  1  4 63 33
 0  0  9693    69   0   0   0   0    0   0 134   96  60 12  4 84  0
 0  0 10193    57   0   0   0   0    0   0 124   29  32  1  3 94  2
 0  0 11194    64   0   0  38 201 1080   0 168   29  57  2  8 62 29
 0  0 11194    63   0   0   0   0    0   0 141  111  65 12  7 81  0
 0  0  5480   755   3   1   0   0    0   0 154  107  71 13  8 78  2
 0  0  5467  5747   0   3   0   0    0   0 167   39  68  1 16 79  5
 0  1  4797  5821   0  21   0   0    0   0 191  192 125 20  5 42 33
 0  1  3778  6119   0  24   0   0    0   0 188  170  98  5  8 41 46
 0  0  3751  6139   0   0   0   0    0   0 145   24  54  1 10 89  0
```

На начальном этапе обратите внимание на колонки *pi* и *po* в разделе `Страница` и четыре колонки в разделе `CPU`.

В колонках *pi* и *po* указаны сведения об операциях загрузки и выгрузки пространства подкачки. Если в отчете появляются эти записи, это может означать, что в системе не хватает памяти.

Если сумма значений в полях *us* и *sy* (соответствующих использованию CPU пользователем и системой) в разделе ресурсов CPU в каком-либо пятисекундном интервале превышает 90%, значит нагрузка на CPU в этом интервале была близка к предельной.

Если значение в колонке *wa* (ожидающие операции ввода-вывода) отлично от нуля, в то время как значения в колонках *pi* и *po* нулевые, то это означает, что некоторая часть времени уходит на ожидание операций ввода-вывода, то есть производительность некоторых процессов в системе ограничена ресурсами устройств ввода-вывода.

Если в отчете команды **vmstat** указано значительное время ожидания ввода-вывода, то рекомендуется воспользоваться командой **iostat** для получения более подробной информации.

Ниже приведен пример команды **iostat**, которая каждые пять секунд выводит отчет, содержащий данные об интенсивности операций ввода-вывода и об использовании CPU; поскольку после значения интервала указана цифра 3, число отчетов будет ограничено тремя:

```
# iostat 5 3
```

Приведенный ниже отчет создан утилитой **iostat** в той же системе, что и примеры отчетов команды **vmstat**, но в другое время. Первый отчет содержит информацию, собранную с момента загрузки системы, а последующие отчеты - информацию, собранную за прошедший 5-минутный интервал:

```
tty:      tin         tout   avg-cpu:  % user    % sys     % idle    %iowait
          0.0       4.3            0.2       0.6       98.8      0.4

Диски :       % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk0           0.0       0.2       0.0              7993      4408
hdisk1           0.0       0.0       0.0              2179      1692
hdisk2           0.4       1.5       0.3             67548     59151
cd0              0.0       0.0       0.0          0         0

tty:      tin         tout   avg-cpu:  % user    % sys     % idle    %iowait
          0.0      30.3              8.8      7.2       83.9     0.2

Диски :       % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk0           0.2       0.8       0.2                 4         0
hdisk1           0.0       0.0       0.0          0         0
hdisk2           0.0       0.0       0.0          0         0
cd0              0.0       0.0       0.0          0         0

tty:      tin         tout   avg-cpu:  % user    % sys     % idle    %iowait
          0.0       8.4              0.2      5.8        0.0      93.8

Диски :       % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk0           0.0       0.0       0.0          0         0
hdisk1           0.0       0.0       0.0          0         0
hdisk2          98.4     575.6      61.9        396      2488
cd0              0.0       0.0       0.0          0         0
```

Из первого отчета видно, что процессы ввода-вывода в системе не сбалансированы. Большая часть операций ввода-вывода (86,9% считанных и 90,7% записанных данных) относится к диску `hdisk2`, на котором расположена операционная система и пространство подкачки. Статистика *использования CPU с момента загрузки системы* обычно не несет в себе полезной информации, за исключением случаев, когда система используется круглосуточно.

Из второго отчета видно, что число обращений к диску hdisk0 невелико. На этом диске расположена файловая система основного пользователя. Нагрузка на процессор увеличилась вследствие запуска двух приложений и самой утилиты **iostat**.

Третий отчет относится к периоду, когда нагрузка была искусственно повышена путем запуска программы, которая захватила большой объем памяти (в этом примере 26 МБ) и записала в него данные. Поэтому создавшаяся ситуация близка к перегрузке памяти. Кроме того, в данном примере диск hdisk2 активен 98,4 процента времени, то есть в 93,8 процентах случаев приложение вынуждено ждать освобождения диска.

Лимитирующий фактор для отдельной программы

Если вы - единственный пользователь системы, то с помощью команды **time** вы можете выяснить, ограничена ли производительность программы ресурсами CPU или ввода-вывода. Ниже приведен пример этой команды:

```
# time cp foo.in foo.out

фактическое    0м0,13с
польз.         0м0,01с
сист.          0м0,02с
```

Примечание: Примеры применения команды **time** относятся к ее версии, встроенной в оболочку Korn (**ksh**). Стандартная команда **time**, /usr/bin/time, отличается меньшей точностью.

В нашем примере тот факт, что фактическое время выполнения программы **cp** (0,13 секунд) значительно превышает сумму времени для пользователя и CPU (0,03 с), означает, что производительность программы ограничена скоростью ввода-вывода. Как правило, это вызвано тем, что файл foo.in не находится в оперативной памяти.

В системах SMP вывод команды интерпретируется по-другому. Дополнительная информация приведена в разделе “Рекомендации по использованию команд time и timex” на стр. 114.

Запустив ту же команду через несколько секунд, вы получите следующий вывод:

```
фактическое    0м0,06с
польз.         0м0,01с
системное     0m0,03s
```

Большая часть страниц файла foo.in все еще находится в памяти, поскольку нет другого процесса, который мог бы захватить память, и поскольку объем файла невелик по сравнению с общим объемом оперативной памяти в системе. Если бы файл foo.out применялся другой программой в качестве файла ввода, эта программа почти не зависела бы от ресурсов диска, так как файл хранился бы в буфере памяти.

Для того чтобы выяснить зависимость производительности программы от ресурсов диска, необходимо обеспечить чистоту эксперимента. Это означает, что если проверяемая программа в обычных условиях обрабатывает файл, с которым давно никто не работал, то необходимо убедиться, что файл, служащий тестовым примером, не находится в памяти. Если же программа обычно выполняется в рамках конвейера, и ее вводом служит вывод предыдущей программы из того же пакета, то необходимо, напротив, поместить файл ввода в память. Например, следующая команда загрузит страницы файла foo.in в память:

```
# cp foo.in /dev/null
```

Положение осложняется, если размер файла сравним с общим объемом памяти. Если вывод одной программы служит вводом для другой, и при этом он не помещается в память целиком, то вторая программа будет считывать страницы с начала файла, которые постепенно будут замещать страницы с конца файла. Несмотря на то, что такую ситуацию очень трудно точно смоделировать, она очень близка к работе с диском без кэширования.

В том случае, если размер файла хотя бы ненамного превышает объем оперативной памяти, необходимо проанализировать ресурсы памяти и диска. Эта ситуация рассматривается в следующем разделе.

Неполадки, связанные с диском или памятью

В то время как большой фрагмент оперативной памяти используется для буферизации файлов, часть данных программы может принудительно выгружаться из памяти на диск. Для временного хранения таких данных служит системное пространство подкачки.

Предположим, что программа считывает очень мало данных или не считывает их вообще, и тем не менее оказывается зависимой от ресурсов ввода-вывода. Кроме того, соотношение между фактическим временем выполнения и суммой времени пользователя и системы не улучшается при предварительном помещении ввода в память. В этом случае производительность программы и эффективность выполнения ее операций ввода-вывода могут быть ограничены ресурсами памяти, и, в частности, ресурсами пространства подкачки. Проверить это предположение можно с помощью следующего сценария **vmstatit**:

```
vmstat -s >temp.file   # данные за все время до запуска команды
time $1                   # тестируемая команда
vmstat -s >>temp.file  # данные за все время до конца выполнения
grep "pagi.*ins" temp.file >>results   # выбираются только те данные,
grep "pagi.*outs" temp.file >>results  # которые необходимы
```

Сценарий **vmstatit** в сжатом виде выдает основную информацию из отчета команды **vmstat -s**, содержащего сведения, накопленные с момента запуска системы.

Если этот сценарий оболочки запущен следующей командой:

```
# vmstatit "cp file1 file2"  2>results
```

То эта команда создаст следующий вывод:

```
фактическое    0м0,03с
польз.         0м0,01с
сист.          0м0,02с
     2323 страниц загружено из пространства подкачки
     2323 страниц загружено из пространства подкачки
     4850 страниц выгружено в пространство подкачки
     4850 страниц выгружено в пространство подкачки
```

Статистика подкачки за время выполнения команды не изменилась. Это говорит о том, что команда **cp** не выполняла подкачку. Для уточнения ситуации можно вызвать другой вариант сценария оболочки **vmstatit**:

```
vmstat -s >temp.file
time $1
vmstat -s >>temp.file
echo "Стандартный ввод:"               >>results
grep "^[ 0-9]*page ins"    temp.file >>results
echo "Стандартный вывод:"               >>results
grep "^[ 0-9]*page outs"   temp.file >>results
echo "Вывод пространства подкачки:"  >>results
grep "pagi.*outs"          temp.file >>results
echo "Ввод пространства подкачки:"   >>results
grep "pagi.*ins"           temp.file >>results
```

Поскольку всеми обычными операциями ввода-вывода в операционной системе управляет VMM, в выводе команды **vmstat -s** обычные операции ввода-вывода интерпретируются как операции загрузки и выгрузки страниц. Запуск приведенной выше версии сценария оболочки **vmstatit** для команды **cp**, обрабатывающей большой файл, который до этого не был загружен в память, дал следующие результаты:

```
фактическое    0м2,09с
пользов.       0м0,03с
сист.          0м0,74с
Обычный ввод:
    46416 страниц загружено
    47132 страниц загружено
Обычный вывод:
   146483 страниц выгружено
   147012 страниц выгружено
Вывод в пространство подкачки:
```

```
     4854 страниц выгружено в пространство подкачки
     4854 страниц выгружено в пространство подкачки
Ввод из пространства подкачки:
     2527 страниц загружено из пространства подкачки
     2527 страниц загружено из пространства подкачки
```

Вывод команды **time** подтверждает предположение о том, что производительность программы ограничена ресурсами ввода-вывода. Увеличение количества загруженных страниц указывает на число операций ввода-вывода, необходимое для выполнения команды **cp**. Увеличение количества выгруженных страниц означает, что файл достаточно велик для того, чтобы при его обработке ожидающие выгрузки страницы (возможно, не относящиеся к этому файлу) принудительно выгружались на диск. Сохранение общего количества операций ввода-вывода, связанных с подкачкой, подтверждает, что команда **cp** не создает очень больших структур данных, способных исчерпать ресурсы памяти системы, на которой проводится эксперимент.

Порядок, в котором эта версия сценария **vmstatit** выдает информацию об операциях ввода-вывода, выбран не случайно. Обычная программа сначала считывает свои входные данные, а затем записывает выходные. Подкачка же обычно начинается с выгрузки ненужной страницы из рабочего сегмента. Эта страница загружается обратно в память только в том случае, если программа пытается обратиться к ней. То обстоятельство, что в тестируемой системе количество `страниц, выгруженных в пространство подкачки` с момента загрузки, почти вдвое превышает количество `страниц, загруженных в пространство подкачки` за то же время, означает, что некоторые программы, выполняемые в системе, помещают в память свои данные и больше не обращаются к ним. Дополнительная информация приведена в разделе “Программы с ограничением по памяти” на стр. 96. Обратитесь также к разделу “Быстродействие памяти” на стр. 128.

Для того чтобы показать, как будут выглядеть результаты выполнения этого сценария в случае, когда производительность программы ограничена объемом памяти, рассмотрим следующую ситуацию. Сначала проанализируем работу программы при достаточно большом объеме памяти (32 МБ), а затем с помощью команды **rmss** (см. раздел “Оценка необходимого объема памяти с помощью команды rmss” на стр. 143) искусственно уменьшим объем памяти. Последовательность команд

```
# cc -c ed.c
# vmstatit "cc -c ed.c" 2>results
```

сначала загружает в память исходный файл длиной 7944 строки и исполняемый файл компилятора C, а затем повторно измеряет количество операций ввода-вывода:

```
фактическое    0м7,76с
польз.         0м7,44с
сист.          0м0,15с
Обычный ввод:
    57192 страниц загружено
    57192 страниц загружено
Обычный вывод:
   165516 страниц выгружено
   165553 страниц выгружено
Вывод в пространство подкачки:
    10846 страниц выгружено в пространство подкачки
    10846 страниц выгружено в пространство подкачки
Ввод из пространства подкачки:
     6409 страниц загружено из пространства подкачки
     6409 страниц загружено из пространства подкачки
```

Очевидно, что ресурсы ввода-вывода не замедляют обработку программы. В данном случае, для считывания исходного кода операции ввода-вывода вообще не нужны. Если теперь с помощью команды

```
# rmss -c 8
```

уменьшить рабочий объем памяти компьютера до 8 МБ, а затем повторно ввести ту же последовательность команд, то результат будет следующим:

```
фактическое    0м9,87с
пользов.       0м7,70с
сист.          0м0,18с
Обычный ввод:
    57625 страниц загружено
    57809 страниц загружено
Обычный вывод:
   165811 страниц выгружено
   165882 страниц выгружено
Вывод в пространство подкачки:
    11010 страниц выгружено в пространство подкачки
    11061 страниц выгружено в пространство подкачки
Ввод из пространства подкачки:
     6623 страниц загружено из пространства подкачки
     6701 страниц загружено из пространства подкачки
```

Появились следующие признаки того, что производительность программы ограничена скоростью ввода-вывода:

- Фактическое время выполнения больше, чем общее время работы процессора
- Значительный объем обычных операций ввода-вывода при *повторном* выполнении команды

Увеличение фактического времени выполнения при уменьшении доступного объема памяти и значительное число операций подкачки свидетельствуют о том, что работа компилятора осложнена из-за недостатка памяти.

Примечание: Этот пример иллюстрирует влияние нехватки памяти на работу программы. Поскольку в данном случае не был уменьшен объем памяти, используемый другими процессами, точный объем памяти, начиная с которого компилятор вынужден применять подкачку, таким способом определить нельзя.

Для снятия ограничений на доступный объем памяти введите команду:

```
# rmss -r
```

Эта команда разрешает операционной системе использовать память, заблокированную командой **rmss**, то есть восстанавливает обычный объем оперативной памяти в системе.

Диагностика рабочей схемы

Управление рабочей схемой по сути заключается в присвоении приоритетов каждому компоненту рабочей схемы.

Если вы попробовали все способы повышения производительности и варианты настройки системы, но, несмотря на это, иногда производительность снижается до недопустимого уровня, то у вас есть следующие возможности:

- Оставить все как есть
- Модернизировать ресурс, ограничивающий производительность
- Изменить рабочую схему

Первый вариант приведет к снижению эффективности работы части пользователей. Второй вариант потребует дополнительных расходов. Таким образом, очевидный выход состоит в изучении возможности изменения рабочей схемы.

Как правило, есть задания, выполнение которых можно отложить. Например, отчет, который потребуется утром, можно составить как в 4 часа вечера накануне, так и в три часа ночи. Разница будет заключаться только в том, что в три часа ночи центральный процессор и другие ресурсы скорее всего будут простаивать. Для того чтобы запланировать запуск программы на определенное время, либо задать период запуска программы, воспользуйтесь командой **at** или **crontab**.

Аналогично, некоторые программы, которые должны выполняться в течение рабочего дня, можно запустить с пониженным приоритетом. Они будут работать дольше, зато более важные приложения получат больше процессорного времени.

Другой способ заключается в перенесении части нагрузки на другие компьютеры. Например, компиляцию можно выполнять в той системе, в которой хранится исходный текст программ. Такой способ распределения нагрузки требует более тщательного планирования и изучения работы системы, так как снижение нагрузки на сеть за счет повышения нагрузки на процессор сервера может привести к снижению общей производительности работы сети.

В ядре операционной системы предусмотрен стандартный компонент Управление AIX WLM (WLM). WLM предоставляет администратору расширенные возможности для управления распределением ресурсов CPU и оперативной памяти путем настройки планировщика и администратора виртуальной памяти (VMM). С помощью WLM можно управлять и распределением дисковой памяти. С его помощью можно предотвратить конфликт заданий различных типов, а также явно указать объем ресурсов, которые должен выделяться различным группам пользователей. Более подробная информация по этому вопросу приведена в документе *Server Consolidation on RS/6000*.

Управление ресурсами

AIX предоставляет тонко настраиваемые компоненты для управления ресурсами, влияющими на производительность системы.

Ниже перечислены главы, в которых приведены рекомендации по настройке:

- “Производительность процессора” на стр. 104.
- “Быстродействие памяти” на стр. 128.
- “Производительность логических томов и дискового ввода-вывода” на стр. 178.
- “Быстродействие сети” на стр. 258.
- “Производительность NFS” на стр. 329.

Производительность планировщика процессора

Следует обратить внимание на несколько ограничений, связанных с планировщиком процессора и влияющих на производительность.

Поддержка нитей

Нитью называется процесс, использующий небольшой объем ресурсов. Это управляемый объект, для создания которого требуется меньше ресурсов, чем для обычного процесса. В операционной системе AIX версии 4 планировщик работает с нитями.

Процесс может состоять из одной или нескольких нитей. Операционная система точно так же, как и раньше, создает процессы приложений, перенесенных из более младших выпусков, и управляет ими. Такие процессы создаются в виде единичных нитей с приоритетом родительского процесса, которые конкурируют за ресурсы процессора с нитями других процессов. Процесс владеет ресурсами, которые используются при выполнении; у нити есть только ее текущее состояние.

Все дополнительные нити новых или обновленных приложений создаются в контексте процесса. Они работают с общим сегментом памяти процесса и другими ресурсами.

С пользовательской нитью процесса связана *область действия*. Нить с *глобальной* областью действия конкурирует за процессор со всеми остальными нитями системы. Нить, создаваемая одновременно с процессом, всегда имеет глобальную область действия. Нить с *локальной* областью действия конкурирует только с нитями этого же процесса за долю времени процессора, выделенную процессу.

Алгоритм выбора следующей нити для выполнения называется *стратегией планирования*.

Процессы и нити

Процесс представляет собой набор действий, выполняемых системой. Он может быть запущен командой, программой оболочки или другим процессом.

С процессом связаны следующие свойства:

- pid
- pgid
- uid
- gid
- environment
- cwd
- дескрипторы файлов
- сигналы
- статистическая информация
- nice

Эти свойства описаны в файле `/usr/include/sys/proc.h`.

С нитью связаны следующие свойства:

- stack
- scheduling policy
- приоритет планирования
- ожидающие сигналы
- блокированные сигналы
- информация о нити

Эти свойства определены в файле `/usr/include/sys/thread.h`.

Каждый процесс состоит из одной или нескольких нитей. Нить представляет собой упорядоченный поток управления. Несколько нитей, или потоков управления, позволяют приложению параллельно выполнять несколько операций, например, считывать данные с терминала и записывать данные в файл.

Кроме того, приложение, разбитое на несколько нитей, может одновременно выполнять запросы нескольких пользователей. Нити позволяют использовать эти возможности, не требуя такого объема ресурсов, как процессы, создаваемые с помощью системных вызовов **fork()**.

В AIX добавлена функция **f_fork()**, предназначенная для быстрого порождения процесса. Эта функция особенно полезна в тех приложениях с несколькими нитями, в которых сразу за вызовом **fork()** вызывается **exec()**. Функция **fork()** работает относительно медленно, так как она вызывает несколько обработчиков для получения блокировок на все библиотеки, и лишь затем порождает дочерний процесс и позволяет ему запустить все необходимые обработчики для инициализации блокировок. Функция **f_fork()** не вызывает эти обработчики, сразу переходя к системному вызову **kfork()**. Примером приложений, в которых может использоваться функция **f_fork()**, являются Web-серверы.

Приоритет процессов и нитей

В операционной системе предусмотрены специальные средства управления приоритетом процессов.

В операционной системе AIX версии 4 приоритет процесса наследуется нитью. При вызове функции **fork()** создается процесс и одна его нить. Нить получает такой приоритет, который раньше был бы присвоен процессу.

Значение *приоритета* нити (который иногда называется *приоритетом планирования*) хранится в ядре. Приоритет - это натуральное число, обратно пропорциональное важности нити. Таким образом, чем меньше приоритет, тем важнее нить. Планировщик выбирает для запуска нить с наименьшим значением приоритета.

Приоритет нити может быть фиксированным или переменным. Фиксированный приоритет нити - это константа, тогда как переменный приоритет зависит от минимального приоритета пользовательских нитей (40), составляющей приоритета, задаваемой командой **nice** или **renice** (по умолчанию равно 20) и штрафа за использование процессора.

С помощью функции **setpri()** для нити можно установить фиксированный приоритет, меньший 40. Приоритет таких нитей не пересчитывается планировщиком. Нити с фиксированным приоритетом меньше 40 будут запущены и выполнены раньше всех пользовательских нитей. Например, нить с фиксированным приоритетом 10 будет запущена раньше, чем нить с фиксированным приоритетом 15.

С помощью команды **nice** пользователь может увеличить переменный приоритет нити. Администратор системы может указать в команде nice отрицательный аргумент, чтобы уменьшить приоритет нити.

Ниже показано несколько способов изменения приоритета.

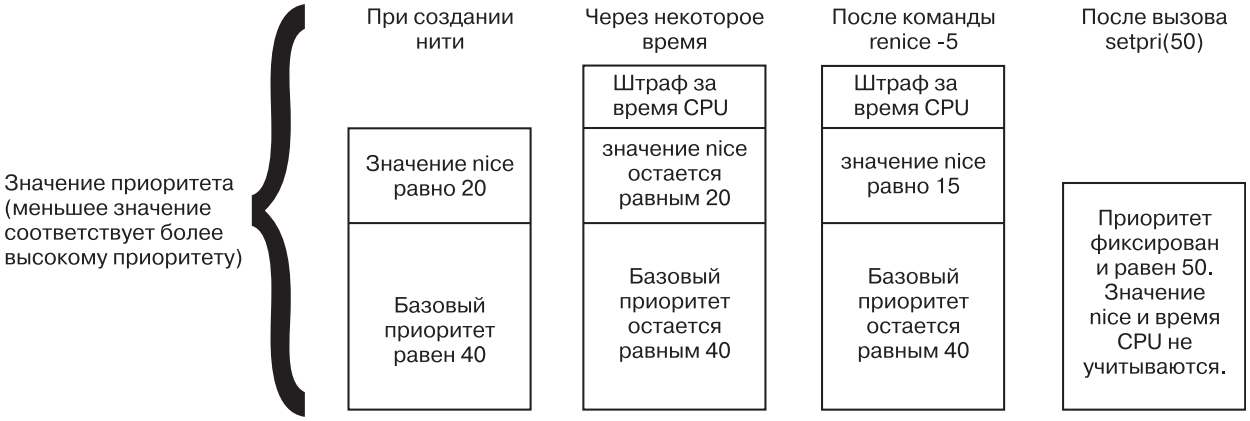


Рисунок 6. Способ определения приоритета. На приведенном рисунке показано, каким образом приоритет планирования нити может измениться во время выполнения или после выполнения команды nice. Чем меньше значение приоритета, тем выше приоритет нити. Вначале значение nice составляет 20, а базовый приоритет - 40. В течение некоторого времени после запуска значение nice остается равным 20, а базовый приоритет - 40. После вызова команды renice —5 значение nice стало равным 15, а базовый приоритет остался равен 40. После вызова функции setpri() с аргументом 50 был установлен фиксированный приоритет 50, не зависящий от значения nice и штрафа за использование процессора.

Значение nice устанавливается при создании нити и изменяется только явным образом с помощью команды **renice**, либо системного вызова **setpri()**, **setpriority()**, **thread_setsched()** или **nice()**.

Штраф за использование процессора - это целочисленное значение, которое пропорционально объему ресурсов процессора, затраченных на выполнение нити за последнее время. Показатель использования процессора увеличивается примерно на 1 в конце каждого интервала времени длиной 10 мсек, если в этот момент управление принадлежит данной нити. Максимальное значение равно 120. Штраф, назначаемый с каждым тактом процессора, возрастает вместе со значением **nice**. Раз в секунду показатели использования процессора пересчитываются для всех нитей.

В результате:

- Переменный приоритет нити увеличивается по мере увеличения нагрузки на процессор (и наоборот). В целом это означает, что чем больше данная нить использовала ресурсы CPU в последнее время, тем менее вероятно, что она будет запущена в следующем кванте времени.
- Переменный приоритет нити увеличивается по мере увеличения значения nice (и наоборот).

Примечание: В многопроцессорных средах с собственными механизмами распределения нагрузки значения **nice** и **renice** не могут непосредственно влиять на приоритет нитей, находящихся в очередях выполнения отдельных процессоров, и возможны ситуации, когда нити с меньшим приоритетом получают больше процессорного времени, чем нити с большим приоритетом. Нити, для которых нужно точно задавать приоритет командами **nice** и **renice**, следует помещать в глобальную очередь выполнения.

Приоритет, значение nice и характеристику использования процессора для процесса можно просмотреть с помощью команды **ps**.

Дополнительная информация о применении команд **nice** и **renice** приведена в разделе “Управление конкуренцией за микропроцессор” на стр. 122.

Более подробная информация о вычислении штрафа за использование процессора и увеличении текущего значения использования процессора приведена в разделе “Вычисление значения приоритета нити” на стр. 125.

Механизм приоритетов применяется в AIX WLM для управления ресурсами процессора. Приоритет нитей, которыми управляет WLM, может отличаться от приоритета остальных нитей.

Стратегии планирования для нитей

В стратегию планирования включены возможные значения для нитей.

SCHED_FIFO
: При таком способе планирования выполнение нити будет прервано только в случае, если эта нить будет блокирована, если она самостоятельно вернет управление процессором, либо если будет готова к выполнению нить с более высоким приоритетом. Стратегия планирования SCHED_FIFO может применяться только для нитей с фиксированным приоритетом.

SCHED_RR
: Если в конце кванта времени управление принадлежит некоторой нити SCHED_RR, то она перемещается в конец очереди готовых к выполнению нитей с таким же приоритетом. Стратегия планирования SCHED_RR может применяться только для нитей с фиксированным приоритетом.

SCHED_OTHER
: Эта стратегия определена в стандарте 1003.4a POSIX как зависящая от реализации. Приоритет активной нити заново вычисляется при каждом прерывании. Это означает что нить может потерять управление, если значение ее приоритета станет больше, чем значение приоритета другой нити, ожидающей запуска.

SCHED_FIFO2
: В целом эта стратегия эквивалентна SCHED_OTHER, однако в ней нить, которая простаивала короткий промежуток времени, помещается в начало, а не в конец очереди выполнения. Этот период времени задается с помощью команды **schedo -o affinity_lim**.

SCHED_FIFO3
: Нить со стратегией планирования SCHED_FIFO3 всегда помещается в начало очереди выполнения. Для того чтобы нити со стратегией планирования SCHED_FIFO2 не помещались перед нитями SCHED_FIFO3, при занесении нити SCHED_FIFO3 в очередь параметры очереди меняются так, что ни одна нить SCHED_FIFO2 не будет соответствовать критерию размещения в начале очереди.

SCHED_FIFO4
: Нить класса планирования SCHED_FIFO4 с высоким приоритетом не замешает текущую нить с низким приоритетом, если их приоритеты отличаются на 1. По умолчанию замещение происходит, если нить с высоким приоритетом становится доступной для выполнения процессором.

Стратегия планирования задается с помощью системного вызова **thread_setsched()**. Он устанавливает стратегию для той нити, из которой отправлен вызов. Для того чтобы изменить стратегию планирования

нити на SCHED_RR можно вызвать функцию **setpri()**, указав ИД процесса. Процесс, из которого был отправлен вызов **setpri()**, не обязательно должен совпадать с процессом, заданным в качестве параметра функции **setpri()**.

Системный вызов **setpri()** может отправляться только процессами с правами доступа root. Кроме того, стратегию планирования могут изменять только нити с правами доступа root. В качестве аргумента они могут указать один из вариантов стратегии планирования SCHED_FIFO или SCHED_RR. Если в качестве параметра функции **thread_setsched()** будет указана стратегия SCHED_OTHER, то этот параметр будет проигнорирован.

В основном нити предназначены для приложений, которые раньше состояли из нескольких асинхронных процессов. Преобразование таких приложений в структуру из нескольких нитей позволит сэкономить ресурсы системы.

Очередь выполнения планировщика

Планировщик поддерживает очередь выполнения, в которой находятся все нити, готовые к выполнению.

На приведенном ниже рисунке приведена схема очереди выполнения.

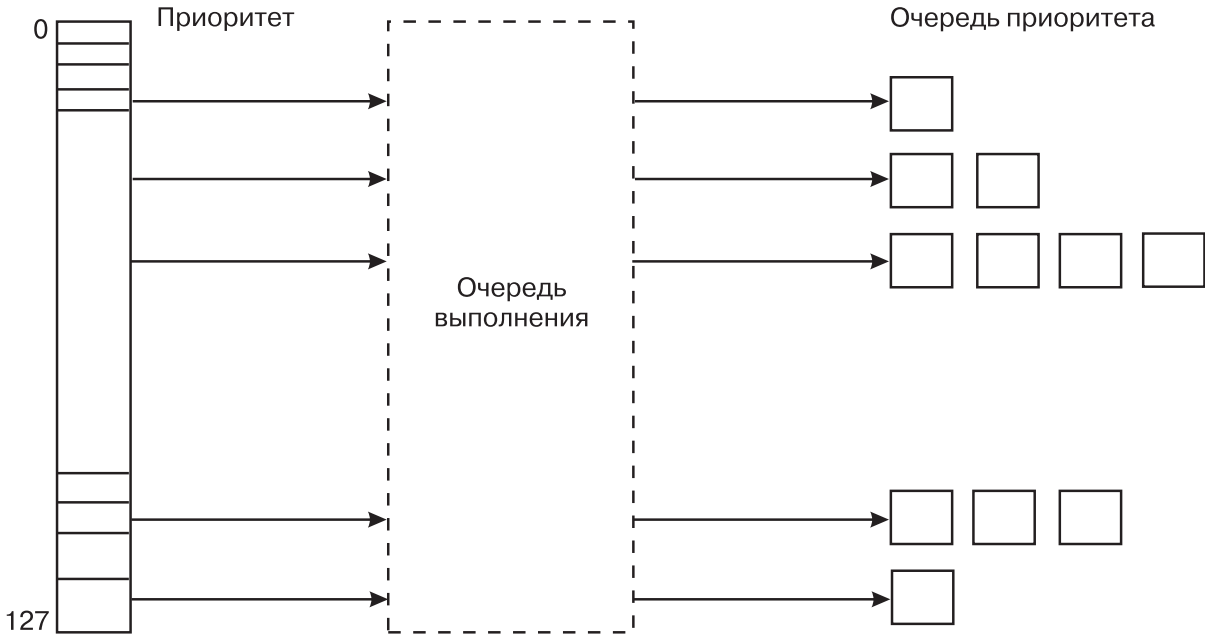


Рисунок 7. Очередь выполнения. На данном рисунке показано, что нити с более низким значением приоритета, помещаются в очередь выполнения перед нитями с более высоким значением приоритета. Число возможных значений приоритета (от 0 - 127) в точности совпадает с числом очередей выполнения (128).

В очереди выполнения расположены все готовые к выполнению нити с одинаковым приоритетом.

Планировщик работает с нитями. AIX обслуживает 256 очередей выполнения. В каждой очереди находятся нити с определенным значением приоритета (от 0 до 255). За счет этого планировщик может быстрее определить, какую нить нужно следующей передать на выполнение. Вместо того чтобы просматривать одну большую очередь выполнения, планировщик создает битовую маску. Если бит маски равен единице, то в соответствующей очереди есть нить, готовая к выполнению.

Приоритет нити изменяется достаточно часто. В таком случае постоянное перемещение нитей в очередях вызвано тем, как планировщик пересчитывает значения приоритетов. Однако это не относится к нитям с фиксированным приоритетом.

В операционной системе AIX версии 6.1 и выше у каждого процессора есть очередь выполнения уровня узла. Информация об очередях выполнения, выдаваемая средствами сбора статистики, будет отражать суммарные значения параметров всех очередей выполнения. Наличие отдельной очереди выполнения для каждого процессора позволяет сократить нагрузку, связанную с управлением блокировками, и более равномерно распределить общую нагрузку по процессорам. Нити реже перемещаются с процессора на процессор. Выполнение нити будет продолжено на другом процессоре только в том случае, если в системе есть простаивающий процессор, и на каком-то процессоре произошло событие, в результате которого нить стала готова к выполнению. Если простаивающего процессора нет, то нить будет дожидаться, пока освободиться тот процессор, на котором она выполнялась раньше, например, в результате прерывания.

В многопроцессорных системах с несколькими очередями выполнения может происходить динамическое изменение приоритета нитей. Вполне возможная такая ситуация, при которой у нескольких нитей в одной очереди выполнения приоритет будет выше, чем у всех нитей в другой очереди выполнения. В AIX предусмотрены механизмы компенсации приоритетов, но если приоритеты быть заданы жестко (например, при работе с приложениями реального времени), то существует переменная среды *RT_GRQ*. Если для переменной среды *RT_GRQ* указано значение ON, то нить помещается в глобальную очередь выполнения. В этом случае планировщик будет выбирать из глобальной очереди выполнения нить с минимальным приоритетом. Такая схема постановки в очередь позволяет ускорить выполнение нитей, для управления которыми применяются прерывания. Нити с фиксированным приоритетом помещаются в глобальную очередь выполнения в том случае, если флаг *fixed_pri_global* команды **schedo** равен 1.

Для того чтобы узнать среднее число нитей в очереди выполнения, вызовите команду **vmstat**. Это значение будет указано в первом столбце вывода. Разделив это значение на число процессоров, вы получите среднее число нитей, которое может выполняться на одном процессоре. Если это значение больше единицы, то указанные нити должны будут ждать своего переключения на процессор (чем больше это значение, тем больше будет время ожидания).

Когда нить помещается в конец очереди выполнения (например, если ей принадлежит управление в конце кванта времени), она занимает место за последней нитью с тем же значением приоритета.

Квант процессорного времени

Квантом процессорного времени называется время, по истечении которого выполнение нити с алгоритмом планирования SCHED_RR будет прервано, а управление будет передано другой нити с тем же приоритетом.

С помощью опции **timeslice** команды **schedo** можно увеличить число тактов процессора в кванте времени. Другими словами, квант времени можно увеличивать приращениями по 10 миллисекунд (см. раздел “Изменение кванта времени планировщика с помощью команды schedo” на стр. 127).

Примечание: Квант времени не является гарантированным квантом времени процессора. Он задает максимальное время, в течение которого нити может принадлежать управление до ее замены другой нитью. Существует множество причин, по которым нить может потерять управление процессором до истечения кванта времени.

Изменение режима

Когда пользовательскому процессу нужно захватить ресурсы системы, он изменяет режим выполнения. Это можно сделать с помощью системных вызовов, либо с помощью прерываний (например, страничных ошибок).

Существует два режима:

- Пользовательский режим
- Системный режим

Время работы процессора в пользовательском режиме (при работе с приложениями и общими библиотеками) указывается в выводах команд **vmstat**, **iostat** и **sar** как пользовательское время. Время работы процессора в системном режиме указывается в выводе этих команд как системное время.

Пользовательский режим:

В пользовательской области защиты выполняются пользовательские процессы.

Такие процессы выполняются в пользовательском режиме. Им предоставлены следующие права доступа:

- Права на чтение и запись пользовательских данных, расположенных в личной области процесса
- Права на чтение пользовательской и общей текстовой области памяти
- Права на обращение к общим областям памяти через соответствующие функции

У программ, выполняемых в пользовательской области защиты, нет прав доступа к области памяти ядра и области памяти данных ядра. Для обращения к этим областям памяти должны применяться системные вызовы. Программа из этой области защиты может изменять только собственную среду выполнения. Она выполняется как обычный процесс в непривилегированном режиме.

Системный режим:

В системной области защиты выполняются обработчики прерываний, основные функции ядра и расширения ядра (драйверы устройств, системные вызовы и функции для работы с файловыми системами).

Программы из системной области защиты выполняются в системном режиме. Им предоставляются следующие права доступа:

- Права на чтение и запись в глобальное адресное пространство ядра
- Права на чтение и запись в область данных ядра, расположенную в области памяти процесса (в случае процесса)

К пользовательским данным, расположенным в адресном пространстве процесса, можно обращаться только через соответствующие службы ядра.

Программы из этой области защиты могут изменять среду выполнения всех программ, так как они обладают следующими свойствами:

- Они могут обращаться к глобальным системным данным
- Они могут вызывать службы ядра
- К ним не применяются никакие ограничения защиты
- Они выполняются в привилегированном режиме.

Переключение режима:

С помощью системных вызовов, процессы, выполняющиеся в пользовательском режиме, могут обращаться к функциям ядра. Функции, которые явно или косвенно используют системные вызовы, обычно расположены в библиотеках программ, предоставляющих доступ к функциям операционной системы.

Следует отличать переключение режима от переключения контекста, которое указывается в выводе команд **vmstat** (столбец **cs**) и **sar** (*cswch/s*). Переключение контекста происходит в том случае, если на процессоре была запущена другая нить.

Планировщик переключает контекст в следующих случаях:

- Когда нить ждет освобождения необходимого ресурса, например, в случае дискового ввода-вывода, передачи данных по сети, перехода в неактивное состояние или блокировок
- Когда появилась нить с более высоким приоритетом, готовая к работе (принудительное переключение)
- Когда закончился квант времени (обычно квант равен 10 мс).

Суммарное время, затраченное процессором на переключение контекста, выполнение системных вызовов, обработку прерываний устройств, ввод-вывод NFS и выполнение других операций в ядре называется системным временем.

Производительность Администратора виртуальной памяти (VMM)

Все пространство виртуальных адресов разделено на сегменты. Сегмент представляет собой непрерывный участок виртуального адресного пространства размером 256 МБ, в который может быть помещен объект данных.

Адресация данных в процессах выполняется на уровне сегментов (или объектов), поэтому сегмент может быть разделен между процессами или заблокирован для других процессов. Например, у процессов может быть общий сегмент кода, но разные сегменты данных.

Физическая память, управление

Управление виртуальной памятью играет важную роль в управлении физической памятью.

Сегменты виртуальной памяти разбиты на блоки фиксированного размера, которые называются *страницами*. AIX 7.1, работающий на процессорах POWER5+, поддерживает четыре размера страниц: 4 КБ, 64 КБ, 16 МБ и 16 ГБ. Дополнительная информация приведена в разделе Поддержка разных размеров страниц. Каждая страница сегмента может располагаться как в оперативной памяти, так и на диске до тех пор, пока она не понадобится. Физическая память также разделена на страницы. Функция Администратора виртуальной памяти состоит в выделении страниц физической памяти и преобразовании ссылок на страницы виртуальной памяти, которые в данный момент выгружены на диск или еще не существуют (например, при первом обращении процесса к сегменту данных).

Так как объем используемой виртуальной памяти может превышать объем оперативной памяти, VMM хранит избыточные данные на диске. С точки зрения производительности к работе VMM предъявляется два требования, отчасти противоположные друг другу:

- Снижение затрат процессорного времени и нагрузки на дисковую подсистему при работе с виртуальной памятью.
- Уменьшение задержек при возникновении страничных ошибок.

Для выполнения этих требований VMM использует *список свободных страниц памяти*, который применяется при возникновении страничных ошибок. Список свободных страниц пополняется на основе алгоритма замены страниц. Алгоритм замены страниц использует несколько механизмов:

- Сегменты виртуальной памяти делятся на постоянные и рабочие.
- Сегменты виртуальной памяти делятся на сегменты, содержащие страницы файлов, и сегменты, применяемые для вычислений.
- Запоминаются страницы виртуальной памяти, обращение к которым приводит к страничной ошибке.
- Страничные ошибки делятся на новые и повторные.
- Для каждого сегмента виртуальной памяти ведется статистика повторных страничных ошибок.
- Решения, принимаемые алгоритмом, учитывают установленные пользователем пороговые значения.

Список свободных страниц:

Список свободных страниц физической памяти применяется VMM для обработки страничных ошибок.

В большинстве систем VMM пополняет список свободных страниц, случайно выбирая страницы физической памяти, принадлежащие некоторым процессам. Страницы виртуальной памяти, которые будут помещены на диск для освобождения оперативной памяти, выбираются на основе алгоритма замены страниц. Число перемещаемых страниц определяется пороговыми значениями VMM.

Постоянные и рабочие сегменты:

В отличие от постоянных, рабочие сегменты являются временными.

Страницам постоянных сегментов соответствует фиксированная область на диске. Постоянные сегменты формируются при работе с файлами данных и исполняемыми программами. Так как для каждой страницы постоянного сегмента существует выделенная область диска, при выгрузке измененных страниц из памяти VMM сразу записывает их в место постоянного хранения. Если страница, выбранная для пополнения списка свободных страниц, не изменилась, то она не записывается на диск. При повторном обращении к странице постоянной памяти она считывается из области постоянного хранения на диске.

Рабочие сегменты являются временными и существуют только во время работы процесса. Их расположение на диске не фиксировано. В рабочих сегментах размещаются стек и области данных процесса, сегменты кода ядра и расширений ядра, а также сегменты кода и данных общих библиотек. Для хранения выгружаемых из памяти страниц рабочих сегментов также выделяются области диска. Они называются пространствами подкачки.

На следующем рисунке проиллюстрированы способы хранения на диске страниц из различных сегментов памяти. На нем также показан пример реального расположения страниц в физической памяти.

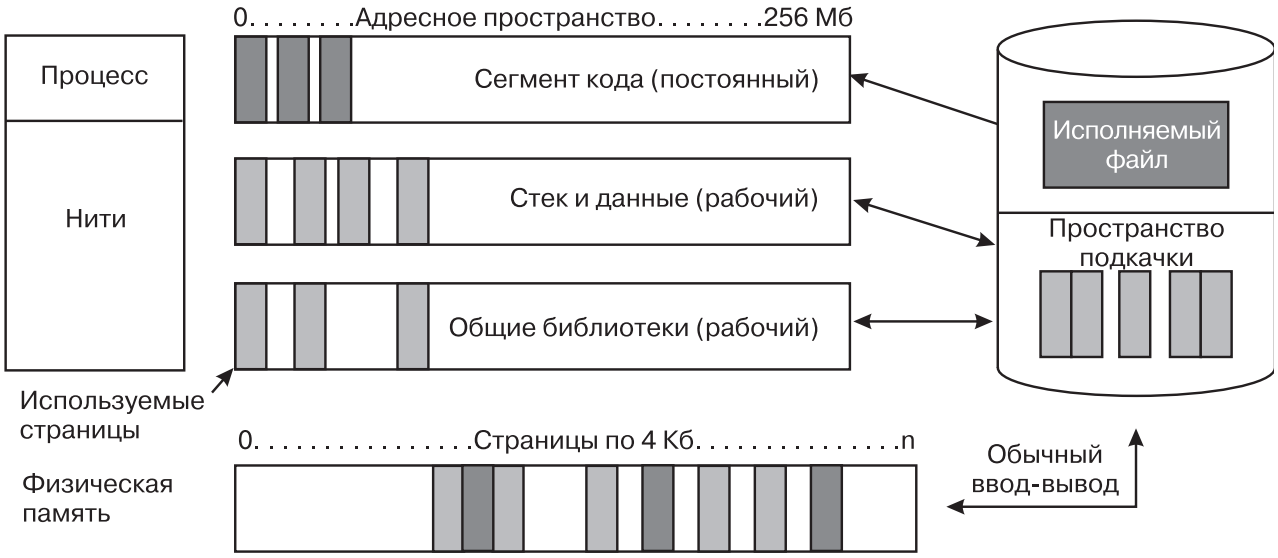


Рисунок 8. Постоянные и рабочие сегменты памяти. На этом рисунке проиллюстрированы способы хранения на диске страниц из различных сегментов памяти. На нем также показан пример реального расположения страниц в физической памяти. Рабочие сегменты являются временными, поскольку они существуют только во время работы процесса. Их расположение на диске не фиксировано. В рабочих сегментах размещаются стек и области данных процесса, сегменты кода ядра и расширений ядра, а также сегменты кода и данных общих библиотек. Для хранения выгружаемых из памяти страниц рабочих сегментов также выделяются области диска. Они называются пространствами подкачки.

Постоянные сегменты также разделяются на типы. *Сегменты клиента* применяются при работе с удаленными файлами (например, с файлами NFS), в том числе и с удаленными исполняемыми файлами. Страницы сегментов клиента считываются из области постоянного хранения на диске удаленной системы, а не из пространства подкачки. *Журнализированные и отложенные сегменты* - это постоянные сегменты, которые должны обновляться целиком. Если страница такого сегмента должна быть выгружена из памяти, она записывается в пространство подкачки на диске и хранится там до тех пор, пока ее нельзя будет зафиксировать (записать в постоянную область памяти файла).

Рабочая и файловая память:

Рабочая память состоит из страниц, хранящихся в рабочих сегментах памяти и сегментах кода программ.

К файловой памяти относятся все остальные страницы. Обычно это страницы стандартных файлов данных, расположенные в постоянных сегментах памяти.

Замена страниц:

Когда в списке свободных страниц физической памяти остается мало страниц, вызывается программа добавления страниц. Эта программа просматривает таблицу страниц физической памяти (PFT) и выбирает страницы для добавления в список.

PFT содержит флаги, которые указывают, было ли зарегистрировано обращение к странице, и была ли страница изменена. Если будет найдена страница, к которой обращался какой-либо процесс, то программа добавления страниц не выберет эту страницу, но сбросит флаг обращения к этой странице. Если при следующем просмотре таблицы программа добавления страниц обнаружит, что флаг обращения по-прежнему не установлен, то она выберет эту страницу. Все страницы, к которым никто не обращался, выбираются сразу.

Флаг изменения указывает, что с момента последней загрузки этой страницы в память хранящиеся в ней данные были изменены. Перед выбором страницы с установленным флагом изменения вызывается процедура выгрузки из памяти. Страницы рабочих сегментов выгружаются в пространство подкачки, а страницы постоянных сегментов - на диск.

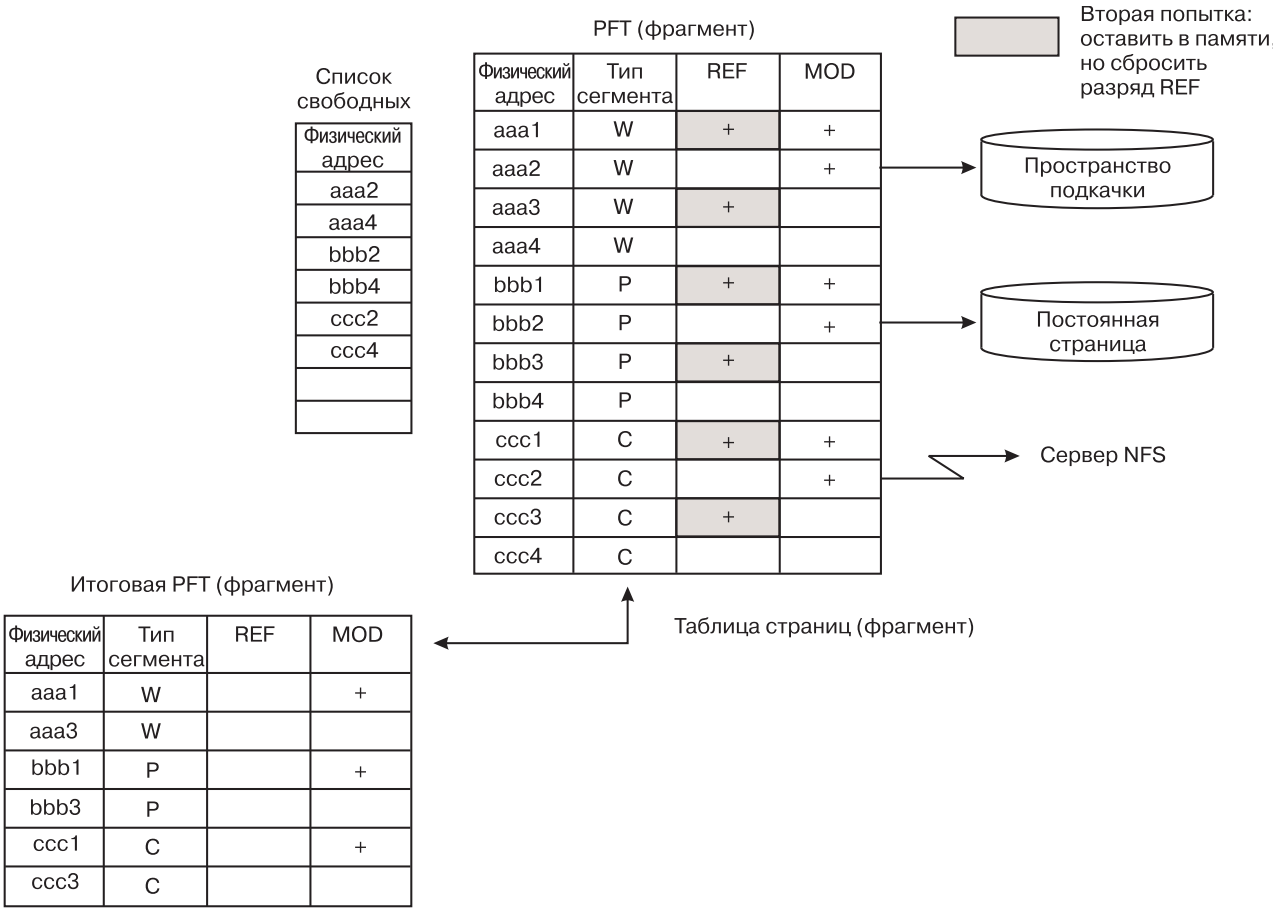


Рисунок 9. Пример замены страниц. На рисунке показаны фрагменты трех таблиц. Первая таблица - это таблица страниц, содержащая четыре столбца, в которых указываются адрес в физической памяти, тип сегмента, флаг обращения и флаг изменения. Вторая таблица содержит список адресов свободных страниц. Третья таблица представляет собой итоговую таблицу страниц после удаления всех адресов свободных страниц.

Алгоритм замены страниц также отслеживает новые страничные ошибки (возникшие при первом обращении к странице) и повторные страничные ошибки (возникшие при обращении к выгруженным страницам). Для этого используется хронологический буфер, который содержит ИД страничных ошибок, возникших за последнее время. После этого алгоритм пытается сбалансировать число выгруженных файловых страниц (с постоянными данными) с числом выгруженных рабочих страниц (страниц из рабочей памяти и страниц с кодом программ).

Сразу же после завершения процесса его рабочая память освобождается, а связанные с ним страницы физической памяти помещаются в список свободных страниц. Однако файлы, открытые этим процессом, могут остаться в оперативной памяти.

При работе в однопроцессорной среде замена страниц выполняется прямо в области нити. При работе в многопроцессорной системе замена страниц выполняется процессом ядра **lrud**. Процессор переключается на этот процесс, когда достигается порог *minfree*. Процесс ядра **lrud** реализован в виде нескольких нитей, каждая из которых работает со своим пулом памяти. Физическая память разделяется на пулы, размер которых кратен двум. Число пулов зависит от числа CPU и объема RAM. Число пулов памяти в системе можно определить командой **vmstat -v**.

Повторные страничные ошибки:

Страничные ошибки разделяются на новые и повторные. Страничная ошибка считается новой, если отсутствуют данные о предыдущих обращениях к странице. Повторная ошибка возникает в том случае, если

с момента недавнего обращения к странице к ней обратились повторно, и страницы не оказалось в оперативной памяти из-за того, что она была удалена из нее (и, возможно, записана на диск).

Идеальная стратегия замены страниц должна полностью исключить появление повторных страничных ошибок (при наличии достаточного количества оперативной памяти), освобождая только те страницы, которые не будут использоваться повторно. Таким образом, число повторных страничных ошибок является показателем, обратным эффективности работы алгоритма замены страниц с точки зрения интенсивности дискового ввода-вывода и общей производительности системы.

Для распознавания повторных страничных ошибок VMM поддерживает хронологический буфер повторных страничных ошибок, содержащий идентификаторы *N* последних страничных ошибок, где *N* - число страниц в оперативной памяти. Например, для оперативной памяти размером 512 МБ будет создан хронологический буфер страничных ошибок размером 128 КБ. При загрузке страницы в память выполняется поиск ее идентификатора в буфере и, если он будет обнаружен, то ошибка считается повторной. Этот буфер также позволяет VMM оценить интенсивность повторных страничных ошибок для рабочей и файловой памяти путем вычисления числа страничных ошибок для памяти каждого типа. При каждом запуске алгоритма замены страниц показатель интенсивности умножается на 0,9, чтобы новая информация о страничных ошибках учитывалась в большей степени, чем устаревшая.

Пороговые значения VMM:

Работа VMM управляется несколькими пороговыми значениями. При достижении одного из пороговых значений VMM выполняет определенные действия для возврата параметров памяти в заданные границы. В этом разделе описаны пороговые значения, которые могут быть изменены системным администратором с помощью команды **vmo**.

Размер списка свободных страниц определяется следующими параметрами:

minfree
Минимально допустимое число страниц физической памяти в списке свободных страниц. Если число страниц в списке снижается до этого значения, VMM начинает процесс освобождения памяти. Этот процесс выполняется до тех пор, пока число страниц в списке не достигнет значения *maxfree*.

maxfree
Максимальное число страниц физической памяти, до которого VMM будет пополнять список свободных страниц. Объем списка свободных страниц может превысить эту границу в результате завершения процесса с освобождением рабочих сегментов или удаления файла, страницы которого загружены в память.

VMM старается поддерживать размер списка свободных страниц не ниже *minfree*. Когда из-за большого числа страничных ошибок или системных запросов размер списка становится меньше *minfree*, запускается алгоритм замены страниц. Размер списка свободных страниц должен превышать определенный уровень (значение *minfree* по умолчанию) по ряду причин. Например, используемый в операционной системе алгоритм упреждающей выборки при последовательном чтении требует наличия сразу нескольких свободных страниц для процесса, выполняющего последовательное чтение. Кроме этого, VMM предотвращает блокировку системы в ситуациях, когда для чтения страницы, необходимой для освобождения страницы оперативной памяти, не хватает памяти.

Описанные ниже пороговые значения выражены в процентах. Они задают долю физической памяти системы, занятую файловыми страницами, то есть страницами, не относящимися к рабочим сегментам.

minperm
Если доля файловой памяти станет меньше данного значения, алгоритм замены страниц начнет освобождение памяти (как файловой, так и рабочей), независимо от интенсивности повторных страничных ошибок.

maxperm
Если доля файловой памяти станет больше данного значения, алгоритм замены страниц начнет освобождение страниц файловой памяти.

maxclient
Если доля файловой памяти станет больше данного значения, алгоритм замены страниц начнет освобождение страниц памяти клиентов.

Если объем физической памяти, занятой файловыми страницами, находится между значениями *minperm* и *maxperm*, VMM пополняет список свободных страниц только за счет файловых страниц, однако если интенсивность повторных страничных ошибок для файловой памяти превышает аналогичный показатель для рабочей памяти, то для пополнения выбираются и страницы рабочей памяти.

Главной задачей алгоритма замены страниц является правильная обработка рабочих страниц. Например, последовательное чтение большого файла данных не должно приводить к выгрузке кодового сегмента программы, который, вероятно, потребуется в будущем. Использование пороговых значений и вычисление интенсивности повторных страничных ошибок позволяет алгоритму замены страниц равномерно обслуживать память обоих типов, с небольшим перевесом в сторону рабочей памяти.

Средство управления нагрузкой на память VMM

Для выполнения процессу необходимы страницы физической памяти. При обращении процесса к странице виртуальной памяти, которая находится на диске в силу того, что она была выгружена из памяти или ни разу не была прочитана, эта страница должна быть загружена в оперативную память. Обычно при этом из памяти должна быть выгружена одна или несколько страниц (если эти страницы были изменены), что требует выполнения ряда операций ввода-вывода и приводит к задержке в работе процесса.

Алгоритм замены страниц операционной системы пытается выгружать только те страницы, к которым вряд ли будут обращаться в ближайшее время. Успешная работа алгоритма замены страниц позволяет операционной системе поддерживать работу достаточного числа процессов и эффективно использовать центральный процессор. Однако при большом числе параллельно работающих процессов в памяти не будет страниц, которые могут быть выгружены на диск, так как все они в ближайшем будущем будут использоваться работающими процессами. Такая ситуация зависит от следующих факторов:

- Общего объема оперативной памяти в системе
- Числа процессов
- Текущих требований к памяти каждого процесса
- Алгоритма замены страниц

В таких случаях страницы будут непрерывно загружаться и выгружаться из памяти. Такая ситуация называется *перегрузкой памяти*. В результате перегрузки памяти возникает избыточный страничный обмен, и страничные ошибки возникают при передаче управления каждому процессу, в результате чего ни один процесс не может нормально работать.

Самым неприятным является то, что перегрузка может быть вызвана кратковременным случайным всплеском активности (например, одновременным нажатием клавиши Enter всеми пользователями), после чего система останется заблокированной на неопределенное время.

В операционной системе применяется алгоритм управления нагрузкой на память, который в системе с перегруженной памятью приостанавливает работу некоторых процессов и откладывает запуск новых процессов на определенное время. Работа алгоритма определяется пятью параметрами. По умолчанию для них установлены значения, которые были протестированы в системах с разной нагрузкой. В операционной системе AIX версии 4 управление нагрузкой на память по умолчанию выключено, если суммарный объем свободных страниц физической памяти составляет 128 Мб или более.

Алгоритм управления нагрузкой на память:

Механизм управления нагрузкой на память раз в секунду оценивает объем памяти, доступный набору активных процессов. При обнаружении перерасхода памяти работа некоторых процессов приостанавливается, что позволяет снизить расход памяти.

Когда процесс приостанавливается, останавливаются все его нити, по мере достижения состояния, в котором они могут быть остановлены. Страницы таких процессов быстро устаревают и выгружаются из памяти алгоритмом замены страниц, освобождая память для работы оставшихся процессов. Если некоторые процессы приостановлены, то новые процессы также не создаются, чтобы нагрузка на систему не увеличивалась. Процессы приостанавливаются на определенный период времени, в течение которого система не будет перегружена. По истечении этого периода времени нити приостановленных процессов постепенно активизируются.

Параметры управления нагрузкой на память **schedo** задают следующие значения:

- Порог перерасхода памяти системы (*v_repage_hi*)
- Длительность периода времени в секундах, на который приостанавливаются процессы (*v_sec_wait*)
- Пороги перерасхода памяти для отдельных процессов, на основании которых выбираются процессы, которые будут приостановлены (*v_repage_proc*)
- Минимальное число активных процессов, которое должно быть оставлено после того, как некоторые процессы будут приостановлены (*v_min_process*)
- Минимальное число секунд, в течение которых процесс должен выполняться после повторной активизации (*v_exempt_secs*)

Информация о настройке этих параметров приведена в разделе “Настройка алгоритма управления нагрузкой на память VMM с помощью команды schedo” на стр. 149.

Раз в секунду планировщик (процесс 0) проверяет описанные выше показатели, измеренные за прошедшую секунду работы, и определяет, какие процессы должны быть приостановлены или активизированы. Если в системе возник перерасход памяти, то все процессы, которые после оценки параметров **-p** и **-e** являются кандидатами на остановку, помечаются для остановки. После того как одному из таких процессов передается управление в пользовательском режиме, он приостанавливается (если после этого число активных процессов не станет меньше **-m**). Для того чтобы не останавливались процессы, от имени которых системой выполняются важные операции, работа процессов приостанавливается только в пользовательском режиме. Если в течение следующей секунды память по-прежнему будет перегружена, для остановки помечаются дополнительные процессы, выбранные на основе значений **-p** и **-e**. После того как работа системы будет стабилизирована, приостановленные процессы будут помещаться в очередь выполнения (активизироваться).

Приостановленные процессы повторно активизируются одним из следующих способов:

1. По величине приоритета
2. В том же порядке, в каком они останавливались

Приостановленные процессы никогда не активизируются одновременно. Число возобновляемых процессов равно одной пятой от текущего числа активных процессов или монотонно возрастающей нижней границе, если она больше. Такая стратегия позволяет увеличивать число параллельно работающих задач на 20 процентов в секунду. Она выбрана для того, чтобы повторная активизация выполнялась относительно плавно, с небольшим увеличением нагрузки в первую секунду после окончания интервала стабилизации. Если перерасход памяти возникнет во время активизации приостановленных процессов, будут выполнены следующие действия:

- Процесс активизации будет прерван
- Все процессы, которые должны быть активизированы, будут снова помечены для остановки

- Кроме того, для остановки будут выбраны дополнительные процессы в соответствии с описанными выше правилами

Выделение и освобождение блоков пространства подкачки

Операционная система поддерживает три способа выделения рабочей памяти.

Операционная система поддерживает три способа выделения рабочей памяти (*блоков пространства подкачки*):

- Динамическое выделение
- Статическое выделение
- Выделение с отсрочкой

Примечание: Блоки пространства подкачки освобождаются только при завершении процесса (а не нити), либо при вызове функции **disclaim()**. Эти блоки не освобождаются системным вызовом **free()**.

Динамическое выделение пространства подкачки

Алгоритм динамического выделения применяется многими программами путем выделения памяти под максимально возможный объем данных с последующим использованием требуемой части выделенной памяти. Неиспользуемые страницы виртуальной памяти не занимают страниц оперативной памяти и блоков пространства подкачки.

Использование этого метода сопряжено с некоторым риском. Если все работающие программы одновременно займут максимально возможную область памяти, пространство подкачки будет полностью израсходовано. Вполне вероятно, что после этого некоторые программы не смогут продолжать работу.

Статическое выделение пространства подкачки

Второй способ выделения блоков пространства подкачки предназначен для применения в системах, в которых описанная выше ситуация вероятна, либо там, где аварийное завершение работы программ недопустимо. В алгоритме статического выделения пространства подкачки необходимое количество блоков пространства подкачки предоставляется одновременно с выделением диапазона адресов виртуальной памяти, например, с помощью функции **malloc()**. Если блоков пространства подкачки недостаточно для выполнения запроса функции **malloc()**, то программе передается код ошибки. Для того чтобы включить алгоритм статического выделения пространства подкачки, необходимо указать следующую строку:

```
# export PSALLOC=early
```

После этого все запускаемые в среде программы будут работать в режиме статического выделения пространства подкачки. Для всех ранее запущенных программ режим выделения пространства подкачки изменен не будет.

Статическое выделение может служить средством анализа производительности, так как оно сопряжено с повышенными требованиями к объему пространства подкачки. При использовании алгоритма статического выделения требования к пространству подкачки возрастают во много раз. В то время как в обычных ситуациях рекомендуют создавать пространство подкачки объемом вдвое больше объема оперативной памяти, в системах с **PSALLOC=early** объем пространства подкачки должен быть как минимум вчетверо больше. Такая оценка весьма приблизительна. Вам потребуется оценить требования программ к виртуальной памяти и создать пространство подкачки исходя из этих требований. Например, при работе в режиме статического выделения пространства подкачки для сервера AIXwindows требуется пространство подкачки размером 250 МБ.

Если указано **PSALLOC=early**, пользователь должен задать обработчик сигнала SIGSEGV путем предварительного выделения и задания памяти в виде стека функцией **sigaltstack**. Даже если указано **PSALLOC=early**, программа может получить сигнал SIGSEGV при попытке расширения стека, если объем пространства подкачки недостаточен.

Выделение пространства подкачки с отсрочкой

По умолчанию применяется метод выделения блоков пространства подкачки третьей операционной системы. В стратегии отложенного выделения пространства подкачки (DPSA) выделение пространства подкачки откладывается до тех пор, пока не потребуется выгрузить страницу. Это приводит к тому, что излишний объем пространства подкачки не выделяется. Этот способ позволяет сэкономить значительный объем пространства подкачки.

В некоторых системах пространство подкачки может не потребоваться даже в том случае, если каждая страница, к которой обращались процессы, изменялась. Это возможно в системах с очень большим объемом оперативной памяти. Однако это может привести к перерасходу пространства подкачки в случае, если объем запрошенной виртуальной памяти будет превосходить объем физической памяти.

Для того чтобы выключить алгоритм DPSA и включить алгоритм динамического выделения пространства подкачки, вызовите команду:

```
# vmo -o defps=0
```

Для того чтобы включить алгоритм DPSA, вызовите команду:

```
# vmo -o defps=1
```

Алгоритм DPSA в целом повышает производительность системы, поскольку он не распределяет страницы памяти, при работе с которыми возникают сбои. За счет этого экономится место в пространствах подкачки.

Дополнительная информация приведена в разделах “Выделение пространства подкачки” на стр. 157 и “Расположение и размер пространств подкачки” на стр. 99.

Аспекты производительности, связанные с управлением дисковым вводом-выводом

Управление дисковой памятью операционной системой осуществляется посредством иерархии структур.

Каждому диску, или физическому тому (PV), присвоено имя, например, `/dev/hdisk0`. Все используемые физические тома образуют группы томов (VG). Все физические тома из группы томов входят в состав физических разделов (PP) равного размера (по умолчанию - 4 МБ в группе, состоящей из томов размером меньше 4 ГБ, и 8 МБ на дисках большего размера).

Для удобства выделения памяти каждый физический том разделен на пять областей. Дополнительная информация приведена в разделе “Расположение данных на физическом томе” на стр. 202. Число физических разделов в разных областях различно и зависит от общего объема диска.

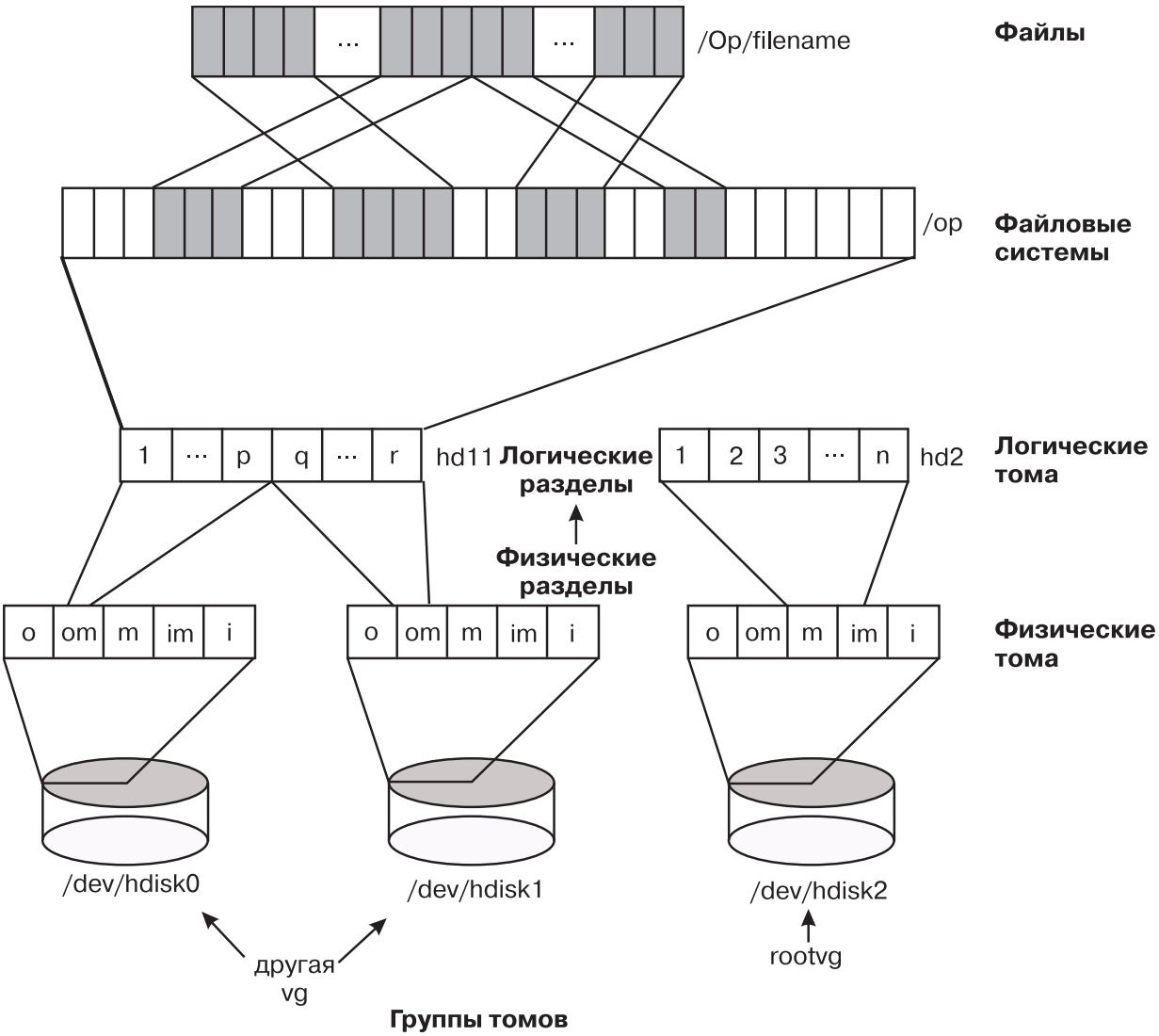


Рисунок 10. Размещение данных на жестком диске без зеркальной копии. На рисунке приведен пример физического тома, на основе которого созданы логические разделы. В таких разделах физического тома, называемых логическими томами, хранятся файловые системы, представляющие собой структуру файлов и каталогов. Файлы хранятся в блоках дорожек физического носителя. Обычно для файла выделяются несмежные блоки. В результате того, что одни файлы стираются, а другие файлы записываются в освободившиеся блоки, расположенные на разных дорожках носителя, происходит фрагментация диска.

В каждой группе томов определен один или несколько логических томов (LV). Логический том состоит из одного или нескольких логических разделов. Каждому логическому разделу соответствует один или несколько физических разделов. Если для данного логического тома применяется зеркальная защита, то для размещения зеркальных копий каждого логического раздела выделяются дополнительные физические разделы. Хотя нумерация логических разделов последовательная, номера входящих в их состав физических разделов могут быть произвольными.

Логические тома могут служить для различных целей, например, для подкачки, однако для тех из них, которые содержат обычные системные или пользовательские данные, либо программы, есть одна журнализированная файловая система (JFS или JFS2). Каждая JFS состоит из нескольких блоков пространства подкачки размером 4096 байт. Если нужно записать данные в файл, для этого файла отводится несколько дополнительных блоков. Эти блоки не обязательно должны следовать друг за другом. Кроме того, они могут быть несмежными с другими блоками файла.

На предыдущем рисунке показана одна из плохих ситуаций (хотя и не худшая из возможных), которые могут возникнуть в нереорганизованной файловой системе. Файл `/op/filename` физически размещен в большом числе блоков, не смежных друг с другом. При последовательном чтении файла необходимо выполнить огромное количество операций, каждая из которых занимает много времени.

Хотя логически файл в операционной системе представлен как непрерывная строка последовательных байтов, физический файл может мало соответствовать такому представлению. Фрагментация может возникнуть вследствие многократного увеличения логического тома, либо интенсивных операций по выделению, освобождению и повторному выделению памяти в файловой системе. Говорят, что файловая система фрагментирована, если свободное дисковое пространство в ней состоит из множества небольших блоков, не позволяющих записать файл в смежные блоки.

При обращении к файлу в файловой системе с высокой степенью фрагментации может потребоваться большое число операций поиска, а время ответа сильно возрастает (основную часть времени ответа составляют задержки, необходимые для выполнения поиска). Например, для последовательного считывания файла, разбитого на множество разбросанных маленьких фрагментов, потребуется больше операций поиска, чем для последовательного чтения файла, состоящего из одного или нескольких больших непрерывных фрагментов. При прямом доступе к файлу, разбросанному по всему диску, потребуется больше операций поиска, чем в случае, когда блоки файла расположены близко друг к другу.

Влияние фрагментации файла на производительность операций ввода-вывода уменьшается при загрузке файла в буфер оперативной памяти. При открытии файла в операционной системе ему выделяется постоянный сегмент данных в виртуальной памяти. Этот сегмент представляет буфер виртуальной памяти для данного файла; блоки файла непосредственно преобразуются в страницы сегмента. VMM управляет страницами сегмента, считывая блоки файла по мере надобности (т.е. по мере доступа к ним) в страницы сегмента. В некоторых условиях VMM записывает страницу памяти обратно в соответствующий блок файла на диске; однако в общем случае VMM после обращения к странице некоторое время хранит ее в памяти. Следовательно, страницы, к которым часто обращаются, долгое время находятся в памяти, и для доступа к соответствующим блокам не нужно обращаться к физическому диску.

Периодически пользователю или системному администратору следует выполнять реорганизацию распределения файлов по логическим томам и логических томов по физическим томам для снижения уровня фрагментации и более равномерного распределения нагрузки на все устройства ввода-вывода. В разделе “Производительность логических томов и дискового ввода-вывода” на стр. 178 приведена дополнительная информация по обнаружению и исправлению ошибок, связанных с размещением файлов и фрагментацией.

Поддержка фиксации памяти

В AIX предусмотрена возможность запрета вытеснения в пространство подкачки определенных страниц памяти. Это называется фиксацией памяти.

Зафиксированные области памяти не могут вытесняться в пространство подкачки. Фиксация допускается как в системном, так и в пользовательских пространствах. После фиксации область памяти находится по одному и тому же адресу оперативной памяти и не перемещается до тех пор, пока фиксация не будет снята. Некоторые части ядра всегда зафиксированы, а прочие фиксируются только на время обращения к ним, а в остальное время могут вытесняться в пространство подкачки.

Смысл фиксации областей памяти заключается в том, что к фиксированной области можно обращаться напрямую, не задействуя средства определения реальных адресов динамических страниц. Побочный эффект заключается в том, что фиксация слишком большого числа областей памяти повышает частоту перемещения незафиксированных областей, и в больших масштабах это может существенно снизить производительность системы.

Переменную **vmo maxpin%** можно использовать для настройки количества закрепляемой памяти. Переменную **vmo maxpin%** задает максимальную долю физической памяти, которая может быть закреплена (в процентах).

Примечание: Ядру необходимо закрепить данные ядра, поэтому не рекомендуется уменьшать значение переменной **maxpin%**, т.к. это может привести к функциональным проблемам.

Приложения пользователя могут закреплять память с помощью различных механизмов. Приложения могут использовать функции **plock()**, **mlock()** и **mlockall()** для закрепления памяти.

Приложение может явно закрепить общую память с помощью опции **SHM_LOCK** функции **shmctl()** или флага **SHM_PIN** для **shmget()**.

Многопроцессорность

В силу технологических причин, быстродействие процессора не может превышать некоторого ограничения. Если требования к производительности системы превышают возможности одного процессора, то одним из вариантов решения проблемы является установка нескольких процессоров.

Эффективность такого решения зависит не только от квалификации проектировщиков системы, но и от того, насколько многопроцессорный режим подходит для выполнения конкретных задач. Можно привести следующую аналогию: для повышения эффективности работы можно увеличить число операторов, отвечающих на телефонные звонки, однако увеличить число водителей автомобиля нельзя.

Ниже указаны предпосылки для перехода от однопроцессорной системы к многопроцессорной с целью повышения производительности:

- Скорость обработки зависит от производительности процессора, причем однопроцессорная система не справляется с нагрузкой.
- Рабочая схема содержит несколько элементов, целиком зависящих от скорости процессора, таких как обработка транзакций или сложные вычисления, которые могут выполняться одновременно.
- Процессор в однопроцессорной системе нельзя заменить на более мощный.

Хотя однонитевые приложения правильно работают в многопроцессорной среде, скорость их выполнения может измениться в любую сторону. Переход к многопроцессорной среде может повысить общую производительность системы и ускорить выполнение сложных многонитевых приложений, однако вряд ли сократит время выполнения отдельных однонитевых команд.

Для достижения максимальной эффективности работы многопроцессорной системы необходимо представлять схему выполнения программ в многопроцессорной среде с точки зрения операционной системы и аппаратных компонентов.

Принцип работы и архитектура симметричних многопроцессорных систем

Как и для любой сложной системы, для обеспечения работы многопроцессорной системы с высокой производительностью нужно создать специальную конфигурацию.

По сравнению с однопроцессорными компьютерами, в такой системе применяется более сложное программное и аппаратное обеспечение, которое предоставляет дополнительные возможности, но требует приложить дополнительные усилия для обеспечения согласованности. Поскольку можно предложить различные варианты конфигурации многопроцессорной системы, каждый из которых имеет свои преимущества, существует множество разнообразных архитектур многопроцессорных систем.

Типы многопроцессорной обработки

Существует несколько типов многопроцессорных систем (MP).

MP без общих ресурсов (простой кластер:

У процессоров нет общих ресурсов (то есть у каждого из них есть своя оперативная память, кэши и диски), однако они соединены между собой. Такой тип многопроцессорных систем также называется *простым кластером*.

Каждый процессор представляет собой автономный компьютер, на котором запущена своя копия операционной системы. Если они соединены с помощью локальной сети, то говорят, что процессоры слабо связаны между собой. Если они соединены при помощи коммутатора, то говорят, что процессоры сильно связаны. Процессоры взаимодействуют друг с другом путем передачи сообщений.

К достоинствам таких систем относят возможность масштабируемости и высокий коэффициент готовности. Недостатком таких систем является то, что в них необходимо использовать особую модель программирования (с передачей сообщений).

MP с общими дисками:

К достоинствам архитектуры с общими дисками относится возможность использовать часть стандартной модели программирования (данные на диске доступны и согласованы, данные в оперативной памяти - нет) и высокий коэффициент готовности, которого намного проще достигнуть, чем в системах с общей оперативной памятью. Недостатком является ограниченная масштабируемость из-за сложностей в организации физического и логического доступа к общим данным.

У процессоров есть собственная оперативная память и кэш. Все процессоры работают параллельно и используют общие диски. Каждый процессор работает со своей копией операционной системы и слабо связан с остальными процессорами (процессоры соединены с помощью локальной сети). Процессоры взаимодействуют друг с другом путем передачи сообщений.

Кластер с общей оперативной памятью:

У всех процессоров в кластере с общей оперативной памятью есть собственные ресурсы (оперативная память, диски и устройства ввода-вывода), и на каждом из них работает своя копия операционной системы.

Процессоры сильно связаны между собой (соединены через коммутатор). Процессоры взаимодействуют друг с другом через общую память.

MP с общей оперативной памятью:

Все процессоры расположены в одном корпусе и сильно связаны между собой с помощью высокоскоростной шины или коммутатора. Процессоры совместно работают с глобальной оперативной памятью, дисками и устройствами ввода-вывода. На всех процессорах работает одна и та же копия операционной системы, поддерживающая такую архитектуру (операционная система с поддержкой нескольких нитей).

У SMP есть несколько достоинств:

- Эта архитектура позволяет повысить общую производительность системы без существенных затрат.
- Поскольку на процессорах работает общая операционная система, с точки зрения пользователей SMP все они работают с одной системой, что упрощает процесс администрирования.
- Для решения задачи могут использоваться несколько процессоров (параллельное программирование).
- Операционная система равномерно распределяет нагрузку на процессоры.
- В SMP может применяться та же модель программирования, что и в однопроцессорной системе.
- В SMP можно создать общий сегмент данных.
- Все процессоры могут работать с любыми данными, при этом согласованность данных обеспечивается аппаратными средствами.

- Для обеспечения взаимодействия процессоров не нужно применять библиотеки передачи сообщений, так как для обмена данными применяется глобальная общая память.
- Для наращивания мощности системы достаточно добавить в нее еще несколько процессоров. Однако вы должны реально оценивать, насколько может возрасти производительность системы SMP при добавлении процессоров.
- Для SMP активно создаются новые приложения и служебные программы. Большинство приложений, разработанных для однопроцессорной системы, можно перенести без изменений в систему SMP.

При работе с системами SMP следует учитывать следующие ограничения:

- Существуют определенные пределы для наращивания системы SMP, связанные с необходимостью синхронизации данных в кэше, механизмом блокировки, общими объектами и т.д.
- Для работы с многопроцессорной системой требуются сотрудники с определенными навыками, в том числе умеющие создавать программы с несколькими нитями и драйверы устройств для SMP.

Распараллеливание приложений

В многопроцессорной системе параллельную обработку приложений можно обеспечить двумя способами.

- Стандартный способ заключается в том, чтобы разбить приложение на несколько процессов. Эти процессы общаются друг с другом с помощью среды межпроцессного взаимодействия (IPC), например, каналов, семафоров или общей памяти. Процессы должны уметь приостанавливать выполнение до возникновения события, например, до получения сообщения от другого процесса. Кроме того, необходимо средство для управления доступом к общим объектам, например, блокировки.
- Другой способ заключается в использовании нитей POSIX UNIX. При работе с нитями возникают те же проблемы с согласованием их работы, что и при работе с процессами, а для решения этих проблем используются те же средства. В этом случае несколько нитей одного процесса могут выполняться одновременно на разных процессорах. Всю работу по согласованию их работы и сериализации доступа к общим данным должен выполнять разработчик приложения.

При выборе способа распараллеливания приложения необходимо сравнить достоинства нитей и процессов. Нити могут выполняться быстрее процессов и, кроме того, при работе с нитями легче организовать доступ к общей памяти. С другой стороны, программу, разбитую на несколько процессов, проще распределить по нескольким компьютерам или кластерам. Если в ходе работы приложения создаются и удаляются новые экземпляры, то реализация с помощью нитей будет работать быстрее, так как для порождения процесса потребуется меньше ресурсов. Во всех остальных случаях варианты реализации с помощью процессов и нитей будут сравнимы по своей эффективности.

Сериализация данных

Любой объект, доступный для чтения или записи нескольким нитям, может быть изменен во время работы программы.

Обычно это правило справедливо и для многопрограммных сред, однако в многопроцессорных системах оно имеет особое значение по двум причинам:

- В многопроцессорной системе с поддержкой нитей проще создавать приложения, нити которых работают с общими данными.
- В такой системе проблема сериализации доступа к данным не может решаться путем запрета прерываний.

Примечание: Во избежание ошибок, программы, работающие с общими данными, должны обращаться к ним последовательно, а не параллельно. Перед записью данных необходимо защитить эти данные на время выполнения операции от изменения другими программами (в том числе, другим экземпляром той же программы). Операции чтения обычно могут выполняться параллельно.

Главный механизм, позволяющий избежать конфликтов между программами - *блокировка*. Блокировкой называется разрешение на доступ к одному или нескольким элементам данных. Запросы на блокировку и разблокирование являются элементарными, то есть во время их выполнения управление не передается другим нитям. Перед обращением к общему объекту программа должна установить блокировку этого

объекта. Если другая программа, либо другая нить той же программы, уже установила блокировку объекта, то запрашивающей программе придется дождаться освобождения объекта.

Время простоя нити увеличивается не только за счет ожидания блокировки, но и за счет сериализации доступа к данным. Во время простоя нити другие нити могут заменить в кэше данные первой нити на свои. В результате, когда исходная нить сможет установить блокировку и получит управление, ее производительность снизится за счет увеличения длительности операций ввода-вывода.

Многие общие объекты данных расположены в ядре операционной системы, поэтому в ядре выполняется внутренняя сериализация. Это означает, что задержка из-за сериализации доступа может возникнуть даже в том случае, когда приложение не работает с общими данными, но применяет службы ядра, которые работают с общими данными ядра.

Блокировки

Блокировки позволяют резервировать и освобождать память операционной системы.

Дополнительная информация приведена в разделе Работа с блокировками.

Типы блокировки:

При создании функций блокировки операционной системы AIX, применяемых при работе в многопроцессорных системах, за основу была взята модель блокировок OSF/1 версии 1.1.

Но поскольку в операционной системе применяется подкачка страниц и вытеснение нитей, к модели блокировок OSF/1 1.1 были добавлены некоторые новые функции. Простые и сложные блокировки являются вытесняющими. Кроме того, в ожидании освобождения простой блокировки нить может быть приостановлена, если владелец этой блокировки в настоящий момент не выполняется. Простые блокировки преобразуются в ждущие и в том случае, когда процессор ожидает освобождения простой блокировки свыше определенного периода времени (этот период задается одной из системных переменных).

Простая блокировка:

В операционной системе версии 4 под простой блокировкой понимается блокировка с ожиданием из-за занятости. Если какая-либо нить ожидает снятия простой блокировки, то, при выполнении определенных условий, эта нить будет приостановлена. Таким образом, нить не будет ждать снятия блокировки бесконечно.

Простые блокировки являются вытесняющими. Это означает, что нить ядра, установившая простую блокировку, может быть вытеснена другой нитью ядра с более высоким приоритетом. В многопроцессорных системах простые блокировки применяются для управления доступом нитей к критическим секциям. Они должны использоваться вместе со средствами управления прерываниями для сериализации выполнения на текущем процессоре и взаимодействия между процессорами.

В однопроцессорной системе достаточно использовать только прерывания. Блокировки в ней не нужны. Простые блокировки предназначены для защиты критических секций типа нить-нить и нить-прерывание. Нить будет ожидать установления простой блокировки до тех пор, пока ресурс не освободится. Различают два состояния простой блокировки: заблокировано и разблокировано.

Сложные блокировки:

В операционной системе AIX сложные блокировки представляют собой блокировки чтения-записи, защищающие критические секции, к которым могут обращаться несколько нитей. Такие блокировки являются вытесняющими.

Сложные блокировки представляют собой блокировку с ожиданием из-за занятости. Это означает, что нить, ожидающая возможности установить блокировку, в некоторых случаях приостанавливается. По умолчанию

такие блокировки не являются рекурсивными. Для работы с рекурсивными блокировками нужно вызвать службу ядра **lock_set_recursive()**. У таких блокировок есть три состояния: исключительная блокировка на запись, общая блокировка с разрешением чтения и отсутствие блокировки.

Дискретность блокировок:

Программист, работающий с многопроцессорной системой, должен решить, сколько отдельных блокировок потребуется установить для доступа к общим данным. Если изменяется несколько элементов большого набора данных, то можно установить блокировку на уровне всего набора, но в этом случае возрастает вероятность конфликтов с другими нитями. Широкое использование блокировок накладывает ограничение на производительность системы.

Можно установить отдельную блокировку для каждого элемента. В этом случае вероятность конфликта с другими нитями достаточно мала. Однако обработка каждого запроса на блокировку и разблокирование занимает процессорное время. Кроме того, при большом числе блокировок может возникнуть тупик. Простейший пример тупика показан на следующем рисунке. Здесь нить 1 установила блокировку A и ожидает снятия блокировки B, которую установила нить 2. В то же время, нить 2 ожидает освобождения блокировки A. Выполнение ни одной из программ не достигнет вызова **unlock()**, поэтому они будут находиться в тупике бесконечно. Для того чтобы избежать тупиков, обычно задаются правила, определяющие последовательность захвата объектов программами, работающими с этими данными.

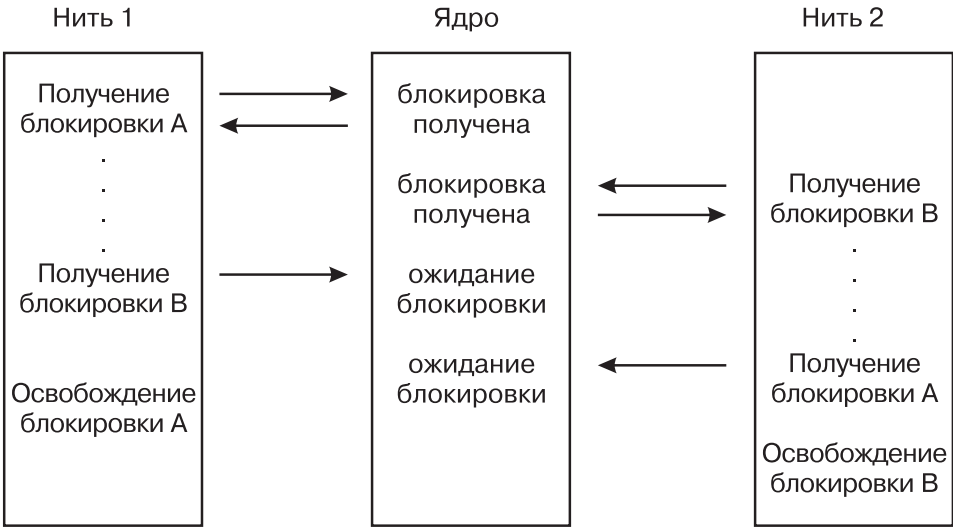


Рисунок 11. Тупик. На следующем рисунке показан пример тупика. Нить 1 установила блокировку A и ожидает освобождения блокировки B, которую установила нить 2. В то же время, нить 2 ожидает освобождения блокировки A. Выполнение ни одной из программ не достигнет вызова unlock, поэтому они будут находиться в тупике вечно.

Согласно теории организации очередей, чем меньше простаивает ресурс, тем больше среднее время ожидания этого ресурса. Эта зависимость не линейная. Если время блокировки увеличится вдвое, то среднее время ожидания снятия этой блокировки увеличится больше чем вдвое.

Самый эффективный способ сократить время ожидания снятия блокировки заключается в снижении уровня объектов, для которых можно установить блокировку. Ниже приведены некоторые рекомендации:

- Уменьшите количество запросов на получение блокировок.
- Блокируйте только тот фрагмент программы, который обращается к общим данным, а не весь компонент (при этом уменьшится время удержания блокировки).
- Блокируйте только отдельные элементы или структуры данных, а не всю функцию.

- Всегда связывайте блокировку только с отдельными элементами или структурами данных, но не со всей функцией.
- В случае больших структур данных блокируйте отдельные элементы этой структуры, а не всю структуру целиком.
- До снятия блокировки не выполняйте операции синхронного ввода-вывода или другие операции, устанавливающие блокировки.
- Если в компоненте требуется несколько раз обратиться к одним и тем же данным, то постарайтесь сгруппировать их вместе так, чтобы блокировку можно было устанавливать только один раз.
- Не отправляйте уведомления о снятии блокировки дважды. Если вы установили блокировку, изменили данные, а теперь хотите уведомить кого-либо о завершении операции, то освободите блокировку перед отправкой уведомления.
- Если вам нужно установить одновременно две блокировки, то блокировку наиболее занятого ресурса нужно запрашивать последней.

С другой стороны, блокировка на очень низком уровне приведет к увеличению запросов на установление и снятие блокировок, что приведет к увеличению сложности программ. Необходимо выбрать золотую середину между слишком низким и слишком высоким уровнями блокировки. Оптимальный уровень блокировок можно найти только путем проб и ошибок. Это одна из самых сложных задач, возникающих при работе с многопроцессорной системой. Приведенная ниже диаграмма показывает взаимосвязь между производительностью системы и уровнем блокировок.

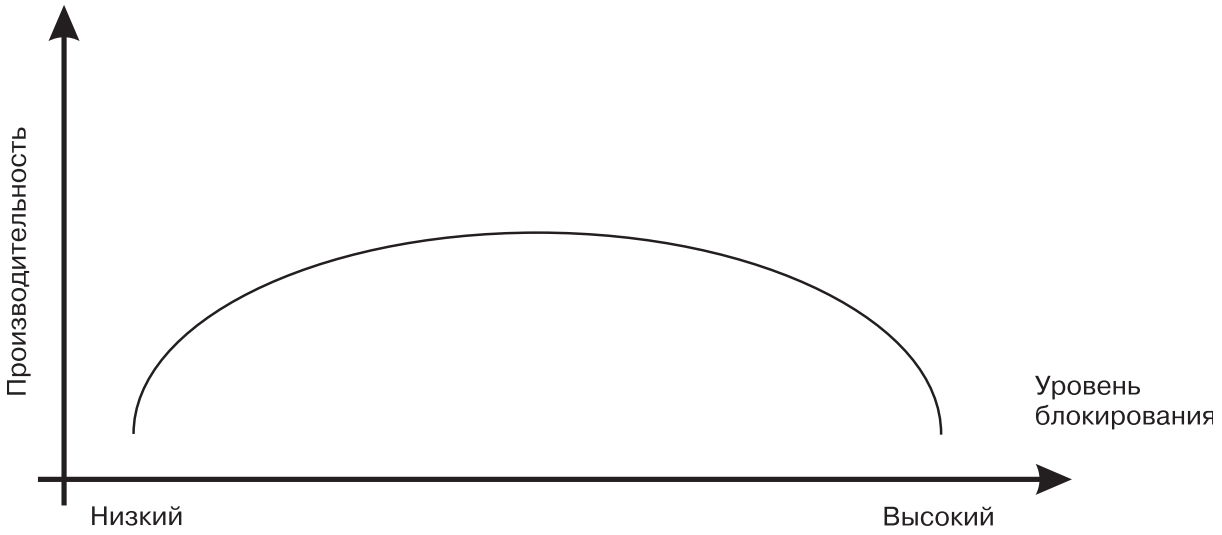


Рисунок 12. Взаимосвязь между производительностью системы и уровнем блокировок. На этом рисунке показана двумерная диаграмма. По вертикальной оси (y) откладывается значение производительности. Горизонтальная ось (x) представляет уровень блокировок. Взаимосвязь между уровнем блокировок и производительностью представлена в виде выпуклой кривой. С увеличением уровня блокировок производительность вначале растет до максимума, а затем начинает медленно снижаться. Из этого следует, что для достижения максимальной производительности нужно выбрать оптимальный уровень блокировок.

Дополнительная нагрузка за счет блокировок:

Запросы и ожидание блокировок, а также разблокирование объектов создают дополнительную нагрузку на процессор.

- Программа, написанная для многопроцессорной системы, всегда запрашивает блокировки объектов, даже если она выполняется в однопроцессорной системе или не конкурирует с другими программами за доступ к данным.

- Если объект, запрошенный нитью, уже заблокирован, то эта нить будет некоторое время работать вхолостую, либо простаивать, в то время как будет выполняться другая нить. В итоге будет расходоваться процессорное время.
- Широкое использование блокировок накладывает ограничение на производительность системы. Например, если программа 20 процентов времени выполнения удерживает взаимно исключающую блокировку, то, независимо от числа процессоров, в системе может одновременно выполняться не более пяти экземпляров этой программы. В действительности, пять экземпляров программы вряд ли удастся синхронизировать таким образом, чтобы они не конкурировали за блокировку (см. “Масштабируемость и производительность в многопроцессорных системах” на стр. 66).

Ожидание блокировки:

Если нить должна захватить блокировку, которая в настоящий момент занята, то она будет ожидать освобождения блокировки.

Есть два варианта ожидания снятия блокировки:

- Блокировки с ожиданием из-за занятости устанавливаются на небольшое время. Они позволяют ожидающей нити сохранить управление, проверяя через небольшие периоды времени бит блокировки в цикле до тех пор, пока блокировка не освободится. Циклическая проверка приводит к возрастанию времени использования CPU (времени работы процессора в служебном режиме при установлении блокировок ядра или расширений ядра).
- Блокировки с выгрузкой из-за занятости устанавливаются на продолжительное время. Если блокировка занята, то нить приостанавливается до тех пор, пока она не освободится. После этого она снова помещается в очередь выполнения. Остановка нити приводит к увеличению времени простоя.

Ожидание снятия блокировки всегда влияет на производительность системы. В случае блокировок с ожиданием из-за занятости процессор не освобождается, несмотря на то, что он не выполняет полезной работы (это приводит к снижению производительности). В случае блокировок с выгрузкой из-за занятости переключение контекста и повторная постановка в очередь выполнения создает дополнительную нагрузку на систему. Помимо этого, возрастает число промахов в кэше.

Разработчики операционной системы могут выбрать один из двух типов блокировок: взаимно исключающие простые блокировки, позволяющие процессам ожидать снятия блокировки в активном состоянии, либо сложные блокировки чтения-записи, выгружающие процессы, которые ожидают снятия блокировки.

Существуют определенные соглашения относительно использования блокировок. Ни аппаратных, ни программных способов активизации или проверки состояния блокировок не существует. Хотя за счет блокировок операционная система AIX версии 4 обеспечивает согласованную работу многопроцессорной системы, разработчики должны самостоятельно определять и реализовывать стратегию блокировки для защиты собственных глобальных данных.

Согласованность кэша

При создании многопроцессорных систем большое внимание уделяется проблеме согласованности кэша. Эта проблема была решена за счет некоторого снижения производительности системы.

Для того чтобы понять, почему это так, необходимо разобраться в сути проблемы:

Если у каждого процессора есть кэш, содержащий некоторые данные из памяти, то сразу несколько кэшей могут содержать одинаковую строку данных. Кроме того, эта строка может содержать сразу несколько блокируемых элементов данных. Если две нити последовательно изменят эти элементы данных, то в кэшах могут оказаться разные версии этой строки памяти. Это означает, что целостность системы будет нарушена, так как в ней будут хранится две версии содержимого определенной области памяти.

Обычно для решения проблемы согласованности кэша при изменении строки памяти все ее копии, кроме одной, объявляются неверными. Это выполняется специальной контрольной логической схемой на

аппаратном уровне без участия программного обеспечения. Однако в результате при следующем обращении к неверной строке кэша произойдет промах и возникнет связанная с ним задержка.

Контрольная логическая схема применяется для решения задач, связанных с поддержанием согласованности кэша. При изменении записи кэша эта логическая схема процессора рассылает оповещающее сообщение по шине. Кроме того, эта логическая схема отслеживает подобные сообщения, отправленные по шине другими процессорами.

Когда процессор обнаруживает, что другой процессор изменил запись с адресом, который есть в его собственном кэше, контрольная логическая схема объявляет эту запись неверной. Такая процедура называется *перекрестным аннулированием*. В ходе перекрестного аннулирования процессору сообщается, что запись в кэше не верна, поэтому ее нужно взять из памяти или другого кэша. Поскольку перекрестное аннулирование увеличивает число промахов в кэше, а протокол контрольной логической схемы увеличивает нагрузку на шину, то поддержание кэша в согласованном состоянии снижает производительность и масштабируемость SMP.

Процессор, связывание

Принадлежностью процесса процессору называется свойство, гарантирующее, что работа нити всегда возобновляется на том же процессоре, на котором она выполнялась раньше. Ценность этого свойства прямо пропорциональна объему данных нити, хранящихся в кэше, и обратно пропорциональна времени простоя нити. Планировщик операционной системы AIX версии 4 по умолчанию применяет это свойство для всех процессоров.

Когда выполнение нити прерывается, а позднее вновь возобновляется на том же процессоре, то в кэше все еще могут находится данные этой нити. Если работа нити возобновляется на другом процессоре, то произойдет ряд промахов кэша, пока данные этой нити не будут записаны в кэш процессора из оперативной памяти или кэша другого процессора. С другой стороны, еще большая задержка может возникнуть в том случае, если нити придется ждать, пока освободится исходный процессор.

Говорят, что нить связана с определенным процессором, если нить полностью принадлежит процессору. Связывание означает, что нить будет выполняться только заданным процессором, независимо от загруженности других процессоров системы. Команда **bindprocessor** и процедура **bindprocessor()** позволяют связать все нити процесса с определенным процессором (см. раздел “Команда bindprocessor” на стр. 78). Связь с процессором, установленная явным образом, наследуется дочерними процессами, созданными с помощью системных вызовов **fork()** и **exec()**.

Связывание может эффективно применяться в программах с большой нагрузкой на процессор и небольшим числом прерываний. Однако иногда связывание может послужить причиной снижения производительности, так как нити придется ждать освобождения процессора после выполнения операции ввода-вывода. Если нить была заблокирована на время выполнения операции ввода-вывода, то маловероятно, что информация о ней сохранилась в кэше процессора. В этом случае продолжение работы нити на другом процессоре было бы эффективнее.

Конкуренция за память и шину

В однопроцессорной системе задержки, связанные с конкуренцией за внутренние ресурсы (устройства памяти, шины ввода-вывода и памяти), обычно невелики. В многопроцессорных системах эти задержки становятся ощутимыми, особенно в том случае, если применяется алгоритм поддержания согласованности кэша, который увеличивает число обращений к оперативной памяти.

Производительность SMP

Что следует принимать во внимание для получения максимальной эффективности симметричных многопроцессорных систем.

Параллельность рабочей схемы

Системы SMP характеризуются параллельной рабочей схемой. Это означает, что для повышения производительности системы нужно решить проблему оптимального распределения нагрузки на *n* процессоров.

Если в четырехпроцессорной системе в каждый момент времени загружен только один процессор, то ее производительность будет не выше, чем у однопроцессорной системы. Более того, производительность будет даже ниже, поскольку потребуется выполнять дополнительные действия для предотвращения конфликтов между процессорами.

Требование параллельности рабочей схемы противоречит требованию сериализации доступа к данным. Чем шире в системном программном обеспечении или рабочей схеме применяется сериализация доступа к данным, тем меньше степень параллельности рабочей схемы.

Степень параллельности рабочей схемы также уменьшается, если в системе активно используется присвоение процессов процессорам. Это свойство повышает эффективность работы кэша, что позволяет ускорить выполнение программы. При этом степень параллельности рабочей схемы уменьшается (если есть простаивающие нити), однако улучшается время отклика.

В понятие параллельности рабочей схемы входит понятие *параллельности процесса*. Оно характеризует наличие готовых к запуску нитей процесса в любой момент времени.

Производительность

Производительность системы SMP главным образом зависит от следующих факторов:

- Уровня параллельности рабочей схемы. При низком уровне параллельности рабочей схемы число готовых к запуску нитей иногда превышает число процессоров, а иногда процессоры простаивают.
- Числа конфликтов доступа к данным.
- Активности присвоения процессов процессорам.

Время ответа

Время ответа программы в системе SMP зависит от нескольких факторов.

- Уровня параллельности ее процесса. Время отклика программ, у которых всегда несколько нитей готовы к выполнению, обычно лучше. Если программа выполняется в одной нити, то ее время отклика в лучшем случае будет сравнимо с временем отклика в аналогичной однопроцессорной системе.
- Числа конфликтов доступа к данным между экземплярами одной программы или различными программами.
- Активности присвоения процессов процессорам. Если работа программы каждый раз возобновляется на новом процессоре, не содержащем в кэше ее данных, то программа может работать даже медленнее, чем в аналогичной однопроцессорной системе.

Рабочая нагрузка SMP

Повышение производительности, связанное с увеличением числа процессоров, зависит от некоторых свойств рабочей схемы. Эти свойства обсуждаются в данном разделе.

Ниже перечислены критерии работоспособности программы в среде SMP:

SMP-защищенность
: Программа удовлетворяет этому критерию, если она не выполняет никаких действий, которые могли бы вызвать неполадки в работе системы SMP (например, параллельный доступ к общим данным). Это минимальное требование, которое предъявляется к программам, работающим в среде SMP.

SMP-эффективность
: Программа удовлетворяет этому критерию, если она не выполняет никаких действий, которые могут вызвать сбой в работе системы SMP или понизить ее производительность. SMP-эффективная

программа всегда является SMP-защищенной. Для выполнения этого требования в программу обычно требуется внести дополнительные изменения, переписав самые неэффективные участки кода.

Использование возможностей SMP
Программа удовлетворяет этому критерию, если в ней используются специальные средства, повышающие производительность системы SMP, например поддержка нескольких нитей. Программа, использующая возможности SMP, обычно является SMP-эффективной и SMP-защищенной.

Рабочая схема мультипроцессорной системы

Когда мы наблюдаем работу многозадачной операционной системы на мощном компьютере, нам кажется, что она выполняет очень много задач одновременно.

На самом деле, даже в больших рабочих схемах в каждый момент времени поддерживается лишь небольшое число нитей. Это справедливо даже для однопроцессорной системы, в которой нет проблемы сериализации доступа к данным. Если число готовых к выполнению нитей в системе меньше числа процессоров, то часть времени один или несколько процессоров будут простаивать.

Число готовых к выполнению нитей равно общему числу нитей в системе

- Минус число нитей, ожидающих выполнения ввода-вывода,
- Минус число нитей, ожидающих освобождения общего ресурса,
- Минус число нитей, ожидающих результатов выполнения другой нити,
- Минус число нитей, ожидающих выполнения собственного запроса.

Рабочая схема подходит для многопроцессорной обработки, если число готовых к выполнению нитей в каждый момент времени равно числу процессоров. Обратите внимание, что речь идет не просто о среднем числе нитей. Так, если половину времени число готовых нитей равно 0, а половину времени - оно в два раза больше числа процессоров, то среднее число нитей будет равно числу процессоров. Тем не менее, все процессоры системы будет простаивать половину времени.

Существует два способа адаптации рабочей схемы к многопроцессорной среде:

- Обнаружение и устранение критических элементов ("узких мест") в системе, которые задерживают выполнение нитей
- Увеличение общего числа нитей в системе

Эти два способа взаимозависимы. Если в системе есть один основной критический элемент, то увеличение общего числа проходящих через него нитей в данной рабочей схеме только повысит число ожидающих нитей. А если в настоящее время в системе таких узких мест нет, то вследствие увеличения числа нитей они могут появиться.

Масштабируемость и производительность в многопроцессорных системах

В системах SMP с реальной рабочей схемой увеличение числа процессоров не всегда приводит к значительному росту производительности.

При масштабируемости системы возникают следующие проблемы:

- При увеличении числа процессоров возрастает конкуренция за шину или коммутатор
- Возрастает конкуренция за оперативную память (поскольку вся оперативная память совместно используется всеми процессорами)
- С отдалением процессора от оперативной памяти возрастает стоимость промахов в кэше
- Для поддержания согласованности кэша некоторые записи объявляются недействительными, и их приходится читать из другого кэша
- Возрастает число промахов в кэше из-за более частого переключения нитей (поскольку требуется распределить больше нитей по процессорам)
- Возрастает объем ресурсов, затрачиваемый на синхронизацию

- Из-за увеличения объема операционной системы и структур данных приложений возрастает число промахов в кэше
- Возрастает длительность блокировок операционной системы и приложений
- Возрастает длительность ожидания снятия блокировки

Все эти факторы влияют на *масштабируемость* рабочей схемы. Масштабируемость указывает, насколько повысится производительность системы с данной рабочей схемой при добавлении новых процессоров. Она равна отношению производительности многопроцессорной системы к производительности однопроцессорной системы с той же рабочей схемой. Например, если однопроцессорная система с данной рабочей схемой обрабатывает 20 запросов в минуту, а четырехпроцессорная - 58 запросов в минуту, то коэффициент масштабируемости равен 2,9. Это довольно высокий коэффициент масштабируемости. У рабочей схемы, состоящей из вычислительных программ с большим периодом выполнения, содержащих небольшое число операций ввода-вывода и не работающих с общими данными, коэффициент масштабируемости на четырехпроцессорной системе будет равен 3,2 - 3,9. У большинства реальных рабочих схем коэффициент намного ниже. Поскольку оценку масштабируемости получить довольно сложно, она всегда должна базироваться на информации о реальных рабочих схемах.

Трудности, возникающие при масштабируемости, проиллюстрированы на приведенном ниже рисунке. Рабочая схема системы представляет собой совокупность последовательностей команд. При выполнении каждой команды треть времени тратится на обычную обработку, треть - на ожидание ввода-вывода, а еще треть - на работу с блокировками. В однопроцессорной системе, независимо от наличия блокировки, в каждый момент времени выполняется только одна команда. За рассматриваемый промежуток времени (время автономного выполнения команды * 5) однопроцессорная система обрабатывает 7,67 команд.

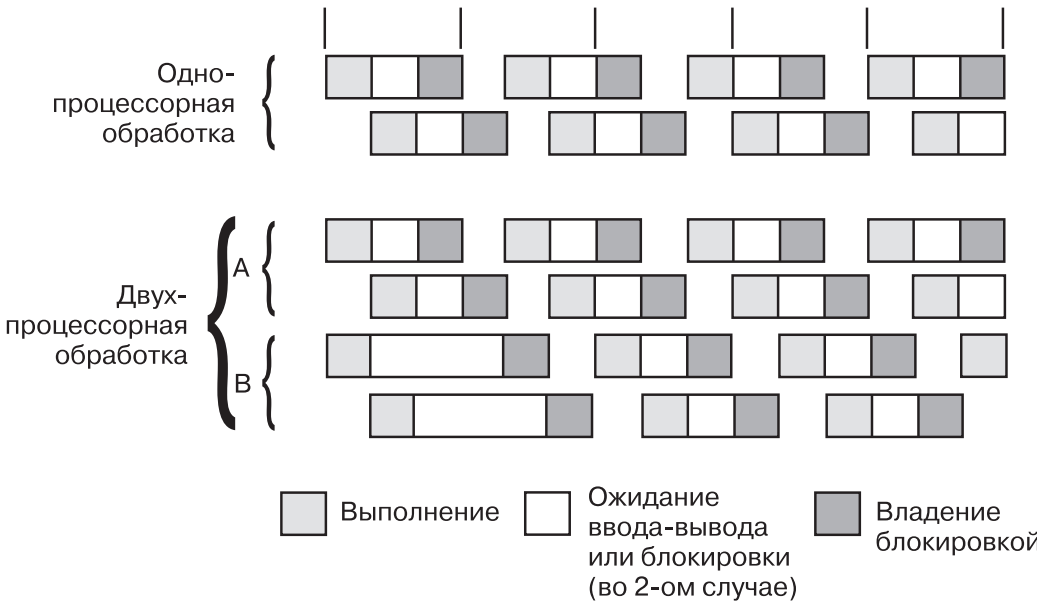


Рисунок 13. Масштабируемость многопроцессорной системы. На этом рисунке проиллюстрированы трудности, возникающие при масштабировании системы. Рабочая схема системы представляет собой совокупность последовательностей команд. При выполнении каждой команды треть времени тратится на обычную обработку, треть - на ожидание ввода-вывода, а еще треть - на работу с блокировками. В однопроцессорной системе, независимо от наличия блокировки, в каждый момент времени выполняется только одна команда. За рассматриваемый интервал времени однопроцессорная система обрабатывает 7,67 команд. За тот же интервал времени многопроцессорная система обрабатывает 14 команд, если коэффициент масштабируемости равен 1,83.

Если программа будет выполняться на двух процессорах, скорость выполнения увеличиться вдвое, так как блокировка всего одна. Для простоты будем считать, что блокировка влияет только на работу процессора B. За показанный промежуток времени многопроцессорная система обрабатывает 14 команд. Таким образом,

коэффициент масштабируемости равен 1,83. В качестве примера мы взяли систему с двумя процессорами, так как в данном случае дополнительные процессоры не увеличили бы производительность (потому что блокировка занята всегда). В четырехпроцессорной системе коэффициент масштабируемости остался бы прежним или даже снизился.

Как правило, в действительности программы не столь симметричны, как это показано на рисунке. Кроме того, в приведенном примере была учтена конкуренция только за один ресурс: блокировку. Если принять во внимание необходимость поддержания кэша в согласованном состоянии и обеспечение связи процессов с процессорами, то коэффициент масштабируемости окажется еще ниже.

Этот пример показывает, что далеко не всегда добавление процессоров в систему может ускорить работу. Для этого необходимо минимизировать число конфликтов между нитями.

Успех масштабируемости в значительной мере зависит от рабочей схемы. Некоторые опубликованные результаты тестирования говорят о том, что высокая степень масштабируемости легко достижима. Однако такие тесты обычно проводятся на примере небольших программ, которые интенсивно используют ресурсы процессора и почти не обращаются к ядру. Эти результаты отражают лишь теоретические, но не реальные возможности.

Кроме того, следует заметить, что в общем случае система SMP с одним процессором будет работать на 5-15 процентов медленнее, чем аналогичная обычная однопроцессорная система, в которой будет работать версия операционной системы для однопроцессорной системы.

Время ответа в многопроцессорных системах

Скорость выполнения программы в многопроцессорной системе увеличится лишь в том случае, если ее процесс можно разбить на несколько нитей.

Существует несколько способов параллельного выполнения компонентов программы:

- Явно указывать вызовы функций `libpthreads.a` (или **fork()** в старых программах) для создания нескольких параллельных нитей.
- Обработать программу с помощью распараллеливающего компилятора или препроцессора. Препроцессор находит последовательности команд, которые можно выполнять одновременно, и создает несколько нитей.
- Использовать программное обеспечение, которое заранее создано с учетом поддержки нескольких нитей.

Если не предпринять перечисленных действий, то в многопроцессорной системе программа не будет работать быстрее. Более того, из-за дополнительных ресурсов, которые будут расходоваться на обработку блокировок и переключение на другой процессор, выполнение программы может даже замедлиться.

Максимальная производительность, которой можно достичь путем применения всех трех способов, вычисляется по закону Амдаля:

- Если x - это доля времени выполнения программы в однопроцессорной системе (t), в течение которого обработка должна быть линейной, то в системе с n процессорами ускорение составит:

$$\text{ускорение} = \frac{\text{время для одного процессора}}{\text{время одно-процессорной обработки} + \text{время много-процессорной обработки}} = \frac{t}{xt + \frac{(1-x)t}{n}} = \frac{1}{x + \frac{1-x}{n}}$$

$$\lim_{n \to \infty} \text{ускорение} = \frac{1}{x}$$

Рисунок 14. Закон Амдаля. В законе Амдаля говорится, что ускорение составляет время выполнения программы в многопроцессорной системе, поделенное на время линейной обработки плюс время выполнения в многопроцессорной системе (1/(x+x/n)). Предельное ускорение при n, стремящемся к бесконечности, составляет единицу, деленную на x.

Например, если 50 процентов вычислений программы распараллелить нельзя, то время отклика можно улучшить не более, чем в 2 раза (в 4-х процессорной системе коэффициент составит 1,6).

Планирование выполнения нитей SMP

Поддержка нитей упрощает задачу создания приложений, использующих возможности SMP.

Система управления выполнением программ оперирует двумя понятиями:

- *Процесс* - это набор физических ресурсов, необходимых для выполнения программы (например, сегмент памяти и файлы с данными).
- *Нить* - это набор параметров, характеризующих текущее состояние запущенного экземпляра программы, например, содержимое регистра адреса команды и регистров общего назначения. Нить выполняется в контексте своего процесса и использует его ресурсы. В составе одного процесса могут выполняться несколько нитей, совместно использующих его ресурсы.

Размножать процессы для создания нескольких потоков управления неудобно и дорого, так как каждый процесс обладает собственным набором ресурсов памяти и требует значительных системных ресурсов для своей настройки. Более эффективно создавать несколько нитей в рамках одного процесса.

Существует два уровня поддержки нитей:

- Поддержка `libpthreads.a` в среде прикладных программ
- Поддержка нитей ядра

Хотя нити представляют собой удобный и эффективный способ организации многозадачности, они обладают ограничениями с точки зрения масштабируемости. Поскольку нити совместно используют ресурсы процесса, необходимо контролировать состояние этих ресурсов, обеспечивать возможность их блокирования и сериализации.

Управление старыми рабочими схемами

Поддержка нитей никак не повлияет на работу уже существующих программ.

Операционная система точно так же, как и раньше, создает процессы приложений, перенесенных из более младших выпусков. Такие процессы создаются в виде единичных нитей (начальных нитей), которые конкурируют за ресурсы CPU с нитями других процессов.

Благодаря новым алгоритмам планирования, а также атрибутам главной нити, устанавливаемым по умолчанию, такой способ выполнения старых программ не приводит к снижению производительности системы.

Для изменения приоритета предназначены команды **nice** и **renice**, а также системные вызовы **setpri()** и **setpriority()** (как и в предыдущих версиях). Планировщик передает управление нити максимум на один квант

времени (как правило, 10 мсек), а затем передает управление следующей готовой к выполнению нити с таким же (или более высоким) приоритетом. Дополнительную информацию можно найти в разделе “Управление конкуренцией за микропроцессор” на стр. 122.

Переменные планирования

Планирование нитей зависит от нескольких переменных.

Некоторые из них относятся исключительно к поддержке нитей, другие относятся к планированию процессов:

По величине приоритета
Значение приоритета - это основной фактор, от которого зависит вероятность захвата нитью очередного кванта времени процессора.

Положение в очереди выполнения планировщика
Положение нитей, готовых к выполнению, в очереди планировщика зависит от числа нитей с более высоким приоритетом.

Стратегия планирования
От этого атрибута зависит, что происходит с активной нитью по истечении выделенного ей кванта времени.

Область действия
Этот атрибут указывает, конкурирует ли нить со всеми нитями в системе, или только с нитями своего процесса. Выполнение нитей pthread, действующих в рамках процесса, планируется библиотекой, а выполнение нитей с глобальной областью действия планируется ядром. Для управления нитями "pthread", конкурирующими только с нитями своего процесса, планировщик библиотеки использует пул нитей ядра. Нити pthread, выполняющие операции ввода-вывода, рекомендуется создавать с глобальной областью действия. Локальную область действия следует использовать при наличии частых синхронизаций внутри одного процесса. Понятие области действия применяется по отношению к нитям `libpthreads.a`.

Принадлежность процесса процессору
Степень связанности процессов с процессорами сказывается на производительности системы.

На первый взгляд, учесть все эти параметры достаточно сложно. Однако на самом деле, существует три основных способа планирования процессов:

По умолчанию
Процесс состоит из одной нити, приоритет которой изменяется в зависимости от нагрузки на CPU, а стратегия планирования равна SCHED_OTHER.

Управление на уровне процесса
Процесс может состоять из одной или нескольких нитей, но их стратегия планирования остается прежней (по умолчанию - SCHED_OTHER). Эта стратегия позволяет использовать существующие функции для изменения значения nice и задания фиксированных приоритетов. Все эти функции применяются ко всем нитям в составе процесса. При выполнении функции **setpri()** для всех нитей процесса устанавливается стратегия планирования SCHED_RR.

Управление на уровне нитей
Процесс может состоять из одной или нескольких нитей. Для этих нитей можно установить стратегию планирования SCHED_RR или SCHED_FIFO*n*. Приоритет нитей фиксирован. Для его изменения используются функции уровня нитей.

Подробное описание стратегий планирования приведено в разделе “Стратегии планирования для нитей” на стр. 42.

Настройка нити

Пользовательские нити представляют собой независимые потоки управления в процессе.

Если пользовательской нити потребуется обратиться к службе ядра (например, при помощи системного вызова), то ее запросы будут выполняться соответствующей нитью ядра. Поддержка пользовательских нитей предусмотрена во многих пакетах программ, одним из которых является общая библиотека нитей pthread (`libpthreads.a`). В реализации libpthreads пользовательские нити расположены над виртуальными процессорами (VP), которые, в свою очередь, расположены над нитями ядра. Пользовательский процесс с несколькими нитями может быть создан по одной из следующих моделей:

Модель нитей 1:1

В модели 1:1 каждой пользовательской нити соответствует ровно одна нить ядра. Эта модель применяется по умолчанию во всех версиях AIX. В этой модели каждая пользовательская нить связана с VP, и ей соответствует ровно одна нить ядра. При этом VP не обязательно связан с реальным CPU (если связывание не установлено явным образом). Нить, связанная с VP, имеет глобальную область действия, так как она планируется вместе с остальными пользовательскими нитями планировщиком ядра.

Модель нитей M:N

Модель M:N была реализована в операционной системе AIX 4.3.1, а сейчас применяется по умолчанию. В этой модели несколько пользовательских нитей могут работать с общим виртуальным процессором или пулом процессоров. Виртуальный процессор играет роль виртуального CPU, который может выполнять пользовательские программы и системные вызовы. Нить, не связанная с VP, имеет локальную область действия, совпадающую с процессом, так как она не планируется вместе с остальными нитями планировщиком ядра. Библиотека pthread распределяет пользовательские нити по виртуальным процессорам, а затем ядро планирует выполнение соответствующих нитей ядра. Как и в операционной системе AIX 4.3.2, по умолчанию одна нить ядра связана с восьмью пользовательскими нитями. Это соотношение можно изменить в самом приложении, либо во всей системе при помощи переменной среды.

Администратор может выбрать ту модель нитей, которая лучше всего подходит для выполнения приложения. Проведенные тесты показали, что некоторые приложения выполняются быстрее в рамках модели 1:1. В AIX 6.1 по умолчанию снова применяется модель 1:1 вместо модели M:N. Во всех версиях AIX, установив для процесса значение переменной среды **AIXTHREAD_SCOPE=S**, вы можете задать модель нитей 1:1, а затем сравнить скорость работы программы в обеих моделях.

Если вы заметите, что приложение создает и удаляет нити, это могут быть нити ядра, *появившиеся* из-за высокого соотношения пользовательских нитей и нитей ядра (8:1). В сочетании с высокой нагрузкой на планировщик библиотеки это может отрицательно повлиять на производительность. С другой стороны, если в системе работает несколько тысяч пользовательских нитей, то затраты на планирование их выполнения в пользовательском пространстве в рамках библиотеки могут быть меньше затрат, которые потребуются на управление тысячами нитей ядра. Если при работе с нитями pthread вы заметите резкое снижение производительности, то в первую очередь нужно попытаться изменить область действия нитей. Во многих случаях нити с глобальной областью действия работают быстрее.

Если приложение выполняется в системе SMP, и пользовательская нить не может захватить взаимную блокировку, то она повторит попытку захвата не более 40 раз. Довольно часто взаимные блокировки освобождаются через небольшой промежуток времени, поэтому иногда имеет смысл увеличить количество попыток. Если с увеличением количества CPU производительность уменьшается, то, скорее всего, это связано с блокировками. В этом случае рекомендуется увеличить время активного ожидания блокировки, задав переменную среды **SPINLOOPTIME=***n*, где *n* - число попыток захвата. В некоторых случаях это значение может исчисляться тысячами - все зависит от мощности и числа процессоров. После того, как число попыток захвата достигнет максимума, нить может быть приостановлена до освобождения взаимной блокировки, либо она может вызвать функцию **yield()** и уступить процессор другой нити, оставшись в очереди выполнения. По умолчанию нить приостанавливается, однако если вы зададите переменную среды **YIELDLOOPTIME**, то нить будет приостановлена только после того, как она указанное число раз уступит управление другой нити. Каждый раз, когда нити будет возвращаться управление, она может попробовать захватить взаимную блокировку.

Некоторые пользовательские процессы с несколькими нитями, активно использующие подсистему malloc, будут работать быстрее, если перед их запуском экспортировать переменную среды **MALLOCMULTIHEAP=1**. В частности, это рекомендуется делать для программ C++ с несколькими нитями, так как они используют подсистему malloc при вызове конструкторов и деструкторов. Повышение производительности будет лучше всего заметно при запуске пользовательских приложений с несколькими нитями в системе SMP, в особенности если для нитей будет установлена глобальная область действия (соотношение M:N равно 1:1). Однако в некоторых случаях повышения производительности можно добиться и в другой среде, в частности, в однопроцессорной системе.

Переменные среды нити

В среде `libpthreads.a` предусмотрен набор переменных для настройки производительности приложения.

При возможности используйте сценарий оболочки команды-клиента для вызова двоичных исполняемых программ. Сценарий оболочки должен указывать новое значение, которое должно переопределить системные значения по умолчанию для переменных среды, описанных в следующих разделах.

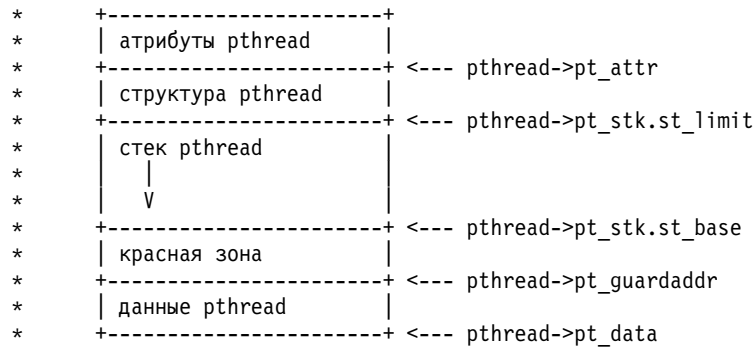
AIXTHREAD_COND_DEBUG

Переменная *AIXTHREAD_COND_DEBUG* поддерживает список переменных условия для использования отладчиком. Если в программе используется много активных переменных условия, то их интенсивное создание и уничтожение может потребовать слишком больших затрат ресурсов. Для отключения списка переменных условия присвойте этой переменной среды значение **OFF**. Если вы оставите эту переменную включенной, то вам будет проще выполнять отладку приложений с несколькими нитями, однако при этом возрастет нагрузка на систему.

AIXTHREAD_ENRUSG

Переменная *AIXTHREAD_ENRUSG* разрешает или запрещает использование набора ресурсов pthread. Если этому параметру присвоено значение ON, то всем нитям pthread процесса будет разрешено применять набор ресурсов, однако при этом возрастет нагрузка на систему.

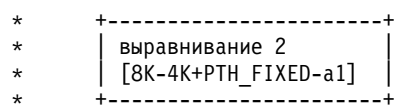
AIXTHREAD_GUARDPAGES=n



Красной зоной на этом рисунке называется *Guardpage*.

Атрибуты pthread attr, структуры pthread и ctx представляют часть PTH_FIXED памяти, выделенной для pthread.

Приблизительные размеры областей (в байтах) для 32-разрядных процессов показаны на диаграмме в квадратных скобках. Для 64-разрядных процессов компоненты PTH_FIXED будут несколько больше, а объем ключевых данных равен 8 КБ.



```
*      | ctx pthread [368]     |
*      +-----------------------+<--- pthread->pt_attr
*      | атриб. pthread [112]  |
*      +-----------------------+ <--- pthread->pt_attr
*      | структ. pthread [960] |
*      +-----------------------+ <--- pthread
*      | pthread stack         |         pthread->pt_stk.st_limit
*      |   |[96K+4K-PTH_FIXED] |
*      |   V                   |
*      +-----------------------+ <--- pthread->pt_stk.st_base
*      | красная зона [4K]     |
*      +-----------------------+ <--- pthread->pt_guardaddr
*      | ключевые данные       |
*      | pthread [4K]          |
*      +-----------------------+ <--- pthread->pt_data
*      | выравнивание 1 (a1)   |
*      | [<4K]                 |
*      +-----------------------+
```

Красной зоной на этом рисунке называется Guardpage.

n задает количество вспомогательных страниц, добавляемых в конец стека pthread. Это значение переопределяет значение атрибута, заданного при создании нити pthread. Если приложение использует собственный стек, то вспомогательные страницы не создаются. *n* должно быть больше либо равно нуля. Значение по умолчанию равно нулю.

Размер красной зоны в байтах определяется путем умножения *n* на системный размер страницы. Размер страницы задается системой.

AIXTHREAD_DISCLAIM_GUARDPAGES

Переменная *AIXTHREAD_DISCLAIM_GUARDPAGES* указывает, запрещено ли использование вспомогательных страниц при создании стека pthread. При **AIXTHREAD_DISCLAIM_GUARDPAGES=ON** вспомогательные страницы не используются. Если у стека pthread нет вспомогательных страниц, переменная **AIXTHREAD_DISCLAIM_GUARDPAGES** ни на что не влияет.

AIXTHREAD_MNRATIO

Параметр *AIXTHREAD_MNRATIO* определяет коэффициент масштабирования библиотеки. Этот коэффициент требуется при создании и завершении работы нитей pthread. Рекомендуется для приложений с очень большим количеством нитей. Рекомендуется всегда пытаться установить коэффициент 1:1, который во многих случаях обеспечивает максимальную производительность.

AIXTHREAD_MUTEX_DEBUG

Переменная *AIXTHREAD_MUTEX_DEBUG* поддерживает список переменных условия для использования отладчиком. Если в программе используется много активных взаимных блокировок, то их интенсивное создание и уничтожение может потребовать слишком больших затрат ресурсов. Указание в этой переменной значения ON упрощает отладку многонитевых приложений, но уменьшает производительность. Если в переменной указано значение OFF, список не создается.

AIXTHREAD_MUTEX_FAST

Если производительность программы заметно снижается из-за взаимных операций, то при назначении для данной переменной значения ON библиотека pthread будет использовать оптимизированный механизм взаимной блокировки, работающий только в частных взаимных блокировках процесса. Эти частные взаимные блокировки процесса должны быть инициализированы с помощью процедуры pthread_mutex_init и должны быть уничтожены с помощью процедуры pthread_mutex_destroy. Задав для переменной значение OFF библиотека pthread будет использовать механизмы взаимной блокировки по умолчанию.

AIXTHREAD_READ_GUARDPAGES

Переменная *AIXTHREAD_READ_GUARDPAGES* устанавливает права на чтение вспомогательных страниц, добавляемых в конец стека pthread. Дополнительная информация о вспомогательных страницах, создаваемых стеком pthread, приведена в разделе “AIXTHREAD_GUARDPAGES=n” на стр. 72.

AIXTHREAD_RWLOCK_DEBUG

Переменная *AIXTHREAD_RWLOCK_DEBUG* поддерживает список переменных условия для использования отладчиком. Если в программе используется много активных блокировок чтения-записи, то их интенсивное создание и уничтожение может потребовать слишком больших затрат. Для отключения списка переменных условия присвойте этой переменной среды значение `OFF`.

AIXTHREAD_SUSPENDIBLE={ON|OFF}

Задав для переменной *AIXTHREAD_SUSPENDIBLE* значение `ON` вы предотвратите тупики в приложениях, использующих следующие процедуры с процедурами **pthread_suspend_np** и **pthread_suspend_others_np**:

- pthread_getrusage_np
- pthread_cancel
- pthread_detach
- pthread_join
- pthread_getunique_np
- pthread_join_np
- pthread_setschedparam
- pthread_getschedparam
- pthread_kill

При использовании этой переменной слегка снижается производительность.

AIXTHREAD_SCOPE={S|P}

Параметр **S** задает область действия уровня *системы* (1:1), а параметр **P** задает область действия уровня *процесса* (M:N). Следует указать одну из этих опций; по умолчанию задано **S**.

Переменная среды *AIXTHREAD_SCOPE* влияет только на нити, при создании которых был установлен атрибут по умолчанию. Он устанавливается в том случае, если параметр *attr* функции `pthread_create()` равен нулю.

Если пользовательская нить создана с глобальной областью действия, она связывается с нитью ядра, и ее выполнение планируется ядром. Эта нить ядра не применяется никакими другими нитями.

Если пользовательская нить создана с областью действия уровня процесса, то с ней работает пользовательский планировщик. Это означает следующее:

- Отсутствует выделенная нить ядра.
- В пользовательском режиме включен спящий режим.
- Она помещена в пользовательскую очередь выполнения во время ожидания процессора.
- Согласно пользовательскому планировщику она предназначена для кванта времени.

Тесты показали, что некоторые приложения гораздо быстрее работают в рамках модели 1:1.

AIXTHREAD_SLPRATIO

Параметр *AIXTHREAD_SLPRATIO* задает число нитей ядра, которые следует зарезервировать для ожидающих нитей. В общем случае, для поддержки ожидающих нитей pthread требуется меньше нитей ядра, так как обычно ожидающие нити активируются по одной. Это позволяет сэкономить ресурсы ядра.

AIXTHREAD_STK=n

Переменная нити *AIXTHREAD_STK=n* задает количество байт, выделяемое каждой нити pthread (значение должно быть указано в десятичной системе счисления). Это значение может быть переопределено значением **pthread_attr_setstacksize** .

AIXTHREAD_AFFINITY={default|strict|first-touch}

Опция *AIXTHREAD_AFFINITY* определяет расположение структур, стеков и локальных хранилищ нитей pthread в системах с включенной опцией enhanced affinity.

- При использовании опции default данные будут распределяться между областями памяти, используемыми процессом, согласно параметрам системы
- При использовании опции strict данные будут размещаться вместе с pthread. При этом время создания pthread может увеличиться, поскольку данные будут перемещаться в другие области памяти, а общая производительность - улучшится.
- При использовании опции first touch данные также будут размещаться вместе с pthread, но переноса данных в памяти осуществляться не будет. Для этих данных нити необходимы внутренние страницы памяти (в том числе страницы пространства подкачки). При использовании данной опции достигается компромисс между временем запуска и общей производительностью.

AIXTHREAD_PREALLOC=n

Переменная *AIXTHREAD_PREALLOC* означает число байтов для предварительного выделения и освобождения в процессе создания нити. Некоторые многопоточные приложения могут воспользоваться этим , избегая вызова sbrk() из нескольких нитей одновременно.

n должно быть больше либо равно нуля. Значение по умолчанию равно нулю.

AIXTHREAD_HRT

Переменная *AIXTHREAD_HRT=true* разрешает тайм-ауты с высокой разрешающей способностью для приложений `pthreads`. Необходимо иметь права доступа `root` или возможности `CAP_NUMA_ATTACH` для включения тайм-аутов с высокой разрешающей способностью. Эта переменная среды игнорируется при отсутствии необходимых полномочий.

MALLOCBUCKETS

Наборы malloc - это расширение стандартного механизма распределения памяти. Оно увеличивает скорость выполнения операций malloc в случаях, когда приложения запрашивают большое количество областей памяти малого размера. Наборы malloc применяются только в строго определенных случаях, когда параметры запроса на выделение области памяти попадают в определенные границы. Все прочие запросы обрабатываются обычным образом.

По умолчанию наборы malloc не задействованы. Для применения наборов malloc нужно настроить и включить их до запуска процесса с помощью переменных среды *MALLOCTYPE* и *MALLOCBUCKETS*.

Дополнительные сведения о сегментах malloc приведены в разделе *Программирование: Разработка и отладка программ*.

MALLOCMULTIHEAP={considersize,heaps:n}

Для того чтобы в приложении с несколькими нитями все нити могли использовать вызовы **malloc()**, **free()** и **realloc()**, должно быть создано несколько куч. Если будет создана только одна куча, то вызовы **malloc()**, **free()** и **realloc()** всех нитей будут сериализованы (другими словами, только одна нить в каждый момент времени может вызывать функцию malloc, free или realloc). Это приведет к значительному снижению производительности при работе в многопроцессорной системе. Если будет создано несколько куч, то каждой нити может быть выделена своя куча. Если все кучи заняты, то нить, пытающаяся сделать вызов, будет ждать освобождения кучи. Сериализация по прежнему будет присутствовать, но вероятность ее возникновения и сила влияния будет существенно ниже.

Для поддержки такой схемы была изменена процедура блокировки, защищающая нити от возникновения конфликтов при доступе к ресурсам. У каждой кучи есть своя блокировка. Процедура блокировки выбирает кучу не случайным образом, а так, чтобы минимизировать вероятность последующей сериализации доступа. Если в переменной среды *MALLOCMULTIHEAP* задано значение **considersize**, то процедура блокировки будет выбирать ту кучу, в которой достаточно свободной памяти для обработки запроса, а не первую освободившуюся кучу.

В этой переменной можно указать несколько параметров через запятую, причем в любом порядке. Например:

```
MALLOCMULTIHEAP=considersize,heaps:3
```

Допустимы следующие параметры:

considersize
: Устанавливает другой алгоритм выбора кучи, который пытается минимизировать размер рабочего набора процесса. По умолчанию этот параметр не учитывается и используется самый быстрый алгоритм.

heaps:*n*
: С помощью этой опции можно изменить число куч. Допустимым диапазоном значений для *n* являются числа от 1 до 32. Если задать *n* равным значению, выходящему за рамки данного диапазона, например *n*<=0 или *n*>32), *n* будет равно 32.

По умолчанию значение *MALLOCMULTIHEAP* не задано (используется только первая куча). Если переменная среды *MALLOCMULTIHEAP* задана (например, `MALLOCMULTIHEAP=1`), то приложение с несколькими нитями сможет использовать все 32 кучи. Если вы укажете `MALLOCMULTIHEAP=heaps:`*n*, то приложение сможет использовать всего *n* куч вместо 32.

Дополнительная информация приведена в разделе Malloc Multiheap в книге *Программирование: Разработка и отладка программ*.

SPINLOOPTIME=n

Переменная *SPINLOOPTIME* определяет количество попыток системы получить занятый мьютекс или спин-блокировку без принятия дополнительных действий, таких как вызов ядра для освобождения процесса. Эта переменная предназначена для использования в системах SMP, где всегда предполагается, что блокировка, занятая другой активной нитью pthread, будет когда-нибудь освобождена. Этот параметр может устанавливаться только в среде libpthreads (для пользовательских нитей). Если обычно блокировки освобождаются через небольшой промежуток времени, вы можете увеличить время ожидания блокировки. *n* указывает число попыток захвата блокировки перед передачей управления другой нити. *n* должно быть больше либо равно нуля. Значение по умолчанию равно 40.

В функциях блокировки ядра для управления временем ожидания снятия блокировки применяется параметр ядра *MAXSPIN* (см. раздел “Применение команды schedo для изменения параметра MAXSPIN” на стр. 80).

YIELDLOOPTIME=n

Переменная *YIELDLOOPTIME* указывает, сколько раз система передает процессор другой нити, когда она пытается захватить занятую взаимную блокировку или блокировку с ожиданием из-за занятости, перед тем как приостановить нить, ожидающую блокировку. Процессор передается только в том случае, если есть другая готовая к выполнению нить с достаточным значением приоритета. Эту переменную рекомендуется применять в сложных приложениях, устанавливающих несколько блокировок. Переменная *n* задает, сколько раз требуется уступить процессор перед повторной попыткой блокировки. *n* должно быть больше либо равно нуля. Значение по умолчанию равно нулю.

Переменные локальной области действия

Ниже описаны переменные среды, влияющие на планирование нитей с локальной областью действия.

AIXTHREAD_MNRATIO=*p*:*k*

где *k* - число нитей ядра, которое будет использоваться для обработки *p* готовых к запуску нитей pthread. Эта переменная среды определяет коэффициент масштабирования библиотеки. Этот коэффициент требуется при создании и завершении работы нитей pthread. Эта переменная может быть задана только для нитей с локальной областью действия. В случае глобальной области действия эта переменная будет проигнорирована. Значение по умолчанию - 8:1.

AIXTHREAD_SLPRATIO=*k*:*p*

где *k* - число нитей ядра, которые следует зарезервировать для *p* нитей pthread, находящихся в состоянии ожидания. Это соотношение отражает число нитей ядра, которое должно поддерживаться для обработки ожидающих нитей. В целом для поддержки ожидающих нитей pthread требуется меньше нитей ядра, так как обычно эти нити активизируются последовательно. Это позволяет сэкономить ресурсы ядра. В качестве *p* и *k* можно указать любое натуральное число. Если *k*>*p*, то устанавливается соотношение 1:1. Значение по умолчанию - 1:12.

AIXTHREAD_MINKTHREADS=*n*

где *n* - минимальное число нитей ядра, которое следует использовать. Планировщик библиотеки не возвращает нити ядра, если их число меньше указанного. Нить ядра может быть возвращена практически в любой момент. Обычно нить ядра возвращается в результате завершения нити pthread. Значение по умолчанию равно 8.

Опции отладки нити

В библиотеке pthread предусмотрен ряд активных взаимных блокировок, событий и блокировок чтения-записи, которые могут использоваться отладчиком.

После инициализации блокировка добавляется в список. При этом предполагается, что ее еще нет в списке. Список компонуется динамически, поэтому в случае большого списка проверка отсутствия новой блокировки в списке потребовала бы значительного объема ресурсов. Ситуация осложняется тем, что список защищен с помощью блокировки (dbx__mutexes), которую необходимо было бы захватывать на время поиска в списке. В этом случае все остальные вызовы функции **pthread_mutex_init()** должны были бы ожидать окончания поиска.

Если перечисленные ниже переменные среды равны OFF (это значение применяется по умолчанию), то соответствующий список отладки создаваться не будет. Это означает, что команда **dbx** (или любой другой отладчик, использующий библиотеку функций отладки pthread) не будет показывать ни одного объекта в списке.

- **AIXTHREAD_MUTEX_DEBUG**
- **AIXTHREAD_COND_DEBUG**
- **AIXTHREAD_RWLOCK_DEBUG**

Для того чтобы присвоить переменной значение ON, выполните следующую команду:

```
# export переменная=ON
```

Инструменты SMP

Все средства по настройке производительности, предусмотренные в операционной системе, поддерживают работу в среде SMP.

Некоторые из них позволяют получить статистическую информацию об использовании отдельных процессоров. Все остальные средства настройки производительности подсчитывают только средние значения параметров, характеризующие интенсивность использования всех процессоров в целом.

В этом разделе описаны те средства, которые поддерживаются только в системах SMP. Информация о других средствах настройки производительности приведена в остальных разделах.

Команда bindprocessor

Команда **bindprocessor** позволяет связать нити процесса с определенным процессором и удалить такую связь.

Эту команду может вызывать только владелец процесса или пользователь с правами доступа root.

Примечание: Команда **bindprocessor** предназначена для использования в многопроцессорных системах. Хотя ее не запрещается вызывать в однопроцессорной системе, эффект от выполнения этой команды будет нулевой.

Для того чтобы узнать, какие процессоры доступны в системе, вызовите следующую команду:

```
# bindprocessor -q
Доступны следующие процессоры:  0 1 2 3
```

В выводе указаны логические номера доступных процессоров, которые и используются в команде **bindprocessor**, как показано ниже.

Для того чтобы связать процесс с идентификатором 14596 с процессором 1, вызовите следующую команду:

```
# bindprocessor 14596 1
```

В случае успешного выполнения команда не выдает никакого сообщения. Для того чтобы убедиться в том, что процессор был связан с процессором, либо такая связь была удалена, вызовите команду **ps -mo THREAD**, как описано в разделе “Применение команды ps” на стр. 115:

```
# ps -mo THREAD
Польз. PID  PPID    TID ST  CP PRI SC    WCHAN        F     TT BND Команда
root  3292  7130      - A    1  60  1        -   240001  pts/0   - -ksh
   -     -     -  14309 S    1  60  1        -      400      -   - -
root 14596  3292      - A   73 100  1        -   200001  pts/0   1 /tmp/cpubound
   -     -     -  15629 R   73 100  1        -        0      -   1 -
root 15606  3292      - A   74 101  1        -   200001  pts/0   - /tmp/cpubound
   -     -     -  16895 R   74 101  1        -        0      -   - -
root 16634  3292      - A   73 100  1        -   200001  pts/0   - /tmp/cpubound
   -     -     -  15107 R   73 100  1        -        0      -   - -
root 18048  3292      - A   14  67  1        -   200001  pts/0   - ps -mo THREAD
   -     -     -  17801 R   14  67  1        -        0      -   - -
```

В столбце *BND* указывается номер процессора, с которым связан процесс. Отсутствие значения (-) означает, что процесс не связан ни с одним процессором.

Для того чтобы удалить связь процесса 14596 с процессором, вызовите следующую команду:

```
# bindprocessor -u 14596
# ps -mo THREAD
Польз. PID  PPID    TID ST  CP PRI SC    WCHAN        F     TT BND Команда
root  3292  7130      - A    2  61  1        -   240001  pts/0   - -ksh
   -     -     -  14309 S    2  61  1        -      400      -   - -
root 14596  3292      - A  120 124  1        -   200001  pts/0   - /tmp/cpubound
   -     -     -  15629 R  120 124  1        -        0      -   - -
```

```
root 15606  3292      - A  120 124  1        -   200001  pts/0   - /tmp/cpubound
   -     -     -  16895 R  120 124  1        -        0      -   - -
root 16634  3292      - A  120 124  0        -   200001  pts/0   - /tmp/cpubound
   -     -     -  15107 R  120 124  0        -        0      -   - -
root 18052  3292      - A   12  66  1        -   200001  pts/0   - ps -mo THREAD
   -     -     -  17805 R   12  66  1        -        0      -   - -
```

Если команда **bindprocessor** выполняется для процесса, то все нити этого процесса связываются с указанным процессором, при этом их связь с предыдущим процессором удаляется (если она была). При удалении связи процесса с процессором удаляется связь для всех его нитей. Команду **bindprocessor** нельзя выполнить для отдельных нитей.

Однако внутри программы можно вызвать функцию **bindprocessor()**, позволяющую связать с процессором отдельные нити. После вызова функции **bindprocessor()** в некотором фрагменте программы, указанные нити останутся связанными с заданным процессором до конца выполнения программы. Удалить эту связь нельзя. Если для процесса этой программы будет вызвана команда **bindprocessor**, то все нити этого процесса будут связаны с новым процессором, включая те, которые раньше были связаны с каким-то другим процессором. При удалении связи процесса с процессором будут удалены связи всех нитей этого процесса с процессорами.

Процесс можно связать с процессором только после запуска программы. Другими словами, можно связать только существующий процесс. Если процесс не существует, то появится следующее сообщение об ошибке:

```
# bindprocessor 7359 1
1730-002: Процесс 7359 не существует
```

Если же не существует указанный процессор, то появится следующее сообщение об ошибке:

```
# bindprocessor 7358 4
1730-001: Процессор 4 недоступен
```

Примечание: Не выполняйте команду **bindprocessor** для ожидающих процессов **kproc**.

Замечания по связыванию:

При работе со связыванием процессов необходимо учитывать определенные факторы.

Связывание может эффективно применяться в программах с большой нагрузкой на процессор и небольшим числом прерываний. В случае обычных программ связывание может привести к снижению производительности, так как нити придется ждать освобождения процессора после выполнения операции ввода-вывода. Если нить была заблокирована на время выполнения операции ввода-вывода, то маловероятно, что информация о ней сохранилась в кэше процессора. В этом случае продолжение работы нити на другом процессоре было бы эффективнее.

Связывание не гарантирует, что на указанном процессоре не будут выполняться другие процессы. В этом смысле связывание отличается от создания разделов. С помощью `rset` или исключительных `rsets` набор логических процессоров можно выделить для выполнения конкретной рабочей нагрузки. Следовательно, процесс с более высоким приоритетом может занять процессор, с которым вы связали исходный процесс. В этом случае этот процесс не сможет занять другой процессор, если он свободен. Таким образом, связывание процесса не приводит к повышению производительности. Более высокой производительности можно добиться путем увеличения приоритета связанного процесса.

В сильно загруженной системе связывание процесса может привести к снижению производительности, так как процесс не сможет занять любой простаивающий процессор, а будет вынужден ждать, пока не освободиться тот процессор, с которым он связан.

Если в процессе есть несколько нитей, то все его нити связываются с одним процессором. Следовательно, этот процесс не сможет воспользоваться возможностями многопроцессорной обработки, и его производительность не увеличится.

Примечание: Будьте внимательны, связывая процессы с процессорами, так как эта операция нарушает естественное распределение нагрузки, которое поддерживается в AIX, и может привести к резкому снижению производительности системы. Если рабочая схема системы изменилась по сравнению с начальный отслеживаемым связыванием, то производительность системы может понизится. После выполнения команды **bindprocessor** периодически отслеживайте работу системы, так как связывание процесса может в любой момент отрицательно сказаться на ее производительности, например, при изменении среды выполнения.

Применение команды schedo для изменения параметра MAXSPIN

Если нить запрашивает блокировку, которая занята нитью, выполняющейся на другом процессоре, то она будет ожидать освобождения блокировки в соответствии с ограничением, указанным в настраиваемом параметре *MAXSPIN*.

В системах SMP значение параметра *MAXSPIN* по умолчанию равно 0x4000 (16384), а в системах UP значение этого параметра равно 1. Если в системе увеличилось время простоя процессора или время ожидания ввода-вывода, то это может быть вызвано тем, что нити стали чаще останавливаться на время ожидания блокировки. Если это привело к снижению производительности системы, то увеличьте значение параметра *MAXSPIN* или присвойте ему значение -1, эквивалентное 0xFFFFFFFF.

Число попыток захвата, при достижении которого нить будет остановлена до освобождения блокировки, можно задать с помощью опции **maxspin** команды **schedo**. Для того чтобы сократить нагрузку на CPU, связанную с большим количеством попыток захвата, уменьшите значение *MAXSPIN*, как показано ниже:

```
# schedo -o maxspin=8192
```

При этом может возрасти количество переключений контекста. Если это приведет к снижению скорости работы приложений, увеличьте значение *MAXSPIN*.

Для изменения значения параметра необходимы права доступа root.

Планирование и реализация требуемой производительности

Программа, которая работает недостаточно эффективно, зачастую оказывается бесполезной. Каждая программа должна удовлетворять требованиям пользователей, число которых может быть достаточно велико. Если производительность программы не устраивает большинство пользователей, то она не будет применяться. Программа, которая не применяется, бесполезна.

Сказанное в равной степени относится к пакетам лицензионного программного обеспечения и пользовательским приложениям, хотя большинство разработчиков пакетов программ понимают, к каким последствиям может привести низкая производительность, и принимают все возможные меры по ее повышению. К сожалению, они не в состоянии предвидеть все возможные варианты конфигурации рабочей среды, в которой будут применяться их программы. В конечном счете, ответственность за обеспечение нужного уровня производительности несут те, кто планирует, выбирает (или создает) и устанавливает пакеты программного обеспечения.

В этой главе описана последовательность действий, с помощью которых программист или системный администратор может убедиться, что только что созданная или приобретенная программа работает достаточно эффективно. (В дальнейшем словом *программист* будут обозначаться также системные администраторы и другие пользователи, отвечающие за успешную работу программы.)

Для того чтобы обеспечить необходимый уровень производительности программы, определите этот уровень перед началом работы над проектом и в дальнейшем постоянно помните о мерах и ресурсах, требуемых для его достижения. Хотя это звучит тривиально, об этом иногда забывают. В некоторых случаях при разработке проекта основное внимание уделяется проектированию, написанию, отладке и, возможно, документированию; меры по повышению производительности принимаются в последнюю очередь, если на это остается время.

Единственный способ, позволяющий гарантированно обеспечить не только правильную, но и достаточно быструю работу программы, заключается в учете требований к производительности во время планирования и разработки программного обеспечения. Роль предварительного планирования становится еще важнее в случае установки уже существующего программного обеспечения, поскольку при установке диапазон возможных мер по повышению производительности значительно уже, чем при разработке.

Подробное описание процесса может показаться излишним в случае небольшой программы; однако помните, что задача этим не исчерпывается. Недостаточно, чтобы новая программа обладала требуемой производительностью. Помимо этого необходимо, чтобы при ее установке в системе не снижалась производительность других программ.

Определение программного компонента

У разработчиков, администраторов и пользователей программы есть представление о том, как новая программа (была ли она создана или приобретена) будет применяться.

Ниже приведены некоторые соображения:

- Кто будет работать с этой программой
- В каких ситуациях будет запускаться программа
- Как часто будет возникать потребность в запуске программы (сколько раз в час, день, месяц, год)
- Должны ли одновременно запускаться другие программы
- В каких системах будет запускаться программа
- Какой объем данных будет обрабатываться, и где они будут храниться
- Будут ли данные, создаваемые программой или для программы, использоваться вне ее

Пока эти соображения не станут частью процесса разработки, они будут выглядеть достаточно неопределенными, причем представления об эффективной работе программы у ее создателей и будущих пользователей почти всегда сильно различаются. Даже в простейшем случае, когда программист - одновременно и пользователь, если не сформулировать требования к программе, то невозможно будет провести четкое сравнение параметров созданной программы с ее предполагаемыми характеристиками. Более того, невозможно сформулировать требования к эффективности, если нет полного представления о том, какую работу будет выполнять программа.

Документация по требованиям к производительности

При определении требований к производительности необходимо обосновать каждое из них. Это обязательная часть процесса планирования распределения ресурсов. Пользователям свойственно формулировать свои требования исходя из логики работы программы, представления о которой могут не совпадать с представлениями программиста.

Минимальный список требований к производительности должен содержать следующие пункты:

- Максимальное время ответа для каждого типа выполняемых в системе операций. Это время не должно превышаться большую часть времени (необходимо также определить, что вы понимаете под *большей частью времени*). Под временем ответа понимается время между завершением передачи запроса пользователя системе и получением ответа, достаточного для продолжения выполнения задачи. Это время ожидания может быть разным для различных пользователей. Не следует путать это значение с промежутком времени от входа в программу до первого оператора записи.

  Если пользователь возразит, что его интересует только результат выполнения операции, а не время выполнения, то узнайте, устроит ли его, если время выполнения станет в десять раз больше текущей оценки. Если пользователь ответит утвердительно, то этот вопрос можно считать закрытым. В противном случае вам придется продолжить дискуссию и узнать все требования пользователя:
- Максимальное время ответа в остальное время. Если время ответа будет слишком большим, то пользователь может решить, что система зависла. Необходимо четко определить, что значит *в остальное*

время; например, одна минута в рабочее время или один процент всех операций. В определенное время дня к времени ответа могут предъявляться особо жесткие требования.

- Сколько раз за определенный промежуток времени и когда именно будет запускаться программа. Очень важно точно определить этот параметр. Предположим, что некоторая программа должна запускаться дважды в день: в 10:00 и 15:15. Если эта программа работает в течение 15 минут в многопользовательской системе и занимает много ресурсов CPU, то в указанные периоды времени работа пользователей может значительно замедлиться.
- Периоды времени, когда производительность должна быть максимальной.
- Набор ожидаемых запросов и его изменение со временем.
- Число пользователей, приходящееся на один компьютер, и полное число пользователей, если это многопользовательское приложение. Этот пункт должен также включать информацию о том, сколько раз пользователи входят в систему и выходят из нее, предполагаемую скорость нажатия клавиш, выполняемые запросы и время обдумывания запроса. Рекомендуется изучить, сколько времени в среднем проходит между переходом от одного запроса к другому.
- Сведения пользователей относительно компьютеров, на которых они будут работать. Если пользователь будет работать на каком-то конкретном компьютере, то вы должны узнать об этом как можно раньше. Аналогично, если у пользователя есть информация о конкретном типе, размере, стоимости, расположении, взаимодействии или о других параметрах, которые могут влиять на предыдущие требования, эта информация также должна быть включена в перечень требований. Может оказаться так, что в системе, применяемой для разработки, тестирования или установки программ, эти требования выполнить нельзя.

Оценка требований к ресурсам рабочей схемы

Оценка необходимых ресурсов является одной из наиболее трудных задач в процессе планирования производительности. Исключение составляет лишь случай, когда был приобретен пакет программ, в документации к которому есть подробный перечень всех необходимых ресурсов.

При решении этой задачи возникают следующие трудности:

- Любую задачу всегда можно решить несколькими способами. Например, можно написать программу на языке C (или на любом другом языке высокого уровня), сценарий оболочки, сценарий **perl**, сценарий **awk**, сценарий **sed**, многооконную программу AIX и т.д. Некоторые способы, которые кажутся особенно эффективными с точки зрения разработки алгоритма и создания программы, могут в будущем привести к значительным затратам при эксплуатации программы.

  В общем случае закономерность такова закономерность: чем выше уровень абстракции, тем сложнее гарантировать, что не возникнет никаких неожиданных проблем с производительностью. Необходимо всегда учитывать реальный объем данных и число итераций, скрытых в безобидных на вид конструкциях.
- Ресурсы, используемые конкретным процессом, определить довольно сложно. Это не только техническая, но также и методическая трудность. Если разные экземпляры данной программы, запущенные несколькими пользователями, совместно обращаются к одним и тем же страницам текста программы, то к какому процессу нужно относить использование этих страниц памяти? Операционная система хранит в кэш-памяти страницы файлов, использованные недавно, чтобы программы могли повторно обращаться в этим данным. Если программа несколько раз обращается к одним и тем же данным, то следует ли учитывать тот объем памяти, который использовался для хранения этих данных? Точность некоторых измерений, например, показаний системных часов, может привести к сложностям при оценке времени использования CPU последовательно запускаемыми экземплярами одной и той же программы.

  Существует два подхода к анализу неоднозначных данных, содержащихся в отчете об использовании ресурсов. Первый основан на том, что неопределенности игнорируются, а изменяющиеся величины исключаются из рассмотрения до тех пор, пока в результате измерений не начнут поступать согласованные данные. При втором подходе по возможности измеряются реальные параметры, а для описания результатов применяются статистические методы. Второй подход позволяет получить результаты, которые больше соответствуют реальным условиям.
- Случаи, когда в системе выполняется только одна программа, очень редки. Обычно в системе выполняются несколько демонов, обслуживаются активные соединения и одновременно работают

несколько пользователей. При этом зависимость объема занятых ресурсов от числа выполняемых операций не линейная. Например, увеличение числа экземпляров программы может привести к использованию только нескольких новых страниц текста программы, поскольку большая часть программы уже была в памяти. В то же время, новый процесс может усилить конкуренцию за использование кэш-памяти процессора. Следовательно, он не только увеличивает конкуренцию за процессорное время, но и приводит к тому, что увеличивается время, затрачиваемое на выполнение одной команды. Это связано с возрастанием числа промахов в кэше, и в итоге приводит к снижению скорости выполнения всех процессов.

При оценке параметров производительности программы постарайтесь быть как можно ближе к реальным условиям, руководствуясь следующими рекомендациями:

- Если программа уже существует, то выполните измерения в условиях, максимально приближенных к исходным требованиям. Для этого лучше всего воспользоваться специальным средством для планирования объема необходимых ресурсов, например, планировщиком BEST/1.
- Если подходящей системы нет, то установите программу в любой другой системе и попробуйте искусственно создать в ней похожую среду.
- Если похожую среду создать не удается, то измерьте отдельные параметры программы и используйте полученные результаты для моделирования.
- Если программа еще не создана, то найдите похожую программу с аналогичной структурой, написанную на том же языке программирования, и измерьте ее параметры. Не забудьте, что чем абстрактнее язык, тем осторожнее нужно подходить к вопросу подбора подобных программ.
- Если похожую программу найти не удалось, создайте прототип программы с основными алгоритмами на используемом языке программирования, измерьте его параметры и смоделируйте рабочую схему.
- Только в том случае, если невозможно сделать никаких практических измерений, можно ограничиться теоретическими оценками. Если объем необходимых ресурсов необходимо определить на стадии планирования, то особенно важно протестировать программу на самых ранних этапах ее разработки.

Обратите внимание, что независимые разработчики программного обеспечения (ISV) часто приводят рекомендации по определению объема ресурсов, необходимых для работы их приложений.

При оценке необходимого объема ресурсов вам должны интересовать четыре параметра:

Процессорное время
: Показатель использования процессора рабочей схемой

Доступ к диску
: Частота, с которой рабочая схема генерирует запросы на чтение с диска или запись на диск

Нагрузка на LAN
: Число пакетов, генерируемых рабочей схемой, и число байтов, передаваемых при обмене данными

Физическая память
: Объем оперативной памяти, необходимой для рабочей схемы

В следующих разделах описано, каким образом можно определить значения этих параметров в различных ситуациях.

Измерение ресурсов для задач

Если для оценки будет применяться исходная программа, ее аналог или прототип, то выбор способа оценки определяется следующими условиями.

Эти факторы таковы:

- Выполняются ли в системе другие задачи, помимо рабочей схемы, характеристики которой будут измеряться?

- Есть ли у администратора права доступа на применение инструментов, которые могут привести к снижению производительности? Например: является ли данная система рабочей, либо она специально предназначена для проведения измерений?
- С какой степенью точности вы можете смоделировать исходную рабочую схему?

Оценка ресурсов, необходимых для рабочей системы:

Использование тестовой (специально выделенной) системы - это идеальный случай, так как в ней можно создать ситуацию искусственной перегрузки, а также оценить объем ресурсов, используемых отдельными процессами.

Для большинства операций производительность системы можно оценить с помощью команды **vmstat**:

```
# vmstat 5 >vmstat.output
```

Эта команда выдает информацию о состоянии системы каждые 5 секунд, пока выполняется измерение. Первый отчет команды **vmstat** содержит информацию, накопленную с момента загрузки системы до вызова команды **vmstat**. В каждом следующем наборе выдается информация, собранная за предшествующий интервал времени (в данном случае - за 5 секунд). Ниже приведен типичный пример вывода команды **vmstat**:

```
нити     память             страница          ошибки        cpu
----- ----------- ----------------------- ------------ -----------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
 0  1 75186   192   0   0   0   0    1   0 344 1998 403  6  2 92  0
```

Для оценки нагрузки на CPU и дисковую память вызовите команду **iostat**:

```
# iostat 5 >iostat.output
```

Эта команда выдает информацию о состоянии системы каждые 5 секунд, пока выполняется измерение. Первый отчет команды **iostat** содержит информацию, накопленную с момента загрузки системы до вызова команды **iostat**. В каждом следующем наборе выдается информация, собранная за предшествующий интервал времени (в данном случае - за 5 секунд). Ниже приведен пример вывода команды **iostat**:

```
tty:      tin         tout   avg-cpu:  % user    % sys     % idle    %iowait
          0.0          0.0              19.4      5.7       70.8       4.1

Диски :       % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk0           8.0      34.5       8.2         12       164
hdisk1           0.0       0.0       0.0          0         0
cd0              0.0       0.0       0.0          0         0
```

Для того чтобы получить информацию об использовании оперативной памяти, вызовите команду **svmon**. Команда **svmon -G** выдает информацию об использовании всей оперативной памяти. Все значения в выводе этой команды измеряются в страницах по 4 КБ:

```
# svmon -G

               разм.     занято      своб.      закр.     вирт.
memory        65527      65406        121       5963      74711
pg space     131072      37218

               work       pers       clnt        lpage
pin            5972          0          0            0
in use        54177       9023       2206            0
```

В этом примере занята вся оперативная память системы объемом 256 МБ. Около 83 процентов памяти занято рабочими сегментами - память, выделенная для ввода-вывода активным программам (все остальное занято файлами кэша). Если в системе запущен какой-либо продолжительный процесс, то можно получить информацию об объеме памяти, который используется этим процессом. Ниже приведен пример вывода команды, содержащий информацию об использовании памяти процессом пользователя `hoetzel`.

```
# ps -fu hoetzel
     UID   PID  PPID   C    STIME    TTY  TIME CMD
 hoetzel 24896 33604   0 09:27:35  pts/3  0:00 /usr/bin/ksh
 hoetzel 32496 25350   6 15:16:34  pts/5  0:00 ps -fu hoetzel

# svmon -P 24896

------------------------------------------------------------------------------
     Pid Command        Inuse      Pin     Pgsp  Virtual   64-bit    Mthrd    LPage
   24896 ksh             7547     4045     1186     7486        N        N        N

  Vsid     Esid Тип  Описание              LPage   Inuse   Pin Pgsp Virtual
     0        0 work kernel seg               -    6324  4041 1186  6324
6a89aa        d work shared library text      -    1064     0    0  1064
72d3cb        2 work process private          -      75     4    0    75
401100        1 pers code,/dev/hd2:6250       -      59     0    -     -
 3d40f        f work shared library data      -      23     0    0    23
16925a        - pers /dev/hd4:447             -       2     0    -     -
```

Экземпляр программы **ksh** занимает 4 страницы в рабочем сегменте (5176). 2619 страниц памяти, отведенных общей библиотеке, и 58 страниц, отведенных программе **ksh**, используются всеми работающими программами и всеми экземплярами **ksh**, соответственно.

Если оперативная память размером 256 МБ слишком велика, то с помощью команды **rmss** можно уменьшить объем оперативной памяти и заново измерить параметры рабочей схемы. Если при этом сильно увеличивается интенсивность подкачки или ухудшается время ответа, то размер уменьшен слишком сильно. Эта процедура может применяться до тех пор, пока не будет найден точный размер, при котором не ухудшаются характеристики рабочей схемы. Дополнительная информация приведена в разделе “Оценка необходимого объема памяти с помощью команды rmss” на стр. 143.

Для измерения нагрузки на сеть в первую очередь применяется команда **netstat**. Ниже приведен пример вывода команда с информацией о работе интерфейса Token-Ring:

```
# netstat -I tr0 5
   input    (tr0)     output            input   (Total)    output
 пакетов ошибок пакетов ошибок конфл. пакетов ошибок пакетов ошибок конфл.
35552822 213488 30283693    0     0  35608011 213488 30338882    0     0
     300     0      426     0     0       300     0      426     0     0
     272     2      190     0     0       272     2      190     0     0
     231     0      192     0     0       231     0      192     0     0
     143     0      113     0     0       143     0      113     0     0
     408     1      176     0     0       408     1      176     0     0
```

Первая строка отчета отражает совокупную нагрузку в сети с момента последней перезагрузки. Остальные строки содержат информацию, получаемую с интервалом в 5 секунд.

Измерение параметров задач в рабочей системе:

Измерение параметров в рабочей системе делается теми же способами, что и в тестовой системе, однако при этом необходимо прилагать дополнительные усилия, чтобы избежать снижения производительности системы.

Наиболее эффективным средством для оценки необходимого объема ресурсов является команда **vmstat**, которая позволяет получить данные об использовании оперативной памяти, устройств ввода-вывода и CPU. Если вы укажете в команде **vmstat** достаточно большой период сбора информации, например, 10 секунд, то выполнение этой команды не потребует много процессорного времени. Дополнительная информация по команде **vmstat** приведена в разделе “Определение ресурсов, ограничивающих производительность” на стр. 33.

Оценка ресурсов, необходимых для неполной загрузки рабочей системы:

Под неполной загрузкой подразумевается измерение части параметров нагрузки реальной системы для будущего переноса этой нагрузки в другую систему.

Поскольку система находится в рабочем состоянии, необходимо свести к минимуму возможное влияние измерений на ее производительность. В то же время мы должны проанализировать рабочую схему достаточно подробно, чтобы можно было установить четкое различие между компонентами рабочей схемы, которые нас интересуют, и остальными ее частями. Для того чтобы выполнить частичные измерения, необходимо выяснить, что общего у интересующих нас элементов рабочей схемы. Это может быть:

- Одна и та же программа или небольшой пакет связанных программ
- Действие, выполняемое одним или несколькими пользователями в системе
- Запрос, поступающий с нескольких конкретных терминалов

В зависимости от того, насколько тесно связаны компоненты рабочей схемы, воспользуйтесь одной из следующих команд:

```
# ps -ef | grep имя-программа
# ps -fuusername, . . .
# ps -ftttyname, . . .
```

для поиска интересующих нас процессов и создания отчета об использовании CPU этими процессами. После этого с помощью команды **svmon** можно узнать объем памяти, занятой этими процессами.

Статистика отдельных программ:

Существует много способов оценки ресурсов, используемых отдельными программами. Некоторые из них позволяют выполнять комплексные измерения рабочей схемы, но они слишком сильно влияют на производительность систем в рабочем состоянии.

Большинство таких процедур обсуждается в главах, посвященных оптимизации потребления конкретных ресурсов. Вот некоторые из них:

svmon Позволяет узнать объем оперативной памяти, занимаемый процессом. Описан в разделе “Использование памяти” на стр. 128.

time Измеряет время выполнения отдельной программы и затраченное на ее выполнение время CPU. Описан в разделе “Применение команды time для измерения нагрузки процессора” на стр. 112.

tprof Измеряет относительную загрузку процессора программами, библиотеками подпрограмм и ядром операционной системы. Описан в разделе Инструмент профилирования *Performance Tools Guide and Reference*.

vmstat -s
Измеряет нагрузку на подсистему ввода-вывода, создаваемую программой. Описан в разделе “Оценка общего объема дискового ввода-вывода с помощью команды vmstat” на стр. 188.

Оценка ресурсов, необходимых для выполнения новой программы

На стадии разработки программы и создания кода трудно делать какие-либо предсказания, однако оценить ресурсы, необходимые программе, можно с помощью следующих приблизительных расчетов.

Если программа еще не написана, то невозможно точно оценить объем ресурсов, требующихся для ее выполнения. В качестве отправной точки можно считать, что для программы необходимо как минимум следующее:

- Около 50 миллисекунд процессорного времени (в основном - в системном режиме)
- Физическая память
  - Одна страница для текста программы
  - Около 15 страниц (из них - 2 прикрепленных) для рабочего сегмента (сегмента данных)

- Библиотека `libc.a`. Обычно эта библиотека используется одновременно всеми программами, поэтому этот ресурс учитывается как часть операционной системы.

- Если программа в последнее время не компилировалась, не копировалась и не запускалась, то потребуется около 12 операций подкачки с диска. В противном случае, не потребуется ни одной операции.

К перечисленному выше необходимо добавить ресурсы, объем которых напрямую зависит от особенностей программы (приведенные ниже значения даны только в качестве примера):

- Процессорное время
  - Обычная программа, не выполняющая много итераций и не содержащая подпрограмм, для которых требуется много ресурсов, практически не использует процессор.
  - Если программа выполняет большой объем вычислений, заложенных в алгоритме, то сделайте необходимые измерения на основе прототипа программы.
  - Если программа использует библиотечные функции, выполняющие большой объем вычислений, например, такие конструкции как функция **printf()** в X или Motif, то измерьте время CPU, требуемое для выполнения этих функций в тривиальной программе.
- Физическая память
  - Предположим, что на странице текста программы приходится 350 строк кода, по 12 байт в каждой строке. Следует помнить, что стиль программирования и опции компилятора могут уменьшить или увеличить это число в несколько раз. Это допущение справедливо для расчета числа страниц, соответствующего стандартному (типичному) сценарию. Если в программе редко вызываемые подпрограммы размещаются в конце текста программы, то такие страницы не будут все время находиться в физической памяти.
  - Ссылки на общие библиотеки, отличные от `libc.a`, увеличивают расход памяти. Однако дополнительная память потребуется только для тех библиотек, которые не используются совместно с другими программами или экземплярами оцениваемой программы. Для того чтобы оценить объем этих библиотек, напишите тривиальную программу, использующую эти библиотеки, и вызовите для процесса этой программы команду **svmon -P**.
  - Оцените объем памяти, необходимый структурам данных, объявленным в программе. Округлите это значение (с избытком) до ближайшего целого числа страниц.
  - В программе с небольшим временем выполнения каждая операция дискового ввода-вывода будет использовать одну страницу памяти. Предположим, что эта страница доступна. Тогда программе не придется ждать, пока освободится страница другой программы.
- Дисковый ввод/вывод
  - При последовательном доступе к диску операция ввода/вывода считывает или записывает 4096 байт (однако если к файлу обращались совсем недавно, то некоторые его страницы все еще находятся в памяти).
  - В случае произвольного доступа к диску, операция ввода/вывода выполняется при обращении к любой 4096-байтовой странице для считывания/записи сколь угодно малого объема данных (однако если к файлу обращались совсем недавно, то некоторые его страницы все еще находятся в памяти).
  - В случае последовательного доступа каждая операция чтения или записи страницы объемом 4 КБ в большой файл занимает примерно 100 единиц работы процессора. В случае произвольного доступа каждая операция чтения или записи страницы размером 4 КБ занимает 300 единиц работы процессора. Помните, что хотя файлы записываются и считываются программой последовательно, в действительности они располагаются на разных участках диска. Следовательно, расход процессорного времени при работе с диском на самом деле будет ближе к значению, достигаемому при прямом, а не при последовательном доступе.
- Средства связи
  - Если дисковый ввод-вывод выполняется в удаленной файловой системе NFS, то фактически он выполняется на сервере. Тем не менее, на клиенте возрастает нагрузка на процессор и память.
  - Значительно увеличивает загрузку процессора Удаленный вызов процедур (RPC). Функции RPC должны быть минимизированы и собраны в пакеты; для них необходимо создать прототип и выполнить измерения.

– В случае последовательного доступа каждая операция чтения или записи страницы размером 4 КБ из файловой системы NFS занимает примерно 600 единиц работы процессора в клиентской системе. В случае произвольного доступа такая операция занимает примерно 1000 единиц процессорного времени в клиентской системе.
– Работа Web-браузеров и Web-серверов создает значительную нагрузку на сеть, поскольку соединения TCP открываются и закрываются довольно часто.

Преобразование оценок на уровне программы в оценки на уровне рабочей схемы

Лучший способ оценить ресурсы, необходимые при работе в режиме обычной и пиковой нагрузки, - это использование специальных средств моделирования, например, планировщика BEST/1.

Можно применять и статические модели, но при этом вы рискуете переоценить или недооценить ресурсы при работе в режиме пиковой нагрузки. В любом случае, вам необходимо понимать, каким образом взаимодействие программ рабочей схемы влияет на объем необходимых ресурсов.

При разработке статической модели используйте интервал времени, равный наихудшему времени ответа для наиболее часто вызываемой или запрашивающей программы (обычно они одинаковы). Определите, какие программы будут запускаться в разное время дня, в зависимости от предполагаемого числа пользователей, времени принятия решения, скорости нажатия клавиш и предполагаемого набора операций.

Ниже приведены некоторые рекомендации:

- Процессорное время
  – Сложите значения процессорного времени для всех программ, запускаемых в течение данного промежутка времени. Добавьте время, необходимое для обработки обращений к диску и устройствам связи.
  – Если значение, полученное для какого-либо интервала времени, превышает 75 процентов процессорного времени, то уменьшите число пользователей или увеличьте число процессоров.
- Физическая память
  – Объем физической памяти, необходимый для работы операционной системы, зависит от общего объема физической памяти. В качестве начального значения можно принять 6 - 8 МБ. Меньшая цифра относится к автономной системе, большая - к системе, подключенной к сети и использующей TCP/IP или NFS.
  – Сложите объем рабочих сегментов для всех экземпляров программ, которые будут запускаться в течение данного интервала времени, а также объем памяти, который выделяется под структуры данных, объявленные в программах.
  – Добавьте к этому необходимую память для сегментов текстов всех программ, которые будут запускаться (одна копия текста программы применяется для всех экземпляров программы). Помните, что в исполняемую программу добавляются только функции из пользовательских библиотек, при этом сами библиотеки в память не загружаются.
  – Добавьте память, занимаемую всеми общими библиотеками, которые будут использоваться какой-либо программой в рабочей схеме. (Одна копия обслуживает все запросы.)
  – Полученная оценка объема памяти не должна превышать 80 процентов от общего объема памяти системы, так как должно оставаться достаточно памяти для кэширования файлов и списка свободных страниц.
- Дисковый ввод/вывод
  – Просуммируйте предполагаемое число операций ввода/вывода для всех экземпляров каждой программы. Отдельно учитывайте ввод-вывод для небольших файлов (или ввод-вывод для больших файлов в режиме произвольного доступа) и ввод-вывод в режиме последовательного доступа при чтении/записи больших файлов (больше 32 КБ).
  – Вычтите соответствующий объем памяти из общего значения. Запись, которая была прочитана или записана в предыдущем интервале, может быть доступной и в текущем интервале. Затем необходимо сравнить объем памяти предполагаемого компьютера с полным объемом памяти, требующейся для

рабочей схемы. Если помимо памяти, необходимой операционной системе и рабочей схеме, остается неучтенная память, то, возможно, в этой области памяти содержатся страницы файлов, которые считывались или записывались совсем недавно. Если ваше приложение с большой вероятностью повторно использует данные, к которым оно уже недавно обращалось, то вы можете рассчитать параметры кэширования. Помните, что повторное использование выполняется на уровне страницы, а не на уровне записи. Если вероятность повторного использования записи мала, но страница содержит много записей, то вполне вероятно, что некоторые записи будут находиться на той же странице, что и другие записи, использованные недавно.

  - Сравните полученную оценку для операций ввода-вывода (объем дискового ввода-вывода в секунду для каждого диска) с возможностями установленных дисков. Если значение, полученное для последовательного или произвольного доступа, составляет более 75 процентов от объема дисков, на которых будут хранится данные приложения, то для работы приложения потребуется выполнить дополнительную настройку (а при необходимости - увеличить число дисков).
- Средства связи
  - Рассчитайте пропускную способность (быстродействие) рабочей схемы. Если необходимая пропускная способность всех узлов локальной сети составляет более 70 процентов от номинальной (50 процентов для Ethernet), то потребуется увеличить пропускную способность сети.
  - Аналогичный анализ требований к CPU, оперативной памяти и подсистеме ввода-вывода необходимо выполнить и для дополнительных операций, выполняемых сервером.

Примечание: Эти рекомендации рассчитаны *только* на тот случай, когда невозможно выполнить более точные измерения параметров. Если существуют результаты измерений для конкретного приложения, то используйте их вместо соответствующих теоретических расчетов, поскольку они дают намного более точную оценку.

Эффективное проектирование программы и ее реализация

Если вы точно знаете, какие ресурсы ограничивают быстродействие программы, то можете сразу перейти к разделу, посвященному способам минимизации расхода соответствующих ресурсов.

В противном случае следует исходить из того, что программа нуждается в оптимизации, и при этом будут использоваться все рассматриваемые в этой главе способы оптимизации. После того, как программа будет готова, перейдите к разделу “Определение ресурсов, ограничивающих производительность” на стр. 33.

Программы, ограничения которых связаны с процессором

Если процессор оказывается недостаточно мощным для обработки программы, так как эта программа выполняет огромное количество вычислений, то низкая производительность обусловлена неудачным выбором алгоритма.

Максимальная скорость обработки программы, если она действительно ограничена ресурсами процессора, зависит от следующих факторов:

- Применяемого алгоритма
- Исходного кода и структур данных, созданных программистом
- Последовательности команд на машинном языке, созданной компилятором
- Размера и структуры кэш-памяти процессора
- Архитектуры и тактовой частоты самого процессора (см. “Определение быстродействия процессора” на стр. 414).

Вопросы, связанные с выбором алгоритма, выходят далеко за рамки этой книги. Поэтому в дальнейшем мы будем исходить из того, что выбран самый эффективный алгоритм.

Помимо выбора алгоритма, от программиста зависят только три фактора из числа перечисленных выше: исходный код, опции компилятора и, возможно, структуры данных. Последующие разделы посвящены

способам повышения производительности конкретной программы в случае, если пользователь располагает ее исходным кодом. Если же исходного кода нет, попробуйте изменить конфигурацию или перераспределить ресурсы.

Проектирование программ для эффективной работы с кэшем

Эффективное использование памяти означает, что она должна быть полностью занята нужными командами и данными.

В процессорах реализована иерархия памяти:

1. Конвейер команд и регистры процессора
2. Кэш команды и данных, а также соответствующие таблицы преобразования адресов
3. Оперативная память
4. Диск

Каждый следующий уровень в этой иерархии обеспечивает меньшее быстродействие, чем предыдущий, зато больше по объему и дешевле. Следовательно, для достижения максимальной производительности на конкретном компьютере программист должен обеспечить максимальную эффективность работы с памятью на каждом уровне.

К сожалению, память выделяется блоками фиксированной длины (например, целыми строками кэша или страницами реальной памяти), а программный код и структуры данных имеют различную длину, поэтому остаются неиспользуемые участки памяти. Если при разработке программ и структур данных не учитывалась иерархическая структура памяти, то память часто распределяется нерационально, что приводит к снижению производительности при работе в небольших или сильно загруженных системах.

Речь идет о том, что программист должен знать и соблюдать основные принципы повышения эффективности программы при работе с кэш-памятью или виртуальной памятью. С помощью средств создания пакетов можно увеличить производительность существующих программ без изменения исходного кода, а при написании новых программ следует стремиться к повышению эффективности работы с памятью.

При рассмотрении вопросов эффективного использования памяти с иерархической структурой следует уделить особое внимание таким понятиям, как *компактность ссылок* и *рабочий набор*.

- Компактность ссылок программы - это степень концентрации в памяти адресов команд и ссылок на данные в течение заданного периода времени (т.е. хранятся ли все эти ссылки в одном небольшом фрагменте памяти или разбросаны).
- Рабочий набор программы в течение того же периода времени - это набор используемых блоков памяти, или, точнее, набор фрагментов кода и данных, записанных в эти блоки памяти.

У программы с высокой компактностью ссылок рабочий набор будет минимальным, поскольку используемые блоки памяти плотно заполнены фрагментами выполняемого кода и данными. В то же время у функционально эквивалентной программы с низким уровнем компактности ссылок рабочий набор будет больше, так как для размещения большого количества ссылок требуется много блоков памяти.

Поскольку загрузка каждого блока в память заданного уровня требует времени, для повышения эффективности программ в системе с иерархической структурой памяти следует разрабатывать и упаковывать программный код таким образом, чтобы рабочий набор был минимальным.

На следующем рисунке приведены примеры рационального и нерационального программирования на уровне процедур. В первом варианте порядок процедур в программе отражает лишь способ мышления автора. Первая функция **PriSub1** содержит точку входа в программу. Она всегда вызывает вспомогательные функции **PriSub2** и **PriSub3**. Дополнительные функции **SecSub1** и **SecSub2** предназначены для выполнения некоторых редко встречающихся задач. Функции обработки ошибок **ErrSub1** и **ErrSub2** вызываются только в исключительных случаях.

Низкая локальность ссылок, большой рабочий набор

Страница 1			Страница 2			Страница 3
PriSub1	SecSub1	ErrSub1	PriSub2	SecSub2	ErrSub2	PriSub3

Высокая локальность ссылок, небольшой рабочий набор

Страница 1			Страница 2			Страница 3
PriSub1	PriSub2	PriSub3	SecSub1	SecSub2	ErrSub1	ErrSub2

Рисунок 15. Компактность ссылок. В верхней области рисунка продемонстрировано, что двоичная программа упакована с низким уровнем компактности ссылок. Инструкции функции **PriSub1** расположены в начале двоичного исполняемого кода. За ними расположены инструкции функций **SecSub1**, **ErrSub1**, **PriSub2**, **SecSub2**, **ErrSub2** и **PriSub3**. В данном случае инструкции функций **PriSub1**, **SecSub1** и **ErrSub1** расположены в первой странице памяти. Инструкции функций **PriSub2**, **SecSub2**, **ErrSub2** расположены во второй странице, а инструкции функции **PriSub3** - в третьей странице памяти. Функции **SecSub1** и **SecSub2** используются редко, а функции **ErrSub1** и **ErrSub2** применяются лишь в исключительных ситуациях. Следовательно, эта программа упакована таким образом, что получился низкий уровень компактности ссылок. В результате программа занимает излишне много памяти. В нижней области рисунка показан альтернативный способ размещения программы в памяти. Функции **PriSub1**, **PriSub2** и **PriSub3** расположены в первой странице памяти. Функции **PriSub3**, **SecSub1**, **SecSub2** и **ErrSub1** расположены во второй странице памяти. Функция **ErrSub2** занимает третью страницу памяти. Поскольку функция **ErrSub2** практически никогда не используется, объем памяти, необходимый для работы программы, уменьшится на одну страницу.

Компактность ссылок первой версии программы низкая, так как в обычной ситуации для ее работы потребуется три страницы памяти. Дополнительные функции и функции обработки ошибок выделены в отдельные блоки, вследствие чего основной путь к программе распадается на три части, физически удаленные друг от друга.

В усовершенствованной версии программы все основные функции расположены в начале исполняемого кода. За ними следуют редко используемые функции. Необходимые (но крайне редко используемые) функции обработки ошибок помещены в конец программы. Теперь для выполнения большинства функций программы достаточно одной операции чтения с диска и одной страницы памяти вместо трех.

Помните, что компактность ссылок и рабочий набор определяются для конкретного момента времени. Если программа выполняется поэтапно, причем каждый этап занимает значительное время, и на каждом этапе вызываются разные функции, следует минимизировать рабочий набор для каждого этапа в отдельности.

Регистры и конвейер

Как правило, размещение информации по регистрам, оптимизация регистров и заполнение конвейера осуществляются компилятором.

Все, что требуется от программиста, - избегать конструкций, плохо поддающихся оптимизации. Например, если некоторая процедура обрабатывается в одном из важнейших циклов программы, компилятор скорее всего попытается сделать эту процедуру внутренней, чтобы не тратить время на загрузку. Если же функция содержится в другом исходном модуле (файле с расширением .c), компилятор не сможет сделать ее внутренней.

Кэш и TLB

Кэш может содержать таблицы преобразования адресов (TLB), на основании которых виртуальные адреса преобразуются в реальные адреса страниц памяти, содержащих тексты инструкций или данные.

В зависимости от архитектуры и модели, процессор содержит один или несколько кэшей, которые предназначены для хранения следующих объектов:

- Компонентов исполняемых программ
- Данных, с которыми работают исполняемые программы
- TLB

Если при чтении из кэша произошел промах, загрузка полной строки кэша может растянуться на десять и более циклов процессора. Если произошел промах при операциях с TLB, на повторное вычисление таблицы преобразования виртуальных адресов в реальные может уйти несколько десятков циклов. Точные цифры зависят от реализации.

Даже если размеры фрагментов программы и блоков данных точно соответствуют размерам кэша, загрузка информации в кэш выполняется тем дольше, чем больше занято строк в кэше или записей TLB (т.е. чем ниже компактность ссылок). За исключением случая многократной обработки одних и тех же инструкций и данных, время на загрузку инструкций и данных составляет значительную часть общего времени выполнения программы, а увеличение объема загружаемых данных снижает быстродействие системы.

Оптимальный стиль программирования заключается в создании максимально компактной основной последовательности выполнения программы. Главная процедура и все подпрограммы, к которым она часто обращается, должны следовать друг за другом. Все маловероятные события, например, неожиданные ошибки, в ходе основной последовательности выполнения программы должны не обрабатываться, а только диагностироваться. Если такая ситуация действительно возникнет, она должна обрабатываться в отдельной процедуре. Все эти процедуры следует сгруппировать и поместить в конец модуля. Такая структура программы снизит вероятность того, что редко используемый код будет занимать место в кэше. Если модуль достаточно велик, то некоторые или все редко используемые процедуры будут расположены на странице, которая крайне редко загружается в память.

Это же правило относится и к структурам данных, хотя иногда приходится изменять код программы, чтобы компенсировать недостатки размещения данных компилятором.

Например, если напрямую запрограммировать операции с матрицами (в частности, умножение матриц), то у них будет очень низкий уровень компактности ссылок. В таких операциях почти всегда требуется перебрать все элементы матрицы по порядку. У каждого компилятора есть свои правила размещения матриц в памяти. Компилятор FORTRAN размещает матрицы по столбцам (то есть, сначала все элементы первого столбца, затем - второго и т.д.). В то же время компилятор C размещает матрицы по строкам. Для матриц небольшого размера все строки и столбцы умещаются в кэше одновременно, так что процессор и математический сопроцессор могут работать с полной скоростью. Однако по мере увеличения размера матрицы компактность ссылок для операций над строками и столбцами снижается, и, в конце концов, хранить данные в кэше становится невозможно. Происходит следующее: если строка матрицы длиннее строки кэша, то в ходе операции над строкой (например, умножения строки на столбец) уже помещенные в кэш элементы строки выбрасываются, а затем снова считываются, но уже с другой позиции. Таким образом, одни и те же данные помещаются в кэш несколько раз.

Общий рецепт для подобных случаев - разбить операцию на блоки так, чтобы над элементами, помещенными в кэш, можно было выполнить сразу несколько операций. Эта техника называется *поблочной выборкой (strip mining)*.

Перед специалистами по численным методам была поставлена задача - создать программы на основе алгоритмов работы с матрицами, использующие технику поблочной выборки и другие способы оптимизации. В результате скорость умножения матриц повысилась в 30 раз. Эти оптимизированные процедуры входят в состав библиотеки BLAS, `/usr/lib/libblas.a`. Большое количество оптимизированных процедур содержится в лицензионной программе ESSL.

Функции и интерфейсы из библиотеки BLAS приведены в документе *AIX версии 7.1 - технический справочник*. Для работы с этой библиотекой необходимо установить среду времени выполнения FORTRAN. При

выполнении операций с матрицами и векторами рекомендуется пользоваться готовыми процедурами из этой библиотеки, поскольку компактность ссылок для этих процедур значительно выше той, которой может достичь обычный пользователь.

Если структуры данных описываются программистом, в его распоряжении находятся и другие средства повышения эффективности. Общий принцип оптимизации данных - разместить часто используемые данные рядом друг с другом, насколько это возможно. Если структура данных содержит часто используемую управляющую информацию и подробную информацию, к которой обращаются сравнительно редко, то убедитесь, что управляющая информация расположена единым блоком, а не рассредоточена. Это повысит вероятность того, что управляющая информация будет целиком загружена в кэш - как минимум, количество промахов будет сведено к минимуму.

Препроцессор и компилятор

Существует несколько различных уровней оптимизации, отличающихся друг от друга уровнем свободы, предоставленной компилятору при переупорядочении инструкций.

Ниже приведены некоторые рекомендации для программиста, который хочет достичь максимальной скорости выполнения программы на конкретном компьютере:

- Препроцессоры могут изменять некоторые структуры исходного кода так, что полученный функционально эквивалентный исходный модуль после компиляции превратится в более эффективный исполняемый код.
- Для каждого варианта архитектуры существует набор опций компилятора, обеспечивающих оптимальную компиляцию для данного варианта архитектуры.
- С помощью директивы **#pragma** программист может сообщить компилятору С о некоторых особенностях программы, что позволит ему создать более эффективный исполняемый код, уделяя меньше внимания тем наиболее неблагоприятным ситуациям, возникновение которых маловероятно.

Если у программиста нет возможности сравнить скорость выполнения оптимизированной и неоптимизированной программы, то он всегда должен использовать оптимизацию. Различие в скорости выполнения между оптимизированным и неоптимизированным кодом почти всегда оказывается столь велико, что в любом случае следует применять хотя бы базовый уровень оптимизации (опция **-O** в командах компиляторов). Единственным исключением являются ситуации, в которых нужна именно прямая компиляция - например, анализ производительности на уровне отдельных операторов с помощью средства **tprof**.

Эти методы способствуют повышению производительности для некоторых программ, однако для определения сочетания методов, дающего наибольший выигрыш в производительности для конкретной программы, может потребоваться огромное количество перекомпиляций и измерений.

Более подробная информация об эффективном использовании компиляторов приведена в книге *Optimization and Tuning Guide for XL Fortran, XL C and XL C++*.

Уровни оптимизации

Степень оптимизации кода компилятором определяется параметром **-O**.

Нет оптимизации
: В этом случае, в отличие от любых флагов типа **-O**, компилятор выполняет прямую компиляцию, не изменяя порядка следования инструкций, и не предпринимая никаких других попыток оптимизации.

-O или -O2
: Эти флаги эквивалентны и означают, что оптимизация должна выполняться практически без изменения порядка следования инструкций. Применяются только явно разрешенные (например, директивой **#pragma**) средства оптимизации. На этом уровне не применяются программный конвейер, преобразование циклов в линейный код и простое опережающее объединение. Кроме того, на этом уровне компилятору выделен ограниченный объем памяти.

-O3 Этот флаг разрешает компилятору применять все доступные средства оптимизации и снимает

ограничения на объем памяти, выделенной компилятору. Оптимизация на этом уровне может привести к неправильной работе программы, если в ней есть исключительные ситуации с числами с плавающей точкой, знаком нуля или точностью округления: Этих побочных последствий оптимизации можно избежать, хотя и за счет некоторого снижения производительности, путем указания опции **-qstrict** вместе с **-O3**. Сочетание опций **-qhot** и **-O3** разрешает опережающее объединение и, в некоторых случаях, развертку циклов. Следовательно, в этой версии при компиляции больших или сложных процедур с опцией **-O3** (возможно, вместе с опцией **-qstrict** или **-qhot**) достигается тот же или более высокий уровень производительности, чем в предыдущих версиях при компиляции с опцией **-O**.

-O4 Этот флаг эквивалентен комбинации **-O3 -qipa** с автоматическим созданием архитектуры и оптимальной настройкой для платформы.

-O5 Этот флаг аналогичен **-O4**, но в данном случае **-qipa = level = 2**.

Компиляция для конкретных аппаратных платформ

Приведены замечания по компиляции программ для определенных аппаратных платформ.

В системах устанавливаются процессоры различных типов. С помощью опций **-qarch** и **-qtune** вы можете оптимизировать программы с учетом специальных команд и особенностей этих процессоров.

Ниже приведены некоторые рекомендации:

- Если программа будет работать только в одной системе, либо в группе систем с процессорами одного типа, то укажите тип процессора с помощью опции **-qarch**.
- Если программа будет выполняться в системах с различными типами процессоров, то с помощью параметров **-qarch** и **-qtune** вы можете указать один тип процессора как наиболее приоритетный. Для динамической настройки этих параметров пользователи FORTRAN и HPF могут воспользоваться командами **xxlf** и **xxlhpf**.
- Если нельзя сказать, что программа предназначена преимущественно для какого-то одного типа процессоров, и она будет использоваться в системах с разными типами процессоров, то параметры **-qarch** и **-qtune** использовать не нужно.

Быстродействие функций string.h и опции языка C

В данной операционной системе функции обработки строк можно встраивать в прикладную программу, а не считывать их из `libc.a`. При этом экономится время загрузки и возврата.

Для встраивания функций обработки строк в программу перед вызовом функции в исходном коде нужно указать следующий оператор:

```
#include <string.h>
```

Стили программирования C и C++ и производительность

В большинстве случаев различия в производительности, связанные со стилем программирования на C или C++, неочевидны, а иногда их вообще можно определить только экспериментально.

Ниже приведены некоторые рекомендации:

- Везде, где только возможно, используйте тип *int* вместо *char* или *short*.

  В большинстве случаев для обработки типов *char* и *short* требуется выполнить больше команд. Выполнение дополнительных команд занимает время, а кроме того, эти команды займут в исполняемом коде больше места, чем будет сэкономлено за счет применения более коротких чисел (если речь не идет о больших массивах данных).
- Если же необходимо использовать тип *char*, то везде, где это возможно, указывайте *unsigned char*.

  Для загрузки данных типа *signed char* в регистр требуется выполнить на две команды больше, чем для данных типа *unsigned char*.
- Везде, где это возможно, заменяйте глобальные переменные локальными.

Для обращения к глобальным переменным требуется выполнить больше команд, чем для обращения к локальным переменным. Кроме того, если явно не указано иное, компилятор предполагает, что вызванная подпрограмма может изменять значения всех глобальных переменных. Таким образом, после вызова подпрограммы значения всех глобальных переменных загружаются в память повторно, что приводит к снижению производительности.

- Если необходимо обратиться к глобальной переменной (которая не является общей переменной для этой и других нитей), скопируйте ее значение в локальную переменную и работайте с копией.

  Использование локальной копии дает выигрыш в производительности, за исключением случая, когда обращение к глобальной переменной осуществляется только один раз.

- При записи результатов, а также при сравнении лучше применять двоичные коды, а не строки. При работе со строками увеличивается как количество команд, так и объем памяти, необходимый для хранения данных. Например, последовательность операторов

  ```
  #define situation_1 1
  #define situation_2 2
  #define situation_3 3
  int situation_val;

  situation_val = situation_2;
  . . .
  if (situation_val == situation_1)
  . . .
  ```

  выполняется намного быстрее, чем последовательность

  ```
  char situation_val[20];

  strcpy(situation_val,"situation_2");
  . . .
  if ((strcmp(situation_val,"situation_1"))==0)
  . . .
  ```

- Если необходимо использовать строки, то везде, где это возможно, применяйте строки фиксированной длины, а не строки переменной длины, оканчивающиеся символом NULL.

  Процедуры семейства **mem*()** (такие как **memcpy()**) выполняются быстрее соответствующих процедур семейства **str*()** (например, **strcpy()**). Это связано с тем, что в процедурах семейства **str*()** каждый символ сравнивается с символом конца строки, в то время как в процедурах **mem*()** этого не делается.

Время работы компилятора

На время работы компилятора влияют разные факторы.

Для вызова компилятора C в операционной системе предусмотрено две команды: **cc** и **xlc**. Команда **cc**, изначально предназначенная для вызова системного компилятора C, запускает компилятор C в режиме **langlevel=extended**. В этом режиме можно скомпилировать старые программы на языке C, не совместимые со стандартом ANSI. При работе в этом режиме тратится больше процессорного времени.

Если компилируемая программа совместима со стандартом ANSI, то компилятор C лучше вызвать с помощью программы **xlc**.

Флаг **-O3** неявно включает в себя опцию **-qmaxmem**. Эта опция снимает ограничения на объем памяти, доступный компилятору. Однако при этом возможны следующие побочные эффекты:

- При работе в многопользовательской системе компиляция больших программ в режиме **-O3** может негативно повлиять на скорость выполнения заданий других пользователей.
- В системах с небольшим объемом оперативной памяти компиляция больших программ в режиме **-O3** может занять столько памяти, что потребует постоянной подкачки, и в итоге компиляция будет выполняться очень медленно.

Программы с ограничением по памяти

Для программистов, привыкших к работе в среде с ограниченным объемом адресуемой памяти, например, в DOS, объем сегмента виртуальной памяти размером 256 МБ кажется просто огромным. В результате программист вообще забывает об ограничениях на объем памяти. Но у этой медали есть и оборотная сторона.

Объем виртуальной памяти действительно велик, но скорость доступа к ней непостоянна. Скорость доступа к виртуальной памяти снижается по мере ее заполнения, причем это отношение нелинейно. Пока общий объем виртуальной памяти, используемой всеми программами (т.е. общий объем всех рабочих наборов) не превышает объема свободной физической памяти, скорость доступа к виртуальной памяти почти не отличается от скорости доступа к физической. Когда же общий объем рабочих наборов всех выполняемых программ становится больше суммарного объема доступных страниц памяти, скорость доступа начинает быстро снижаться (при выключенном механизме управления нагрузкой VMM) и в конце концов уменьшается на два порядка. Такое состояние называется *перегрузкой памяти*. Почти все время система занимается перемещением страниц, и ни одно задание не может завершиться, поскольку каждый процесс пытается отвоевать у остальных объем памяти, необходимый для размещения соответствующего рабочего набора. Если включен механизм управления нагрузкой на память VMM, то он может предотвратить загрузку, но за счет значительного увеличения времени ответа.

Снижение производительности вследствие неэффективного использования памяти намного более ощутимо, чем снижение производительности вследствие неэффективного использования кэша, поскольку скорости памяти и диска отличаются намного больше, чем скорости кэша и памяти. В то время как обработка промаха в кэше занимает несколько десятков циклов процессора, обработка страничной ошибки занимает не менее 10 миллисекунд, что эквивалентно по крайней мере 400000 циклам процессора.

Хотя средство управления нагрузкой на память VMM не позволяет довести перегрузку памяти до тупиковой ситуации, дополнительные страничные ошибки приводят к увеличению времени ответа и снижению производительности (см. “Настройка алгоритма управления нагрузкой на память VMM с помощью команды schedo” на стр. 149).

Вытесняемая в виртуальную память структура кода:

Общая рекомендация по минимизации рабочего набора программы заключается в том, чтобы выделить небольшую область для часто используемых фрагментов кода, отделив их от редко используемых фрагментов.

Ниже приведены некоторые рекомендации:

- Не включайте большие блоки обработки ошибок в основную программу. Для этой цели следует создавать особые процедуры, которые лучше всего размещать в отдельном исходном модуле. Это относится не только к процедурам обработки ошибок, но и к любым другим редко вызываемым функциям.
- Структура загрузочного модуля не должна быть произвольной. Попытайтесь разместить часто вызываемые объектные модули как можно ближе друг к другу. Объектные модули, содержащие (предположительно) редко вызываемые процедуры, должны располагаться в конце загрузочного модуля. Эти страницы будут считываться редко.

Вытесняемая в виртуальную память структура данных:

Для минимизации рабочего набора данных попытайтесь собрать все часто обрабатываемые данные вместе и удалить ненужные ссылки на страницы виртуальной памяти.

Ниже приведены некоторые рекомендации:

- В функциях **malloc()** и **calloc()** следует указывать такой объем памяти, который действительно необходим. Никогда не запрашивайте память для всего массива, если в реальной ситуации будет использоваться только часть этого массива. Когда при инициализации элементов массива потребуется еще одна страница,

VMM будет вынужден отобрать эту страницу у другого процесса. Позднее, когда этот процесс попытается повторно обратиться к странице, произойдет страничная ошибка. Функции **malloc()** и **calloc()** работают принципиально по разному.

- Функция **calloc()** заполняет всю выделенную память нулями, и поэтому обращается ко всем выделенным страницам памяти, в то время как **malloc()** обращается только к первой из них. Если операция **calloc()** будет вызвана для получения большой области памяти, из которой впоследствии будет использоваться лишь небольшая часть, то получится, что значительная часть работы была проделана системой впустую. Вначале нужно будет инициализировать все страницы памяти, а затем при возврате страниц физической памяти те из них, которые никогда не были использованы, придется выгрузить в пространства подкачки. Таким образом будут выполнены лишние операции ввода-вывода и заняты дополнительные блоки пространства подкачки.
- Те же проблемы могут возникнуть при обработке связанных списков больших структур (например, буферов). Если в программе содержится большое количество операций выбора по ключу, храните ссылки и ключи отдельно от данных или применяйте хеширование.
- Компактность ссылок - это компактность во времени, а не только в адресном пространстве. Структуры данных нужно инициализировать непосредственно перед работой с ними. В сильно загруженной системе, если от инициализации структур данных до их использования проходит много времени, соответствующие кадры могут быть захвачены другими процессами. Когда программа начнет работать с этими структурами данных возникнут излишние страничные ошибки.
- Аналогично, если большая структура будет обработана сразу же, а потом останется неизменной до конца выполнения программы, следует освободить занятую этой структурой память. При этом для освобождения памяти, выделенной функцией **malloc()** или **calloc()**, недостаточно вызвать функцию **free()**. Функция **free()** освобождает только тот диапазон адресов, который занят самой структурой. Для того чтобы освободить физическую память и пространство подкачки, необходимо также вызвать функцию **disclaim()**. Вызов **disclaim()** должен быть выполнен до вызова **free()**.

Рекомендации по работе с закрепленной памятью:

Для предотвращения зацикливания и тайм-аутов небольшая часть системы должна располагаться в закрепленной области оперативной памяти.

Для этих фрагментов кода и данных понятие рабочего набора лишено смысла, поскольку вся информация постоянно находится в памяти, вне зависимости от того, используется она или нет. Программы, в которых применяется резидентный код или данные (например, пользовательские драйверы устройств), нуждаются в особо тщательной разработке (а готовые программы - в тщательном анализе), чтобы эти программы не занимали слишком много памяти. Ниже приведено несколько примеров:

- Резидентный код в загрузочном модуле (исполняемом файле). Если в программе некоторые объектные модули должны быть резидентными, а другие модули могут загружаться по мере необходимости, резидентные объектные модули следует выделить в отдельный загрузочный модуль.
- Для создания резидентных модулей и структур данных должны быть веские основания. Разработчик программы должен иметь четкое представление о том, при каких условиях эта информация может оказаться необходимой, и о том, допустимы ли при таких условиях страничные ошибки.
- Системному администратору должны быть предоставлены средства для настройки резидентных структур, размер которых зависит от нагрузки, например, пула буферов.

Производительность, рекомендации по установке

Перед началом процесса установки необходимо продумать ряд вопросов.

Подготовка к установке операционной системы

Существует две возможные ситуации:

- Установка операционной системы в новой системе

  Перед началом установки определите размер и параметры расположения файловых систем и пространств подкачки, а затем выясните, каким образом эти параметры устанавливаются в операционной системе.

- Установка нового выпуска операционной системы в существующей системе

  Перед переходом к новому выпуску операционной системы выполните следующие действия:
  - Проверьте, применяется ли файл `/etc/tunables/nextboot`
  - Если файл `/etc/tunables/nextboot` применяется, то проверьте файл `/etc/tunables/lastboot.log` после первой перезагрузки.

Рекомендации перед установкой процессора

Не рекомендуется изменять значения по умолчанию параметров планировщика процессора, например, длительность кванта времени.

Изменять эти параметры во время установки можно только в том случае, если вы выполнили тщательный анализ системы с аналогичной конфигурацией и рабочей схемой.

За рекомендациями после установки обратитесь к разделу “Производительность процессора” на стр. 104.

Рекомендации перед установкой памяти

Изменение пороговых значений оперативной памяти рекомендуется отложить до тех пор, пока вы не соберете достаточную статистику об использовании памяти в рабочей системе.

За рекомендациями после установки обратитесь к разделу “Быстродействие памяти” на стр. 128.

Рекомендации перед установкой дисков

По умолчанию алгоритм определения и расширения логических томов выбирает оптимальные значения параметров. Однако производительность работы подсистемы ввода-вывода будет выше, если при определении размера и размещении логических томов программа установки будет учитывать предполагаемый объем данных и требования рабочей схемы.

Ниже приведены общие рекомендации:

- По возможности, группа томов по умолчанию, rootvg, должна состоять из одного физического тома, на который первоначально была установлена система. Для других физических томов нужно определить другие группы томов. Выполнение этих правил позволяет повысить производительность и упростить задачи администрирования.
- Если группа томов состоит из нескольких физических томов, то следующие операции позволят повысить производительность дисковой подсистемы:
  - Создание группы томов на основе одного физического тома.
  - Определение логического тома в новой группе томов. В результате логический том с журналом группы томов будет расположен на первом физическом томе.
  - Добавление оставшихся физических томов в группу томов.
  - Определение файловых систем с высокой интенсивностью обращений в добавленных физических томах.
  - Определение файловых систем с очень низкой интенсивностью обращений (если такие есть) в физическом томе, содержащем журнальный логический том. Этот параметр влияет на производительность только в том случае, если при обработке запросов на ввода-вывод изменяется протокол журналируемой файловой системы (JFS).

    Такой подход позволяет разделить операции ввода-вывода, заносимые в журнал, и операции по работе с данными, повышая возможность параллельной работы. Применение такого подхода дает особенно ощутимый эффект на сервере NFS, так как на нем запись считается выполненной только тогда, когда записаны как сами данные, так и информация журнала.
- При первой возможности определяйте или расширяйте логические тома до максимального предполагаемого размера. В первую очередь рекомендуется расширять логические тома, производительность которых особенно важна, так как это позволит разместить их данные в смежных сегментах диска.

- Логические тома с высокой интенсивностью использования рекомендуется размещать на нескольких накопителях. Если параметр **Диапазон физических томов** в меню SMIT Добавить физический том (**smitty mklv**) равен **maximum**, то новый логический том будет распределен между физическими томами, входящими в группу томов (или между явно заданными физическими томами).
- Если в системе установлены накопители разных типов (или вы хотите упорядочить накопители), то примите во внимание следующее:
  - Большие файлы с последовательным доступом рекомендуется размещать на диске, обращения к которому обрабатываются быстрее всего.
  - Если к таким файлам часто обращаются для выполнения операций ввода-вывода, то рекомендуется сократить число накопителей, подключенных к одному адаптеру.
  - Диски, обмен с которыми всегда должен выполняться быстро, рекомендуется подключать к высокоскоростному адаптеру. Такие адаптеры поддерживают ряд функций, отсутствующие в других адаптерах.
  - В случае небольших дисков логические тома, предназначенные для хранения больших файлов с высокой интенсивностью обращений, должны располагаться ближе к внешнему краю. В этих областях на дорожке размещается больше блоков, поэтому скорость обработки последовательных операций возрастает.
  - На шине, поддерживающей исходный стандарт SCSI, диски с самым большим номером (т.е. с самым большим адресом SCSI) имеют самый высокий приоритет. Все последующие стандарты обычно совместимы с исходным. Номера в порядке убывания приоритета упорядочены следующим образом: 7-6-5-4-3-2-1-0-15-14-13-12-11-10-9-8.

    В большинстве ситуаций это незаметно, но блочные операции с файлами большого объема могут привести к тому, что диски с маленькими номерами не получат доступа к шине. Дискам, содержащим данные, время чтения и записи которых должно быть минимальным, нужно присваивать старшие адреса SCSI.

    Информацию о текущих адресах устройств, подключенных к шине SCSI, можно узнать с помощью команды **lsdev -Cs scsi**. Для стандартного адаптера SCSI адрес устройства на шине SCSI указывается в четвертой паре цифр. В приведенном ниже примере указано, что первый диск объемом 400 ГБ расположен по адресу 4, второй диск - по адресу 5, накопитель на 8-мм магнитной ленте - по адресу 1, а дисковод CD-ROM - по адресу 3.

    ```
    cd0    Доступно  10-80-00-3,0 Дисковод CD-ROM SCSI
    hdisk0 Доступно  10-80-00-4,0 16-разрядный диск SCSI
    hdisk1 Доступно  10-80-00-5,0 16-разрядный диск SCSI
    rmt0   Доступно  10-80-00-1,0  Накоп. на магн. ленте 8-мм 2,3 ГБ
    ```
  - Большие файлы с произвольным доступом, такие как файлы баз данных, должны располагаться на нескольких физических томах.

Понятия, связанные с данным:

“Производительность логических томов и дискового ввода-вывода” на стр. 178
В этой главе основное внимание уделено производительности логических томов и локальных дисков.

Расположение и размер пространств подкачки:

Общая рекомендация заключается в том, чтобы сумма размеров пространств подкачки была не меньше удвоенного объема оперативной памяти компьютера, если размер памяти не превышает 256 МБ (размер пространства подкачки - 512 МБ).

Примечание: Для объемов памяти, превышающих 256 МБ, рекомендуется следующая формула:

```
общий размер пространства подкачки = 512 МБ + (объем оперативной памяти - 256 МБ) * 1,25
```

Однако поскольку применяется алгоритм выделения пространства подкачки с задержкой, значение, рассчитанное по этой формуле, может оказаться несколько больше, чем требуется. Дополнительная информация приведена в разделе “Выделение пространства подкачки” на стр. 157.

В идеальном случае в системе должно быть несколько областей подкачки равных размеров, расположенных на разных физических дисках. Дополнительные пространства подкачки рекомендуется создавать на тех физических томах, которые загружены меньше, чем rootvg. При выделении блоков области подкачки VMM берет по 4 блока из каждой области, в которой есть свободное пространство. Во время загрузки системы активна только основная область подкачки (hd6). Поэтому во время загрузки блоки пространства подкачки выделяются только в основной области. Из этого следует, что основная область подкачки должна быть больше вспомогательных областей. Для эффективной работы алгоритма поочередного выделения блоков размер дополнительных пространств подкачки должен совпадать.

Команда **lsps -a** показывает информацию об использовании всех пространств подкачки системы. Для того чтобы узнать, насколько пространство подкачки близко к заполнению, можно вызвать функцию **psdanger()**. Ниже приведен пример использования функции **psdanger()** в программе, выводящей предупреждающее сообщение при достижении порогового значения:

```
/* psmonitor.c
  Отслеживает ситуации, в которых пространство подкачки близко к
  заполнению. Если такая ситуация
  обнаружена, в stderr выводится
  сообщение.
  Формат:        psmonitor [Интервал_времени [Число_интервалов]]
  По умолчанию:  psmonitor 1 1000000
*/
#include <stdio.h>
#include <signal.h>
main(int argc,char **argv)
{
  int interval = 1;        /* интервал в секундах */
  int count = 1000000;     /* число интервалов */
  int current;             /* текущий интервал */
  int last;                /* проверка последнего цикла */
  int kill_offset;         /* возвращается psdanger() */
  int danger_offset;       /* возвращается psdanger() */


  /* проверка параметров командной строки */
  if (argc > 1) {
    if ( (interval = atoi(argv[1])) < 1 ) {
      fprintf(stderr,"Формат: psmonitor [ Интервал_времени [ Число_интервалов ] ]\n");
      exit(1);
    }
    if (argc > 2) {
      if ( (count = atoi( argv[2])) < 1 ) {
         fprintf(stderr,"Формат: psmonitor [ Интервал_времени [ Число_интервалов ] ]\n");
         exit(1);
      }
    }
  }
  last = count -1;
  for(current = 0; current < count; current++) {
    kill_offset = psdanger(SIGKILL); /* проверка отсутствия свободного пространства подкачки */
    if (kill_offset < 0)
      fprintf(stderr,
        "НЕХВАТКА ПРОСТРАНСТВА ПОДКАЧКИ! %d блоков до порога SIGKILL.\n",
        kill_offset*(-1));
    else {
      danger_offset = psdanger(SIGDANGER); /* проверка нехватки свободного пространства подкачки */
      if (danger_offset < 0) {
        fprintf(stderr,
          "ПРЕДУПРЕЖДЕНИЕ: Недостаточно пространства подкачки. %d блоков до SIGDANGER.\n",
          danger_offset*(-1));
        fprintf(stderr,
          "                           %d блоков до порога SIGKILL.\n",
          kill_offset);
      }
    }
```

```
      if (current < last)
        sleep(interval);
  }
}
```

Влияние зеркальной защиты дисков на производительность:

С точки зрения производительности, зеркальная защита создает значительную нагрузку на систему, а зеркальная защита с проверкой записи снижает скорость работы еще сильнее (при каждой записи выполняется повторное обращение к секторам). Следовательно, зеркальная защита с проверкой записи и контролем целостности замедляет работу системы еще больше (повторное обращение к секторам и позиционирование на нулевую дорожку).

Если включена зеркальная защита с функцией контроля целостности (которая включена по умолчанию), то рекомендуется размещать зеркальные копии на внешнем крае диска, так как информация о целостности зеркальных копий всегда размещается в цилиндре 0. С фискальной точки зрения, больших затрат требует только зеркальная защита с записью. Несмотря на то, что команда **lslv** сообщает о включении проверки целостности для логических томов без зеркальной защиты, никаких действий не выполняется, если значение COPIES равно единице. Проверка записи по умолчанию выключена, так как эта функция не имеет смысла (и не влияет на производительность) для логических томов без зеркальной защиты.

Доступен дополнительный режим обеспечения целостности при записи на накопители с зеркальной защитой. Этот режим называется пассивной целостностью зеркальных дисков (пассивной MWC). По умолчанию применяется активная MWC. В режиме активной целостности восстановление данных происходит с минимальными затратами времени при первой загрузке системы после сбоя. Однако такой подход приводит к снижению производительности дисковой подсистемы при записи, особенно при работе с большим количеством мелких файлов. В режиме пассивной MWC снижения производительности нет, но при перезагрузке после сбоя нужно полностью синхронизировать поврежденную группу томов с помощью команды **syncvg -f**. Для этого требуется отключить автоматическое включение групп томов.

Пассивная MWC выигрывает не только за счет отсутствия снижения производительности, но и за счет автоматической синхронизации логических томов по мере обращения к разделам дисковой памяти. Администратору не требуется вручную синхронизировать логические тома и отключать режим автоматического включения групп томов. Недостаток пассивной MWC заключается в снижении скорости операций чтения вплоть до окончания синхронизации всех разделов.

Нужную опцию обеспечения целостности при записи на диски с зеркальной защитой можно указать в SMIT при создании и изменении логических томов. Данная опция имеет смысл только для логических томов с зеркальной защитой.

Влияние зеркальных томов с чередованием данных на производительность:

Создание таких томов позволяет программным образом объединить большую эффективность доступа к данным, свойственную массиву RAID 1, с высокой производительностью массива RAID 0.

Зеркальные тома можно создавать с чередованием данных. Группы томов, содержащие зеркальные тома с чередованием данных, нельзя импортировать в операционные системы AIX.

Рекомендации перед установкой устройств связи

Информация о правильном размещении адаптеров, а также различные рекомендации, связанные с настройкой производительности, приведены в разделе *Справочник по установке адаптеров PCI*.

См. инструкции по настройке, описанные в разделах “Настройка производительности TCP и UDP” на стр. 258 и “Настройка производительности пула mbuf” на стр. 292.

Системы на базе POWER4

Системы на базе POWER4 имеют несколько ограничений, связанных с производительностью.

Связанная информация: “Производительность файловой системы” на стр. 235, “Управление ресурсами” на стр. 39 и публикацию IBM Redbooks *The POWER4 Processor Introduction and Tuning Guide*.

Повышение производительности POWER4

В процессорах POWER4 были выполнены несколько оптимизаций, повышающих производительность.

- Процессоры оптимизированы для работы в симметричных многопроцессорных системах (SMP) и позволяют добиться более высокой степени распараллеливания инструкций.
- Улучшены алгоритмы предсказания инструкций и упреждающей выборки данных.
- По сравнению с процессорами POWER3 повышена пропускная способность внутренней памяти и достигнуты более высокие тактовые частоты.

Сравнение микропроцессоров

В следующей таблице приведены основные сравнительные характеристики процессоров IBM.

Таблица 1. Характеристики процессоров

	POWER3	RS64	POWER4
Тактовая частота	450 МГц	750 МГц	> 1 ГГц
Число блоков для операций с фиксированной точкой	3	2	2
Число блоков для операций с плавающей точкой	2	1	2
Число блоков чтения и записи данных	2	1	2
Число блоков ветвления и прочих блоков	1	1	2
Ширина шины диспетчера	4	4	5
Алгоритм предсказывания	Динамический	Статический	Динамический
Размер кэша инструкций	32 КБ	128 КБ	64 КБ
Размер кэша данных	128 КБ	128 КБ	32 КБ
Размер кэша второго уровня	1, 4, 8 МБ	2, 4, 8, 16 МБ	1.44
Размер кэша третьего уровня	Нет	Нет	Зависит от числа процессоров
Возможность упреждающей выборки данных	Да	Нет	Да

Масштабируемость систем на базе POWER4

Для систем на базе процессоров POWER4 предусмотрены расширенные средства масштабирования, ощутимо повышающие производительность и снижающие общую нагрузку на системы.

Термином *масштабируемость нагрузки* обозначается способность системы реагировать на увеличение нагрузки. Термином *масштабируемость производительности* обозначается способность системы сохранять высокую производительность при повышении нагрузки за счет выделения дополнительных ресурсов.

Ниже перечислены основные изменения, относящиеся к масштабируемости.

Фиксация общей памяти для работы с базами данных

В AIX предусмотрена возможность запрета вытеснения в пространство подкачки определенных страниц памяти. Это называется *фиксацией памяти*.

Зафиксированные области памяти не могут вытесняться в пространство подкачки.

Поддержка больших объемов памяти

Максимальный объем физической памяти в системе с 64-разрядным ядром зависит от аппаратных систем.

Данное значение обусловлено ограничениями некоторых аппаратных ресурсов. В 64-разрядном ядре нет ограничений на минимальный объем пространства подкачки.

64-разрядное ядро

В данной системе AIX применяется 64-разрядное ядро, и это позволяет обойти большинство "узких мест", которые были свойственны системам с 32-разрядным ядром.

Начиная cAIX 7.1, 64-разрядные ядра - единственно доступные. Системы на базе процессоров POWER4 оптимизированы для работы с 64-разрядным ядром в целях улучшения масштабируемости систем RS/6000 System p. В свою очередь, операционная система AIX оптимизирована для выполнения 64-разрядных приложений в системах POWER4. Еще одно важное улучшение в 64-разрядном ядре (с точки зрения масштабируемости) заключается в увеличении максимального объема физической памяти.

Также, JFS2 - это файловая система по умолчанию для AIX 7.1. Можно использовать JFS или JFS2. Дополнительная информация о расширенной JFS приведена в разделе Быстродействие файловой системы.

64-разрядные приложения в 32-разрядном ядре

В системах POWER4 с 64-разрядным ядром 64-разрядные приложения должны выполняться быстрее или по крайней мере с такой же скоростью, чем в таких же системах с 32-разрядным ядром.

Это связано с тем, что 64-разрядному ядру для выполнения 64-разрядных приложений не требуется выполнять преобразование параметров системных вызовов. Кроме того, 64-разрядные приложения создаются с помощью усовершенствованного компилятора, оптимизированного для систем POWER4.

32-разрядные приложения в 64-разрядном ядре

Большинство 32-разрядных приложений могут выполняться в 64-разрядном ядре без снижения производительности.

Скорость выполнения 32-разрядных вызовов в 64-разрядном ядре несколько снижается за счет необходимости преобразования параметров. Как правило, скорость выполнения снижается не более чем на 5%. Примером 32-разрядного вызова, медленно выполняющегося в 64-разрядном ядре, может служить функция **fork()**.

64-разрядные приложения в 64-разрядном ядре в системах, отличных от POWER4

В системах с процессором, отличным от POWER4, скорость выполнения 64-разрядных приложений в 64-разрядном ядре может быть ниже, чем в 32-разрядном ядре.

Системы с процессорами, отличными от POWER4, рассматриваются как промежуточная стадия при переходе к системам POWER4, и в них отсутствует ряд механизмов, необходимых для достижения оптимальной производительности 64-разрядного ядра.

64-разрядные расширения ядра в системах, отличных от POWER4.

Производительность расширений 64-разрядного ядра в системах POWER4 должна быть не ниже, чем производительность аналогичных расширений 32-разрядного ядра.

Однако производительность расширений 64-разрядного ядра в системах с процессорами, отличными от POWER4, может быть ниже, чем производительность расширений 32-разрядного ядра в таких системах. Это связано с тем, что системы с процессорами, отличными от POWER4, не оптимизированы для работы с 64-разрядным ядром.

Расширенная журналируемая файловая система

Расширенная JFS, или JFS2, - это еще одна внутренняя файловая система AIX. Это - файловая система по умолчанию для AIX 6.1 и более поздних версий.

Дополнительная информация о расширенной JFS приведена в разделе “Производительность файловой системы” на стр. 235.

Производительность процессора

В этом разделе рассмотрены способы выявления зациклившихся программ и программ, интенсивно использующих процессор, а также способы снижения их негативного влияния на общую производительность системы.

Если вы не знакомы с принципами планирования ресурсов процессора, то сначала ознакомьтесь с разделом “Производительность планировщика процессора” на стр. 39.

Отслеживание производительности микропроцессора

Процессор - это один из самых быстрых компонентов системы.

Одна программа редко способна полностью загрузить процессор (когда процессор не совсем не простаивает и не находится в состоянии ожидания) больше чем на несколько секунд. Даже в многопользовательских системах с большой нагрузкой иногда возникают периоды длительностью 10 миллисекунд, когда все нити находятся в состоянии ожидания. Если в выводе монитора указано, что процессор загружен на 100 процентов в течение длительного времени, то, скорее всего, одна из программ зациклилась. Даже если программа не содержит ошибку, а просто потребляет много процессорного времени, необходимо изменить алгоритм ее работы.

Команда vmstat

В первую очередь нужно вызвать команду **vmstat**, которая предоставляет краткую информацию о различных ресурсах системы и связанных с ними неполадках, приводящих к снижению производительности.

Отчет команды **vmstat** содержит статистическую информацию о нитях ядра, находящихся в очереди выполнения и ожидающих запуска, а также об оперативной памяти, подкачке, дисках, прерываниях, системных вызовах, переключении контекста и работе процессора. В информации о процессоре указывается доля времени, в течение которой процессор работал в пользовательском режиме, системном режиме, простаивал и ожидал завершения операции дискового ввода-вывода.

Примечание: Если в команде **vmstat** не указан интервал, выводится один отчет. Этот отчет является содержит усредненные значения за период с момента запуска системы. Можно указать только параметры Count и Interval. Если параметр Interval указан без параметра Count, отчеты будут генерироваться непрерывно.

Для получения информации о работе процессора удобнее использовать команду **vmstat**, а не **iostat**, так как ее построчный вывод проще анализировать, а в случае, когда к системе подключено много дисков, ее вывод намного менее объемный. Приведенный ниже пример иллюстрирует применение vmstat для обнаружения программ, которые потребляют слишком много ресурсов CPU.

```
# vmstat 2
нити     память             страница          ошибки        cpu
----- ----------- ------------------------ ------------ -----------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
 1  0 22478  1677   0   0   0   0    0   0 188 1380 157 57 32  0 10
 1  0 22506  1609   0   0   0   0    0   0 214 1476 186 48 37  0 16
 0  0 22498  1582   0   0   0   0    0   0 248 1470 226 55 36  0  9

 2  0 22534  1465   0   0   0   0    0   0 238  903 239 77 23  0  0
 2  0 22534  1445   0   0   0   0    0   0 209 1142 205 72 28  0  0
```

```
 2  0 22534  1426   0   0   0   0    0   0 189 1220 212 74 26  0  0
 3  0 22534  1410   0   0   0   0    0   0 255 1704 268 70 30  0  0
 2  1 22557  1365   0   0   0   0    0   0 383  977 216 72 28  0  0

 2  0 22541  1356   0   0   0   0    0   0 237 1418 209 63 33  0  4
 1  0 22524  1350   0   0   0   0    0   0 241 1348 179 52 32  0 16
 1  0 22546  1293   0   0   0   0    0   0 217 1473 180 51 35  0 14
```

Этот пример показывает результаты запуска зацикленной программы в многопользовательской системе с большой нагрузкой. Первые три отчета (итоговый отчет был удален) показывают, что система была загружена на 50-55 процентов пользовательскими процессами, на 30-35 процентов - задачами системы, а 10-15 процентов пришлось на задержки ввода-вывода. После запуска зацикленной программы она забирает все свободные ресурсы CPU. Так как зацикленная программа не запрашивает операции ввода-вывода, она может получить все процессорное время, которое не используется другими программами, ожидающими завершения операций ввода-вывода. Кроме этого, такая программа всегда готова забрать ресурсы CPU, когда другой процесс их освобождает. При этом она не обязательно отдаст эти ресурсы другому процессу, так как приоритеты всех фоновых процессов одинаковы. Программа работает примерно 10 секунд (группа из пяти отчетов), после чего система возвращается в обычный режим работы (об этом говорят следующие отчеты команды **vmstat**).

Оптимальной является ситуация, когда процессор работает 100 процентов времени. Это условие всегда выполняется в однопользовательской системе, если в ней нет конкуренции за процессор. В общем случае, если суммарное время `us` + `sy` меньше 90 процентов, то считается, что производительность однопользовательской системы не ограничена ресурсами CPU. Однако если значение `us` + `sy` в многопользовательской системе превышает 80 процентов, то некоторые процессы часть времени находятся в состоянии ожидания. Это может отрицательно сказаться на времени ответа и скорости работы приложений.

Для того чтобы узнать, служат ли ресурсы процессора причиной снижения производительности системы, изучите значения, указанные в четырех столбцах `cpu` и двух столбцах `нить` (нить ядра) вывода команды **vmstat**. Кроме того, обратите внимание на столбец `ошибки`:

- **cpu**

  Распределение времени CPU в течение интервала времени. В столбцах `cpu` указываются следующие значения:

  – **us**

    В столбце `us` указывается, какая доля времени CPU (в процентах) была затрачена на работу в пользовательском режиме. Процессы UNIX могут выполняться в пользовательском или системном режиме (режиме ядра). При работе в пользовательском режиме процесс выполняется в рамках приложения. Ему не требуются ресурсы ядра для выполнения вычислений, управления памятью и настройки переменных.

  – **sy**

    В столбце `sy` указывается, какая доля времени процессора была затрачена на выполнение процессов в системном режиме. Это значение отражает ресурсы CPU, которые были затрачены на выполнение процессов ядра (**kprocs**) и других процессов, использующих ресурсы ядра. Для получения доступа к ресурсам ядра процесс отправляет системный вызов, в результате чего он переключается в системный режим. Например, при выполнении операции чтения или записи данных в файл ресурсы ядра применяются для открытия файла, смещения указателя в нужную позицию и последующего чтения или записи данных (если файл еще не загружен в оперативную память).

  – **id**

    В столбце `id` указывается, какую долю времени (в процентах) процессор простаивал, не ожидая завершения локальной операции дискового ввода-вывода. Если ни одна нить не готова к работе (очередь выполнения пуста), то система запускает специальную нить **wait**, иначе называемую **idle kproc**. В системе SMP нить **wait** может быть запущена на каждом процессоре. The report generated by the **ps** command (with the **-k** or **-g 0**) этот процесс указывается как `kproc` или `wait`. Если в отчете **ps** указано, что на эту нить в целом затрачивается много времени, то это означает, что существуют большие

промежутки времени, в течение которых ни одна нить не готова к работе и не ожидает запуска. Следовательно, система преимущественно *простаивает*, *ожидая* появления новых задач.

– **wa**

В столбце wa указывается доля времени, в течение которой процессор *простаивал*, ожидая завершения операции ввода-вывода на локальный диск и на диски NFS. Если во время выполнения процесса **wait** в системе есть хотя бы одна ожидающая выполнения операция ввода-вывода, то это время рассматривается как время ожидания завершения операции ввода-вывода. За исключением случаев, когда операции ввода-вывода выполняются асинхронно с процессом, процесс блокируется на время выполнения запросов на обмен данными с диском. После выполнения запроса на ввод-вывод процесс снова помещается в очередь выполнения. Чем быстрее выполняется обмен данными с диском, тем большая часть времени CPU будет тратиться на выполнение процессов.

Если значение wa больше 25 процентов, то можно сделать вывод, что дисковая подсистема плохо сбалансирована, либо в системе выполняется очень много дисковых операций ввода-вывода.

Информация о change+ приведена в разделе “Получение информации о времени ожидания ввода-вывода” на стр. 179.

- **kthr**

Число нитей ядра в различных очередях, усредненное по секундам интервала опроса. В столбцах нити указывается следующее:

– **r**

Среднее число активных нитей ядра, включающее число выполняемых нитей и число нитей, ожидающих освобождения процессора. Если данное число превышает количество процессоров, по крайней мере одна нить ожидает освобождения процессора; чем больше таких нитей, тем больше снижается производительность.

– **b**

Среднее число нитей ядра, которые каждую секунду находятся в очереди ожидания VMM. В это число входят нити, ожидающие ввода-вывода в файловую систему, а также нити, выполнение которых приостановлено из-за перегрузки памяти памяти.

Если процессы в системе приостанавливаются из-за перегрузки памяти, то прирост нитей отражает именно значение в столбце b отчета **vmstat**, а не число нитей в очереди выполнения.

– **p**

В выводе команды **vmstat -I** это значение показывает число нитей, ожидающих выполнение ввода-вывода на устройство с линейным доступом, каждую секунду. Нити, ожидающие выполнение ввода-вывода в файловую систему, сюда не включены.

- **faults**

Здесь указывается информация об управлении процессами, в том числе число прерываний. В столбцах ошибки содержатся следующие значения:

– **in**

Число прерываний, поступавших от устройств каждую секунду в течение интервала сбора информации. См. раздел “Оценка производительности дисков командой vmstat” на стр. 182.

– **sy**

Число системных вызовов, отправлявшихся каждую секунду в течение интервала сбора информации. Пользовательские процессы получают доступ к ресурсам путем отправки стандартных системных вызовов. Эти вызовы отправляют ядру запрос на выполнение необходимой операции, а также служат для обмена данными между ядром и процессом. Допустимое число системных вызовов в секунду зависит от особенностей конкретной рабочей схемы и ее приложений, а также функций, выполняемых системными вызовами. Однако если значение sy достигает 10000 вызовов в однопроцессорной системе (в системе SMP - 10000 вызовов в секунду для каждого процессора), то рекомендуется провести дальнейшее исследование ситуации. Одной из причин увеличения числа системных вызовов может быть запуск опрашивающей процедуры, например, процедуры **select()**. Рекомендуется измерить значение sy при выполнении базовых операций в системе, и в дальнейшем применять его для сравнения.

– **cs**

Число переключений контекста, выполнявшихся каждую секунду в течение интервала сбора информации. Все время работы процессора разделяется на логические кванты времени по 10 миллисекунд. Любая нить выполняется до тех пор, пока не истечет квант времени, пока она не будет замещена нитью с более высоким приоритетом, либо пока она добровольно не передаст управление другой нити. Когда управление передается другой нити, контекст, или рабочая среда, предыдущей нити сохраняется, а вместо него загружается контекст текущей нити. В операционной системе предусмотрена очень эффективная процедура переключения контекста, поэтому на переключение контекста тратится лишь незначительная часть ресурсов. Однако при значительном росте числа переключений контекста, например, когда значение `cs` становится намного больше скорости дискового ввода-вывода и скорости передачи пакетов по сети, необходимо дополнительно проанализировать ситуацию.

Команда iostat

Команда **iostat** позволяет быстро обнаружить неполадки дискового ввода-вывода, приводящие к снижению производительности.

Обратитесь к разделу “Оценка производительности дисков командой iostat” на стр. 179. С ее помощью можно также получить статистическую информацию об использовании процессора.

В приведенном ниже примере показан фрагмент вывода команды **iostat**. В первом разделе указываются суммарные значения параметров, накопленные с момента запуска системы.

```
# iostat -t 2 6
tty:      tin         tout   avg-cpu:  % user    % sys     % idle    %iowait
          0.0          0.8               8.4      2.6       88.5       0.5
          0.0         80.2               4.5      3.0       92.1       0.5
          0.0         40.5               7.0      4.0       89.0       0.0
          0.0         40.5               9.0      2.5       88.5       0.0
          0.0         40.5               7.5      1.0       91.5       0.0
          0.0         40.5              10.0      3.5       80.5       6.0
```

В столбцах с информацией о процессоре (*% user*, *% sys*, *% idle* и *% iowait*) указывается распределение времени процессора. В выводе команды **vmstat** эта информация выводится в столбцах *us*, *sy*, *id* и *wa*. Дополнительная информация приведена в разделе “Команда vmstat” на стр. 104. Кроме того, ознакомьтесь с информацией о новом способе вычисления значения *%iowait*, приведенной в разделе “Получение информации о времени ожидания ввода-вывода” на стр. 179.

Задачи, связанные с данной:

“Оценка производительности дисков командой iostat” на стр. 179
Начните оценку с запуска команды **iostat** во время максимальной нагрузки на систему или работы важного приложения, задержки ввода-вывода для которого необходимо свести к минимуму.

Ссылки, связанные с данной:

“Непрерывное отслеживание производительности системы командой iostat” на стр. 14
Команда **iostat** позволяет получить информацию об использовании дисков и CPU.

Команда sar

Команда **sar** собирает статистическую информацию о работе системы.

Хотя она позволяет получить полезную информацию о производительности, ее выполнение создает значительную нагрузку на систему, что может привести к дальнейшему снижению производительности. Это особенно важно учитывать, если команда **sar** создает отчеты через короткие промежутки времени. Тем не менее, команда **sar** удобнее, чем пакет программ для учета ресурсов. В системе предусмотрен ряд счетчиков для различных операций, выполняемых в системе. Команда **sar** создает отчеты на основе показаний этих счетчиков. Значения этих счетчиков обновляются автоматически, поэтому их обновление никак не связано с запуском команды **sar**. Эта команда просто считывает и сохраняет показания счетчиков, причем число измерений и их периодичность задаются при запуске команды **sar**.

В команде **sar** предусмотрено большое число опций, позволяющих получить информацию об очередях, подкачке, терминалах и других устройствах. Одной из важных особенностей команды **sar** является то, что

она может создавать как отчеты с информацией об использовании всех процессоров системы в целом (такие отчеты содержат средние значения в процентах и суммарные значения), так и отчеты с информацией по каждому процессору в отдельности. Таким образом, эта команда будет особенно полезна в системах SMP.

Существует три способа применения команды **sar**:

Сбор и просмотр информации в реальном времени:

Для того чтобы запустить команду, которая собирает статистическую информацию о работе системы и сразу же выводит ее на экран, воспользуйтесь утилитой **sar**.

Выполните следующую команду:

```
# sar -u 2 5

AIX ses12 1 6 000126C5D600    04/08/08

Конфигурация системы: lcpu=2 mode=Capped

19:42:43    %usr    %sys    %wio   %idle   physc
19:42:45       0       2       1      97    0.98
19:42:47       0       0       0     100    1.02
19:42:49       0       0       0     100    1.00
19:42:51       0       0       0     100    1.00
19:42:53       0       0       0     100    1.00

Среднее        0       1       0      99    1.00
```

Выше приведен пример вывода команды, запущенной на однопользовательской рабочей станции для сбора информации об использовании CPU.

Показ ранее собранных данных:

С помощью опций **-o** и **-f** вы можете разбить работу команды на два этапа: сначала записать собранные данные в пользовательский файл данных, а затем просмотреть данные из этого файла. Это позволяет сэкономить ресурсы, когда команда запускается в системе, искусственно переведенной в то состояние, в котором возникла неполадка.

Полученные данные можно проанализировать на другом компьютере, перенеся в него двоичный файл вывода, так как он содержит всю информацию, необходимую для работы команды **sar**.

```
# sar -o /tmp/sar.out 2 5 > /dev/null
```

Приведенная выше команда запускает утилиту **sar** в фоновом режиме, собирает данные о работе системы пять раз с периодичностью в 2 секунды и сохраняет полученный неотформатированный вывод **sar** в файле `/tmp/sar.out`. Вывод перенаправляется в файл для того, чтобы он не отображался на экране.

Приведенная ниже команда считывает информацию об использовании CPU из созданного ранее файла и выводит отформатированный отчет на стандартное устройство вывода:

```
# sar -f/tmp/sar.out

AIX ses12 1 6 000126C5D600    04/08/08

Конфигурация системы: lcpu=2 mode=Capped

20:17:00    %usr    %sys    %wio   %idle   physc
20:18:00       0       1       0      99    1.00
20:19:00       0       1       0      99    1.00
20:20:00       0       1       0      99    1.00
```

```
20:21:01       0       1       0      99    1.00
20:22:00       0       0       0      99    1.00

Среднее        0       1       0      99    1.00
```

Собранный двоичный файл данных содержит всю информацию, необходимую для создания отчетов. С его помощью можно получить любой отчет команды **sar**. За счет этого вы можете просмотреть информацию об использовании процессоров системы SMP в однопроцессорной системе.

Учет ресурсов системы с помощью демона cron:

Демон **cron** запускает два сценария оболочки (/usr/lib/sa/sa1 и /usr/lib/sa/sa2), которые каждый день создают отчеты и выдают статистическую информацию.

Для доступа к системным данным команда **sar** вызывает процесс **sadc**. Файл crontab /var/spool/cron/crontabs/adm содержит разделы, которые изначально помещены в комментарий. В этих разделах указывается, когда демон **cron** должен запускать сценарии оболочки.

Ниже приведен вариант файла crontab для пользователя adm. Он отличается от исходного файла только тем, что в нем удалены символы комментария для интервалов сбора данных:

```
#==================================================================
#  Отчеты о работе системы создаются:
#  В рабочие дни с 8 утра до 5 вечера - каждые 20 минут;
#  В субботу и воскресенье - каждый час;
#  Каждый день с 6 вечера до 7 утра - каждый час.
#  Ежедневный отчет создается в 18:05.
#==================================================================
0 8-17 * * 1-5 /usr/lib/sa/sa1 1200 3 &
0 * * * 0,6 /usr/lib/sa/sa1 &
0 18-7 * * 1-5 /usr/lib/sa/sa1 &
5 18 * * 1-5 /usr/lib/sa/sa2 -s 8:00 -e 18:01 -i 3600 -ubcwyaqvm &
#==================================================================
```

Такой способ сбора информации позволяет получить представление о том, насколько интенсивно система используется в течение определенного периода времени, и определить часы пиковой нагрузки.

Полезные опции процессора:

Существует большое количество полезных опций команды **sar**, относящихся к процессору.

Наиболее используемые опции:

- **sar -P**

  Опция **-P** позволяет получить статистическую информацию об указанных процессорах. Если будет указано ключевое слово **ALL**, то команда выдаст информацию по каждому процессору в отдельности и средние значения по всем процессорам. Если опция **-P ALL** применяется в среде раздела рабочей схемы, то помимо статистики уровня системы отображается статистика RSET; процессоры, входящие в состав RSET, обозначаются с помощью префикса (*). Статистика RSET отображается только в том случае, если среда раздела рабочей схемы связана с RSET. С флагом **-P** можно указывать только флаги **-a**, **-c**, **-m**, **-u** и **-w**.

  Ниже приведен пример вывода команды, запущенной в системе, в которой процессор 0 связан с определенным процессом:

  ```
  # sar -P ALL 2 2

  AIX tooltime2 1 6 00CA52594C00    04/02/08
  Конфигурация системы: lcpu=4 mode=Capped
  05:23:08 cpu    %usr    %sys    %wio   %idle   physc
  05:23:11  0        8      92       0       0    1.00
            1        0      51       0      49    0.00
            2        0       0       0     100    0.51
            3        0       0       0     100    0.48
  ```

```
          -        4      46       0      50    1.99
05:23:13  0       10      89       0       0    1.00
          1        0       7       0      93    0.00
          2        0       3       2      95    0.51
          3        0       0       0     100    0.49
          -        5      45       0      49    2.00

Среднее   0        9      91       0       0    1.00
          1        0      12       0      88    0.00
          2        0       2       1      98    0.51
          3        0       0       0     100    0.48
          -        5      46       0      49    1.99
```

В поле cpu последней строки каждого раздела, начинающейся с дефиса (-), указано среднее значение по всем процессорам. Строка средних значений (-) выводится только в том случае, если задана опция **-P ALL**. Если в команде указан конкретный процессор, то эта строка удаляется. Последний раздел, в начале которого вместо системного времени указано слово Average, содержит средние значения параметров отдельных процессоров, вычисленные исходя из значений этих параметров во всех предыдущих разделах.

Ниже приведен пример вывода команды **vmstat**, полученного за тот же период времени:

```
Конфигурация системы:  lcpu=4 mem=44570MB
нити     память             страница          ошибки        cpu
----- ----------- ---------------------- ------------ -----------
 r  b   avm   fr       re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
 2  0 860494 6020610   0   0   0   0    0   0  16 14061 409  5 45 49  0
 2  0 860564 6020540   0   0   0   0    0   0   4 14125 400  5 45 50  0
 1  0 860669 6020435   0   0   0   0    0   0   3 14042 388  5 46 49  0
 2  0 860769 6020335   0   0   0   0    0   0   3 13912 398  5 45 50  0
```

В первой строке указаны суммарные значения, накопленные с момента запуска системы. Вторая строка отражает запуск команды **sar**, а все последующие строки сравнимы с отчетом команды sar. Команда **vmstat** выводит только средние значения использования всех процессоров. Эти значения аналогичны значениям, указанным в строках отчета команды **sar**, помеченных дефисами (-).

В случае выполнения за пределами среды WPAR эта команда возвращает следующий вывод:

```
AIX wpar1 1 6 00CBA6FE4C00    04/01/08
Конфигурация wpar1: lcpu=2  memlim=204MB  cpulim=0.06  rset=Regular
05:23:08 cpu    %usr    %sys    %wio   %idle   physc
05:23:11 *0        8      92       0       0    1.00
         *1        0      51       0      49    0.00
          2        0       0       0     100    0.51
          3        0       0       0     100    0.48
          R        4      71       0      24    1.00
          -        4      46       0      50    1.99
05:23:13 *0       10      89       0       0    1.00
         *1        0       7       0      93    0.00
          2        0       3       2      95    0.51
          3        0       0       0     100    0.49
          R        5      48       0      46    1.00
          -        5      45       0      49    2.00

Среднее   0        9      91       0       0    1.00
         *1        0      12       0      88    0.00
          2        0       2       1      98    0.51
          3        0       0       0     100    0.48
          R        4      51       0      44    1.00
          -        5      46       0      49    1.99
```

WPAR со связанным реестром RSET. Процессоры 0 и 1 подключены к RSET. Строка R содержит уровень использования RSET, связанного с WPAR. Процессоры из RSET обозначаются префиксом (*).

- **sar –P** RST - отображает показатели использования процессоров из RSET. Если среда WPAR не связана с RSET, то отображаются все показатели процессов.

  Пример выполнения команды **sar –P** RST в среде WPAR:

  ```
  AIX wpar1 1 6 00CBA6FE4C00    04/01/08

  Конфигурация wpar1: lcpu=2  memlim=204MB  cpulim=0.06  rset=Regular
  ```

```
05:02:57 cpu    %usr    %sys    %wio   %idle   physc
05:02:59  0       20      80       0       0    1.00
          1       10       0       0      90    0.00
          R       15      40       0      45    1.00
05:03:01  0       20      80       0       0    1.00
          1        8       0       0      92    0.00
          R       14      40       0      46    1.00

Среднее   0        20      80       0       0    1.00
          1        9       0       0      91    0.00
          R       14      40       0      46    1.00
```

- **sar -u**

  Эта команда выводит информацию об использовании процессора. Такое действие выполняется по умолчанию, если не заданы другие флаги. Вывод этой команды совпадает со статистической информацией о работе процессора, которая выдается командами **vmstat** и **iostat**.

  В приведенном ниже примере во время работы команды была запущена команда копирования:

```
# sar -u 3 3
AIX wpar1 1 6 00CBA6FE4C00    04/01/08
Конфигурация wpar1: lcpu=2  memlim=204MB  cpulim=0.06  rset=Regular
05:02:57 cpu    %usr    %sys    %wio   %idle   physc
05:02:59  0       20      80       0       0    1.00
          1       10       0       0      90    0.00
          R       15      40       0      45    1.00
05:03:01  0       20      80       0       0    1.00
          1        8       0       0      92    0.00
          R       14      40       0      46    1.00

Среднее   0        20      80       0       0    1.00
          1        9       0       0      91    0.00
          R       14      40       0      46    1.00
```

  Вывод команды в случае выполнения в разделе рабочей схемы:

```
AIX wpar1 1 6 00CBA6FE4C00    04/01/08

Конфигурация wpar1: lcpu=2  memlim=204MB  cpulim=0.06  rset=Regular

05:07:16    %usr    %sys    %wio   %idle   physc   %resc
05:07:19      17      83       -       -    0.11   181.6
05:07:22      19      81       -       -    0.08   133.5
05:07:26      16      84       -       -    0.10   173.4

Среднее       17      83       -       -    0.10   164.3
```

  Столбец %resc содержит информацию о разделах рабочей схемы, ресурсы процессоров которых ограничены. Этот показатель отражает процентную долю ресурсов процессора, используемых разделом рабочей схемы.

- **sar -c**

  Опция **-c** позволяет просмотреть частоту отправки системных вызовов.

```
# sar -c 1 3
19:28:25 scall/s sread/s swrit/s  fork/s  exec/s rchar/s wchar/s
19:28:26     134      36       1    0.00    0.00 2691306    1517
19:28:27      46      34       1    0.00    0.00 2716922    1531
19:28:28      46      34       1    0.00    0.00 2716922    1531

Среднее       75      35       1    0.00    0.00 2708329    1527
```

  В отличие от команды **vmstat**, команда **sar** выдает информацию о том, какие именно системные вызовы выполнялись: **read()**, **write()**, **fork()**, **exec()** и т.д. Обратите особое внимание на столбец `fork/s`. Если значение в этом столбце велико, то рекомендуется собрать более подробную информацию с помощью утилит учета ресурсов, команды **trace** или команды **tprof**.

- **sar -q**

  Опция **-q** позволяет просмотреть размер очереди выполнения и очереди подкачки.

```
# sar -q 5 3

19:31:42 runq-sz %runocc swpq-sz %swpocc
19:31:47     1.0     100     1.0     100
19:31:52     2.0     100     1.0     100
19:31:57     1.0     100     1.0     100

Среднее      1.3      95     1.0      95
```

runq-sz
: Среднее число нитей, доступных для выполнения каждую секунду, и доля времени, в течение которой очередь выполнения была занята (поле % может содержать ошибку).

swpq-sz
: Среднее число нитей в очереди ожидания VMM и доля времени, в течение которой очередь подкачки была занята. (Поле % может содержать неточные значения.)

С помощью опции **-q** можно проверить, что в системе не запущено слишком большое число заданий (`runq-sz`), и что подкачка не служит причиной снижения производительности. В системе с большим числом транзакций, например, Enterprise Resource Planning (ERP), очередь выполнения может содержать сотни нитей, так как на выполнение каждой транзакции затрачивается очень маленькая доля времени процессора. Если подкачка служит причиной снижения производительности, вызовите команду **vmstat**. Если в выводе этой команды будет указано, что на ожидание выполнения операций ввода-вывода затрачивается много времени, то в системе многие процессы конкурируют за доступ к диску, либо из-за нехватки памяти часто выполняются операции загрузки и выгрузки страниц.

Применение наборов ресурсов:

Приведены рекомендации по использованию наборов ресурсов.

Рекомендации

1. Если набор ресурсов является субъядром, то содержит несколько процессоров из каждого ядра, то рекомендуется включить основную нить и последующие нити SMT (логические процессоры) таким образом, чтобы процессоры в ядре были последовательными.
2. Набор ресурсов содержит одинаковое число процессоров из каждого предоставленного ядра.

Рекомендации

В случае применения наборов ресурсов выполняются следующие условия:

- Наборы ресурсов могут содержать ядра из нескольких доменов планирования выделения ресурсов (SRAD). При этом число ядер из каждого SRAD необязательно должно быть одинаковым.
- Набор ресурсов может содержать один процессов из каждого ядра; при этом он может не относиться к основной нити SMT.
- Если для наборов ресурсов включено распределение нагрузки, то подсистема складки процессора, такая как VPM, принимает решения относительно складки с учетом ядер, необходимых для наборов ресурсов. Ядра, с которым связано наибольшее число наборов ресурсов, обладают более высоким приоритетом.

Примечание: Если включен статический энергосберегающий режим, то VPM выбирает ядра с поддержкой управления питанием даже в том случае, если для наборов ресурсов включено распределение нагрузки.

Программа xmperf

Программа **xmperf** позволяет просмотреть информацию об использовании CPU в виде динамически изменяющейся диаграммы.

Применение команды time для измерения нагрузки процессора

Команда **time** позволяет просмотреть параметры производительности отдельной программы и ее дочерних процессов, работающих в синхронном режиме.

Команда **time** показывает время, прошедшее с момента запуска до завершения работы программы, то есть *фактическое* время. Кроме того, она показывает, сколько времени процессор затратил на выполнение этой программы. Время процессора делится на две категории: `пользовательское` и `системное`. `Пользовательское` время - это время выполнения самой программы и всех вызовов библиотечных функций. `Системное` время - это время обработки вызванных программой (напрямую или косвенно) системных процедур и функций.

В сумме `пользовательское` и `системное` время дают общее время процессора, затраченное на выполнение программы. Сюда не входит обработка процессором компонентов ядра, которые могут быть запущены в ходе выполнения программы, но в настоящее время не запущены в данной нити. Например, захват страниц физической памяти для замещения страниц, полученных из списка свободных страниц при запуске программы, не учитывается при подсчете времени процессора, затраченного на выполнение программы.

On a uniprocessor, the difference between the `real` time and the total microprocessor time, that is:

```
фактическое - (пользовательское + системное)
```

обусловлена влиянием всех факторов, которые могут замедлить обработку программы; кроме того, в него входят не учтенные ранее затраты времени на обработку самой программы. В системе SMP эту разницу можно вычислить по следующей формуле:

```
фактическое * число-процессоров - (пользовательское + системное)
```

Ниже в порядке убывания значимости перечислены некоторые из этих факторов:

- Операции ввода-вывода, необходимые для считывания кода и данных программы
- Операции ввода-вывода, связанные с выделением памяти для программы
- Процессорное время, затраченное на выполнение других программ
- Процессорное время, затраченное на выполнение задач операционной системы

В приведенном ниже примере программа откомпилирована с опцией **-O3** для повышения скорости ее работы. Разность между фактическим временем выполнения программы и суммой пользовательского и системного времени в этом примере очень мала. Программе было предоставлено все запрошенное время - возможно, за счет других программ в системе.

```
# time looper
фактическое    0m3.58s
пользов.       0m3.16s
сист.          0m0.04s
```

В следующем примере перед запуском той же программы ее приоритет был понижен путем добавления 10 к значению nice. Время выполнения программы увеличилось почти вдвое, однако при этом другие программы не были приостановлены:

```
# time nice -n 10 looper
фактическое    0m6.54s
пользов.       0m3.17s
системное     0m0,03s
```

Обратите внимание, что команда **nice** указана после команды **time**, а не наоборот. Если бы мы ввели команду

```
# nice -n 10 time looper
```

то была бы выполнена другая команда **time** (**/usr/bin/time**), отличающаяся меньшей точностью по сравнению с применяемой нами командой **time**, встроенной в оболочку **ksh**. Если команда **time** указана первой, то запускается встроенная версия, если явно не указано полное имя команды **/usr/bin/time**. Если команду **time** вызывает другая команда, то запускается программа **/usr/bin/time**.

Рекомендации по использованию команд time и timex

При работе с командами **time** и **timex** следует обратить внимание на следующие особенности.

Особенности:

- Не рекомендуется применять команды `/usr/bin/time` и `/usr/bin/timex`. Всегда, когда это возможно, запускайте команду time из оболочки Korn или C.
- Для сбора дополнительных статистических данных команда **timex -s** запускает команду **sar**. Поскольку команда **sar** создает значительную нагрузку на систему, команда **timex -s** также отличается этим свойством. В частности, при запуске на небольшое время данные, возвращаемые командой **timex -s**, могут не соответствовать показателям выполнения программы в той системе, в которой не запущена программа сбора информации.
- Из-за длительности такта системных часов (приблизительно 10 мс) и правил распределения времени CPU при работе с несколькими нитями, повторный запуск команды **time** может дать другой результат. Поскольку время процессора дискретно, между двумя запусками программы всегда существует неизбежная разница. Разница значений выражается в тактах. Разница (в процентах от результата выполнения команды) будет тем больше, чем меньше время работы программы (см. “Работа с таймером процессора” на стр. 411).
- Команды **time** и **timex** из каталога `/usr/bin` и встроенную функцию **time** не рекомендуется применять для измерения пользовательского или системного времени выполнения конвейера команд, запущенного из командной строки. Один из возможных источников ошибок заключается в том, что вследствие несогласованности синтаксических правил команда **time** может измерить время выполнения только одной команды и не выдать никакого сообщения об ошибке. Команда введена синтаксически правильно, но выполняет не те действия, которые ожидал от нее пользователь.
- Хотя формат команды **time** не изменился, в системах SMP ее вывод интерпретируется немного по-другому:

  В системе SMP фактическое время может быть меньше пользовательского времени процесса. Пользовательское время теперь представляет собой суммарное время, затраченное на выполнение нитей процесса всеми процессорами.

  Если в однопроцессорной системе будет запущен процесс, состоящий из четырех нитей, то фактическое время будет больше пользовательского времени:

  ```
  # time 4threadedprog
  фактическое    0m11.70s
  пользов.       0m11.09s
  сист.          0m0.08s
  ```

  Если тот же процесс запустить в четырехпроцессорной системе, то фактическое время может составлять около 1/4 от пользовательского времени. Как видно из приведенного ниже примера, процесс с несколькими нитями распределяет нагрузку на несколько процессоров, за счет чего фактическое время выполнения уменьшается. Следовательно, скорость выполнения процесса в системе увеличилась.

  ```
  # time 4threadedprog
  фактическое    0m3.40s
  пользов.       0m9.81s
  сист.          0m0.09s
  ```

Обнаружение программ, загружающих процессор

Существует две стандартных команды, позволяющих определить, какие процессы используют больше всего ресурсов процессора: команды **ps** и **acctcom**.

Кроме того, для этой цели можно запустить монитор **topas**, который описан в разделе “Непрерывное отслеживание производительности системы с помощью команды topas” на стр. 16.

Применение команды ps

Команда **ps** является очень гибким инструментом для определения работающих в системе программ и оценки используемых ими ресурсов. Она выводит статистику и информацию о состоянии процессов в системе, в том числе ИД процесса или нити, объем выполняемого ввода-вывода и используемый объем ресурсов процессора и памяти.

В этом разделе описаны опции и поля вывода, которые относятся к процессору.

Информацию об использовании процессора содержат три столбца вывода команды **ps**.

Столбец
Значение:

C Показатель использования процессора процессом (в тактах)

TIME Общее процессорное время, затраченное на выполнение процесса (в минутах и секундах)

%CPU
Процессорное время, затраченное на выполнение процесса с момента его запуска, поделенное на время, прошедшее с момента запуска процесса. Этот параметр отражает зависимость программы от процессора.

Интенсивность использования процессора

Следующий сценарий оболочки:

```
# ps -ef | egrep -v "STIME|$LOGNAME" | sort +3 -r | head -n 15
```

позволяет найти пользовательские процессы, которые потребляют больше всего ресурсов CPU (для большей наглядности была вставлена строка заголовка).

```
     UID   PID  PPID   C    STIME    TTY  TIME CMD
    mary 45742 54702 120 15:19:05 pts/29  0:02 ./looper
    root 52122     1  11 15:32:33 pts/31 58:39 xhogger
    root  4250     1   3 15:32:33 pts/31 26:03 xmconsole allcon
    root 38812  4250   1 15:32:34 pts/31  8:58 xmconstats 0 3 30
    root 27036  6864   1 15:18:35      -  0:00 rlogind
    root 47418 25926   0 17:04:26      -  0:00 coelogin <d29dbms:0>
    bick 37652 43538   0 16:58:40  pts/4  0:00 /bin/ksh
    bick 43538     1   0 16:58:38      -  0:07 aixterm
     luc 60062 27036   0 15:18:35 pts/18  0:00 -ksh
```

Столбец (C) означает недавно использованный процессор. В начале списка находится процесс зацикленной программы. Значение C может минимизировать использование процессором зацикленной программы, так как планировщик остановит процесс при счете 120. Для многопотокового процесса в этом поле отображается сумма для всех потоков в этом процессе.

В следующем примере показана простая программа с пятью потоками, которые находятся в бесконечной зацикленной программе:

```
ps -lmo THREAD -p 8060956
    USER PID   PPID  TID   ST CP PRI SC WCHAN F      TT    BND COMMAND
    root 8060956 6815882         - A  720 120  0        -   200001  pts/0   - ./a.out
       -       -       -   8716483 R  120 120  0        -   400000      -   - -
       -       -       -  17105017 R  120 120  0        -   400000      -   - -
       -       -       -  24182849 R  120 120  0        -   400000      -   - -
       -       -       -  24510589 R  120 120  0        -   400000      -   - -
       -       -       -  30277829 R  120 120  0        -   400000      -   - -
       -       -       -  35913767 R  120 120  0        -   400000      -   - -
```

Значение 720 в столбце CP означает сумму отдельных потоков, указанных под этим значением: (5 * 120) + (120).

процессор, коэффициент использования

В столбце TIME вывода команды **ps**, запускаемой периодически, указывается обычное время работы процессора, а в столбце %CPU - отношение времени работы процессора к фактическому времени. Найдите процессы, выполнение которых отнимает у процессора больше всего времени. Опции **au** и **v** позволяют получить такую информацию для пользовательских процессов. Опции **aux** и **vg** выводят информацию как о пользовательских, так и о системных процессах.

Ниже приведен пример вывода команды, запущенной в четырехпроцессорной системе:

```
# ps au
USER       PID %CPU %MEM   SZ  RSS    TTY STAT    STIME TIME COMMAND
root     19048 24.6  0.0   28   44  pts/1 A    13:53:00 2:16 /tmp/cpubound
root     19388  0.0  0.0  372  460  pts/1 A      Feb 20 0:02 -ksh
root     15348  0.0  0.0  372  460  pts/4 A      Feb 20 0:01 -ksh
root     20418  0.0  0.0  368  452  pts/3 A      Feb 20 0:01 -ksh
root     16178  0.0  0.0  292  364      0 A      Feb 19 0:00 /usr/sbin/getty
root     16780  0.0  0.0  364  392  pts/2 A      Feb 19 0:00 -ksh
root     18516  0.0  0.0  360  412  pts/0 A      Feb 20 0:00 -ksh
root     15746  0.0  0.0  212  268  pts/1 A    13:55:18 0:00 ps au
```

Значение %CPU указывает, какой процент процессорного времени был потрачен на выполнение процесса с момента его запуска. Это значение подсчитывается по следующей формуле:

```
(время CPU, затраченное на выполнение процесса / время выполнения
процесса) * 100
```

Представим, что было запущено два процесса: первый проработал с момента запуска пять секунд, а затем был прерван, после чего был запущен второй процесс, который также проработал пять секунд, а затем был прерван. В результате в столбце %CPU вывода команды **ps** для первого процесса будет указано значение 50 процентов (процессор работал в течение 5 из 10 прошедших секунд), а для второго процесса - 100 процентов (процессор работал в течение всех пяти секунд, прошедших с момента запуска процесса).

В системе SMP это значение делится на число CPU, работающих в системе. По этой причине значение %CPU для процесса **cpubound** из предыдущего примера никогда не будет больше 25, так как он выполняется в четырехпроцессорной системе. Процесс **cpubound** полностью загрузил один из процессоров, однако значение %CPU будет равно всего лишь 25, так как оно делится на число работающих процессоров.

Опция THREAD

Команда **ps** позволяет получить информацию о том, с какими процессорами связаны нити и процессы. Для этого нужно вызвать команду **ps -mo THREAD**. Пример приведен ниже:

```
# ps -mo THREAD
USER PID   PPID  TID   ST CP PRI SC WCHAN F      TT    BND COMMAND
root 20918 20660 -     A  0  60  1  -     240001 pts/1 -   -ksh
-    -     -     20005 S  0  60  1  -     400    -     -   -
```

В столбце TID указывается ИД нити, а в столбце BND - процессор, с которым связана нить или процессор.

В некоторых случаях может быть показано, что в системе запущен процесс kproc (в операционной системе версии 4 его PID равен 516), на выполнение которого тратится значительная доля процессорного времени. Если ни одна нить в системе не готова к выполнению в течение следующего кванта времени, то планировщик выделяет этот квант времени процессора процессу ядра (kproc), который также называется процессом *idle* или *wait*. В системах SMP для каждого процессора есть свой *idle* kproc.

Более подробная информация о команде **ps** приведена в книге в книге *Справочник по командам*.

Работа с командой acctcom

Команда **acctcom** выводит хронологическую информацию об использовании CPU. Для ее работы должна быть активирована система учета ресурсов.

Включайте систему учета ресурсов только в случае необходимости, так как она создает значительную нагрузку на систему. Для активации системы учета ресурсов выполните следующие действия:

1. Создайте пустой файл учета ресурсов:
   ```
   # touch acctfile
   ```
2. Включите систему учета ресурсов:
   ```
   # /usr/sbin/acct/accton acctfile
   ```
3. После сбора всей необходимой информации выключите систему учета ресурсов:
   ```
   # /usr/sbin/acct/accton
   ```
4. Просмотрите информацию, собранную системой учета ресурсов, с помощью следующей команды:
   ```
   # /usr/sbin/acct/acctcom acctfile
   Имя                Имя     Время    Время       Фактич.    CPU   Средний
   команды    Польз.  TTY     запуска  заверш.     время(с)   (с)  разм.(КБ)
   #accton    root    pts/2   19:57:18 19:57:18     0.02     0.02   184.00
   #ps        root    pts/2   19:57:19 19:57:19     0.19     0.17    35.00
   #ls        root    pts/2   19:57:20 19:57:20     0.09     0.03   109.00
   #ps        root    pts/2   19:57:22 19:57:22     0.19     0.17    34.00
   #accton    root    pts/2   20:04:17 20:04:17     0.00     0.00     0.00
   #who       root    pts/2   20:04:19 20:04:19     0.02     0.02     0.00
   ```

Если вы еще раз запустите систему учета ресурсов с тем же файлом, то все новые процессы будут следовать в списке за процессом **accton** (процесс, с помощью которого работа системы учета ресурсов была завершена в первый раз).

Оценка интенсивности использования процессора нитями ядра с помощью команды pprof

Команда **pprof** отображает показатели использования процессора для всех нитей ядра, выполнявшихся в указанный период времени. Работа этой команды основана на утилите трассировки.

Исходная информация о процессах сохраняется в файле `pprof.flow`. На ее основе команда может создать отчеты пяти типов. Если флаги не заданы, то выдаются все пять отчетов.

Для того чтобы узнать, установлена ли команда **pprof**, вызовите следующую команду:

```
# lslpp -lI bos.perf.tools
```

Эта программа создает отчеты следующих типов:

pprof.cpu
: Перечислены все нити ядра, упорядоченные по фактическому времени обработки процессором. Отчет содержит следующую информацию: имя процесса, ИД процесса, ИД родительского процесса, состояние процесса в начале и в конце работы программы, ИД нити, ИД родительской нити, фактическое время процессора, время запуска, время завершения и время работы программы.

pprof.famcpu
: Перечислена информация обо всех семействах (процессы с общим предком). Имя и ИД процесса и его предка не обязательно совпадают. Отчет содержит следующую информацию: время запуска, имя процесса, ИД процесса, число нитей, общее время процессора.

pprof.famind
: Перечислены все процессы, сгруппированные по семействам (процессы с общим предком). Имена дочерних процессов указаны с отступом по отношению к именам их родителей. Отчет содержит следующие столбцы: Начальное время, Конечное время, Фактическое время процессора, ИД процесса, ИД родительского процесса, ИД нити, ИД родительской нити, Состояние процесса в начале и в конце, Уровень, Имя процесса.

pprof.namecpu
Перечислена информация о каждом типе нити ядра (все исполняемые нити с одним именем). Отчет содержит следующую информацию: имя процесса, число нитей, время процессора, доля от общего времени процессора.

pprof.start
Список всех нитей ядра, переданных процессору на обработку в течение работы команды **pprof**. Нити упорядочены по времени запуска. Отчет содержит следующую информацию: имя процесса, ИД процесса, ИД родительского процесса, состояние процесса в начале и в конце работы программы, ИД нити, ИД родительской нити, фактическое время процессора, время запуска, время завершения и время выполнения программы.

Ниже приведен пример файла `pprof.namecpu`, содержащего отчет о выполнении программы **tthreads32**. Программа порождает четыре нити, каждая из которых, в свою очередь, порождает собственный процесс. Эти процессы запускают несколько программ **ksh** и **sleep**:

```
Отчет Pprof о процессах

Упорядочено по времени процессора

От: четверг 19 октября 2000 г. 17.53.07
До: четверг 19 октября 2000 г. 17.53.22


    Процесс  # нитей     Время_CPU       %
  ======== ==========  ========  ========
tthreads32         13     0.116    37.935
        sh          8     0.092    30.087
      Idle          2     0.055    17.987
       ksh         12     0.026     8.503
     trace          3     0.007     2.289
      java          3     0.006     1.962
     kproc          5     0.004     1.308
   xmservd          1     0.000     0.000
   trcstop          1     0.000     0.000
   swapper          1     0.000     0.000
       gil          1     0.000     0.000
        ls          4     0.000     0.000
     sleep          9     0.000     0.000
        ps          4     0.000     0.000
   syslogd          1     0.000     0.000
      nfsd          2     0.000     0.000
          ==========  ========  ========
                  70     0.306   100.000
```

Соответствующий файл `pprof.cpu` выглядит следующим образом:

```
Отчет Pprof об использовании процессора

Упорядочено по фактическому времени процессора

От: четверг 19 октября 2000 г. 17.53.07
До: четверг 19 октября 2000 г. 17.53.22


E = Выполнен    F = Разветвлен
X = Завершен    A = Активен (на момент начала или конца трассировки)
C = Создана нить

      Процесс     PID     PPID  BE      TID     PTID      Учет    Начало     Конец  Разность
      =====    =====    ===== ===    =====    =====  ========  ========  ========  ========
       Idle      774        0  AA      775        0     0.052     0.000     0.154     0.154
  tthreads32     5490    11982  EX    18161    22435     0.040     0.027     0.154     0.126
         sh    11396     5490  EE    21917     5093     0.035     0.082     0.154     0.072
         sh    14106     5490  EE    16999    18867     0.028     0.111     0.154     0.043
         sh    13792     5490  EE    20777    18179     0.028     0.086     0.154     0.068
```

```
       ksh     5490    11982  FE    18161    22435     0.016     0.010     0.027     0.017
tthreads32     5490    11982  CX     5093    18161     0.011     0.056     0.154     0.098
tthreads32     5490    11982  CX    18179    18161     0.010     0.054     0.154     0.099
tthreads32    14506     5490  FE    17239    10133     0.010     0.128     0.143     0.015
       ksh    11982    13258  AA    22435        0     0.010     0.005     0.154     0.149
tthreads32    13792     5490  FE    20777    18179     0.010     0.059     0.086     0.027
tthreads32     5490    11982  CX    18867    18161     0.010     0.057     0.154     0.097
tthreads32    11396     5490  FE    21917     5093     0.009     0.069     0.082     0.013
tthreads32     5490    11982  CX    10133    18161     0.008     0.123     0.154     0.030
tthreads32    14106     5490  FE    16999    18867     0.008     0.088     0.111     0.023
     trace     5488    11982  AX    18159        0     0.006     0.001     0.005     0.003
     kproc     1548        0  AA     2065        0     0.004     0.071     0.154     0.082
      Idle      516        0  AA      517        0     0.003     0.059     0.154     0.095
      java    11612    11106  AA    14965        0     0.003     0.010     0.154     0.144
      java    11612    11106  AA    14707        0     0.003     0.010     0.154     0.144
     trace    12544     5488  AA    20507        0     0.001     0.000     0.001     0.001
        sh    14506     5490  EE    17239    10133     0.001     0.143     0.154     0.011
     trace    12544     5488  CA    19297    20507     0.000     0.001     0.154     0.153
       ksh     4930     2678  AA     5963        0     0.000     0.154     0.154     0.000
     kproc     6478        0  AA     3133        0     0.000     0.154     0.154     0.000
        ps    14108     5490  EX    17001    18867     0.000     0.154     0.154     0.000
tthreads32    13794     5490  FE    20779    18179     0.000     0.154     0.154     0.000
        sh    13794     5490  EE    20779    18179     0.000     0.154     0.154     0.000
        ps    13794     5490  EX    20779    18179     0.000     0.154     0.154     0.000
        sh    14108     5490  EE    17001    18867     0.000     0.154     0.154     0.000
tthreads32    14108     5490  FE    17001    18867     0.000     0.154     0.154     0.000
        ls    13792     5490  EX    20777    18179     0.000     0.154     0.154     0.000
:
:
:
```

Обнаружение эмулируемых инструкций с помощью утилиты emstat

Для обеспечения совместимости со старыми версиями программ в ядре AIX предусмотрены функции эмуляции инструкций, которые не входят в набор инструкций данного конкретного процессора. Попытка выполнить неподдерживаемую инструкцию приводит к возникновению исключительной ситуации "запрещенная инструкция". Ядро расшифровывает недопустимую инструкцию и, если она относится к числу неподдерживаемых, запускает функцию эмуляции этой инструкции.

Поскольку эмуляция неподдерживаемых инструкций повышает нагрузку на процессор, она может привести к снижению производительности. Это зависит от частоты выполнения неподдерживаемых инструкций и числа команд в процедурах их эмуляции. Более того, даже если доля операций эмуляции мала, снижение производительности может быть весьма ощутимым. В приведенной ниже таблице указано примерное число команд в функциях эмуляции некоторых неподдерживаемых инструкций:

Инструкция	Среда эмуляции	Приблизительное число команд
abs	ассемблер	117
doz	ассемблер	120
mul	ассемблер	127
rlmi	C	425
sle	C	447
clf	C	542
div	C	1079

Инструкции, поддерживаемые не всеми платформами, следует удалить из кода, написанного на ассемблере, поскольку повторная компиляция влияет только на код высокого уровня. Необходимо также изменить коды возврата ассемблера, чтобы они не использовали отсутствующие инструкции.

Для того чтобы узнать, эмулируются ли некоторые инструкции, воспользуйтесь утилитой **emstat**.

Для того чтобы определить, установлена и доступна ли программа **emstat**, введите следующую команду:

```
# lslpp -lI bos.perf.tools
```

Как и в команде **vmstat**, в команде **emstat** указывается интервал сбора информации в секундах и, при необходимости, число интервалов. Первый столбец вывода содержит общее число эмулированных инструкций с момента загрузки системы, а второй - число инструкций, эмулированных за указанный интервал. Если в секунду эмулируется до нескольких тысяч инструкций, то это может служить причиной снижения производительности.

Ниже приведен пример вывода команды **emstat 1**:

```
# emstat 1

Эмуляция     Эмуляция
с загрузки   новое
        0           0
        0           0
        0           0
```

Обнаружив, что эмуляция действительно происходит, необходимо найти приложение, в котором эмулируются инструкции. Это гораздо более трудоемкая операция. Один из способов заключается в поочередном запуске приложений и отслеживании их работы с помощью программы **emstat**. Некоторые случаи эмуляции можно обнаружить с помощью точки трассировки. Их можно найти в текстовом отчете о трассировке по словам `PROGRAM CHECK`. Процесс или нить, связанные с этим событием трассировки, эмулируют либо собственные инструкции, либо инструкции применяемых библиотечных функций или модулей.

Обнаружение исключительных ситуаций при выравнивании с помощью утилиты alstat

Неправильное выравнивание данных вызывает особые исключительные ситуации.

Компиляторы AIX используют естественное выравнивание типов данных. Например, данные типа short (длина - 2 байта) выравниваются по границам 4-байтовых областей. Некоторые приемы программирования - например, преобразование типов и директивы принудительного выравнивания, могут привести к неправильному выравниванию данных. Однако оптимизация под процессоры С процессором POWER, применяемая компиляторами AIX, подразумевает правильное выравнивание данных. Поэтому на чтение неправильно выравненных данных может потребоваться больше операций обращения к памяти. Исключительная ситуация неправильного выравнивания данных вынуждает ядро эмулировать операции обращения к памяти с помощью нескольких операций. Эта разновидность эмуляции, как и эмуляция инструкций, приводит к снижению скорости выполнения программ.

В набор файлов `bos.perf.tools` включена утилита **alstat**, позволяющая отслеживать возникновение исключительных ситуаций, связанных с выравниванием. Опция **-v** этой утилиты позволяет сгруппировать сведения об исключительных ситуациях по отдельным процессорам.

Поскольку утилиты **alstat** и **emstat** представляют собой одну и ту же программу, они полностью взаимозаменяемы с точки зрения эмуляции инструкций и обработки исключительных ситуаций, связанных с выравниванием. Для эмуляции инструкций следует пользоваться опцией **-e** утилиты **alstat**. Для просмотра исключительных ситуаций, связанных с выравниванием, применяется опция **-a** утилиты **emstat**.

Утилита **alstat** выдает данные примерно следующего вида:

```
# alstat -e 1
  Выравн.    Выравн.    Эмуляция   Эмуляция
  с загрузки    дельта  с загрузки     дельта
          0              0           0            0
          0              0           0            0
          0              0           0            0
```

Изменение структуры исполняемых программ с помощью программы fdpr

Программа **fdpr** оптимизирует исполняемые модули, сокращая время выполнения программы и повышая эффективность использования оперативной памяти.

Для того чтобы определить, установлена и доступна ли программа **fdpr**, введите следующую команду:

```
# lslpp -lI perfagent.tools
```

Команда **fdpr** относится к утилитам настройки производительности. Она позволяет повысить производительность пользовательских приложений и эффективность использования оперативной памяти. Для работы программы **fdpr** не нужен исходный код оптимизируемого приложения. Однако, оптимизация программ без информации о компиляции не поддерживаются. Если у вас есть исходный код программы, то вы можете скомпилировать его с флагом **-qfdpr**. В этом случае исполняемая программа будет содержать информацию, позволяющую гарантировать работоспособность кода, измененного программой **fdpr**. Флаг **-qfdpr** должен указываться при компиляции всех объектных модулей программы. Если указан флаг **-qfdpr**, то статическое связывание не повышает производительность программы.

Утилита **fdpr** изменяет порядок инструкций исполняемой программы, улучшая работу кэша инструкций, таблицы преобразования адресов (TLB) и оперативной памяти за счет следующих операций:

- Размещение рядом часто вызываемых фрагментов кода (эти фрагменты выбираются по итогам анализа программы)
- Изменение условных переходов для повышения вероятности их аппаратного предсказания
- Удаление из основного потока исполнения редко используемых фрагментов кода

Например, анализируя оператор "if-then-else", программа **fdpr** может прийти к выводу, что альтернативная (стоящая после else) ветвь используется чаще, чем основная (стоящая после then). В этом случае программа инвертирует условие и поменяет местами ветви, как показано на следующем рисунке.

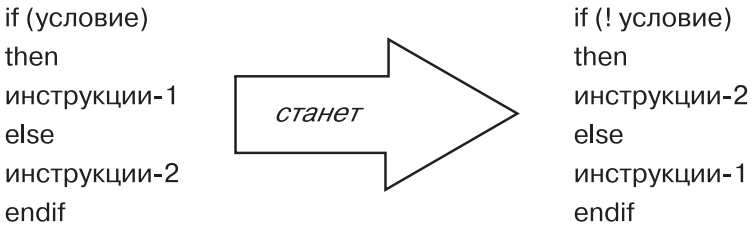


Рисунок 16. Пример изменения оператора условия. На рисунке показано, каким образом программа изменит оператор условия. Например, выражение If (условие) будет заменено на If (! условие), операторы then, else и endif останутся без изменений, а инструкции из основной и альтернативной части оператора будут поменяны местами.

В результате оптимизации увеличение производительности больших (>5 МБ) приложений, активно использующих процессор, достигает 23 процентов, хотя обычно составляет 5-20 процентов. Сокращение объема физической (невыгружаемой) памяти, занимаемого кодом программы, достигает 70 процентов, хотя обычно составляет от 20 до 50 процентов. Конкретные значения зависят от типа приложения и опций программы **fdpr**.

Работа программы **fdpr** состоит из трех этапов:

1. Сначала оптимизируемый исполняемый модуль настраивается для сбора данных о производительности.
2. Затем программа запускается в рабочей системе. При выполнении программы собирается информация о производительности.
3. На основе этих данных выполняется оптимизация. Полученный исполняемый модуль должен работать с указанными при запуске данными более эффективно. Очень важно, чтобы утилита **fdpr** применялась для

оптимизации программы в той рабочей схеме, в которой эта программа будет действительно применяться. Производительность оптимизированной программы при работе с сильно отличными данными заранее предсказать невозможно. В частности она может оказаться ниже, чем у первоначальной версии.

Например команда **# fdpr -p ProgramName -R3 -x test.sh** будет использовать тестовый набор `test.sh` для сбора данных о работе программы **ProgramName**. На основании полученного вывода будет выполнена максимальная (**-R3**) оптимизация программы. В результате будет создан новый модуль, которому по умолчанию присваивается имя `имя-программы.fdpr`. Степень улучшения характеристик оптимизированной программы по сравнению с исходной в существенной степени зависит от того, насколько точно тестовый набор `test.sh` имитирует рабочую нагрузку.

Примечание: Программа **fdpr** применяет для оптимизации сложные алгоритмы, которые иногда изменяют логику работы программы. В связи с этим перед первым запуском с реальными данными *обязательно следует* тщательно и всесторонне протестировать оптимизированную программу, чтобы убедиться в корректности результатов ее работы.

При работе с программой **fdpr** придерживайтесь следующих рекомендаций:

- При запуске программы **fdpr** укажите параметры и файлы данных, характерные для дальнейшего применения оптимизированной программы.
- Тщательно и всесторонне проверьте работу оптимизированного кода.
- Запускайте полученную программу только в той рабочей схеме, для которой он был оптимизирован.

Управление конкуренцией за микропроцессор

Несмотря на то, что ядро AIX распределяет нити между процессорами, большинство инструментов управления системой по-прежнему работают не с нитями, а с процессами, к которым относятся эти нити.

Управление приоритетом пользовательских процессов

Для изменения приоритета пользовательских процессов предусмотрены команды **nice** и **renice**, а также функция **setpri()**. Для просмотра приоритета предназначена команда **ps**.

Обзор приоритетов приведен в разделе “Приоритет процессов и нитей” на стр. 40.

Вычисление приоритета позволяет решить следующие задачи:

- Разделить процессорное время между нитями
- Предотвратить "дискриминацию" отдельных нитей
- Назначить штраф нитям, потребляющим много ресурсов процессора
- Динамически изменять степень важности нитей

Изменение значения приоритета с помощью команды nice:

С помощью утилиты **nice** любой пользователь может запустить команду с пониженным приоритетом.

Для запуска команд с повышенным приоритетом воспользоваться **nice** может только пользователь root. В этом случае возможные значения параметра команды **nice** лежат в пределах от -20 до 19.

Значение, указанное в команде **nice**, добавляется или вычитается из стандартного значения `nice`. Изменение значения `nice` влияет только на приоритет процесса, выполняющего указанную команду. Приоритет этого процесса не фиксируется, а изменяется в зависимости от нагрузки на процессор, значения `nice` и минимального приоритета пользовательского процесса.

Для интерактивных процессов значение `nice` по умолчанию равно 20 (для фоновых процессов ksh - 24). Приведенная ниже команда запускает команду **vmstat** в интерактивном режиме со значением `nice`, равным 25 (вместо 20), таким образом понижая приоритет этой команды.

```
# nice -n 5 vmstat 10 3 > vmstat.out
```

Пользователь root может повысить приоритет команды **vmstat** с помощью следующей команды:

```
# nice -n -5 vmstat 10 3 > vmstat.out
```

Если пользователь, запустивший указанную выше команду **nice**, не является пользователем root, то значение `nice` команды **vmstat** не изменится (останется равным 20). При этом команда **nice** не выдаст никакого сообщения об ошибке.

Настройка фиксированного приоритета с помощью функции setpri:

Приложение, запущенное от имени пользователя root, может установить приоритет своего или любого другого процесса с помощью функции **setpri()**.

Например:

```
retcode = setpri(0,59);
```

установит для текущего процесса фиксированный приоритет 59. Если функции **setpri()** не удается изменить приоритет, она возвращает значение -1.

Приведенная ниже программа получает из командной строки значение приоритета и список идентификаторов процессов и устанавливает для всех процессов указанный приоритет.

```
/*
   fixprocpri.c
   Формат: fixprocpri приоритет PID . . .
*/

#include <sys/sched.h>
#include <stdio.h>
#include <sys/errno.h>

main(int argc,char **argv)
{
   pid_t ProcessID;
   int Priority,ReturnP;

   if( argc < 3 ) {
      printf(" Формат: setpri приоритет список-PID \n");
      exit(1);
   }

   argv++;
   Priority=atoi(*argv++);
   if ( Priority < 50 ) {
      printf(" Значение приоритета должно быть >= 50 \n");
      exit(1);
   }

   while (*argv) {
      ProcessID=atoi(*argv++);
      ReturnP = setpri(ProcessID, Priority);
      if ( ReturnP > 0 )
          printf("pid=%d новый приоритет=%d  старый приоритет=%d\n",
            (int)ProcessID,Priority,ReturnP);
      else {
          perror(" ошибка setpri ");
            exit(1);
      }
   }
}
```

Просмотр приоритета процесса с помощью команды ps

Команда **ps** с флагом **-l** (строчная L) позволяет просмотреть значения `nice` и текущие приоритеты указанных процессов.

Например, для того чтобы узнать приоритеты процессов, запущенных пользователем nobody, введите следующую команду:

```
# ps -lu user1
      F S UID   PID  PPID   C PRI NI ADDR    SZ    WCHAN    TTY  TIME CMD
  241801 S 200 7032 7286   0  60 20 1b4c   108           pts/2  0:00 ksh
  200801 S 200 7568 7032   0  70 25 2310    88  5910a58  pts/2  0:00 vmstat
  241801 S 200 8544 6494   0  60 20 154b   108           pts/0  0:00 ksh
```

Данная команда отражает результат запуска команды **nice -n 5**, описанной выше. Для процесса 7568 установлен приоритет 70. (Команда **ps** выполнялась в отдельном сеансе в режиме root - поэтому показаны два разных терминала).

Если один из процессов установит для себя фиксированный приоритет с помощью функции **setpri(10758, 59)**, то вывод команды **ps -l** будет выглядеть следующим образом:

```
      F S UID   PID  PPID   C PRI NI ADDR    SZ    WCHAN    TTY  TIME CMD
  200903 S   0 10758 10500   0  59 -- 3438    40  4f91f98  pts/0  0:00 fixpri
```

Изменение приоритета с помощью команды renice

Команда **renice** изменяет значение `nice` (а, следовательно, и приоритет) уже запущенного процесса. Для выбора процесса можно задать идентификатор процесса, ИД группы процессов или имя пользователя, которому он принадлежит.

С помощью команды **renice** нельзя изменить фиксированный приоритет процесса. Увеличить значение `nice` может любой пользователь. Уменьшить это значение может только пользователь root. Данная команда изменяет значения `nice` указанных процессов. Однако, приоритет не фиксируется. Пользователь root может указать в команде **renice** значение от -20 до 20. Таким образом, он может уменьшить значение `nice` процесса.

В продолжение нашего примера изменим командой **renice** значение `nice` процесса **vmstat**, запущенного с помощью команды **nice**.

```
# renice -n -5 7568
# ps -lu user1
      F S UID   PID  PPID   C PRI NI ADDR    SZ    WCHAN    TTY  TIME CMD
  241801 S 200 7032 7286   0  60 20 1b4c   108           pts/2  0:00 ksh
  200801 S 200 7568 7032   0  60 20 2310    92  5910a58  pts/2  0:00 vmstat
  241801 S 200 8544 6494   0  60 20 154b   108           pts/0  0:00 ksh
```

Теперь процесс работает с более высоким приоритетом, чем другие интерактивные процессы. Для восстановления исходного значения приоритета вызовите команду:

```
# renice -n 5 7568
# ps -lu user1
      F S UID   PID  PPID   C PRI NI ADDR    SZ    WCHAN    TTY  TIME CMD
  241801 S 200 7032 7286   0  60 20 1b4c   108           pts/2  0:00 ksh
  200801 S 200 7568 7032   1  70 25 2310    92  5910a58  pts/2  0:00 vmstat
  241801 S 200 8544 6494   0  60 20 154b   108           pts/0  0:00 ksh
```

В данном примере предполагалось, что команда **renice** запускается пользователем root. Если команда **renice** запускается обычным пользователем, действуют два важных ограничения:

- В команде можно указать лишь те процессы, которые были запущены этим пользователем.
- Значение `nice` нельзя уменьшить, даже если до этого оно было увеличено с помощью команды **renice**.

Формат команд nice и renice

В командах **nice** и **renice** приращение значения `nice` можно указать разными способами.

Команда Nice	Команда Renice	Итоговое значение nice	Оптимальное значение приоритета
nice -n 5	**renice -n 5**	25	70
nice -n +5	**renice -n +5**	25	70
nice -n -5	**renice -n -5**	15	55

Вычисление значения приоритета нити

В данном разделе описана настройка с помощью вычисления приоритета в команде **schedo**.

Команда “Команда schedo” на стр. 126 позволяет изменять некоторые параметры планировщика заданий, применяемые для вычисления приоритета отдельных нитей. См. раздел “Приоритет процессов и нитей” на стр. 40 для получения дополнительной информации о приоритете.

Для того чтобы определить, установлена и доступна ли программа **schedo**, введите следующую команду:

```
# lslpp -lI bos.perf.tune
```

Вычисление приоритета:

Значение приоритета нити (который иногда называется приоритетом планирования) хранится в ядре. Приоритет - это натуральное число, обратно пропорциональное важности нити. Таким образом, чем меньше приоритет, тем важнее нить. Планировщик выбирает для запуска нить с наименьшим значением приоритета.

Приоритет вычисляется по следующей формуле:

```
приоритет = базовый-приоритет + значение-nice + штраф-за-использование-процессора
```

Показатель использования процессора, связанный с нитью, увеличивается на единицу каждый раз, когда при возникновении прерывания (раз в 10 миллисекунд) выясняется, что процессор выполняет данную нить. Его значение указано в столбце *C* вывода команды **ps**. Максимальное значение показателя равно 120.

По умолчанию штраф за использование процессора вычисляется путем умножения показателя использования процессора на коэффициент 0,5. Данный коэффициент зависит от параметра *R* (по умолчанию равен 16). Ниже приведена точная формула:

```
Штраф-за-использование-процессора = C * R/32
```

Каждую секунду алгоритм умножает показатели использования процессора всех нитей на коэффициент затухания 0,5. Данный коэффициент зависит от параметра *D*, который по умолчанию равен 16. Ниже приведена точная формула:

```
C = C * D/32
```

Для того чтобы определить приоритет нитей в процессе, алгоритм вычисления приоритета использует значение **nice** этого процесса. С ростом времени процессора, затраченного на выполнение процесса, приоритет процесса уменьшается пропорционально значению `nice`. С помощью команды `schedo -r -d` можно повлиять на алгоритм вычисления приоритета, изменив значения R и D. За дополнительной информацией обратитесь к разделу “Команда schedo” на стр. 126.

Вначале вычисляется следующее значение:

```
базовый_приоритет = стандартный_приоритет + значение_nice
```

Затем вычисляется вспомогательная величина x_nice по следующей формуле:

```
Если базовый_приоритет > 60,
   то x_nice = (базовый_приоритет * 2) - 60,
   иначе x_nice = базовый_приоритет.
```

Это означает, что если значение nice больше 20, то оно в два раза сильнее влияет на значение приоритета, чем если бы оно было меньше или равно 20. Ниже приведена новая формула, по которой вычисляется приоритет (без учета округления до целого значения):

```
приоритет = x_nice + [(x_nice + 4)/64 * C*(R/32)]
```

Команда schedo:

Для изменения способа вычисления приоритета в команде **schedo** предусмотрены две опции: **sched_R** и **sched_D**.

В каждой опции указывается целое значение от 0 до 32. Существующие значения коэффициентов умножаются на значения параметров, а затем делятся на 32. По умолчанию значения *R* и *D* равны 16. При этом (D=R=16)/32=0,5. Эти значения существенно влияют на расчет приоритетов. Например:

```
# schedo -o sched_R=0
```

[(R=0)/32=0, (D=16)/32=0,5] означает, что штраф за использование процессора будет равен 0. В этом случае приоритет будет зависеть только от значения nice. Фоновые процессы не получат процессорного времени до тех пор, пока можно выполнять хотя бы один интерактивный процесс. Приоритеты нитей изменяться не будут, хотя формально их приоритет не фиксирован.

```
# schedo -o sched_R=5
```

[(R=5)/32=0,15625, (D=16)/32=0,5] установит режим, при котором интерактивный процесс будет выполняться бесконечно, если параллельно запущен фоновый процесс с помощью команды **nice -n 10**. Ограничение в 120 квантов времени означает, что максимальный штраф за использование процессора, который может быть назначен интерактивному процессу, равен 18.

```
# schedo -o sched_R=6 -o sched_D=16
```

В случае [(R=6)/32=0,1875, (D=16)/32=0,5] фоновый процесс, запущенный с помощью команды **nice -n 10**, получит управление не ранее, чем через одну секунду. При этом интерактивный процесс будет сохранять приоритет при использовании процессора. Долго работающие интерактивные процессы (которые могли бы быть запущены в фоновом режиме) получат достаточно большой штраф за использование процессора, чтобы не мешать фоновым процессам.

```
# schedo -o sched_R=32 -o sched_D=32
```

В случае [(R=32)/32=1, (D=32)/32=1] долго работающие нити достигнут показателя использования процессора, равного 120, после чего будут конкурировать только на основе значений nice. Новые нити будут получать более высокий приоритет независимо от значений nice до тех пор, пока штраф за использование процессора не понизит их приоритет до уровня существующих нитей.

Ниже приведены некоторые рекомендации по настройке значений R и D:

- При уменьшении значения R диапазон приоритетов сужается, а значение nice начинает оказывать большее влияние на значение приоритета.
- При увеличении значения R диапазон приоритетов расширяется, и приоритет начинает меньше зависеть от значения nice.
- Уменьшение значения D приводит к тому, что показатель использования процессора затухает скорее, и в итоге нити интенсивно использующие процессор, начинают выполняться быстрее.
- Увеличение значения D приводит к замедлению затухания показателя использования процессора. В итоге нитям, интенсивно использующим процессор, назначается более крупный штраф (за счет этого выигрывают интерактивные нити).

Пример расчета приоритетов:

В данном примере предполагается, что значение R равно 4, значение D равно 31, и в системе не выполняются другие нити.

```
действующий_приоритет
                  | базовый_приоритет
                  |     | значение_nice
                  |     |     | счетчик (затраченных квантов времени)
                  |     |     |     | (schedo -o sched_R)
                  |     |     |     |     |
  время 0          p = 40 + 20 + (0   * 4/32) =   60
  время 10 мс      p = 40 + 20 + (1   * 4/32) =   60
  время 20 мс      p = 40 + 20 + (2   * 4/32) =   60
  время 30 мс      p = 40 + 20 + (3   * 4/32) =   60
  время 40 мс      p = 40 + 20 + (4   * 4/32) =   60
  время 50 мс      p = 40 + 20 + (5   * 4/32) =   60
  время 60 мс      p = 40 + 20 + (6   * 4/32) =   60
  время 70 мс      p = 40 + 20 + (7   * 4/32) =   60
  время 80 мс      p = 40 + 20 + (8   * 4/32) =   61
  время 90 мс      p = 40 + 20 + (9   * 4/32) =   61
  время 100 мс      p = 40 + 20 + (10  * 4/32) =   61
        .
  (пропуск до 1000 мс, или 1 с)
        .
  время 1000 мс    p = 40 + 20 + (100 * 4/32) =   72
  время 1000 мс    планировщик пересчитывает счетчики использования
                  процессора для всех процессов. Для данного процесса:
                  новый_показатель_использования_процессора = 100 * 31/32 = 96  (если d=31)
                  после уменьшения планировщиком: p = 40 + 20 + ( 96 * 4/32) = 72
                  (если d=16, то p = 40 + 20 + (100/2 * 4/32) = 66)
  время 1010 мс    p = 40 + 20 + ( 97 * 4/32) =   72
  время 1020 мс    p = 40 + 20 + ( 98 * 4/32) =   72
  время 1030 мс    p = 40 + 20 + ( 99 * 4/32) =   72
                 ..
  время 1230 мс    p = 40 + 20 + (119 * 4/32) =   74
  время 1240 мс    p = 40 + 20 + (120 * 4/32) =   75     счетчик <= 120
  время 1250 мс    p = 40 + 20 + (120 * 4/32) =   75
  время 1260 мс    p = 40 + 20 + (120 * 4/32) =   75
                 ..
  время 2000 мс    p = 40 + 20 + (120 * 4/32) =   75
  время 2000 мс    планировщик пересчитывает счетчики всех процессов.
                  Для данного процесса 120 * 31/32 = 116
  время 2010 мс    p = 40 + 20 + (117 * 4/32) =   74
```

Изменение кванта времени планировщика с помощью команды schedo

Команда **schedo** позволяет изменить квант времени планировщика. Для этого нужно вызвать команду **schedo -o timeslice=***значение*.

Параметр **-t** задает число тактов в кванте времени. Новое значение кванта времени будет применяться только нитями, для которых установлен алгоритм планирования SCHED_RR (описание нитей с фиксированным приоритетом приведено в разделе “Стратегии планирования для нитей” на стр. 42).

Изменение кванта времени вступает в силу немедленно, так что перезагрузка не требуется.

Нить со стратегией планирования SCHED_OTHER или SCHED_RR может использовать процессор в течение всего кванта времени (по умолчанию квант равен одному такту, длительность которого составляет 10 мс).

В некоторых случаях накладные расходы, связанные с частым переключением контекста нитей и планированием запуска нитей, столь велики, что требуется увеличить квант времени. В таких случаях увеличение кванта времени позволяет повысить скорость выполнения нитей с фиксированным приоритетом. Для того чтобы узнать, сколько раз в секунду переключается контекст, вызовите команду **vmstat** или **sar**.

Не все приложения поддерживают работу в среде с увеличенным квантом времени (для некоторых он будет избыточным, а для некоторых - недопустимым). Такие приложения могут освобождать ресурсы процессора

явным образом с помощью системного вызова **yield()** (как и программы в системе со стандартным квантом времени). После вызова **yield()** нить помещается в очередь выполнения за последней нитью с тем же приоритетом.

Управление ИД пользователей для экономии ресурсов процессора с помощью команды mkpasswd

В системе с большим числом пользователей можно ускорить вход в систему и сэкономить ресурсы процессора за счет применения индексированного файла `/etc/passwd` для поиска идентификаторов пользователей. В этом случае обычный файл `/etc/passwd` сохраняется в системе, но не применяется для обработки запросов на вход в систему.

Для создания индексированного файла применяется команда **mkpasswd**. Если индексированная версия файла устарела, то при входе в систему поиск идентификатора пользователя выполняется путем последовательного перебора записей обычного файла `/etc/passwd`. Для выполнения такого поиска требуется значительно больше ресурсов процессора.

Для создания индексированных версий файла паролей применяется команда **mkpasswd -f**. Эта команда создает индексные версии файлов `/etc/passwd`, `/etc/security/passwd` и `/etc/security/lastlog` . В результате создаются файлы `/etc/passwd.nm.idx`, `/etc/passwd.id.idx`, `/etc/security/passwd.idx` и `/etc/security/lastlog.idx`. Обратите внимание, что создание таких файлов позволяет значительно повысить производительность приложений, для запуска которых требуется ввести зашифрованный пароль (например, процедуры входа в систему и других программ, запрашивающих у пользователя пароль).

Приложение, не запрашивающее у пользователя зашифрованный пароль, будет выполняться быстрее, если для проверки имени и пароля пользователя будет применяться функция **_getpwent()** вместо **getpwent()**, **_getpwnam_shadow(name,0)** вместо **getpwnam(name)** и **_getpwuid_shadow(uid,0)** вместо **getpwuid(uid)**. Это связано с тем, что указанные функции не выполняют поиск в файле `/etc/security/passwd`.

Не изменяйте файлы паролей вручную, так как при этом будет изменено время последнего обновления файла. Это время перестанет совпадать со временем изменения файлов базы данных (`.idx`), и в итоге будет применяться алгоритм поиска по умолчанию (последовательный перебор записей). При вызове команд **passwd**, **mkuser**, **chuser** и **rmuser** (или аналогичных команд SMIT) индексные файлы обновляются автоматически. Если файл `/etc/passwd` был изменен вручную или с помощью команды **pwdadm**, то необходимо обновить индекс.

Примечание: Команда **mkpasswd** не изменяет базы данных пользователей NIS, DCE и LDAP.

Быстродействие памяти

В этом разделе рассмотрены способы оценки и повышения эффективности использования памяти.

Оперативная память системы почти все время заполнена. Даже если выполняемые в данный момент программы не занимают всю оперативную память, операционная система хранит в ней страницы программ, выполнявшихся ранее, и файлы, которые применялись этими программами. Такое хранение не требует затрат, поскольку в противном случае память все равно бы не использовалась. Вместе с тем вполне возможно, что программы или файлы понадобятся вновь, и тогда они будут обработаны значительно быстрее.

Если вы не знакомы с принципами управления виртуальной памятью операционной системы, то сначала ознакомьтесь с разделом “Производительность Администратора виртуальной памяти (VMM)” на стр. 46.

Использование памяти

Сведения об использовании памяти можно получить несколькими способами.

Наиболее полезная информация содержится в выводе команд **vmstat**, **ps** и **svmon**.

Оценка интенсивности использования памяти с помощью команды vmstat

Команда **vmstat** подсчитывает общий объем *активной* виртуальной памяти, занятой всеми процессами в системе, а также число страниц физической памяти в списке свободных страниц.

Объем активной виртуальной памяти определяется как число страниц из рабочих сегментов виртуальной памяти, к которым фактически обращались процессы. Это значение может превосходить объем оперативной памяти компьютера, так как некоторые из страниц активной виртуальной памяти могли быть выгружены в пространство подкачки.

Для того чтобы узнать, хватает ли в системе оперативной памяти и нужно ли настроить некоторые параметры ее использования, запустите команду **vmstat**, задав несколько интервалов сбора данных, а затем проанализируйте значения *pi* и *po* в отчете этой команды. Указанные значения задают число страниц, загружаемых из пространства подкачки и выгружаемых в пространство подкачки в секунду, соответственно. Если эти значения в большинстве случаев отличны от нуля, то вполне вероятно, что недостаток оперативной памяти ограничивает производительность системы. Если в отчете команды есть всего несколько ненулевых значений, то их можно проигнорировать, так как подкачка всегда выполняется в системах с виртуальной памятью.

```
# vmstat 2 10
нити     память             страница          ошибки        cpu
----- ----------- ------------------------ ------------ -----------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
 1  3 113726   124   0  14   6 151  600   0 521 5533 816 23 13  7 57
 0  3 113643   346   0   2  14 208  690   0 585 2201 866 16  9  2 73
 0  3 113659   135   0   2   2 108  323   0 516 1563 797 25  7  2 66
 0  2 113661   122   0   3   2 120  375   0 527 1622 871 13  7  2 79
 0  3 113662   128   0  10   3 134  432   0 644 1434 948 22  7  4 67
 1  5 113858   238   0  35   1 146  422   0 599 5103 903 40 16  0 44
 0  3 113969   127   0   5  10 153  529   0 565 2006 823 19  8  3 70
 0  3 113983   125   0  33   5 153  424   0 559 2165 921 25  8  4 63
 0  3 113682   121   0  20   9 154  470   0 608 1569 1007 15  8  0 77
 0  4 113701   124   0   3  29 228  635   0 674 1730 1086 18  9  0 73
```

В примере, приведенном выше, обратите внимание на время ожидания выполнения ввода-вывода и число блокированных нитей. Ожидание выполнения ввода-вывода может быть связано и с другими операциями, но в данном случае оно с наибольшей вероятностью связано с загрузкой данных в пространство подкачки и выгрузкой данных из него.

Для того чтобы выяснить, связано ли снижение производительности системы с VMM, проанализируйте значения, указанные в разделах *память* и *страница*:

- **память**

  Содержит информацию о физической и виртуальной памяти.

  – **avm**

    В поле `avm` (активная виртуальная память) указано число активных страниц виртуальной памяти на момент сбора сведений командой **vmstat**. Стратегия динамического выделения пространства подкачки применяется по умолчанию. В этом случае значение параметра `avm` может быть больше числа используемых страниц пространства подкачки. В статистике `avm` не учитываются файловые страницы.

  – **fre**

    Значение `fre` отражает среднее число свободных страниц памяти. Страницей называется блок физической памяти размером 4 КБ. В операционной системе создается буфер страниц, называемый списком свободных страниц, который применяется VMM для выделения дополнительной памяти. Минимальное число страниц в этом списке можно задать в параметре *minfree* команды **vmo**. Более подробная информация приведена в разделе “Настройка алгоритма замены страниц VMM” на стр. 153.

    Когда работа приложения завершается, все страницы из его рабочего сегмента памяти добавляются в список свободных страниц. Страницы постоянной памяти (страницы файлов) остаются в оперативной памяти до тех пор, пока они не будут принудительно добавлены в список свободных страниц

приложением VMM для их выделения другим программам. Страницы постоянной памяти автоматически освобождаются только при удалении файла.

По этой причине значение *fre* не позволяет точно оценить объем физической памяти, доступной процессам. Если какому-то приложению требуется страница памяти, то в первую очередь освобождаются страницы постоянной памяти тех приложений, которые уже завершили свою работу.

Если значение *fre* намного больше значения *maxfree*, то маловероятно, что в системе возникнет *перегрузка памяти*. Считается, что память перегружена, если все ресурсы системы уходят на выполнение подкачки. В случае перегрузки памяти значение *fre* всегда очень мало.

- **страница**

  Содержит информацию о страничных ошибках и подкачке страниц. В этом разделе указаны средние значения, подсчитанные за интервал сбора данных. Они отражают число событий, происходящих в секунду.

  – **re**

  Примечание: В настоящий момент это поле не поддерживается.

  – **pi**

  В поле `pi` указывается число страниц, загружаемых из пространства подкачки в секунду. Пространством подкачки называется область виртуальной памяти, расположенная на диске. Страницы выгружаются в пространство подкачки при переполнении оперативной памяти. Пространство подкачки представляет собой набор логических томов, предназначенных для хранения страниц рабочего набора, выгруженных из физической памяти. Когда процесс обращается к выгруженной на диск странице, возникает страничная ошибка. В этом случае страница загружается из пространства подкачки в оперативную память.

  В связи с тем, что существует множество разнообразных конфигураций аппаратного и программного обеспечения, нельзя указать точное значение, которым следует руководствоваться. Данное значение прежде всего отражает интенсивность подкачки. Если страница была загружена в память, значит ранее она была выгружена на диск. Кроме того, в системе с небольшим объемом оперативной памяти для загрузки страницы в оперативную память часто требуется выгрузить одну из страниц на диск.

  – **po**

  В поле *po* указывается число страниц, выгружаемых в пространство подкачки в секунду. При освобождении страницы из рабочего сегмента памяти она записывается в пространство подкачки, если эта страница еще ни разу не выгружалась на диск или была изменена. Если к этой странице не поступит повторных обращений, то она останется на устройстве подкачки до тех пор, пока процесс не завершит свою работу или не освободит занимаемую им память. Все последующие обращения к данным, хранящимся в выгруженных страницах, будут приводить к возникновению страничных ошибок, в результате чего система будет загружать необходимые страницы в память по-отдельности. Если процесс завершает работу в обычном режиме, то область пространства подкачки, выделенная этому процессу, освобождается. Если система загружает в память большое число постоянных страниц (файлов), то значение *po* может возрасти без пропорционального увеличения значения *pi*. Это не обязательно свидетельствует о перегрузке памяти. В этом случае нужно проанализировать шаблоны доступа к данным, применяемые приложениями.

  – **fr**

  Указывает число страниц, освобождаемых в секунду алгоритмом замены страниц. Функция замены страниц VMM выбирает страницы из таблицы страниц PFT для пополнения списка свободных страниц на основании определенного критерия. Этот критерий применяется к страницам из сегментов рабочей памяти и сегментов постоянной памяти. Для освобождения страницы не всегда требуется выполнять операции ввода-вывода. Например, если страница постоянной памяти (страница файла) не была изменена, то она не будет записана на диск. Когда не требуется выполнять операции ввода-вывода, на освобождение страницы затрачиваются минимальные ресурсы системы.

  – **sr**

Указывает число страниц, просматриваемых алгоритмом замены страниц в секунду. При этом соответствие порогам замены страниц может быть получено после проверки большого числа страниц. Чем больше значение **sr** по сравнению с **fr**, тем труднее алгоритму найти подходящие страницы для освобождения.

– **cy**

Указывает число циклов, выполняемых в секунду алгоритмом замены страниц. Этот алгоритм применяется VMM для выбора страниц, которые нужно освободить. Он просматривает флаги обращений к страницам и выбирает те страницы, для которых этот флаг не установлен. Когда вызывается функция добавления страниц, она по кругу просматривает таблицу PTF, проверяя флаги обращений к страницам.

Значение *cy* указывает, сколько раз в секунду алгоритм замены страниц просматривает таблицу PTF. Поскольку обычно список свободных страниц заполняется до того, как будет просмотрена вся таблица PTF, а в выводе команды **vmstat** указываются только целочисленные значения, значение этого параметра чаще всего равно нулю.

Для определения необходимого объема оперативной памяти найдите максимальное значение *avm* в выводе команды **vmstat**. Умножьте это значение на 4 КБ и сравните полученный результат с объемом оперативной памяти системы. В идеале значение, вычисленное на основе *avm*, должно быть меньше объема оперативной памяти. Если это не так, то в системе будет выполняться подкачка. Интенсивность подкачки зависит от разности между этими значениями. Основное назначение виртуальной памяти состоит в расширении адресного пространства системы (часть страниц виртуальной памяти хранится в оперативной памяти, а часть - в пространстве подкачки). Однако если объем виртуальной памяти будет намного превосходить объем физической памяти, то придется выполнять много операций подкачки, что приведет к значительным задержкам в работе системы. Если в системе выполняется подкачка, несмотря на то, что значение *avm* меньше объема оперативной памяти, то, скорее всего, оперативная память переполнена файловыми страницами. В этом случае число операций подкачки можно уменьшить, изменив значения *minperm*, *maxperm* и *maxclient*. За дополнительной информацией обратитесь к разделу “Настройка алгоритма замены страниц VMM” на стр. 153.

Команда vmstat -I:

Команда **vmstat -I** позволяет получить дополнительную информацию об использовании оперативной памяти, включая сведения о количестве файловых страниц, загружаемых в оперативную память и выгружаемых из оперативной памяти в секунду, т.е. число операций загрузки и выгрузки страниц, не связанных с пространством подкачки.

В выводе команды, вызванной с этим флагом, отсутствуют столбцы `re` и `cy`.

Команда vmstat -s:

Команда, вызванная с опцией **-s**, записывает в стандартный поток вывода итоговый отчет, который содержит суммарные значения показателей использования памяти, подсчитанные за все время, прошедшее с момента инициализации системы.

Эту команду рекомендуется вызывать дважды: до запуска оцениваемой рабочей схемы и после выполнения этой рабочей схемы. Дальше нужно оценить разницу между значениями, полученными в двух отчетах команды. В разделе “Неполадки, связанные с диском или памятью” на стр. 36 приведен сценарий **awk** с именем **vmstatit**, который автоматически создает список различий.

```
# vmstat -s
  3231543 ошибок преобразования адреса
    63623 страниц загружено
   383540 страниц выгружено
      149 страниц загружено из пространства подкачки
      832 страниц выгружено в пространство подкачки
        0 восстановлений
   807729 ошибок при обращении к страницам, заполненных нулями
     4450 ошибок при обращении к страницам с исполняемым кодом
```

```
   429258 страниц проверено алгоритмом замены страниц
        8 циклов алгоритма замены страниц
   175846 страниц освобождено алгоритмом замены страниц
    18975 операций возврата
        0 неудачных захватов блокировок
       40 задержек из-за ожидания свободной страницы
        0 задержек, связанных с расширенным XPT
    16984 ожидающих операций ввода-вывода
   186443 запущенных операций ввода-вывода
   186443 выполненных операций ввода-вывода
141695229 переключений контекста процессора
317690215 прерываний устройств
        0 программных прерываний
        0 условных прерываний
 55102397 системных вызовов
```

Число загруженных и выгруженных страниц отражает интенсивность обмена страницами с областью подкачки и постоянной памятью. Число загруженных и выгруженных страниц из пространства подкачки учитывает только страницы, хранящиеся в пространстве подкачки.

Оценка интенсивности использования памяти с помощью команды ps

Команда **ps** позволяет получить информацию об использовании оперативной памяти для отдельного процесса.

Команда **ps v** *ИД-процесса* создает отчет с подробной информацией об использовании памяти указанным процессом. Этот отчет содержит:

- Число страничных ошибок
- Общий объем страниц из рабочего сегмента памяти, к которым обращался процесс
- Объем рабочего сегмента памяти и сегмента, содержащего двоичный код программы
- Размер сегмента, содержащего исходный текст программы
- Размер постоянной области памяти
- Долю оперативной памяти, занятую данным процессом (в процентах)

Пример приведен ниже:

```
# ps v
   PID    TTY STAT  TIME PGIN  SIZE   RSS   LIM  TSIZ   TRS %CPU %MEM COMMAND
 36626  pts/3 A     0:00    0   316   408 32768    51    60  0.0  0.0 ps v
```

Ниже описаны те поля отчета команды **ps**, на которые стоит обратить внимание:

PGIN Число операций чтения страниц, вызванных страничными ошибками. Поскольку все операции ввода-вывода связаны со страничными ошибками, данный параметр является основной характеристикой интенсивности ввода-вывода.

SIZE Размер области данных процесса в пространстве подкачки в КБ (при вызове команды с другими флагами это значение указывается в поле SZ). Данное значение равно числу страниц из рабочего сегмента памяти, к которым обращался процесс, умноженному на 4. Если некоторые страницы из рабочего сегмента были выгружены на диск, то это значение будет больше объема физической памяти. Значение SIZE учитывает страницы из сегмента процесса и из сегмента данных общих библиотек, используемых процессом.

RSS Объем физической памяти, занятой процессом, в килобайтах. Это значение равно числу страниц из рабочего сегмента памяти и из сегмента кода программы, находящихся в оперативной памяти, умноженному на 4. Не забывайте, что сегмент кода программы применяется всеми запущенными экземплярами программы. Если запущено 26 процессов **ksh**, в оперативной памяти будет находиться только одна копия страниц с исполняемым кодом **ksh**, однако в выводе команды **ps** размер этих страниц будет прибавлен к значению RSS всех экземпляров программы **ksh**.

TSIZ Размер текста (общего для всех экземпляров программы). Эта величина представляет собой размер

текстовой части исполняемого файла программы. Страницы с текстовой частью исполняемой программы загружаются в память только при обращении к этим страницам. Данная величина представляет собой возможный верхний предел количества загруженного текста. Значение TSIZ не отражает фактического объема занятой памяти. Значение TSIZ можно узнать и с помощью команды **dump -ov**, указав в качестве параметра исполняемый файл (например, **dump -ov** `/usr/bin/ls`).

TRS Размер резидентной (размещенной в реальной памяти) командной части программы. Это значение равно числу страниц с кодом программы, умноженному на 4. Это значение искажает фактический объем памяти, занятый программой, если запущено несколько экземпляров программы. Значение TRS может превосходить значение TSIZ за счет того, что сегмент с кодом программы содержит не только текст программы, но и заголовок XCOFF, а также описание загружаемых библиотек.

%MEM
Это значение указывает, какую долю (в процентах) от общего объема оперативной памяти компьютера составляет значение RSS. Это значение должно характеризовать долю физической памяти, занятой процессом. К сожалению, как и RSS, эта величина переоценивает загрузку памяти процессами, которые совместно используют командную часть программы. Кроме того, округление до ближайшего целого значения приводит к тому, что значение %MEM будет равно 0,0, если отношение RSS к объему оперативной памяти меньше 0,005.

Примечание: Команда **ps** не позволяет узнать размер общих сегментов памяти и закрепленных сегментов памяти. Поскольку многие приложения пользуются общей памятью и закрепленными сегментами, сведения о сегментах проще всего получать с помощью команды **svmon**.

Команда svmon

Команда **svmon** позволяет получить более подробную информацию об использовании памяти. В то же время, на ее выполнение затрачивается больше ресурсов системы, чем на выполнение команд **vmstat** и **ps**. Команда **svmon** собирает информацию о текущем состоянии оперативной памяти. Тем не менее, нельзя считать, что значения всех показателей были измерены в одно и то же время, поскольку эта команда выполняется в пользовательском режиме, допускающем прерывания.

Для того чтобы узнать, установлена ли в системе программа **svmon**, вызовите следующую команду:

```
# lslpp -lI bos.perf.tools
```

Для вызова команды **svmon** необходимы права доступа пользователя root.

Если при вызове команды с помощью опции **-i** был задан интервал сбора данных то статистика будет выдаваться до тех пор, пока процесс команды не будет убит, либо пока не будет достигнуто указанное число интервалов сбора информации (если оно задано).

На основе собранной информации можно создать отчеты следующих типов:

Глобальный (-G)
Содержит информацию о том, какой суммарный объем физической памяти и пространства подкачки занят всеми процессами системы.

Процесс (-P)
Содержит информацию об использовании памяти указанными активными процессами. Если список процессов не указан, то отображается информация о всех активных процессах.

Сегмент (-S)
Содержит информацию об использовании памяти для указанных сегментов. Если список сегментов не указан, то отображается информация о всех определенных сегментах.

Подробный отчет о сегментах (-D)
Содержит подробную информацию об указанных сегментах.

Пользователь (-U)

Содержит информацию об использовании памяти для указанных пользователей. Если список имен не задан, то выводится информация для всех пользователей, определенных в системе.

Команда (-C)

Содержит информацию об объеме памяти, выделенном для процесса указанной команды.

Класс WLM (-W)

Содержит информацию об использовании памяти для указанных классов управления рабочей схемой. Если классы не заданы, то выводится информация для всех классов, определенных в системе.

Страница (-F)

Содержит информацию об использовании страниц памяти. Если номер страницы не задан, то выводится доля занятой оперативной памяти в процентах. Учитываются только страницы, содержащие флаг обращения к странице. В процессе выполнения все флаги обращения к страницам сбрасываются. Таким образом при повторном применении опции **-f** команда **svmon** выводит долю физической памяти в процентах, доступ к которой осуществлялся с момента предыдущего применения опции **-f**. Если в системе определен зарезервированный пул, выводится соответствующее значение для каждого определенного пула.

Приоритет (-T)

Содержит информацию о приоритетах, в том числе значение приоритета, суперкласс (если задан флаг **-a**) и общее число страниц физической памяти, расположенных в сегментах с данным приоритетом.

Текущий объем памяти:

Команда **svmon** позволяет просмотреть объем используемой памяти.

Для того чтобы просмотреть глобальную статистику, вызовите команду с флагом **-G**. В приведенном ниже примере она выполняется дважды с интервалом в одну секунду.

```
# svmon -G -i 1 2

               разм.     занято      своб.      закр.     вирт.
память      1048576     425275     623301      66521     159191
пр. подкачки 262144      31995

               раб.       пост.       клнт.
закр.         46041          0          0
занято       129600     275195          0

разм.стр.   разм.пула   занято   пр.пдкчк.      закр.      вирт.
мал. 4 КВ          -     404795      31995      46041     159191
L  16 МБ           5          0          0          5          0
               разм.     занято      своб.      закр.     вирт.
память      1048576     425279     623297      66521     159195
пр. подкачки 262144      31995

               раб.       пост.       клнт.
закр.         46041          0          0
занято       129604     275195          0

разм.стр.   разм.пула   занято   пр.пдкчк.      закр.      вирт.
мал. 4 КВ          -     404799      31995      46041     159195
L  16 МБ           5          0          0          5          0
```

Обратите внимание на то, что если в системе доступны только страницы размером 4 КБ, раздел, делящий информацию в соответствии с размером страниц, не отображается.

Ниже описаны поля отчета команды **svmon**:

память

Статистическая информация об использовании физической памяти. Все значения представляют собой число страниц размером по 4 КБ.

размер Общий размер памяти в страницах по 4 КБ.

занято Число страниц оперативной памяти, занятых процессом, плюс число страниц постоянной памяти, принадлежащих завершенному процессу, и еще не выгруженных из памяти. Это значение равно разности между общим числом страниц в оперативной памяти и числом страниц в списке свободных страниц.

своб Число страниц в списке свободных страниц.

закр Число страниц, закрепленных в оперативной памяти (закрепленной называется страница, постоянно расположенная в оперативной памяти; такая страница не выгружается на диск).

вирт Число страниц, размещенных в виртуальной памяти процесса.

пр.пдкч.

Статистическая информация об использовании пространства подкачки. Все значения задают число страниц размером по 4 КБ. В этом разделе отчета указывается фактическое число страниц, занятых в пространстве подкачки, т.е. количество страниц, которые были выгружены в пространство подкачки. Это значение не совпадает со значением `avm` из вывода команды **vmstat**, задающим число страниц виртуальной памяти, к которым поступали обращения, так как не все такие страницы при вызове **vmstat** выгружаются на диск.

размер Общий размер пространства подкачки в страницах по 4 КБ.

занято Общее число выделенных страниц.

закр Статистическая информация об области физической памяти, содержащей закрепленные страницы. Все значения представляют собой число страниц размером по 4 КБ.

раб Число рабочих страниц, закрепленных в оперативной памяти.

пост Число постоянных страниц, закрепленных в оперативной памяти.

клнт Число страниц клиентов, закрепленных в оперативной памяти.

занято Статистическая информация об области занятой физической памяти. Все значения представляют собой число страниц размером по 4 КБ.

раб Число рабочих страниц в оперативной памяти.

пост Число постоянных страниц в оперативной памяти.

клнт Число страниц клиентов в оперативной памяти (страница клиента - это страница удаленного файла).

разм.стр.

Доступна только в том случае, если в системе присутствуют страницы с размером, отличным от 4 КБ. Показывает данные по каждому из размеров страниц, доступных в системе.

разм.стр.
Размер страницы

разм.пула
Число страниц в зарезервированном пуле памяти.

занято Число занятых страниц

пдкч Число страниц, размещенных в пространстве подкачки

закр Число закрепленных страниц

вирт Число страниц, размещенных в виртуальной памяти системы.

В данном примере рассматривается 1 048 576 страниц памяти. Для того чтобы определить общий объем физической памяти, это значение нужно умножить на 4096 (получится 4 ГБ). 425 275 страниц занято, 623 301 страница находится в списке свободных страниц, и 66 521 страница закреплена в оперативной памяти. Среди занятых страниц 129 600 страниц относятся к рабочим сегментам памяти, 275 195 страниц относятся к сегментам постоянной памяти, а 0 страниц являются страницами клиентов. Сумма этих трех значений, а также памяти, зарезервированной, но, возможно, не используемой зарезервированными пулами, дает значение *занято* из раздела *память*. В разделе *закр.* указано, какая часть закрепленных страниц является рабочими, постоянными и клиентскими. Сумма этих значений, а также памяти в зарезервированных пулах, дает значение *pin* в разделе *memory*. Пространство подкачки содержит 262 144 страниц (1 ГБ), 31995 из которых занято. Как правило, значение *занято* из раздела *память* больше значения *занято* из раздела *пр. подкачки*, так как страницы файлов не освобождаются после завершения программы, в отличие от страниц, выделенных из пространства подкачки.

Использование памяти процессами:

Команда **svmon -P** позволяет просмотреть статистику использования памяти для всех запущенных процессов системы.

Ниже приведен пример вывода команды **svmon -P**:

```
# svmon -P

--------------------------------------------------------------------------------
     Pid Command          Inuse        Pin     Pgsp    Virtual 64-bit Mthrd 16MB
   16264 IBM.ServiceRM    10075     3345     3064    13310      H     Д     H

     Разм.Стр.     Занято      Закр.   Пр.пдкчк. Виртуальн.
     мал.  4 КВ     10075       3345       3064      13310
     L  16 МБ          0          0          0          0

    Vsid      Esid Тип  Описание                PSize  Inuse   Pin Pgsp Virtual
   f001e         d раб. текст общ. библ.            s   4857     0   36  6823
       0         0 раб. сегм. ядра                  s   4205  3335 2674  5197
   b83f7         2 раб. частн. процесс              s    898     2  242  1098
   503ea         f раб. данные общ. библ.           s     63     0   97   165
   c8439         1 лич. код,/dev/hd2:149841         s     28     0    -     -
   883f1         - раб.                             s     21     8   14    26
   e83dd         - лич. /dev/hd2:71733              s      2     0    -     -
   f043e         4 раб. сегм. общ. пам.             s      1     0    1     1
   c0438         - лич. больш. файл /dev/hd9var:243 s      0     0    -     -
   b8437         3 свз. связ. с sid a03f4           s      0     0    -     -
   583eb         - лич. больш. файл /dev/hd9var:247 s      0     0    -     -

--------------------------------------------------------------------------------
     Pid Command          Inuse        Pin     Pgsp    Virtual 64-bit Mthrd 16MB
   17032 IBM.CSMAgentR     9791     3347     3167    12944      H     Д     H

     Разм.Стр.     Занято      Закр.   Пр.пдкчк. Виртуальн.
     мал. 4 КВ       9791      3347       3167      12944
     L  16 МБ          0          0          0          0

    Vsid      Esid Тип  Описание                PSize  Inuse   Pin Pgsp Virtual
   f001e         d раб. текст общ. библ.            s   4857     0   36  6823
       0         0 раб. сегм. ядра                  s   4205  3335 2674  5197
     400         2 раб. частн. процесс              s    479     2  303   674
   38407         f раб. данные общ. библ.           s    120     0  127   211
   a83f5         1 лич. код,/dev/hd2:149840         s     99     0    -     -
   7840f         - раб.                             s     28    10   27    39
   e83dd         - лич. /dev/hd2:71733              s      2     0    -     -
   babf7         - лич. /dev/hd2:284985             s      1     0    -     -
   383e7         - лич. больш. файл /dev/hd9var:186 s      0     0    -     -
   e03fc         - лич. больш. файл /dev/hd9var:204 s      0     0    -     -
   f839f         3 свз. связ. с sid 5840b           s      0     0    -     -
[...]
```

Вывод команды содержит подробные сведения как о глобальном использовании памяти процессами, так и сведения о том, как каждый процесс использует различные сегменты памяти. По умолчанию процессы сортируются по убыванию числа занятых страниц. Правило сортировки можно изменить с помощью команды **svmon** с любым из следующих флагов: **-u**, **-p**, **-g** или **-v**.

Просмотреть общие сведения о 15 процессах, использующих самый большой объем памяти системы, можно с помощью следующей команды:

```
# svmon -Pt15 | perl -e 'while(<>){print if($.==2||$&&&!$s++);$.=0 if(/^-+$/)}'
-------------------------------------------------------------------------------
     Pid Command          Inuse        Pin     Pgsp    Virtual 64-bit Mthrd 16MB
   16264 IBM.ServiceRM    10075     3345     3064    13310      Y     N     Y
   17032 IBM.CSMAgentR     9791     3347     3167    12944      Y     N     Y
   21980 zsh               9457     3337     2710    12214      Y     Y     Y
   22522 zsh               9456     3337     2710    12213      Y     Y     Y
   13684 getty             9413     3337     2710    12150      Y     Y     Y
   26590 perl5.8.0         9147     3337     2710    12090      Y     Y     Y
    7514 sendmail          9390     3337     2878    12258      Y     Y     Y
   14968 rmcd              9299     3340     3224    12596      Y     N     Y
   18940 ksh               9275     3337     2710    12172      Y     Y     Y
   14424 ksh               9270     3337     2710    12169      Y     Y     Y
    4164 errdemon          9248     3337     2916    12255      Y     Y     Y
    3744 cron              9217     3337     2770    12125      Y     Y     Y
   11424 rpc.mountd        9212     3339     2960    12290      Y     N     Y
   21564 rlogind           9211     3337     2710    12181      Y     Y     Y
   26704 rlogind           9211     3337     2710    12181      Y     Y     Y
```

Процесс с pid 16264 занимает больше всего памяти. В столбце Команда указывается имя команды, в данном случае - IBM.ServiceRM. В столбце Занято указывается общее число страниц физической памяти, расположенных в сегментах процесса. В данном случае оно равно 10075. Размер страницы - 4 КБ. В столбце Закр. указывается общее число закрепленных страниц, расположенных в сегментах процесса. В данном случае оно равно 3345. В столбце Пр. подкачки указывается общее число страниц пространства подкачки, выделенных процессу. В данном случае оно равно 3064. Столбец Вирт. (общее число страниц виртуальной памяти процесса) содержит значение 13310.

Во втором разделе вывода содержится информация о сегментах процессов, указанных в первом разделе вывода. К ним относятся идентификаторы виртуальных (ВСег) и эффективных (Эсег) сегментов. Значение ЭСег соответствует реестру сегментов, который применяется для доступа к соответствующим страницам. Эта информация включает в себя тип сегмента и его описание (содержащее имя тома и i-узел файла в случае сегмента постоянной памяти). Кроме того, отчет содержит информацию о размере страниц сегмента, где s указывает на страницы по 4 КБ, а L - на страницы по 16 МБ, число страниц в оперативной памяти (Занято), число закрепленных страниц в памяти (Закр.), число страниц в пространстве подкачки (Пр.пдкчк.), а также число виртуальных страниц (Виртуальн.).

Получить более подробную информацию можно с помощью дополнительных опций. Опция **-j** позволяет просмотреть полное имя файла для существующих сегментов. С помощью опции **-l** можно просмотреть подробные сведения о сегментах, а с помощью опции **-r** - диапазоны памяти, используемые каждым сегментом. Ниже приведен пример команды **svmon** с опциями **-l**, **-r** и **-j**:

```
# svmon -S f001e 400 e83dd -l -r -j

    Vsid      Esid Тип  Описание                PSize  Inuse   Pin Pgsp Virtual
   f001e         d раб. текст общ. библ.            s   4857     0   36  6823
                   Диап. адр.: 0..60123
                   Сегмент текста общ. библ.
     400         2 раб. частн. процесс              s    480     2  303   675
                   Диап. адр.: 0..969 : 65305..65535
                   pid(s)=17032
   e83dd         - лич. /dev/hd2:71733              s      2     0    -     -
                        /usr/lib/nls/loc/uconvTable/ISO8859-1
                   Диап. адр.: 0..1
                   pid(s)=17552, 17290, 17032, 16264, 14968, 9620
```

В поле Диапазон адресов указывается один диапазон для сегментов постоянной памяти и сегментов клиентов, и два диапазона для рабочих сегментов. Диапазон адресов сегментов постоянной памяти и сегментов клиентов имеет вид '0..x,' где x - максимальное число виртуальных страниц, когда либо выделявшихся процессу. Для рабочих сегментов указывается значение вида '0..x : y..65535', где диапазон 0..x содержит глобальные данные, а диапазон y..65535 содержит стек. У первого диапазона изменяется правая граница, а у второго - левая. Таким образом, пространство адресов рабочего сегмента представляет собой два диапазона, которые увеличиваются навстречу друг другу. Все вышесказанное относится только к личным рабочим сегментам процесса и не относится к сегментам ядра и общих библиотек.

В приведенном выше примере указан личный рабочий сегмент процесса 400; его диапазон адресов равен 0..969 : 65305..65535. В сегменте f001e хранится рабочий текстовый сегмент общей библиотеки; его диапазон равен 0..60123.

Некоторые сегменты могут применяться несколькими процессами. В этом случае все страницы из этого сегмента будут прибавлены к значению Занято каждого из процессов. Таким образом, сумма всех значений Занято может быть больше общего числа страниц физической памяти. То же самое относится к полям Пр. подкачки и Закр.. Значения в итоговом разделе равны сумме полей Занято, Закр. и Пр. подк., а также значений Виртуальн. всех используемых процессами сегментов.

В приведенном выше примере сегмент e83dd используется несколькими процессами со следующими PID: 17552, 17290, 17032, 16264, 14968 и 9620.

Подробная информация об отдельных сегментах:

Опция **-D** позволяет получить статистику использования памяти в отдельных сегментах.

Пример приведен ниже:

```
# svmon -D 38287 -b
ИД сегм: 38287
Тип:  рабочий
Размер страницы: м (4 КБ)
Диапазон адресов: 0..484
Выделено в пространстве подкачки: 2 страницы (0,0 МБ)
Выделено в виртуальной памяти: 18 фреймов (0,0 МБ)
Занято: 16 фреймов (0,0 МБ)

          Стран. разм.стр. Фрейм   Закр.      Отн.      Изм.    ИДВнешСег   ВнешСтр
           341    м       527720    Н          Н         Н         -          -
           342    м       996079    Н          Н         Н         -          -
           343    м       524936    Н          Н         Н         -          -
           344    м       985024    Н          Н         Н         -          -
           347    м       658735    Н          Н         Н         -          -
           348    м       78158     Н          Н         Н         -          -
           349    м       174728    Н          Н         Н         -          -
           350    м       758694    Н          Н         Н         -          -
           404    м       516554    Н          Н         Н         -          -
           406    м       740622    Н          Д         Н         -          -
           411    м       528313    Н          Д         Д         -          -
           412    м       1005599   Н          Д         Н         -          -
           416    м       509936    Н          Н         Д         -          -
           440    м       836295    Н          Н         Д         -          -
           443    м       60204     Н          Н         Д         -          -
           446    м       655288    Н          Н         Д         -          -
```

Ниже приведено описание столбцов вывода:

Стран. Задает индекс страницы в сегменте памяти.

Разм.стр.
Задает размер страницы (**м** для 4 КБ, **ср** для 64 КБ, **Б** для 16 МБ и **О** для 16 ГБ).

Фрейм Задает индекс страницы физической памяти, которая соответствует странице сегмента.

Закр. Флаг, указывающий, закреплена ли страница.

Отн. Задается только с флагом **-b**. Флаг, указывающий, установлен ли бит обращения к странице.

Изм. Задается только с флагом **-b**. Флаг, указывающий, изменена ли страница.

ИДВнешСег
В случае, если страница относится к внешнему сегменту, связанному с проверяемым, в этом поле указывается виртуальный идентификатор внешнего сегмента.

ВнешСтр
В случае, если страница относится к внешнему сегменту, связанному с проверяемым, в этом поле индекс страницы этого внешнего сегмента.

Если внешний сегмент связан с проверяемым, отчет будет выглядеть примерно следующим образом:

```
 Стран. разм.стр. Фрейм   Закр.    Отн.      Изм.    ИДВнешСег   ВнешСтр
65574     м       345324     Н          Н          Н    288071         38
65575     м       707166     Н          Н          Н    288071         39
65576     м       617193     Н          Н          Н    288071         40
```

Вызвав команду с флагом **-b**, можно просмотреть флаги обращения к страницам и изменения страниц. После вывода этой информации флаг обращения к странице сбрасывается. Если дополнительно будет указана опция **-i**, то флаг обращения будет установлен для тех страниц, к которым процесс обращался в течение последнего интервала сбора данных.

Примечание: При использовании флага **-b** необходимо проявлять повышенное внимание, поскольку это может привести к снижению производительности.

Получение списка сегментов, занимающих больше всего памяти:

Команда с опцией **-S** позволяет получить список сегментов, отсортированных по объему занимаемой памяти, со статистическими данными для выбранных сегментов. Если список сегментов не указан, то отображается информация о всех определенных сегментах.

Приведенная ниже команда сортирует системные и прочие сегменты по числу страниц, находящихся в физической памяти. Опция **-t** позволяет ограничить число отображаемых сегментов указанным числом. Флаг **-u** позволяет отсортировать вывод в порядке убывания по общему числу страниц, находящихся в физической памяти.

Ниже приведен пример вывода команды **svmon** с опциями **-S**, **-t** и **-u**:

```
# svmon -Sut 10

    Vsid      Esid Тип  Описание                 PSize  Inuse   Pin Pgsp Virtual
   70c4e         - лич. больш. файл /dev/lv01:26     s  84625     0    -     -
   22ec4         - раб.                              s  29576     0    0 29586
   8b091         - лич. /dev/hd3:123                 s  24403     0    -     -
   7800f         - раб. куча ядра                    s  22050  3199 19690 22903
   a2db4         - лич. /dev/hd3:105                 s  15833     0    -     -
   80010         - раб. таблица фреймов страницы     s  15120 15120    0 15120
   7000e         - раб. разн. таблицы ядра           s  13991     0 2388 14104
   dc09b         - лич. /dev/hd1:28703               s   9496     0    -     -
   730ee         - лич. /dev/hd3:111                 s   8568     0    -     -
   f001e         - раб.                              s   4857     0   36  6823
```

Зависимость между выводом команд svmon и vmstat

Отчеты команд **svmon** и **vmstat** связаны между собой.

Ниже приведен пример вывода команды **svmon**:

```
# svmon -G
               разм.     занято      своб.      закр.     вирт.
память      1048576     417374     631202      66533     151468
пр. подкачки 262144      31993

               раб.       пост.       клнт.
закр.         46053           0           0
занято       121948     274946           0

разм.стр.   разм.пула   занято   пр.пдкчк.      закр.      вирт.
s   4 КБ           -     397194     262144      46053     151468
L  16 МБ           5          0          0          5          0
```

Команда **vmstat** была запущена в другом окне параллельно с командой **svmon**. Ниже приведен пример отчета **vmstat**:

```
# vmstat 3
нити     память              страница          ошибки        cpu
----- ----------- ------------------------ ------------ -----------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
 1  5 205031 749504   0   0   0   0    0   0 1240  248 318  0  0 99  0
 2  2 151360 631310   0   0   3   3   32   0 1187 1718 641  1  1 98  0
 1  0 151366 631304   0   0   0   0    0   0 1335 2240 535  0  1 99  0
 1  0 151366 631304   0   0   0   0    0   0 1303 2434 528  1  4 95  0
 1  0 151367 631303   0   0   0   0    0   0 1331 2202 528  0  0 99  0
```

Глобальный отчет команды **svmon** содержит аналогичные значения. Столбец fre вывода команды **vmstat** связан со столбцом свобод. память команды **svmon**. Значение avm (активная виртуальная память) в отчете команды **vmstat** соответствует значению в отчете команды **svmon**.

Зависимость между выводом команд svmon и ps

Выводы команд **svmon** и **ps** связаны между собой.

Пример 1

Ниже приведен пример вывода команд **svmon** и **ps**:

```
# # ps v 405528
    PID    TTY STAT  TIME PGIN  SIZE   RSS   LIM  TSIZ   TRS %CPU %MEM COMMAND
 405528  pts/0 A    43:11    1   168   172 32768     1     4 99.5  0.0 yes

(0) root @ clock16: 6.1.2.0: /
# svmon -O unit=KB,segment=category,filtercat=exclusive -P 405528
Ед. изм.: Кб
-------------------------------------------------------------------------------
     Pid Command          Inuse      Pin     Pgsp  Virtual
  405528 yes                172       16        0      168

................................................................................
Сегменты EXCLUSIVE                  Inuse      Pin     Pgsp  Virtual
                                       172       16        0      168

    Vsid      Esid Тип  Описание                PSize  Inuse   Pin Pgsp Virtual
   554f1         f раб. данные общ. библ.            s     92     0    0      92
   49416         2 раб. проц. прив.                  s     76    16    0      76
   6d49f         1 clnt code,/dev/hd2:338            s      4     0    -       -
```

Команда **ps** показывает, что SIZE имеет значение 168, RSS - 172. При использовании команды **svmon** выводятся оба значения.

Значения из вывода команды **svmon** можно использовать в следующем равенстве для вычисления SIZE и RSS:

```
SIZE = Используемая рабочим процессом собственная память в Кб + Рабочие данные общей библиотеки в Кб
RSS = SIZE + размер кода в текстовом формате (Type=clnt, Description=code,)
```

Значения из примера дают следующие результаты:

```
SIZE = 92 + 76 = 168
RSS = 168 + 4 = 172
```

Пример 2

Ниже приведен пример вывода команд **svmon** и **ps**:

```
# ps v 282844
    PID    TTY STAT  TIME PGIN  SIZE   RSS   LIM  TSIZ   TRS %CPU %MEM COMMAND
 282844      - A    15:49  322 24604 25280    xx   787   676  0.0  3.0 /opt/rsct/b

(0) root @ clock16: 6.1.2.0: /
# svmon -O unit=KB,segment=category,filtercat=exclusive -P 282844
Ед. изм.: Кб
-------------------------------------------------------------------------------
     Pid Command          Inuse      Pin     Pgsp  Virtual
  282844 IBM.CSMAgentR    25308       16        0    24604

...............................................................................
Сегменты EXCLUSIVE                  Inuse      Pin     Pgsp  Virtual
                                    25308       16        0    24604

    Vsid      Esid Тип  Описание                PSize  Inuse   Pin Pgsp Virtual
   2936e         2 раб. проц. прив.                s  23532    16    0   23532
   2d36f         f раб. данные общ. библ.          s   1072     0    0    1072
    1364         1 clnt code,/dev/hd2:81988        s    676     0    -       -
   154c1         - clnt /dev/hd9var:353            s     16     0    -       -
   41494         - clnt /dev/hd2:82114             s      8     0    -       -
   4d3d7         - clnt /dev/hd9var:357            s      4     0    -       -
   7935a         - clnt /dev/hd9var:307            s      0     0    -       -
   4d377         3 mmap maps 2 source(s)           s      0     0    -       -
   3934a         - clnt /dev/hd9var:300            s      0     0    -       -
```

Команда **ps** показывает, что SIZE имеет значение 24604, RSS - 25280.

Значения из вывода команды **svmon** можно использовать в следующем равенстве для вычисления SIZE и RSS:

```
SIZE = Используемая рабочим процессом собственная память в Кб + Рабочие данные общей библиотеки в Кб
RSS = SIZE + размер кода в текстовом формате (Type=clnt, Description=code,)
```

Значения из примера дают следующие результаты:

```
SIZE = 23532 + 1072 = 24604
RSS = 24604 + 676 = 25280
```

Вычисление минимального объема памяти

Минимальный объем памяти, необходимый для работы программы, вычисляет по простой формуле.

```
Общее число страниц (в блоках по 4 КБ) = T + ( N * ( PD + LD )) + F
```

где:

T = Число страниц с текстом программы (общих для всех пользователей)

N = Число одновременно выполняемых экземпляров программы

PD = Число страниц в личном сегменте рабочей памяти процесса

LD = Число страниц с данными общих библиотек, применяемых процессом

F = Число страниц файлов (общих для всех пользователей)

Для того чтобы получить объем памяти в килобайтах, умножьте полученное значение на 4. К полученному значению можно добавить размер сегментов ядра, расширения ядра и текстовых сегментов общих библиотек, однако учтите, что эти сегменты применяются всеми процессами системы. Например, некоторые

приложения, в частности, CATIA и базы данных, применяют очень большие модули общих библиотек. Поскольку эта формула вычисляется на основе показателей использования памяти, измеренных один раз в течение работы процесса, полученный результат не всегда дает правильную оценку минимального размера рабочего набора процесса. Для того чтобы получить более точный результат, необходимо воспользоваться утилитой **rmss**, либо измерить показатели использования памяти несколько раз в течение работы процесса, а затем подсчитать средние значения. Дополнительная информация приведена в разделе “Оценка необходимого объема памяти с помощью команды rmss” на стр. 143.

Программы с утечками памяти

Утечкой памяти называется ошибка в программе, заключающаяся в том, что программа многократно запрашивает память, не освобождая ее.

Утечка памяти в долго работающей программе, например, в интерактивном приложении, представляет серьезную проблему, поскольку может привести к фрагментации памяти и накоплению в физической памяти большого количества страниц, заполненных "мусором". Известны случаи, когда утечка памяти в одной программе приводит к переполнению пространства подкачки.

Для обнаружения утечек памяти вызовите команду **svmon** и найдите процессы, рабочие сегменты которых постоянно увеличиваются. Утечка в сегменте ядра может быть вызвана утечкой mbuf, утечкой в драйвере устройства, в расширении ядра и даже в ядре. Для того чтобы найти постоянно растущие сегменты памяти, вызовите команду **svmon** с флагом **-i** и проанализируйте изменение размера сегментов отдельных процессов и групп процессов.

Определить конкретную функцию или строку кода, служащую причиной утечки памяти, намного сложнее. Особенно трудно это сделать в приложениях AIXwindows, которые содержат большое число вызовов **malloc()** и **free()**. Для анализа использования памяти и обнаружения утечек в приложениях на C++ предусмотрена программа HeapView Debugger. Кроме того, существуют программы других фирм для анализа утечек памяти, однако для их работы необходим исходный код программы.

Некоторые вызовы **realloc()**, сами по себе не являясь ошибкой, также могут служить причинами утечки памяти. Если программа часто вызывает функцию **realloc()** для увеличения размера области данных и память, освобожденная функцией **realloc()**, не используется программой для других целей, то рабочий сегмент процесса становится сильно фрагментированным.

Для освобождения памяти рекомендуется использовать функции **disclaim()** и **free()**. Функция **disclaim()** должна вызываться перед функцией **free()**. Однако освобождать память, выделенную функцией **malloc()**, не нужно, если эта функция вызывается незадолго до завершения программы. Это связано с тем, что при завершении программы ее рабочий сегмент памяти удаляется, а страницы физической памяти, содержащие данные из рабочего сегмента, добавляются в список свободных страниц. Ниже приведен пример анализа программы с утечкой памяти. Значения показателей `Inuse`, `Pgspace` и `Address Range` личного рабочего сегмента этой программы все время увеличиваются:

```
# svmon -P 13548 -i 1 3
  Pid Command        Inuse      Pin     Pgsp  Virtual   64-bit    Mthrd    LPage
13548                         pacman         8535        2178      847     8533      N     N     N

Vsid     Esid Type  Description          LPage  Inuse    Pin  Pgsp  Virtual
    0        0 work  kernel seg               -   4375   2176   847     4375
48412        2 work  process private          -   2357      2     0     2357
6c01b        d work  shared library text      -   1790      0     0     1790
4c413        f work  shared library data      -     11      0     0       11
3040c        1 pers  code,/dev/prodlv:4097    -      2      0     -        -
ginger :svmon -P 13548 -i 1 3

  Pid Command        Inuse      Pin     Pgsp  Virtual   64-bit    Mthrd    LPage
13548                         pacman         8589        2178      847     8587      N     N     N

Vsid     Esid Type  Description          LPage  Inuse    Pin  Pgsp  Virtual
   0        0 work  kernel seg               -   4375   2176   847     4375
```

```
48412       2 work  process private          -   2411      2     0     2411
6c01b       d work  shared library text      -   1790      0     0     1790
4c413       f work  shared library data      -     11      0     0       11
3040c       1 pers  code,/dev/prodlv:4097    -      2      0     -        -

  Pid Command        Inuse      Pin     Pgsp  Virtual   64-bit    Mthrd    LPage
13548                        pacman         8599      2178       847     8597      N     N     N

 Vsid     Esid Type  Description          LPage  Inuse    Pin  Pgsp  Virtual
    0        0 work  kernel seg               -   4375   2176   847     4375
48412        2 work  process private          -   2421      2     0     2421
6c01b        d work  shared library text      -   1790      0     0     1790
4c413        f work  shared library data      -     11      0     0       11
3040c        1 pers  code,/dev/prodlv:4097    -      2      0     -        -
```

Оценка необходимого объема памяти с помощью команды rmss

Команда **rmss** (эмулятор системы с сокращенным объемом памяти, позволяет уменьшить размер оперативной памяти компьютера, доступной программам, не удаляя карты памяти из системного блока. Кроме того, команда **rmss** позволяет протестировать работу приложения в системе с различными объемами памяти и просмотреть собранную статистику с информацией о времени ответа приложения и числе операций подкачки.

Команда **rmss** позволяет получить ответ на следующий вопрос: “Какой объем физической памяти необходим для работы операционной системы и заданного приложения с высокой производительностью?”. В многопользовательской системе этот вопрос можно переформулировать следующим образом: “Сколько пользователей могут одновременно работать с данным приложением на компьютере с оперативной памятью объемом *X* МБ?”

Как правило, команда **rmss** применяется при планировании нагрузки на систему для определения объема памяти, необходимого для выполнения заданной рабочей схемы. Однако она может применяться и для обнаружения неполадок, особенно в тех случаях, когда увеличение объема памяти привело к снижению производительности системы.

Для того чтобы узнать, установлена ли команда **rmss** в системе, вызовите следующую команду:

```
# lslpp -lI bos.perf.tools
```

При каждом изменении командой **rmss** размера памяти minperm и maxperm не подстраиваются под новые параметры и число страниц lruable не изменяется в соответствии с имитированным размером памяти. Это может привести к неожиданному поведению, при котором кэш буфера будет не пропорционально расти. Вследствие этого, в системе может оказаться недостаточно памяти.

Важно помнить о том, что команда **rmss** ограничивает общий объем оперативной памяти, который применяется операционной системой и другими приложениями, выполняющимися в системе. Этот объем не совпадает с объемом памяти, используемой самим приложением. Поскольку применение команды **rmss** может привести к резкому снижению производительности системы, ее разрешено вызывать только пользователю root и пользователям, входящим в группу system.

Команда rmss

Команду **rmss** можно использовать для одноразового изменения размера памяти, но она также может запускать указанное приложение несколько раз, каждый раз с разным объемом памяти, при этом собирая информацию о производительности приложения.

Первый вариант применяется в том случае, если приложение, работу которого при различных объемах памяти вы хотите оценить, слишком сложно и не может быть вызвано одной командой, или если вы хотите запустить несколько экземпляров приложения. Второй вариант применяется в том случае, если приложение можно вызвать как исполняемую программу или сценарий оболочки.

Флаги -c, -p и -r команды rmss:

Преимущество флагов **-c**, **-p** и **-r** команды **rmss** заключается в том, что они позволяют протестировать работу сложных приложений, которые нельзя представить в виде одного исполняемого файла или сценария оболочки. Недостатком флагов **-c**, **-p** и **-r** является то, что они не обеспечивают автоматический сбор информации о производительности. Информацию об использовании пространства подкачки во время работы приложения можно получить с помощью команды **vmstat -s**.

Для того чтобы узнать, сколько страниц было загружено в память во время работы приложения, вызовите команду **vmstat -s**, запустите приложение, а затем снова вызовите команду **vmstat -s**. Разница между числом загруженных в память страниц до запуска приложения и во время работы приложения даст искомый результат. Поделив полученное значение на время, прошедшее с момента запуска приложения, вы получите среднее число страниц, загружаемых в память в секунду.

Для каждого объема оперативной памяти нужно проводить несколько измерений, запуская приложение несколько раз. Это обусловлено следующими причинами:

- При изменении объема оперативной памяти команда **rmss** часто очищает значительный объем памяти. В результате, когда вы в первый раз исполняете приложение после изменения объема памяти, приложение может затратить значительное время на загрузку файлов в оперативную память. После окончания работы приложения файлы могут остаться в памяти, так что в следующий раз выполнение приложения может занять меньше времени.
- Это позволяет узнать среднюю производительность приложения в системе с таким объемом памяти. Производительность приложения в значительной мере зависит от текущего состояния системы. В связи с этим в разное время приложение может выполняться с разной скоростью.

Ниже приведен общий сценарий применения команды **rmss**:

```
пока вы хотите проверять приложение при других объемах памяти:
  {
  Установите необходимый объем памяти с помощью команды rmss -c;
  выполните приложение один раз для "разогрева";
  Для нескольких  итераций:
    {
    Вызовите команду vmstat -s и запомните число обращений к страницам подкачки;
    Выполните приложение и измерьте время выполнения;
    Вызовите команду vmstat -s и запомните новое число обращений к страницам подкачки;
    Вычтите исходное количество обращений из итогового и получите
       количество обращений к страницам подкачки за время работы приложения;
    Разделите количество обращений на время исполнения приложения
       для получения средней частоты обращения к страницам подкачки;
    }
  }
Восстановите исходный объем оперативной памяти, вызвав команду
rmss -r или перезагрузив систему
```

Разницу между исходным числом обращений к страницам подкачки и числом обращений, полученным после выполнения приложения, можно вычислить автоматически с помощью сценария **vmstatit**, описанного в разделе “Неполадки, связанные с диском или памятью” на стр. 36.

Изменение объема памяти:

Для того чтобы изменить объем памяти системы, вызовите команду **rmss** с флагом **-c**.

Например, для того чтобы уменьшить объем памяти до 128 МБ, вызовите следующую команду:

```
# rmss -c 128
```

В качестве объема памяти можно указать целое или дробное число мегабайт (например, 128,25). Кроме того, новый объем памяти не должен быть меньше 8 МБ или больше фактического объема оперативной памяти компьютера. Значение, до которого можно сократить объем памяти системы с помощью команды **rmss**,

зависит и от размера стандартных структур системы, например, ядра. Когда команде **rmss** не удается изменить объем памяти системы, она выдает сообщение об ошибке.

Для сокращения объема доступной оперативной памяти команда **rmss** удаляет некоторые страницы из списка свободных страниц VMM. Удаленные страницы перемещаются в пул непригодных к использованию страниц и возвращаются в список свободных при восстановлении объема физической памяти. Кроме того, команда **rmss** динамически обновляет некоторые системные переменные и структуры данных, значение или размер которых зависит от объема доступной оперативной памяти.

Для изменения объема памяти может потребоваться некоторое время (до 15 - 20 с). В общем случае, чем больше нужно сократить объем памяти, тем дольше выполняется команда **rmss**. По окончании выполнения команды **rmss** появляется следующее сообщение:

```
Моделируемый объем памяти установлен равным 128,00 МБ.
```

Для того чтобы узнать текущий объем памяти, вызовите команду с флагом **-p**:

```
# rmss -p
```

Команда **rmss** выдаст примерно следующее сообщение:

```
Моделируемый объем памяти равен 128,00 МБ.
```

Для того чтобы восстановить исходный объем памяти компьютера, вызовите команду с флагом **-r**:

```
# rmss -r
```

Независимо от текущего моделируемого объема памяти, команда rmss с флагом **-r** устанавливает объем памяти равным объему физической памяти компьютера.

Если предположить, что команда **rmss** была вызвана на компьютере с оперативной памятью объемом 256 МБ, то появится следующее сообщение:

```
Моделируемый объем памяти установлен равным 256,00 МБ.
```

Примечание: Команда **rmss** сообщает объем доступной физической памяти. Другими словами, объем памяти в выводе команды **rmss** представляет собой общий объем физической памяти минус объем поврежденной и занятой памяти. Например, команда **rmss -r** может выдать следующее сообщение:

```
Моделируемый объем памяти установлен равным 79,9062 МБ.
```

Это может означать, что некоторые страницы памяти помечены как поврежденные или зарезервированы некоторым устройством (и поэтому недоступны для пользователя).

Запуск приложения с различными объемами памяти с помощью команды rmss:

В режиме драйвера команда **rmss** вызывает указанное приложение несколько раз, каждый раз изменяя объем памяти, и собирает информацию о производительности приложения.

Для вызова команды **rmss** в качестве драйвера укажите флаги **-s**, **-f**, **-d**, **-n** и **-o rmss**. Ниже описан формат вызова для запуска команды **rmss** в этом режиме:

```
rmss [ -s начальный-размер-памяти ] [ -f итоговый-размер-памяти ] [ -d приращение ]
     [ -n число-итераций ] [ -o файл-вывода ] команда
```

Ниже приведено описание флагов команды. Флаги **-s**, **-f** и **-d** задают диапазон объемов памяти.

-n Сколько раз характеристики должны измеряться при каждом значении объема памяти.

-o Файл, в который записывается отчет команды **rmss**. Вместо команды нужно указать приложение, характеристики которого нужно измерить в системе с различными объемами памяти.

-s Начальный объем памяти.

-f Итоговый объем памяти.

-d Шаг приращения.

Все значения указываются в мегабайтах в виде целых числе или десятичных дробей. Например, если характеристики программы нужно измерить в системе с объемом памяти 256, 224, 192, 160 и 128 МБ, то укажите следующие флаги:

```
-s 256 -f 128 -d 32
```

Если характеристики программы нужно измерить в системе с объемом памяти 128, 160, 192, 224 и 256 МБ, то укажите следующие флаги:

```
-s 128 -f 256 -d 32
```

Если флаг **-s** не указан, то в качестве начального объема памяти команда **rmss** применяет фактический объем оперативной памяти компьютера. Если не задан флаг **-f**, то команда **rmss** выполняет последнее измерение в системе с объемом памяти 8 МБ. Если не указан флаг **-d**, то объем памяти каждый раз изменяется на 8 МБ.

Какие значения следует указать с флагами **-s**, **-f** и **-d**? Самый простой критерий выбора значений заключается в том, чтобы заданный диапазон включал в себя объем памяти тех систем, в которых планируется выполнять приложение. Однако если приращение будет составлять меньше 8 МБ, то вы сможете узнать, сколько памяти остается у системы в запасе при выполнении приложения в системе со стандартным объемом памяти. Например, если при выполнении приложения в системе с объемом памяти 120 МБ возникает перегрузка памяти, а в системе с объемом памяти 128 МБ операции подкачки не выполняются, то важно знать, при каком объеме памяти начинается перегрузка. Если перегрузка начинается при объеме памяти 127 МБ, то рекомендуется запускать приложение в системах с объемом памяти больше 128 МБ, либо изменить приложение таким образом, чтобы для его выполнения требовалось меньше памяти. Если же перегрузка начинается при объеме памяти 121 МБ, то 128 МБ оперативной памяти вполне достаточно для работы приложения.

Флаг **-n** указывает, сколько раз характеристики программы должны измеряться при каждом значении объема памяти. После того как приложение будет выполнено указанное число раз, команда **rmss** выдаст статистическую информацию о средней скорости выполнения приложения в системе с данным объемом памяти. Для того чтобы при каждом объеме памяти команда выполнялась трижды, укажите:

```
-n 3
```

Если флаг **-n** не задан, то во время инициализации команда **rmss** определяет, сколько раз нужно выполнить приложение, чтобы общее время выполнения составило 10 секунд. Такая оценка гарантирует, что на информацию о производительности быстро выполняющегося приложения, собираемую командой **rmss**, не оказывает значительного влияния работа других программ, например, демонов.

Примечание: Если программа выполняется за очень незначительное время, то для того чтобы истратить 10 секунд процессорного времени на выполнение программы, ее придется запустить большое число раз. Поскольку команда **rmss** подготавливает очередной запуск программы как минимум 2 секунды, она может выполняться очень долго. В связи с этим для таких программ рекомендуется явно задавать число итераций с флагом **-n**.

Какое значение нужно задать с флагом **-n**? Если время выполнения приложения составляет больше 10 секунд, укажите **-n 1**. В этом случае команда будет выполняться дважды, однако при каждом объеме памяти измерение будет выполняться только один раз. Флаг **-n** позволяет сократить время выполнения команды **rmss** за счет того, что во время инициализации ей не придется определять, сколько раз нужно запускать программу. Это особенно важно при тестировании долго работающих и интерактивных приложений.

Обратите внимание, что перед выполнением измерений для очередного объема памяти команда **rmss** делает пробный запуск программы. "Разогрев" позволяет устранить операции ввода/вывода, связанные с загрузкой

приложения в оперативную память. Хотя такие операции влияют на производительность, это влияние не обязательно связано с нехваткой оперативной памяти. Пробный запуск не учитывается в числе итераций, заданном с флагом **-n**.

Флаг **-o** позволяет задать файл, в который будет записан отчет команды **rmss**. Если флаг **-o** не задан, то отчет записывается в файл `rmss.out`.

В параметре **команда** необходимо указать приложение, характеристики которого нужно измерить. Можно указать любую исполняемую программу или сценарий оболочки, в том числе с аргументами. Есть определенные ограничения на формат команды. В частности, в ней не должен перенаправляться поток ввода или вывода (то есть команды в формате **foo > output** и **foo < input** недопустимы). Это связано с тем, что команда **rmss** рассматривает все значения справа от имени программы, как аргументы этой программы. Для того чтобы перенаправить поток ввода или вывода, напишите сценарий оболочки.

Для того чтобы сохранить вывод команды **rmss** в файле, укажите опцию **-o**. Если вы хотите перенаправить стандартный поток вывода команды **rmss** (например, добавить вывод в конец существующего файла), то вызовите команду **rmss** в оболочке Korn, заключив ее в скобки:

```
# (rmss -s 24 -f 8 foo) >> output
```

Оценка результатов команды rmss

Команда **rmss** сообщает много полезной информации.

Пример в разделе “Отчёт, созданный для программы foo” был создан командой **rmss** для одной из реально существующих программ. Здесь эта программа условно названа `foo`. Для создания отчета была вызвана следующая команда:

```
# rmss -s 16 -f 8 -d 1 -n 1 -o rmss.out foo
```

Отчёт, созданный для программы foo:

Отчёт для программы **foo** был получен с помощью команды **rmss**.

```
Имя хоста:  aixhost1.austin.ibm.com
Реальный объем оперативной памяти:   16.00 МБ
Время:  четверг 18 марта 2004 г. 19:04:04
Команда:  foo

Начальное значение моделируемого объема памяти: 16.00 МБ.

Количество итераций для одного значения объема памяти = 1 разогрев + 1 измерение = 2.

Объем памяти Подкачка Время выполнения Частота подкачки
(мегабайт)                (с)           (стр/с)
-------------------------------------------------------------
16.00            115.0          123.9                0.9
15.00            112.0          125.1                0.9
14.00            179.0          126.2                1.4
13.00             81.0          125.7                0.6
12.00            403.0          132.0                3.1
11.00            855.0          141.5                6.0
10.00           1161.0          146.8                7.9
9.00            1529.0          161.3                9.5
8.00            2931.0          202.5               14.5
```

Отчет содержит четыре столбца. В левом столбце указан объем памяти, а в столбце *Подкачка* - среднее число страниц, загружаемых из пространства подкачки при выполнении приложения в системе с таким объемом памяти. Важно отметить, что значение *Подкачка* учитывает все операции подкачки, то есть операции чтения кода, данных и файлов, выполняемые всеми приложениями системы. В столбце *Время выполнения* указано среднее время выполнения приложения, а в столбце *Частота подкачки* - среднее число операций подкачки в секунду.

Обратите внимание на значения из столбца *Частота подкачки*. Когда объем памяти составляет от 16 до 13 МБ, частота подкачки сравнительно невелика (< 1,5 операций подкачки в секунду). Однако в диапазоне от 13 до 8 МБ частота подкачки начинает расти, достигая максимального значения в системе с объемом памяти 8 МБ. В столбце *Время отклика* значения распределяются аналогичным образом: с уменьшением объема памяти время отклика начинает расти все быстрее, достигая максимума в системе с объемом памяти 8 МБ.

Заметим, что в системе с объемом памяти 13 МБ частота подкачки меньше (0,6 операций в секунду), чем в системе с объемом памяти 14 МБ (1,4 операции в секунду). Это не связано с ошибкой. Если измерения проводятся в рабочей системе, то изменение значений никогда не бывает плавным. Важно лишь то, что при объеме памяти 14 и 13 МБ число операций подкачки относительно невелико.

Сделайте выводы на основании полученного отчета. Если в системе с объемом памяти 8 МБ приложение выполняется очень медленно (что вполне вероятно), то увеличение объема памяти позволит значительно повысить производительность. Обратите внимание, что в системе с объемом памяти 16 МБ время отклика составляет 124 секунды, а в системе с объемом памяти 8 МБ - 202 секунды, то есть время отклика возросло на 63 процента. С другой стороны, если в системе с объемом памяти 16 МБ приложение тоже выполняется слишком медленно, увеличение объема памяти не позволит повысить производительность, так как в этом случае операции подкачки слабо влияют на время выполнения приложения.

Отчет о копировании удаленного файла размером 16 МБ:

Ниже приведен пример отчета **rmss**, созданного на компьютере-клиенте для команды, копирующей файл размером 16 МБ с удаленного сервера NFS.

```
Имя хоста: aixhost2.austin.ibm.com
Реальный объем оперативной памяти:   48.00 МБ
Время:  понедельник 22 марта 2004 г. 18:16:42
Команда:  cp /mnt/a16Mfile /dev/null

Начальное значение моделируемого объема памяти 48.00 МБ.

Количество итераций для одного значения объема памяти = 1 разогрев + 4 измерение = 5.

Объем памяти Подкачка Время выполнения Частота подкачки
(мегабайт)                (с)           (стр/с)
----------------------------------------------------------------
48.00              0.0            2.7                  0.0
40.00              0.0            2.7                  0.0
32.00              0.0            2.7                  0.0
24.00              1520.8        26.9                 56.6
16.00              4104.2        67.5                 60.8
8.00               4106.8        66.9                 61.4
```

Вначале время выполнения и частота подкачки невелики, однако при объеме памяти 24 МБ они начинают резко расти и достигают максимальных значений, когда объем памяти составляет 16 и 8 МБ. Этот пример говорит о том, что команду **rmss** следует вызывать для как можно большего числа значений объема памяти. Если бы в данном случае диапазон значений составил от 24 до 8 МБ, то пользователь не смог бы определить объем памяти, при котором приложение выполняется без подкачки.

Рекомендации по использованию флагов -s, -f, -d, -n и -o:

Одним из полезных свойств команды **rmss** является то, что когда ее выполнение прерывается нажатием клавиш прерывания (в качестве которых по умолчанию применяются (**Ctrl + C**), файл вывода не удаляется. Кроме того, при этом восстанавливается исходный размер оперативной памяти компьютера, измененный командой **rmss**.

Команду **rmss** можно запустить в фоновом режиме (даже после выхода из системы) с помощью команды **nohup**. Для этого укажите перед командой **rmss** команду **nohup** и введите в конце командной строки & (амперсанд):

```
# nohup rmss -s 48 -f 8 -o foo.out foo &
```

Рекомендации по работе с командой rmss

Независимо от того, для какой цели применяется команда **rmss**, перед ее запуском необходимо как можно точнее воссоздать ту среду, в которой обычно запускается исследуемое приложение.

Совпадает ли аппаратная конфигурация и конфигурация сети той системы, в которой будут проводиться измерения, с конфигурацией рабочей системы? Будут ли пользователи применять файлы приложений, монтируемые с удаленного узла с помощью NFS или другой распределенной файловой системы? Ответ на последний вопрос особенно важен, поскольку страницы удаленных файлов учитываются VMM отдельно от страниц локальных файлов.

Кроме того, желательно устранить все задания в системе, не имеющие отношения к исследуемой конфигурации и рассматриваемому приложению. В частности, во время выполнения команды **rmss** в системе не должны работать пользователи, если их приложения не относятся к исследуемой рабочей схеме.

Примечание: В системе нельзя запустить сразу несколько экземпляров команды **rmss**.

После того как вы закончите работать с командой **rmss**, рекомендуется перезагрузить систему. При этом будут аннулированы все изменения, внесенные командой **rmss** в параметры системы. В частности, будут восстановлены значения параметров управления нагрузкой на память VMM.

Настройка алгоритма управления нагрузкой на память VMM с помощью команды schedo

С помощью команды **schedo** пользователь root может изменить критерий перегрузки памяти, критерий выбора процессов, работа которых будет остановлена, интервал времени между окончанием перегрузки и возобновлением работы процессов, а также минимальное число процессов, останавливать которые запрещено. Кроме того, с помощью этой команды можно восстановить значения параметров по умолчанию.

Функция управления нагрузкой на память VMM, описанная в “Средство управления нагрузкой на память VMM” на стр. 51, предотвращает перегрузку памяти в системе.

В старых версиях операционной системы не была предусмотрена защита от перегрузки памяти, поэтому если в системе запускалось большое число процессов, то оперативная память переполнялась и производительность системы резко падала. Впоследствии для защиты от перегрузки памяти был разработан алгоритм управления нагрузкой на память. Работа этого алгоритма зависит от нескольких параметров.

Для того чтобы узнать, установлена ли в системе команда **schedo**, вызовите следующую команду:

```
# lslpp -lI bos.perf.tune
```

Настройка параметров управления нагрузкой на память

Функция управления нагрузкой на память позволяет сгладить редкие пики активности, которые могут привести к перегрузке оперативной памяти.

Функция управления нагрузкой на память повышает производительность системы за счет уменьшения числа программ, которые могут одновременно выполняться в системе. Эта функция не предназначена для непрерывного применения в системе, объем оперативной памяти которой недостаточен для выполнения повседневных задач. Она ориентирована на пакетные задания, и не накладывает серьезных ограничений на работу системы. Для защиты наиболее важных приложений предусмотрены другие функции управления рабочей схемой AIX WLM.

При постоянной нехватке физической памяти нужно увеличить объем оперативной памяти, а не тратить время на настройку VMM. Функцию управления нагрузкой на память имеет смысл настраивать только в том случае, если объем оперативной памяти системы больше, чем предполагался при выборе значений по умолчанию. Примером может служить система, для которой значения по умолчанию задают слишком жесткие ограничения.

Параметры управления нагрузкой на память можно изменять лишь в том случае, если в системе применяется стабильная рабочая схема, для которой не подходят значения по умолчанию.

Параметры управления нагрузкой на память всегда применяются в системе, пока они не будут явно изменены. По умолчанию для них установлены значения, которые были протестированы в системах с разной нагрузкой. Изменения значений параметров действуют только до следующей загрузки системы. Все действия по настройке управления нагрузкой на память могут выполняться только пользователем root. Команда **schedo** позволяет системному администратору настроить алгоритм управления нагрузкой на память для отдельной рабочей схемы или выключить этот алгоритм.

В следующем примере вывод команды **schedo** содержит текущие значения параметров:

```
# schedo -a
           v_repage_hi = 0
         v_repage_proc = 4
            v_sec_wait = 1
         v_min_process = 2
         v_exempt_secs = 2
              pacefork = 10
               sched_D = 16
               sched_R = 16
             timeslice = 1
               maxspin = 1
              %usDelta = 100
          affinity_lim = n/a
idle_migration_barrier = n/a
      fixed_pri_global = n/a
         big_tick_size = 1
             force_grq = n/a
```

Первые пять параметров задают пороговые значения для алгоритма управления нагрузкой на память. Они определяют частоту выполнения операций и числовые ограничения. Если выявлено, что объем зарезервированной оперативной памяти слишком велик, то применяются значения параметров *v_repage_proc* , *v_min_process*, *v_sec_wait* и *v_exempt_secs*. В остальных случаях значения этих параметров игнорируются. Если функция управления нагрузкой на память выключена, то эти значения никогда не применяются.

После тестирования работы системы с новыми значениями параметров управления нагрузкой на память, можно восстановить значения по умолчанию, вызвав команду **schedo -D**.

Параметр v_repage_hi:

Параметр *v_repage_hi* задает пороговое значение, превышение которого означает, что память перегружена. Механизм управления нагрузкой на память приостанавливает некоторые процессы, если это значение превышается в течение одной секунды.

Пороговое значение определяет взаимосвязь между двумя показателями: числом страниц, записанных в пространство подкачки за последнюю секунду (`po`), и числом страниц, добавленных в список свободных страниц за последнюю секунду (`fr`). Значения обоих показателей можно узнать с помощью команды **vmstat**. Число страниц, записанных в пространство подкачки, обычно намного меньше числа страниц, добавленных в список свободных страниц. Память перегружена, если выполнено следующее соотношение:

```
po/fr > 1/v_repage_hi or po*v_repage_hi > fr
```

Команда **schedo -o v_repage_hi=0** выключает управление нагрузкой на память. Если в системе не менее 128 МБ оперативной памяти, то значение по умолчанию равно 0. В противном случае значение по умолчанию равно 6. В системах с оперативной памятью объемом 128 МБ и более обычные функции VMM как правило лучше защищают от перегрузки памяти, чем функция управления нагрузкой на память.

В некоторых случаях требуется выключить функцию управления нагрузкой на память. Например, если для моделирования многопользовательской рабочей схемы применяется эмулятор терминала с функцией тайм-аута, то включение функции управления нагрузкой на память может привести к тому, что функция тайм-аута уничтожит некоторые ответы, передача которых процессу долго откладывалась. Другим примером является применение команды **rmss** для тестирования работы приложений в системе с меньшим объемом оперативной памяти. В этом случае функция управления нагрузкой на память может исказить результаты измерений.

Если выключение механизма управления нагрузкой на память приводит к учащению перегрузок системы (и, соответственно, к увеличению времени ответа), это означает, что механизм играет важную роль в поддержании работоспособности системы. В этом случае попробуйте настроить параметры управления нагрузкой на память, а если производительность системы не повысится, то увеличьте объем оперативной памяти.

Чем меньше значение *v_repage_hi*, тем выше порог обнаружения перегрузки. Это означает, что процессы начнут приостанавливаться за меньшее время до начала перегрузки памяти. Если в системе отношение `po/fr` невелико, то перегрузка маловероятна. Это правило справедливо для системы с любой конфигурацией.

Для того чтобы установить пороговое значение, равное 4, введите следующую команду:

```
# schedo -o v_repage_hi=4
```

В данном случае увеличивается объем нагрузки на память системы, при достижении которой начинают приостанавливаться процессы.

Параметр v_repage_proc:

Параметр *v_repage_proc* устанавливает критерий для отбора процессов, работа которых будет остановлена при возникновении перегрузки памяти. Он представляет собой отношение двух показателей, измеряемых для всех процессов: числа повторных обращений к страницам (*r*) и числа страничных ошибок, возникших при обращении процесса к страницам памяти за последнюю секунду (*f*).

Высокая интенсивность повторных страничных ошибок означает перерасход памяти отдельным процессом. Работа процесса останавливается (процесс расходует слишком много памяти или усиливает общую перегрузку памяти), если выполнено следующее условие:

```
r/f > 1/v_repage_proc or r*v_repage_proc > f
```

По умолчанию значение параметра *v_repage_proc* равно 4. Это означает, что будет остановлена работа тех процессов, у которых отношение числа повторных обращений к числу страничных ошибок, возникших за последнюю секунду, больше 25 процентов. Чем меньше значение *v_repage_proc*, тем выше уровень перегрузки памяти, при достижении которого процесс может быть остановлен.

Для того чтобы запретить функции управления нагрузкой на память останавливать процессы, вызовите следующую команду:

```
# schedo -o v_repage_proc=0
```

Обратите внимание, что функция управления нагрузкой на память никогда не приостанавливает работу процессов с фиксированным приоритетом и процессов ядра.

Параметр v_min_process:

Параметр *v_min_process* задает минимальное число активных процессов в системе. Активными процессами называются процессы, готовые к выполнению, и процессы, ожидающие загрузки или выгрузки страниц. Процессы, ожидающие возникновения событий, процессы, работа которых приостановлена, и процесс wait не относятся к активным.

Если задано минимальное число активных процессов *v_min_process*, то в случае перегрузки памяти работа этих процессов не будет приостановлена. Предположим, что для работы системы как минимум десять активных процессов должны постоянно находиться в оперативной памяти, однако функция управления нагрузкой на память приостанавливает слишком много процессов. Если системный администратор вызовет команду **schedo -o v_min_process=10**, то число приостановленных процессов никогда не будет столь велико, что в системе останется менее десяти активных процессов. Значение параметра *v_min_process* не учитывает:

- Процессы ядра
- Процессы, закрепленные в оперативной памяти с помощью системного вызова **plock()**
- Процессы с фиксированным приоритетом меньше 60
- Процессы, ожидающие возникновения события

По умолчанию устанавливается значение **v_min_process=2**. Оно означает, что в оперативной памяти всегда будут находиться процессы ядра, все закрепленные процессы и как минимум два пользовательских процесса.

Хотя значение **v_min_process=2** вполне подходит для однопользовательской системы, оно недостаточно для многопользовательской системы (например, сервера) с большим объемом оперативной памяти.

Если размер оперативной памяти системы составляет от 32 до 128 МБ, и в системе одновременно будет работать не менее пяти пользователей, то рекомендуется увеличить параметр управления нагрузкой на память, задающий минимальное число активных процессов.

Например, если в системе должно выполняться как минимум четыре приложения, активно использующих оперативную память, при условии, что 16 МБ памяти отведено операционной системе, а 25 процентов памяти занято страницами файлов, то вы можете увеличить минимальное число активных процессов с 2 до 4, вызвав следующую команду:

```
# schedo -o v_min_process=4
```

Как правило, в таких системах оптимальная производительность достигается при значениях *v_min_process* от 4 до 6. Уменьшение *v_min_process*, хотя оно и разрешено, может привести к тому, что не будет работать ни один пользовательский процесс.

Если вы знаете объем памяти, занимаемый приложениями, выполнение которых приводит к перегрузке памяти, то несложно определить значение *v_min_process*. Предположим, что перегрузка памяти возникает при выполнении нескольких экземпляров приложения размером *M*. Если размер оперативной памяти системы равен *N*, то параметру *v_min_process* нужно присвоить значение, приблизительно равное *N*/*M*. Если вы присвоите параметру *v_min_process* меньшее значение, то это приведет к излишнему ограничению числа процессов, которые на самом деле могли бы выполняться в системе.

Параметр v_sec_wait:

Параметр *v_sec_wait* задает интервал времени в секундах, в течение которого отношение `po/fr` должно оставаться меньше `1/v_repage_hi`, после чего будет возобновлена работа приостановленных процессов.

Значение по умолчанию равно 1. Минимальное значение равно 0. Значение по умолчанию обеспечивает повторную активацию процессов сразу после восстановления нормальной работы системы. Слишком большое значение параметра *v_sec_wait* может привести к значительному увеличению времени ответа приостановленных процессов, тогда как процессор будет простаивать из-за недостатка активных процессов.

Для того чтобы работа процессов возобновлялась через две секунды, введите следующую команду:

```
# schedo -o v_sec_wait=2
```

Параметр v_exempt_secs:

После повторной активации приостановленного процесса этот процесс будет работать как минимум *v_exempt_secs* секунд. В этом случае задержки, связанные с загрузкой страниц процесса в память при выполнении операций дискового ввода-вывода, будут оправданы.

Значение *v_exempt_secs* по умолчанию составляет 2 секунды.

Для того чтобы изменить этот параметр, вызовите следующую команду:

```
# schedo -o v_exempt_secs=1
```

Предположим, что перегрузка памяти в системе была случайно вызвана приложением, занимающим большой объем памяти. Пусть это приложение выполняется *T* секунд. Если параметру *v_exempt_secs* присвоено значение 2 секунды, то в сильно загруженной системе приложению придется выполнять подкачку *T*/2 раз. В этом случае увеличение значения параметра *v_exempt_secs* ускорит выполнение приложения. Чем быстрее будет выполнено это приложение, тем раньше повысится производительность системы.

Настройка алгоритма замены страниц VMM

Алгоритм управления памятью сохраняет размер списка свободных страниц и долю физической памяти, занятую страницами постоянной памяти, в допустимых пределах.

Предельные значения, описанные в “Физическая память, управление” на стр. 46, можно изменить с помощью команды **vmo**. Эту команду разрешено выполнять только пользователю root. Изменения, внесенные с помощью этой команды, действуют до следующей загрузки системы. Для того чтобы узнать, установлена ли команда **vmo** в системе, вызовите следующую команду:

```
# lslpp -lI bos.perf.tune
```

Команда **vmo** с опцией **-a** выдает список текущих значений параметров.

Примечание: Команда **vmo** является самодокументируемой. Вывод команды может отличаться от приведенного здесь примера.

```
# vmo -a
               ame_cpus_per_pool = n/a
                 ame_maxfree_mem = n/a
             ame_min_ucpool_size = n/a
                 ame_minfree_mem = n/a
                 ams_loan_policy = n/a
    enhanced_affinity_affin_time = 1
  enhanced_affinity_vmpool_limit = 10
                  esid_allocator = 1
             force_relalias_lite = 0
               kernel_heap_psize = 65536
                    lgpg_regions = 0
                       lgpg_size = 0
                 low_ps_handling = 1
                         maxfree = 1088
                         maxperm = 843105
                          maxpin = 953840
                         maxpin% = 90
                   memory_frames = 1048576
                   memplace_data = 0
           memplace_mapped_file = 0
          memplace_shm_anonymous = 0
              memplace_shm_named = 0
                  memplace_stack = 0
                   memplace_text = 0
          memplace_unmapped_file = 0
                         minfree = 960
                         minperm = 28103
                        minperm% = 3
```

```
        msem_nlocks = 0
          nokilluid = 0
            npskill = 1024
            npswarn = 4096
num_locks_per_semid = 1
          numpsblks = 131072
         pgz_lpgrow = 2
           pgz_mode = 2
    pinnable_frames = 781272
relalias_percentage = 0
              scrub = 0
      thrpgio_inval = 1024
     thrpgio_npages = 1024
           v_pinshm = 0
      vm_cpu_thresh = 0
       vm_mmap_bmap = 1
   vmm_default_pspa = 0
     vmm_klock_mode = 2
 wlm_memlimit_nonpg = 1
```

Значения для параметров minfree и maxfree

Список свободных страниц пополняется за счет страниц физической памяти, освобождаемых при завершении процессов. Страницы из этого списка выделяются другим процессам без задержек, связанных с принудительным освобождением страниц и сопутствующими операциями ввода-вывода.

Ограничение *minfree* задает минимальный размер списка свободных страниц. При достижении этого ограничения запускается функция пополнения списка свободных страниц. Параметр *maxfree* задает размер списка, при достижении которого список прекращает пополняться и значение *maxfree* применяется для запуска пополнения страниц. Вытеснение страниц запускается, когда число постоянных страниц равно либо меньше разницы между значениями параметров *maxfree* и *minfree*, или когда число клиентских страниц равно либо меньше разницы между значениями параметров *maxclient* и *minfree*.

При настройке этих ограничений преследуются следующие цели:

- Любой процесс, для которого чрезвычайно важно низкое время ответа, может получить страницы из списка свободных страниц.
- Не происходит резкого роста числа операций ввода-вывода, так как список свободных страниц пополняется заблаговременно.

Значения по умолчанию для параметров *minfree* и *maxfree* устанавливаются в зависимости от размера оперативной памяти компьютера. При работе с JFS разница между параметрами *maxfree* и *minfree* всегда должна быть равна либо превышать значение параметра *maxpgahead*. При работе с Enhanced JFS разница между параметрами *maxfree* и *minfree* всегда должна быть равна либо превышать значение параметра *j2_maxPageReadAhead*. При использовании и JFS, и Enhanced JFS следует установить значение параметра *minfree*, большее или равное наибольшему значению заголовка страниц для этих двух файловых систем.

Если в системе есть несколько пулов памяти, то для каждого из них значения *minfree* и *maxfree* настраиваются отдельно. Пулы памяти применяются в многопроцессорных системах с большим объемом оперативной памяти. Каждому пулу памяти принадлежат значения *minfree* и *maxfree*. В ранних версиях AIX значения *minfree* и *maxfree*, показываемые по команде **vmo** равны сумме значений *minfree* и *maxfree* для всех пулов памяти. Значения, возвращаемые командой **vmo**, указываются для каждого пула памяти.

Команда **vmstat** предоставляет менее точный, но более удобный способ для вычисления значения *minfree*. Ниже приведен фрагмент вывода команды **vmstat**, выполненной в системе, в которой достигнуто значение *minfree*:

```
# vmstat 1
нити     память             страница          ошибки        cpu
----- ----------- ------------------------ ------------ -----------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
 2  0 70668   414   0   0   0   0    0   0 178 7364  257  35 14  0 51
```

```
 1  0 70669   755   0   0   0   0    0   0 196 19119 272  40 20  0 41
 1  0 70704   707   0   0   0   0    0   0 190 8506  272  37  8  0 55
 1  0 70670   725   0   0   0   0    0   0 205 8821  313  41 10  0 49
 6  4 73362   123   0   5  36 313 1646   0 361 16256 863  47 53  0  0
 5  3 73547   126   0   6  26 152  614   0 324 18243 1248 39 61  0  0
 4  4 73591   124   0   3  11  90  372   0 307 19741 1287 39 61  0  0
 6  4 73540   127   0   4  30 122  358   0 340 20097 970  44 56  0  0
 8  3 73825   116   0  18  22 220  781   0 324 16012 934  51 49  0  0
 8  4 74309    26   0  45  62 291 1079   0 352 14674 972  44 56  0  0
 2  9 75322     0   0  41  87 283  943   0 403 16950 1071 44 56  0  0
 5  7 75020    74   0  23 119 410 1611   0 353 15908 854  49 51  0  0
```

В приведенном выше примере видно, что значение *minfree*, равное 120, достигается постоянно. В результате выполняется замена страниц и в данном случае список свободных страниц оперативной памяти один раз достигает нулевого значения. При этом нити, которым требуются свободная память, блокируются до тех пор, пока не будут освобождены дополнительные страницы. Для того чтобы избежать подобной ситуации рекомендуется увеличить значения *minfree* и *maxfree*.

Для того чтобы в системе всегда была доступна по крайней мере 1000 свободных страниц для пула памяти, выполните следующую команду:

```
# vmo -o minfree=1000 -o maxfree=1008
```

Для того чтобы сделать это изменение постоянным, укажите флаг **-p**:

```
# vmo -o minfree=1000 -o maxfree=1008 -p
```

Значение по умолчанию параметра *minfree* увеличено до 960 для каждого пула памяти, а для параметра *maxfree* - до 1088.

Списки LRU

Алгоритм LRU использует списки. В предыдущих версиях AIX также был доступен способ, основанный на работе с таблицей страниц оперативной памяти. Основанный на списках алгоритм предоставляет список страниц для выбора по каждому типу сегмента.

Ниже перечислены типы сегментов:

- Рабочий
- Постоянный
- Клиентский
- Сжатый

При включении WLM также существуют списки классов.

Повышение эффективности выбора страниц оперативной памяти с помощью параметра lrubucket

Настройка параметра *lrubucket* позволяет повысить эффективность выбора страниц в системах с большим объемом оперативной памяти.

Алгоритм замены страниц перебирает страницы физической памяти, выбирая свободные страницы. Во время такого перебора сбрасывается флаг обращения к странице. Если свободная страница не была найдена, то перебор страниц начинается заново. Если во время повторного перебора флаг обращения к странице будет по-прежнему сброшен, то эта страница физической памяти освобождается для размещения другой страницы виртуальной памяти (выполняется замена страниц).

В системах с большим объемом оперативной памяти приходится перебирать большое число страниц физической памяти. Для повышения эффективности перебора вся память была поделена на блоки страниц. Алгоритм замены страниц дважды просматривает один и тот же блок страниц, и лишь затем переходит к следующему блоку. По умолчанию блок содержит 131072 страниц физической памяти, суммарный объем

которых составляет 512 МБ. Размер блока можно изменить с помощью команды **vmo -o lrubucket=***новое-значение*, указав в ней число блоков по 4 КБ.

Значения для параметров minperm и maxperm

Пользуясь тем, что требования к оперативной памяти динамически изменяются, операционная система оставляет в оперативной памяти страницы файлов, которые были считаны или записаны.

Если обращение к таким страницам происходит до того, как они были преданы под другие нужды, то для выполнения запроса не требуется тратить время на обмен с диском. Такие страницы могут относиться как к локальным файлам, так и к файлам из удаленных файловых систем (например, NFS).

Отношение числа страниц файлов к числу страниц, применяемых для вычислений (рабочих страниц и страниц с текстом программы), косвенно зависит от значений *minperm* и *maxperm*:

- Если доля оперативной памяти, занимаемой страницами файлов, становится меньше значения *minperm*, то алгоритм замены страниц начинает освобождать как страницы файлов, так и страницы, применяемые для вычислений.
- Если доля оперативной памяти, занятая страницами файлов, лежит в диапазоне *minperm* - *maxperm*, то алгоритм замены страниц пополняет список свободных страниц только за счет страниц файлов.

В одних рабочих схемах нужно стремиться к сокращению числа операций ввода-вывода страниц файлов. В других рабочих схемах выгоднее сохранять в памяти сегменты, применяемые для вычислений. Для того чтобы узнать текущие значения этих параметров, выполните команду **vmstat** с опцией **-v**.

```
# vmstat -v
              1048576 страниц памяти
               936784 страниц lru
               683159 свободных страниц
                    1 пулов памяти
               267588 закрепленных страниц
                 90.0 процентов от maxpin
                  3.0 процента от minperm
                 90.0 процентов от maxperm
                  5.6 процента от numperm
                52533 страниц файлов
                  0.0 процентов сжатых
                    0 сжатых страниц
                  5.6 процента от numclient
                 90.0 процентов от maxclient
                52533 станиц клиентов
                    0 запланированных удаленных выгрузок страниц
                    0 блокированных операций дискового ввода-вывода без pbuf
                    0 блокированных операций ввода-вывода из области подкачки без psbuf
                 2228 блокированных операций ввода-вывода файловых систем без fsbuf
                   31 блокированная операция ввода-вывода клиента без fsbuf
                    0 блокированных операций ввода-вывода для области подкачки внешних файловых систем без fsbuf
                 29.8 процентов памяти, используемой для вычислительных страниц
```

Значение *numperm* задает процентную долю физической памяти, занятой сегментами файлов. Значение 5.6% соответствует 52533 страницам файлов в памяти.

Ограничение кэша расширенной файловой системы с помощью параметра maxclient

Параметр *Maxclient* задает максимальное количество страниц клиентов, которые могут использоваться для кэша буфера.

В расширенной файловой системе JFS для кэша буфера применяются страницы клиентов. Ограничение для страниц клиентов в физической памяти устанавливается с помощью параметра *maxclient*.

Демон LRU запускается, когда число страниц клиентов находится в пределах числа страниц *minfree* порогового значения параметра *maxclient*. Демон LRU пытается вытеснить в пространство подкачки

страницы клиентов, к которым давно не было обращений. Если число страниц файлов меньше значения параметра *minperm*, для замещения выбирается любая страница, к которой не происходит обращений.

Кроме того, от значения параметра *Maxclient* зависит способ обработки клиентов NFS и сжатых страниц. Также обратите внимание, что значение параметра *maxclient* должно быть не больше значения параметра *maxperm*.

Выделение пространства подкачки

В AIX имеется несколько стратегий выделения пространства подкачки.

- Отложенное выделение пространства подкачки (DPSA)
- Статическое выделение пространства подкачки (EPSA)

Отложенное выделение пространства подкачки

Стратегия отложенного выделения пространства подкачки применяется по умолчанию в AIX.

В стратегии отложенного выделения пространства подкачки место на диске выделяется только тогда, когда нужно выгрузить страницу из памяти. В результате все выделенное пространство подкачки действительно используется. Отложенный алгоритм может попытаться выделить пространство подкачки, размер которого превышает объем доступной памяти. Это приводит к фиксации слишком большого пространства подкачки.

Если выгруженная в пространство подкачки страница была снова загружена в оперативную память, то за ней будет зарезервирована та область диска, в которой она была расположена. Следовательно, доля занятой памяти в области подкачки учитывает не только страницы, выгруженные в пространство подкачки, но и страницы, которые были загружены обратно в оперативную память. Если страница, загруженная обратно в оперативную память, относится к рабочей памяти нити, и эта нить освободила данную страницу памяти или завершила свою работу, то область диска, выделенная для страницы, будет освобождена. Области диска из пространства подкачки для страниц, считанных обратно в основную память, можно освободить с помощью функции сбора мусора пространства подкачки. Дополнительная информация приведена в разделе “Сбор мусора в пространстве подкачки” на стр. 159.

Если функция сбора мусора пространства подкачки не включена, необходимо правильно настроить объем пространства подкачки. Если размер кэша файлов меньше minperm и не выделен достаточный объем пространства подкачки, может потребоваться настроить систему таким образом, чтобы предотвратить вытеснение страниц рабочей памяти при обработке страниц файлов. Если необходимый объем рабочей памяти меньше объема физической памяти, и система настроена таким образом, что при обработке страниц файлов не происходит вытеснение страниц рабочей памяти, можно выделить минимальный объем для пространства подкачки. Отдельные сегменты области таблицы страниц (PTA), которые не являются сегментами отложенного выделения, известны как внутренние сегменты памяти ядра AIX. С учетом этих сегментов рекомендуется выделить пространство подкачки размером 512 МБ. Если система использует большой объем области PTA, то потребуется увеличить пространство подкачки. Это можно определить с помощью команды **svmon -S**.

Если необходимый объем рабочей памяти больше объема физической памяти, следует выделить пространство подкачки, по крайней мере равное объему рабочей виртуальной памяти. В противном случае объем пространства подкачки может оказаться недостаточным.

Статическое выделение пространства подкачки

Для того чтобы процесс не был убит вследствие нехватки пространства подкачки, ему следует заранее выделить область пространства подкачки с помощью стратегии статического выделения пространства подкачки.

Для этого переменной среды *PSALLOC* нужно присвоить значение early. Это может сделать сам процесс, либо пользователь с помощью команды **PSALLOC=early**. Когда процесс вызывает функцию **malloc()** для получения некоторого объема памяти, на диске, содержащем пространство подкачки, будут зарезервированы блоки для данного процесса. Таким образом, если процессу потребуется выгрузить данные

на диск, для него гарантированно будет найдена свободная область пространства подкачки. В системе со стратегией статического выделения пространства подкачки можно несколько сэкономить ресурсы процессора, установив переменную среды **NODISCLAIM=true**. В этом случае при обработке системного вызова **free()** не будет вызываться функция **disclaim()**.

Пространство подкачки и виртуальная память

Число страниц виртуальной памяти, к которым действительно обращались процессы, можно узнать с помощью команды **vmstat** (столбец *avm*), команды **ps** (поле *SIZE* или *SZ*) и других утилит. В системах со стратегией DPSA это число не обязательно совпадает с числом страниц, занятых в пространстве подкачки.

Для определения объема свободной памяти в пространстве подкачки лучше использовать команду **lsps -s**, а не команду **lsps -a**, так как команда **lsps -a** учитывает только ту часть пространства подкачки, которая действительно занята. Однако команда **lsps -s** учитывает как объем занятой памяти в пространстве подкачки, так и объем памяти, зарезервированной с помощью стратегии EPSA.

Настройка порогов для пространства подкачки

Когда в пространстве подкачки остается мало свободного места, операционная система освобождает часть пространства подкачки. Для этого процессам отправляется предупреждение о том, что нужно освободить пространство подкачки. Если после этого свободной памяти в пространстве подкачки будет по-прежнему недостаточно для выполнения текущих процессов, операционная система уничтожит некоторые процессы.

Значения для параметров npswarn и npskill

Пороговые значения *npswarn* и *npskill* указывают, когда следует рассылать процессам предупреждения и когда нужно уничтожать некоторые процессы.

Значения этих параметров можно изменить с помощью команды **vmo**:

npswarn
: Задает число свободных страниц в пространстве подкачки, при достижении которого операционная система начинает отправлять сигнал SIGDANGER процессам. Получив этот сигнал при достижении порога *npswarn*, процесс может либо проигнорировать его, либо выполнить какие-либо действия по своему усмотрению, например, освободить занимаемую память с помощью функции **disclaim()** или завершить работу.

  Значение *npswarn* должно быть больше нуля и меньше общего числа страниц в пространстве подкачки. Его можно изменить с помощью команды **vmo -o npswarn=value**.

npskill
: Задает число свободных страниц в пространстве подкачки, при достижении которого система начинает убивать процессы. При достижении порогового значения *npskill* недавно запущенным процессам отправляется сигнал SIGKILL. Система не убивает процессы, обрабатывающие сигнал SIGDANGER, и процессы, применяющие алгоритм статического выделения пространства подкачки (в этом алгоритме пространство подкачки выделяется при получении запроса на расширение памяти). Значение по умолчанию для параметра *npskill* вычисляется по следующей формуле:

  ```
  npskill = maximum (64, число страниц в пространстве подкачки/128)
  ```

  Значение **npskill** должно быть больше нуля и меньше общего числа страниц в пространстве подкачки. Его можно изменить с помощью команды **vmo -o npskill=value**.

nokilluid
: Если опции *nokilluid* с помощью команды **vmo -o nokilluid** присвоить ненулевое значение, то процессы пользователей, идентификаторы которых меньше указанного значения, не будут уничтожаться в случае нехватки пространства подкачки. Например, если для **nokilluid** задано значение 1, процессы, принадлежащие корневому субъекту, не будут убиваться при достижении порога **npskill**.

Интервал между повторными вызовами fork()

Если процесс не удалось породить из-за нехватки пространства подкачки, планировщик повторит попытку еще пять раз. По умолчанию интервал между повторными попытками составляет 10 тактов.

С помощью параметра *pacefork* команды **schedo** можно изменить число тактов, по истечении которых система может повторить вызов **fork()**, который в предыдущий раз не был выполнен. Например, если вызов **fork()** не был выполнен из-за того, что в системе не хватило памяти для создания нового процесса, то система повторит вызов по истечении указанного числа тактов. Значение по умолчанию равно 10 тактам. Поскольку один такт равен 10 мс, то система повторит вызов **fork()** через 100 мс.

Если нехватка пространства подкачки связана с кратковременным пиком активности, то увеличение интервала между повторными вызовами, как это сделано в следующем примере, позволяет дождаться освобождения памяти:

```
# schedo -o pacefork=15
```

В этом случае повторный вызов **fork()** с большой вероятностью будет выполнен успешно, поскольку за указанный интервал некоторые процессы могли завершить работу и, следовательно, освободить страницы, занимаемые в пространстве подкачки.

Сбор мусора в пространстве подкачки

Можно использовать функцию сбора мусора в пространстве подкачки для освобождения при определенных условиях блоков на диске в пространстве подкачки. Это позволяет не выделять пространство подкачки, равное объему виртуальной памяти, применяемой для выполнения конкретной задачи. Функция сбора мусора доступна только при использовании стратегии отложенного выделения пространства подкачки.

Сбор мусора в пространстве подкачки после возврата страниц в память

Способ освобождения блока в пространстве подкачки после записи страницы из пространства подкачки обратно в память применяется по умолчанию.

Причина для отказа от освобождения блока после каждой операции записи страницы обратно в память заключается в том, что если блоки остаются в пространстве подкачки, это может повысить производительность при наличии неизмененных страниц рабочей памяти, вытесненных демоном LRU. Если страницы были вытеснены, то отсутствует необходимость выполнения функции повторного считывания страниц.

С помощью команды **vmo** можно настроить следующие параметры:

Тонкая настройка параметра **npsrpgmin**:

Элемент	Описание
Назначение:	Задает пороговое значение для числа свободных блоков в пространстве подкачки, при достижении которого начинается сбор мусора с записью страниц обратно в память.
Значения:	По умолчанию: MAX (768, **npswarn**+ (**npswarn**/2)
Диапазон:	от 0 до общего числа блоков пространства подкачки в системе.

Тонкая настройка параметра **npsrpgax**:

Элемент	Описание
Назначение:	Задает пороговое значение для числа свободных блоков в пространстве подкачки, при достижении которого сбор мусора с записью страниц обратно в память прекращается.
Значения:	По умолчанию: MAX (1024, **npswarn***2)

Тонкая настройка параметра **rpgclean**:

Элемент	Описание
Назначение:	Включает или выключает освобождение блоков страниц в пространстве подкачки при чтении этих страниц в стратегии отложенного выделения пространства подкачки.
Значения:	По умолчанию: 0, что означает освобождение блоков пространства подкачки только при загрузке страниц, которые были изменены. Значение 1 означает освобождение блоков при загрузке страниц, которые были изменены, прочитаны или к которым осуществлялся доступ.
Диапазон:	0 | 1

Тонкая настройка параметра **rpgcontrol**:

Элемент	Описание
Назначение:	Включает или выключает освобождение блоков в пространстве подкачки при загрузке страниц в стратегии отложенного выделения пространства подкачки.
Значения:	По умолчанию: 2, что означает, что блоки в пространстве подкачки всегда освобождаются при загрузке страниц, независимо от пороговых значений. **Примечание:** Чтение страниц выполняется только в том случае, если значение параметра **rpgcontrol** равно 1. По умолчанию всегда выполняются только операции записи.Значение 0 отключает освобождение блоков в пространстве подкачки при загрузке страниц.
Диапазон:	0 | 1 | 2

Сбор мусора посредством очистки памяти

Другим способом сбора мусора в пространстве подкачки является очистка памяти, осуществляемая процессом **psgc** ядра.

Процесс **psgc** ядра освобождает блоки пространства подкачки для измененных страниц памяти, которые еще не были загружены обратно, или для неизмененных страниц, для которых существует блок в пространстве подкачки.

Процесс **psgc** ядра применяет следующие параметры, которые можно настроить с помощью команды **vmo**:

Тонкая настройка параметра **npsscrubmin** включает следующие поля:

Элемент	Описание
Назначение:	Задает число свободных блоков пространства подкачки, при котором начинается процесс очистки памяти для освобождения блоков от страниц из стратегии отложенного выделения пространства подкачки.
Значения:	По умолчанию: MAX (768, значение параметра **npsrpgmin**)
Диапазон:	от 0 до общего числа блоков пространства подкачки в системе.

Тонкая настройка параметра **npsscrubmax** включает следующие поля:

Элемент	Описание
Назначение:	Задает число свободных блоков пространства подкачки, при котором прекращается процесс очистки памяти для освобождения блоков от страниц из стратегии отложенного выделения пространства подкачки.
Значения:	По умолчанию: MAX (1024, значение параметра **npsrpgmax**)
Диапазон:	от 0 до общего числа блоков пространства подкачки в системе.

Тонкая настройка параметра **scrub** включает следующие поля:

Элемент	Описание
Назначение:	Включает или выключает освобождение блоков в пространстве подкачки от страниц в памяти из стратегии отложенного выделения пространства подкачки.
Значения:	По умолчанию: 0, что полностью выключает очистку памяти. Если значение этого параметра равно 1, очистка в памяти блоков пространства подкачки начинается, если число свободных блоков пространства подкачки системы меньше значения параметра **npsscrubmin**, но больше значения параметра *npsscrubmax*.
Диапазон:	0 | 1

Тонкая настройка параметров **scrubclean** включает следующие поля:

Элемент	Описание
Назначение:	Включает или выключает освобождение блоков в пространстве подкачки от страниц в памяти из стратегии отложенного выделения пространства подкачки, которые не были изменены.
Значения:	По умолчанию: 0, что означает освобождение блоков пространства подкачки только для измененных страниц в памяти. Если это значение равно 1, блоки пространства подкачки освобождаются для измененных и не измененных страниц.
Диапазон:	0 | 1

Общая память

С помощью функций **shmat()** и **mmap()** файл можно напрямую разместить в памяти. Это позволяет избежать буферизации и вызова системных функций.

Некоторые области памяти в системе выделяются под сегменты общей памяти. При выполнении 32-разрядных приложений сегмент 14 освобождается под сегмент общей памяти. Всего в системе применяется 11 сегментов общей памяти без учета сегментов, содержащих данные и код общих библиотек. Этот метод применим к процессам с сегментами 3-12 и 14. Размер этих сегментов составляет 256 МБ. Приложения могут считывать или записывать файл путем чтения или записи сегмента. Таким образом, приложение может отказаться от системных вызовов чтения-записи, работая с указателями в этих сегментах.

Файлы и данные могут применяться несколькими процессами или нитями. При этом приложение должно синхронизировать доступ процессов или нитей к общим объектам. Как правило, общая память применяется приложениями баз данных, которые применяют базу данных в качестве большого кэша буферов базы данных.

Для общих сегментов памяти пространство подкачки выделяется аналогично личным сегментам процессов. Если стратегия отложенного выделения пространства подкачки выключена, то пространство подкачки применяется при обращении к страницам.

Расширенная общая память

Расширенная общая память позволяет 32-разрядному процессу резервировать сегменты общей памяти размером один байт, с учетом округления до ближайшей страницы. Расширенная общая память может применяться процессами, для которых переменная среды **EXTSHM** задана равной **ON**, **1SEG** или **MSEG**.

Расширенная общая память фактически устраняет ограничение 11 областей общей памяти. 64-разрядные процессы никак не затрагиваются переменной **EXTSHM**.

Если **EXTSHM** задана равной **ON**, то это равнозначно заданию ее равной **1SEG**. С любым из этих значений любой фрагмент общей памяти размером меньше 256 МБ создается в системе как сегмент mmap, и тем самым влияет на производительность аналогично mmap. Фрагмент общий памяти размером более 256 МБ создается в системе как рабочий сегмент.

Если **EXTSHM** задана равной **MSEG**, то вся общая память создается в системе как сегмент mmap, что позволяет улучшить работу с памятью.

Число общих областей памяти, которые может создать процесс, не ограничено. Процессы по-прежнему могут размещать файлы в памяти, однако при этом будет выделяться адресное пространство, объем которого кратен 256 МБ (размеру сегмента). Изменить размер общей области памяти нельзя. Все вышесказанное справедливо и для процессов ядра.

При работе с расширенной общей памятью следует учитывать следующие ограничения:

- Для работы с данными в общих областях памяти нужно напрямую обращаться к указателю, как и при работе с областями памяти, в которых размещаются файлы.
- Поддерживается только ввод-вывод с помощью функции **uphysio()** (прямой доступ к данным не поддерживается).
- Общие области памяти нельзя использовать в качестве буферов ввода-вывода, если обработчик прерываний удаляет память буфера из числа закрепленной памяти. Например, общие области памяти не могут применяться в качестве буферов асинхронного ввода-вывода.
- Сегменты нельзя закрепить с помощью функции **plock()**, поскольку функция **plock()** не поддерживает сегменты, напрямую размещенные в памяти.

Использование псевдонимов для сегмента 1 ТБ

Использование псевдонимов для сегмента 1 ТБ повышает производительность, используя преобразование области общей памяти в сегменты размером 256 МБ. Поддерживается всеми 64-разрядными приложениями, использующими области общей памяти. Использование псевдонимов допустимо для направленных и ненаправленных присоединений общей памяти.

Если приложение работает с областями общей памяти, используя псевдонимы 1 ТБ, операционная система AIX использует преобразование сегментов в 1 ТБ без изменения приложения. Это требует использования переменных `shm_1tb_shared VMO`, `shm_1tb_unshared VMO` и `esid_allocator VMO`.

Переменная `shm_1tb_shared VMO` может быть задана для каждого процесса с помощью переменной среды *"SHM_1TB_SHARED=" VMM_CNTRL*. Значение по умолчанию устанавливается динамически во время загрузки в соответствии с возможностями процессора. Оно автоматически меняется на общий псевдоним, если одна область общей памяти имеет необходимое количество ESID. Допустимые значения находятся в интервале от 0 до 4 КБ (приблизительно требуется ESID 256 МБ для 1 ТБ).

Переменная `shm_1tb_unshared VMO` может быть задана для каждого процесса с помощью переменной среды *"SHM_1TB_UNSHARED=" VMM_CNTRL*. Значение по умолчанию - 256. Допустимые значения находятся в интервале от 0 до 4 КБ. Требуется внимательно задавать значение по умолчанию (требуется до 64 ГБ адресного пространства), прежде чем переходить к неразделяемому псевдониму 1 ТБ. Пороговое значение - сегменты в 256 МБ, при котором область общей памяти использует псевдоним 1 ТБ. Более низкие значения необходимо использовать осторожно. Это может снизить количество промахов буфера SLB, но может увеличить число промахов элементов таблицы страниц (PTE), если многие области общей памяти, которые не используются для процессов используют псевдонимы.

Переменная `esid_1tb_allocator VMO` может быть задана для каждого процесса с помощью переменной среды *"ESID_1TB_ALLOCATOR=" VMM_CNTRL*. Для AIX версии 6.1 значение по умолчанию - 0, для AIX версии 7.0 - 1. Значения могут быть 0 или 1. Если задано значение 0 - включен прежний распределитель для ненаправленных присоединений. В противном случае используется новая стратегия выделения адресного пространства. Новый распределитель адрессного пространства прикрепляет любое ненаправленное выделение (например, SHM или MMAP) нового адресного интервала 0x0A00000000000000 - 0x0AFFFFFFFFFFFFFF в адресное пространство приложения. Распределитель оптимизирует распределение с тем, чтобы обеспечить лучшую поддержку для продвижения псевдонимов 1 ТБ. Такая оптимизация может привести к дырам в адресном пространстве, что считается нормальным при использовании ненаправленных присоединений. Направленные присоединения выполняются для интервала 0x0700000000000000 - 0x07FFFFFFFFFFFFFF, таким образом сохраняя совместимость с более ранней версией. В некоторых случаях, когда новая стратегия распределения создает проблему двоичной совместимости, поведение устаревшего распределителя может быть восстановлено путем установки переменной в 0.

Области общей памяти, не отвечающие требованию продвижения общего псевдонима, сгруппированы в области 1 ТБ. Для группы областей общей памяти в адресном пространстве приложения размером в 1 Тб, если приложение превышает пороговое значение сегментов в 256 Мб, происходит переход на использование неразделяемого псевдонима 1 Тб. В приложениях, в которых общая память подключается и отключается часто, низкое пороговое значение может привести к снижению производительности.

Для того, чтобы не допускать засорения пространства имен, все переменные среды используются под главной переменной `VMM_CNTRL`. Главная переменная задана символом `@`, разделяющим команды. Пример использования `VMM_CNTRL`:

```
VMM_CNTRL=SHM_1TB_UNSHARED=32@SHM_1TB_SHARED=5
```

Все значения переменных среды наследуются дочерним процессом функции `fork()` и инициализируются системными значениями по умолчанию в функции `exec()`. На 32-разрядные приложения изменения VMO или переменных среды не влияют.

Переменные VMO и переменные среды имеют аналогичные команды **vm_pattr**. Исключение составляет переменная `esid_allocator`. Эта переменная отсутствует в опциях `vm_pattr`, чтобы избежать ситуации, когда части адресного пространства общей памяти выделяются до выполнения команды.

Поддержка сродства памяти в AIX

AIX предоставляет возможность выделения памяти для процесса из модуля, содержащего процессор, вызвавший страничную ошибку. Эту возможность можно использовать в том случае, если в системе включена поддержка сродства памяти. Для этого следует задать переменную среды *MEMORY_AFFINITY*. Поддержка сродства памяти включена по умолчанию, однако ее можно отключить.

Системы IBM Платформа с процессором POWER с архитектурой POWER содержат модули, поддерживающие одно-, двух- и многопроцессорную конфигурацию. В каждом модуле установлены несколько процессоров и оперативная память. Хотя каждому процессору доступна вся память системы, доступ к локальной памяти выполняется быстрее, и ее пропускная способностью выше.

Если включена поддержка сродства памяти, каждому модулю принадлежит собственный пул vmpool, содержащий один или несколько пулов памяти. Каждому пулу памяти принадлежит собственный демон замещения страниц, **lrud**. Объем памяти в каждом пуле зависит от доступного объема памяти в модуле или от объема памяти, выделенного для VMM уровнем гипервизора.

При работе с AIX и при выключенной поддержке сродства памяти число пулов памяти зависит от объема памяти и числа CPU в системе.

Поддержку сродства памяти в AIX можно выключить с помощью следующей команды **vmo**:

```
vmo -o memory_affinity=0
```

Примечание: Для вступления изменений в силу необходимо выполнить команду `bosboot` или `reboot`. Значение по умолчанию равно 1, т. е. поддержка сродства памяти включена.

После включения поддержки сродства памяти операционная система выровняет структуры данных по границам блоков. По умолчанию память выделяется последовательно во всех MCM. Для включения преимущественно локального выделения памяти приложение должно экспортировать переменную среды MEMORY_AFFINITY:

```
MEMORY_AFFINITY=MCM
```

Этот параметр сохраняется при порождении новых процессов. Однако, для сохранения этого параметра в вызове функции **exec** необходимо передать ему соответствующую переменную среды.

Связанная информация

Команда **vmo** и “Настройка алгоритма замены страниц VMM” на стр. 153.

Команда или функция **bindprocessor**.

Класс WLM и атрибуты набора ресурсов.

Влияние локального выделения памяти на производительность

Действие локального выделения памяти на конкретное приложение трудно предсказать заранее. Производительность одних приложений может не измениться, других - увеличиться, третьих - уменьшиться.

Для увеличения производительности при включении данной опции приложение, как правило, должно быть связано с процессорами. Это необходимо для предотвращения перемещения приложения между процессорами, принадлежащими различным MCM.

Рекомендуется ограничить приложение процессорами, принадлежащими одному MCM. Приложения связываются с процессорами командой **bindprocessor** или функцией **bindprocessor()**. Кроме того, это можно сделать с помощью команд и служб сродства **набора ресурсов**.

Если приложению требуется больше процессоров, чем содержится в одном MCM, увеличение производительности от включения сродства памяти зависит от характера выделения и доступа к памяти различных нитей приложения. Производительность приложений, нити которых работают с памятью независимо, может увеличиться. Производительность приложений, нити которых работают с памятью совместно, может снизиться.

Размещение в памяти командой vmo

Можно выделять пользовательскую память с помощью параметров команды **vmo**. Можно также выбрать стратегию планирования: по первому касанию или карусельный метод.

При использовании стратегии планирования по первому касанию память выделяется из того модуля, на котором была запущена нить при первом касании сегмента памяти, являющегося первой страничной ошибкой. При использовании карусельного метода (определенного по умолчанию для всех типов памяти) при выделении памяти пулы vmpool чередуются.

С помощью следующих параметров команды **vmo** можно управлять размещением пользовательской памяти. Значение 1 этих параметров указывает на стратегию планирования по первому касанию, а значение 2 - на применение карусельного метода:

memplace_data
Этот параметр задает размещение памяти для следующих типов данных:

- Данные основного исполняемого файла, который либо инициирован, либо не инициирован
- Данные сегмента кучи
- Данные общей библиотеки
- Данные динамически загружаемых объектных модулей

Значение по умолчанию для этого параметра равно 2.

memplace_mapped_file
Этот параметр задает размещение памяти для файлов, отображенных в адресное пространство процесса, например, функции **shmat()** и функции **mmap()**. Значение по умолчанию для этого параметра равно 2.

memplace_shm_anonymous
Этот параметр задает размещение анонимной общей памяти, служащей в качестве рабочей памяти, созданной вызовом функции **shmget()** или функции **mmap()**. Доступ к такой памяти имеет только

создавший ее процесс либо его потомки. Эта память не связана с каким-либо именем или ключом. Значение по умолчанию для этого параметра равно 2.

memplace_shm_named
Этот параметр задает размещение именованной общей памяти, служащей в качестве рабочей памяти, созданной вызовом функции **shmget()** или функции **shm_open()**. Эта память связана с именем или ключом, что допускает одновременный доступ к ней нескольких процессов. Значение по умолчанию для этого параметра равно 2.

memplace_stack
Этот параметр задает размещение памяти для стека программ. Значение по умолчанию для этого параметра равно 2.

memplace_text
Этот параметр задает размещение памяти для текста приложения главного исполняемого файла, но не для подчиненных файлов. Значение по умолчанию для этого параметра равно 2.

memplace_unmapped_file
Этот параметр задает размещение памяти для не отображенного доступа к файлам, например, с помощью функций **read()** или **write()**. Значение по умолчанию для этого параметра равно 2.

Размещение в памяти с переменной среды MEMORY_AFFINITY

На уровне процесса можно настроить размещение пользовательской памяти с помощью переменной среды *MEMORY_AFFINITY*, которая переопределяет размещение памяти, выполненное с помощью параметров команды **vmo**.

В следующей таблице перечислены возможные значения переменной среды *MEMORY_AFFINITY*:

Значение	Поведение
MCM	Частная и общая память являются локальными.
SHM=RR	Как System V, так и Posix Real-Time общая память чередуются в MCM. Применяется к общим объектам памяти страниц по 4 КБ и больших страниц. Это значение доступно только в случае 64-разрядного ядра, а также если определено значение MCM.
LRU=EARLY	Демон LRU запускается в локальной памяти при достижении минимальных пороговых значений, например значения параметра *minfree*. При этом он не ожидает, пока все пулы системы достигнут минимальных пороговых значений. Это значение доступно, только если значение MCM также определено.

Можно задать несколько значений переменной среды *MEMORY_AFFINITY*, разделяя их знаком @.

Большие страницы

Основная задача функции поддержки больших страниц заключается в повышении производительности системы при работе с приложениями, требующими большого объема ресурсов (HPC), а также с приложениями, интенсивно обращающимися к памяти и использующими большие блоки виртуальной памяти. Это связано с меньшим количеством промахов в Таблице преобразования адресов (TLB), поскольку больший размер страницы позволяет TLB охватывать больший диапазон виртуальной памяти.

Большие страницы также увеличивают эффективность предварительной выборки из памяти, уменьшая число возобновлений предварительной выборки на границах страниц. AIX поддерживает работу с большими страницами как для 32-, так и для 64-разрядных приложений.

Для применения архитектуры больших страниц POWER4 необходимо, чтобы все виртуальные страницы в сегменте памяти 256 МБ были одного размера. В AIX поддержка этой архитектуры реализована с помощью модели смешанных процессов, в которой некоторые сегменты процесса сохраняются в страницах размером 4 КБ, а некоторые - в страницах размером 16 МБ. Приложения могут сохранять сегменты кучи или памяти в больших страницах. Дополнительная информация приведена в разделе “Настройка приложений для работы с большими страницами” на стр. 166.

В AIX поддерживаются независимые пулы физической памяти для страниц размером 4 КБ и 16 МБ. Задать объем физической памяти для пула страниц размером 16 МБ можно с помощью команды **vmo**. Пул больших страниц является динамическим, поэтому указанный объем физической памяти вступает в силу немедленно, без перезагрузки системы. Оставшийся объем физической памяти выделяется для хранения страниц размером 4 МБ.

В AIX обработка больших страниц выполняется так же, как и обработка зафиксированной памяти. AIX не поддерживает подкачку больших страниц памяти. Данные приложений, хранящиеся в больших страницах, находятся в физической памяти до момента завершения работы приложения. Механизм защиты управления доступом не позволяет приложениям без соответствующих прав доступа применять большие страницы или выделенную для их хранения физическую память. Он также не позволяет пользователям, у которых нет нужных прав доступа, запускать приложения, применяющие такие страницы. Для того чтобы пользователь, отличный от root, мог применять большие страницы, необходимо включить опцию **CAP_BYPASS_RAC_VMM** с помощью команды **chuser**. Ниже приведен пример предоставления прав доступа **CAP_BYPASS_RAC_VMM** главным администратором:

```
# chuser capabilities=CAP_BYPASS_RAC_VMM,CAP_PROPAGATE <ИД-пользователя>
```

Настройка приложений для работы с большими страницами

Существуют разные способы настройки приложений для работы с большими страницами памяти.

Хранение данных и сегментов кучи в больших страницах:

Выбрать большие страницы в качестве способа для хранения данных или кучи необходимо при запуске приложения, поскольку изменить режим работы уже запущенного приложения нельзя. Режим использования больших страниц наследуется дочерним процессом функции **fork()**.

В приложении можно задать хранение сегментов кучи, инициализированных данных программы и неинициализированных данных программы (BSS) и с помощью следующих методов:

- “Настройка в исполняемом файле работы с большими страницами”
- “Установка переменной среды для применения больших страниц” на стр. 167

Можно выбрать один из следующих режимов хранения сегментов кучи и данных приложений в больших страницах:

- “Необязательный режим” на стр. 167
- “Обязательный режим” на стр. 167

Из-за дискретности защиты страниц в 32-разрядных приложениях, хранящих сегменты данных и кучи в больших страницах, применяется 32-разрядная модель обработки. В других моделях обработки применяются страницы размером 4 КБ с различными атрибутами защиты, что не позволяет реализовать защиту при дискретности страниц в 16 МБ.

Настройка в исполняемом файле работы с большими страницами:

Заголовок XCOFF исполняемого файла содержит флаг **blpdata**, указывающий, следует ли хранить сегменты данных и кучи в больших страницах.

Чтобы включить для исполняемого файла применение больших страниц, воспользуйтесь следующей командой:

```
# ldedit -blpdata <имя-файла>
```

Отключить хранение сегментов кучи и данных исполняемого файла в больших страницах можно, убрав флаг с помощью следующей команды:

```
# ldedit -bnolpdata <имя-файла>
```

Можно также задать опцию **blpdata** при компоновке приложения с помощью команды **cc**.

Установка переменной среды для применения больших страниц:

Переменная среды *LDR_CNTRL* позволяет настроить в приложении хранение сегментов кучи и данных в больших страницах. Переменная среды имеет приоритет перед флагом **blpdata** исполняемого файла.

Допустимы следующие значения переменной среды *LDR_CNTRL*:

- Значение **LDR_CNTRL=LARGE_PAGE_DATA=Y** указывает, что сегменты кучи и данные запускаемого приложения следует хранить в больших страницах; это значение аналогично установке флага больших страниц.
- Значение **LDR_CNTRL=LARGE_PAGE_DATA=N** указывает, что сегменты кучи и данные запускаемого приложения не следует хранить в больших страницах; имеет приоритет перед флагом больших страниц.
- Значение **LDR_CNTRL=LARGE_PAGE_DATA=M** указывает, что для хранения сегментов кучи и данных запускаемого приложения необходимо применять только большие страницы

Примечание: Переменную среды больших страниц рекомендуется устанавливать только для определенных приложений, для которых это может привести к повышению производительности. В противном случае быстродействие системы может снизиться.

Необязательный режим:

В необязательном режиме часть сегментов кучи приложения может храниться в больших страницах, а часть - в страницах по 4 КБ. В том случае, если в системе недостаточно больших страниц, для хранения сегментов кучи и данных применяются страницы размером 4 КБ.

В необязательном режиме большие страницы применяются, если это возможно. В качестве критериев используются следующие условия:

- У пользователя есть права доступа на применение больших страниц.
- Аппаратное обеспечение системы поддерживает работу с большими страницами.
- В системе определен пул памяти больших страниц.
- Размер пула памяти больших страниц позволяет хранить в нем весь сегмент с большими страницами.

Если какое-либо из перечисленных условий не выполнено, то сегменты данных и кучи приложения хранятся в страницах размером 4 КБ.

Исполняемые файлы, в которых установлен флаг больших страниц, выполняются в необязательном режиме.

Обязательный режим:

Если в обязательном режиме приложение запрашивает сегмент кучи, и в системе недостаточно больших страниц для обработки этого запроса, то возникает сбой; в большинстве случаев работа приложения, отправившего запрос, завершается с выдачей сообщения об ошибке.

Если применяется обязательный режим, необходимо отслеживать размер пула больших страниц, и не допускать отсутствия в пуле больших страниц. В противном случае в работе приложений, использующих обязательный режим, будут возникать сбои.

Хранение сегментов общей памяти в больших страницах:

Для хранения сегментов общей памяти приложения в больших страницах необходимо установить флаги **SHM_LGPAGE** и **SHM_PIN** функции **shmget()**. Если большие страницы недоступны, сегмент общей памяти будет храниться в страницах размером 4 КБ.

Физическая память для хранения сегментов общей памяти, данных и кучи выделяется из пула физической памяти больших страниц. Необходимо обеспечить наличие в этом пуле достаточного числа страниц для размещения общей памяти, данных и кучи.

Настройка системы для поддержки больших страниц

В системе необходимо настроить поддержку больших страниц и указать объем физической памяти, который будет применяться для их размещения.

По умолчанию пулу больших страниц физической памяти память не выделяется. Объем физической памяти, используемой в режиме больших страниц, задается командой **vmo**. Следующая команда выделяет 1 ГБ памяти для пула больших страниц физической памяти:

```
# vmo -r -o lgpg_regions=64 -o lgpg_size=16777216
```

Для применения больших страниц физической памяти необходимо разрешить системный вызов **SHM_PIN shmget()** с помощью следующей команды, которую не нужно повторять при перезагрузках системы:

```
# vmo -p -o v_pinshm=1
```

Просмотреть количество используемых в системе больших страниц можно с помощью команды **vmstat -l**, как показано ниже:

```
# vmstat -l

нити     память             страница          ошибки        cpu      большая-страница

----- ----------- ---------------------- ------------ ----------- ------------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa   alp   flp
 2  1 52238 124523   0   0   0   0    0   0 142   41  73  0  3 97  0     16     16
```

В приведенном выше примере система содержит 16 активных больших страниц (параметр `alp`), и 16 свободных больших страниц (параметр `flp`).

Замечания по работе с большими страницами

Большие страницы - это специальное средство оптимизации производительности; в обычных случаях применять это средство не рекомендуется. Обратите внимание на то, что при использовании больших страниц далеко не все приложения начинают работать быстрее. Более того, производительность некоторых приложений, в том числе тех, в которых используется большое число функций **fork()** может снизиться.

В большинстве случаев рекомендуется не использовать переменную среды *LDR_CNTRL*, а создавать специальные исполняемые файлы, поскольку при этом большие страницы будут применяться только приложениями, для которых это оправдано.

Перед тем как начать использование больших страниц, оцените возможное влияние данного решения на производительность системы. Несмотря на то, что производительность некоторых приложений может возрасти, общее быстродействие системы может снизиться из-за снижения суммарного объема памяти страниц размером 4 КБ. Большие страницы рекомендуется применять в случае, если общий объем физической памяти системы позволяет включить эту функцию без существенного снижения производительности, вызванного нехваткой страниц размером 4 КБ.

Поддержка разных размеров страниц

Процессор POWER5+ поддерживает четыре размера страниц виртуальной памяти: 4 КБ, 64 КБ, 16 МБ и 16 ГБ. Серверы на основе процессоров IBM Power Systems также поддерживают страницы размером 64 КБ в сегментах с базовым размером страницы - 4 КБ. AIX использует этот процесс для увеличения быстродействия при использовании страниц размером 64 КБ, когда необходимо перейти на страницы 4 КБ, чтобы избежать потерь памяти.

Увеличение размера страниц виртуальной памяти до 64 КБ и больше может значительно улучшить быстродействие и производительность в силу архитектуры аппаратного обеспечения. Это может уменьшить

латентность, т.к. требуется меньше времени на преобразование адреса виртуальной страницы в адрес физической страницы, это связано с повышением эффективности аппаратных кэшей преобразования, таких как таблица преобразования адресов (TLB). В этих кэшах может храниться только ограниченное число записей, а использование страниц большего размера увеличивает объём виртуальной памяти, соответствующей каждой такой записи. Таким образом, увеличивается объём памяти, при обращении к которой не требуются дополнительные действия, влекущие за собой задержки.

Страницы размером 16 МБ и 16 ГБ предназначены прежде всего для высокопроизводительных сред, тогда как страницы размером 64 КБ могут применяться для любых задач, и в большинстве случаев позволяют увеличить быстродействие по сравнению со страницами 4 КБ.

Размеры страниц, поддерживаемые процессорами разных типов

Для получения размеров страниц, поддерживаемых AIX на данной системе используйте команду **pagesize** с ключом **-a**.

AIX 6.1 и выше поддерживает сегменты с двумя видами размера страницы: 4 КБ и 64 КБ. По умолчанию процессы используют эти размеры страниц. Эти размеры можно переопределить с помощью механизма выбора размера страницы.

Таблица 2. Поддержка размеров страниц памяти AIX на разном аппаратном обеспечении System p

Размер страницы	Аппаратное обеспечение	Требуется пользовательская конфигурация	Ограничено
4 КБ	Все	Нет	Нет
64 КБ	POWER5+ или более новая версия	Нет	Нет
16 МБ	POWER4 и более поздние модели	Да	Да
16 ГБ	POWER5+ и более поздние модели	Да	Да

Таблица 3. Поддерживаемые размеры страниц сегмента

Базовый размер страницы сегмента	Поддерживаемые размеры страниц	Минимальные требования к аппаратному обеспечению
4 КБ	4 КБ/64 КБ	POWER6
64 КБ	64 КБ	POWER5+
16 Мб	16 МБ	POWER4
16 ГБ	16 ГБ	POWER5+

Как и во всех предыдущих версиях AIX, размер страниц по умолчанию равен 4 КБ. Это значит, что процесс использует страницы с размером 4 КБ если пользователь явно не укажет другой размер.

Поддержка страниц размером 64 КБ

Страницы размером 64 КБ использовать проще, и многие приложения будут работать с ними быстрее, поэтому в AIX поддержка страниц размером 64 КБ реализована в полной мере.

Для работы со страницами размером 64 КБ никаких изменений конфигурации системы не требуется. Если система поддерживает страницы размером 64 КБ, то ядро AIX автоматически включает эти функции. Страницы размером 64 КБ включаются в динамический пул памяти, размером которого управляет система AIX. AIX будет изменять буферы со страницами размером 4 КБ и 64 КБ для задач, которым требуются страницы разного размера. Отслеживать число страниц размером 4 КБ и 64 КБ в системе можно командами **svmon** и **vmstat**.

Поддержка страниц с динамически изменяющимся размером

Процессоры до POWER6 поддерживали только размер одной страницы в сегменте. Системный администратор или пользователь должны были подбирать оптимальный размер страницы для приложения на основании его потребности в памяти.

Выбор страниц, размером 4 КБ, означал наименьшее потерю памяти, т.к. фактически использовались только 4 КБ. Больший размер страницы означал потерю памяти (зарезервированной, но не используемой) в зависимости от адреса рабочего набора, и увеличение быстродействия с меньшим количеством трансляций из виртуальной в физическую память. Пользователь должен явно задавать размер страниц, больший чем 4 КБ.

POWER6 позволяет использовать страницы разного размера в одном сегменте. Архитектура поддерживает различные размещения разных размеров страниц. Однако, POWER6 поддерживает совместное использование страниц только размеров 4 КБ и 64 КБ. AIX 6.1 использует данную возможность для применения страниц размером 4 КБ в местах редкого обращения к памяти и 64 КБ - в местах частого обращения. Данная процедура выполняется автоматически. Эта функция AIX называется Поддержка динамически изменяемого размера страниц (VPSS). Для избежания ошибок совместимости с предыдущими версиями Поддержка динамически изменяемого размера страниц выключена в сегментах, в которых размер страницы задан явно пользователем (См. Поддержка приложений с изменяемым размером страницы).

По умолчанию размер страницы равен 4 KB и размер трансляции - 4 КБ. Трансляции происходит до тех пор, пока не произойдет обращение ко всем 16 страницам, размером 4 КБ (64 КБ). После обращения ко всем 16 страницам происходит проверка того, что они все находятся в одинаковом состоянии - защита чтение/запись, защита от выполнения, ключ защиты памяти и состояние защиты ввода-вывода). Если состояния одинаковые, то трансляции, размером 4 КБ, заменяются на 64 КБ.

Трансляции размером 64 KB используются до тех пор, пока все страницы размером 15 КБ имеют одинаковое состояние. Их состояние может изменяться, например, с помощью функции **mprotect**. В этом случае они разбиваются на страницы размером 4 КБ до тех пор, пока их состояние не станет одинаковым.

Некоторые приложения используют размер 64 КБ, даже если это приводит к малому использованию областей. Изменив Фактор агрессивности продвижения размера страницы (PSPA), можно понизить требование к загрузке памяти, и в этом случае размер страницы может быть увеличен до 64 КБ. Данный фактор задается для всей системы с помощью переменной **vmm_default_pspa vmo**, а для отдельного процесса - с помощью команды **vm_pattr**.

С помощью команды **svmon** можно получить отчет использования страниц памяти изменяемого размера. Дополнительная информация о команде **vmo** приведена в разделе *Справочник по командам, том 6*.

Размеры страниц для высокопроизводительных сред

В дополнение к размерам страниц 4 КБ и 64 КБ AIX поддерживает страницы с размером 16 МБ, также называемые *большими* (large), и страницы с размером 16 ГБ *крупными* (huge). Станицы таких размеров должны использоваться только в высокопроизводительных средах и по умолчанию не задаются в AIX.

Для использования страниц таких размеров требуется настроить AIX вручную. Также необходимо указать число страниц, имеющих такой размер. AIX не может автоматически изменять число настроенных страниц с размером 16 МБ и 16 ГБ по требованию.

Память, выделяемая для страниц с размером 16 МБ может быть использована только для них, то же самое и для страниц с размером 16 ГБ. Поэтому страницы таких размеров следует использовать исключительно в высокопроизводительных средах. Кроме того, их использование ограничено: для выделения для них памяти пользователь должен обладать **CAP_BYPASS_RAC_VMM** и **CAP_PROPAGATE**, либо иметь права **root**.

Настройка числа больших страниц:

Число больших страниц размером 16 МБ можно настроить командой **vmo**.

В следующем примере выделяется область 1 ГБ из страниц размером 16 МБ:

```
# vmo -r -o lgpg_regions=64 -o lgpg_size=16777216
```

Изменения в конфигурации больших страниц вступят в силу только после выполнения команды **bosboot** и перезагрузки системы. В системах с поддержкой динамическое распределение ресурсов между разделами (DLPAR) опцию **-r** можно не указывать, потому что страницы размером 16 МБ будут настроены динамически без необходимости перезагрузки системы.

Дополнительная информация о страницах размером 16 МБ приведена в разделе “Большие страницы” на стр. 165.

Настройка числа очень больших страниц:

Настройка очень больших страниц выполняется в Консоль аппаратного обеспечения (HMC).

1. Выберите в управляемой системе **Свойства > Память > Дополнительные опции > Показать сведения**, чтобы изменить число страниц размером 16 ГБ.
2. Подключите очень большие страницы размером 16 к разделу в профайле раздела.

Поддержка разных размеров страниц приложениями

Размер страниц можно указать для каждой из четырех областей адресного пространства 32-разрядного или 64-разрядного процесса.

Задать размеры страниц можно посредством переменной среды или в параметрах двоичного файла XCOFF приложения командой **ldedit** или **ld**:

Область	Опция ld или ldedit	Переменная среды LDR_CNTRL	Описание
Данные	-bdatapsize	DATAPSIZE	Инициализированные данные, bss, куча
Стек	-bstackpsize	STACKPSIZE	Стек главной нити
Текст	-btextpsize	TEXTPSIZE	Текст главного исполняемого файла
Общая память	нет	SHMPSIZE	Общая память, выделенная процессом

Для каждой из четырех областей адресного пространства процесса можно указывать разный размер страницы. Для любого интерфейса размер указывается в байтах. Допустимо использование следующий суффиксов:

- K (килобайт)
- M (мегабайт)
- G (гигабайт)

Регистр суффикса может быть как верхним, так и нижним.

Для всех четырех областей поддерживаются только размеры 4 КБ и 64 КБ. Использование страниц размером 16 МБ поддерживается только для областей данных процесса, текста процесса и общей памяти процесса. Использование страниц размером 16 ГБ поддерживается только для области общей памяти процесса.

Задав в явном виде размер страницы, вы запрещаете использование меньших размеров в сегменте.

Если указан неподдерживаемый размер, то будет использован наименьший ближайший из поддерживаемых. Если указанный размер не превышает 4 КБ, то будет использован размер 4 КБ.

Доступна поддержка указания размера страницы для использования общей памятью процесса в переменной среды SHMPSIZE. В предыдущих версиях AIX переменная среды SHMPSIZE игнорировалась. Переменная среды SHMPSIZE применяется только к областям общей памяти, созданным процессом при его вызове подпрограмм **shmget**, **ra_shmget** и **ra_shmgetv**. Переменная среды SHMPSIZE не применяется к областям

общей памяти EXTSHM и областям общей памяти POSIX в реальном времени. Переменная среды SHMPSIZE процесса не применяется к областям общей памяти, так как процесс использует области общей памяти , которые были созданы другими процессами.

Задание предпочитаемых размеров страниц для приложения с помощью команды ldedit или ld:

Предпочитаемые размеры страниц для приложения можно задать в его двоичном файле XCOFF/XCOFF64 с помощью команды **ldedit** или **ld**.

Команды **ld** и **cc** позволяют задать предпочитаемый размер страниц на этапе компоновки:

```
ld -o mpsize.out -btextpsize:4K -bstackpsize:64K sub1.o sub2.o
cc -o mpsize.out -btextpsize:4K -bstackpsize:64K sub1.o sub2.o
```

Команда **ldedit** позволяет задать предпочитаемый размер страниц для уже скомпонованного приложения:

```
ldedit -btextpsize=4K -bdatapsize=64K -bstackpsize=64K mpsize.out
```

Примечание: Для команды **ldedit** размер передаётся с использованием знака равенства (=), а для команд **ld** и **cc** с использованием двоеточия (:).

Задание предпочитаемых размеров страниц для приложения с помощью переменной среды:

Предпочитаемых размеров страниц для процесса можно задать с помощью переменной среды *LDR_CNTRL*.

Например, следующая команда устанавливает размер блока данных 4 КБ, текстового блока, стека и общей памяти - 64 КБ для процесса `mpsize.out` на поддерживающем такую конфигурацию аппаратном обеспечении:

```
$ LDR_CNTRL=DATAPSIZE=4K@TEXTPSIZE=64K@SHMPSIZE=64K mpsize.out
```

Эта переменная среды имеет больший приоритет, чем соответствующий параметр в заголовке XCOFF. Переменная среды *DATAPSIZE* имеет больший приоритет, чем *LARGE_PAGE_DATA*.

Замечания по поддержке разных размеров страниц:

На поддержку AIX разных размеров страниц могут повлиять ограничения, связанные с 32-разрядными процессорами, стеками нитей, общими библиотеками и данными страниц большого объёма.

32-разрядные процессоры

При модели 32-разрядного адресного пространства AIX по умолчанию стек главной нити и данные процесса размещаются в одном сегменте PowerPC с размером 256 МБ. На данный момент в одном сегменте могут располагаться только страницы одного размера. Поэтому, если для стека и данных стандартного 32-разрядного процесса указаны разные размеры страниц, для обоих будет использован наименьший размер.

Для использования страниц действительно разных размеров необходимо выбрать одну из альтернативных моделей адресного пространства, предназначенных для поддержки больших и очень больших программ и размещающих кучу процесса в сегменте, отличном от сегмента стека.

Стеки нитей

По умолчанию стеки нитей многопоточных процессов помещаются в куче данных. Поэтому для многопоточных процессов указанный размер страницы стека применяется только к главной нити процесса. Стеки для остальных нитей будут выделяться из кучи данных процесса и, соответственно, для них будет использоваться размер страницы кучи.

Использование страниц с размером 64 КБ вместо 4 КБ для данных многопоточных процессов уменьшает максимальное число нитей, которое может быть создано для него из-за выравнивания вспомогательных страниц стека. Для приложений, требующих большего числа нитей, необходимо отключить вспомогательные страницы стека заданием значения 0 для переменной *AIXTHREAD_GUARDPAGES*.

Общие библиотеки

В системах, поддерживающих страницы с размером 64 КБ AIX будет использовать их для областей текста глобальных общих библиотек в целях повышения производительности.

Данные больших страниц

Переменная среды *DATAPSIZE* имеет больший приоритет, чем *LARGE_PAGE_DATA*. Кроме того, параметр *DATAPSIZE* в двоичном файле XCOFF приложения имеет больший приоритет, чем любой параметр lpdata в том же файле.

Поддержка страниц с изменяющимся большим размером

Серверы на основе процессоров IBM Power Systems поддерживают в одном сегменте страницы размером 4 КБ, 64 КБ и 16 МБ.

Операционная система AIX поддерживает работу со страницами размером 16 МБ для обеспечения высокопроизводительных сред, однако такие страницы не отличаются гибкостью и простотой управления. Эти страницы нельзя удалить из памяти и создать автоматически.

Серверы на основе процессоров IBM Power Systems поддерживают работу со смешанными страницами размером 16 МБ, что дает операционной системе гибкость в управлении памятью с помощью дробления страниц на фрагменты размером 4 и 64 КБ при сохранении преимущества обращения к памяти с использованием аппаратных страниц размером 16 МБ. Использование этой аппаратной функции в операционной системе AIX носит название *Поддержка страниц с изменяющимся большим размером* (VLPSS).

VLPSS размещает области пользовательской памяти размером 16 МБ в блок смежных физических страниц размером 4 или 64 КБ. Эти страницы памяти доступны за одно обращение 16 МБ. Так как используется одиночное обращение к памяти размером 16 МБ, составляющие ее страницы размером 4 и 64 КБ должны иметь одинаковые атрибуты страниц и должны располагаться в памяти. Возможные атрибуты страницы: защита страницы от чтения/записи, защита ключа памяти и защита от выполнения.

Страницы VLPSS размером 16 МБ могут быть разделены на исходные страницы размером 4 или 64 КБ средствами операционной системы. Страницы разделяются, если операционной системе требуется вытеснить страницы из памяти в устройство подкачки или если приложение изменяет атрибуты страницы для области размером 16 МБ таким образом, что страницы перестают быть одинаковыми. Настоящие страницы размером 16 МБ не предоставляют такой гибкости.

Для получения преимуществ функции VPLSS приложения могут использовать системный вызов vm_pattr либо задать команду **VM_PA_SET_PSIZE_EXTENDED**. Операционная система может выбрать: принять рекомендацию системного вызова vm_pattr или отклонить его, если это может повлиять на систему.

Размер страницы 16 МБ имеет больший объем непрерывной памяти по сравнению с размером 64 КБ динамически изменяемых страниц. Размещение и предоставление памяти для использования VLPSS - это дорогостоящая операция, отрицательно влияющая на производительность всей системы. Поэтому предоставление памяти с размером страниц 16 МБ имеет ограничения, которых не имеет поддержка динамических страниц переменного размера.

Использование функции VLPSS запрещено для пользователей с возможностями CAP_BYPASS_RAC_VMM и CAP_PROPAGATE, а также для пользователей с правами root.

Для использования функции VLPSS область памяти размером 16 МБ должна полностью размещаться в памяти. Для применения функции VLPSS операционная система должна иметь большой объем памяти. Минимальный необходимый объем памяти: 16 ГБ.

Размер страницы и общая память

Размер страниц общей памяти System V можно задать с помощью команды **SHM_PAGESIZE** системного вызова shmctl().

Задав в явном виде размер страницы, вы запрещаете использование меньших размеров в сегменте.

Дополнительная информация о задании размера страницы для области общей памяти с помощью shmctl() доступна в соответствующем разделе *Technical Reference: Base Operating System and Extensions, Volume 2*.

Определение размеров страниц процесса с помощью команды ps

Команда **ps** позволяет отслеживать размеры базовых страниц, используемые для блока данных, стека и текста процесса.

Ниже приведен пример вывода команды **ps -Z**. В столбце *DPGSZ* показывается размер страницы для блока данных; в столбце *SPGSZ* - для стека; в *TPGSZ* - для текстового блока процесса.

```
# ps -Z
    PID    TTY  TIME DPGSZ SPGSZ TPGSZ CMD
 311342  pts/4  0:00    4K    4K    4K ksh
 397526  pts/4  0:00    4K    4K    4K ps
 487558  pts/4  0:00   64K   64K    4K sleep
```

Наблюдение за размером страниц с помощью команды vmstat

Команда **vmstat** имеет две опции, связанных с выводом статистики по использованию памяти для данного размера страницы.

vmstat -p
: Вывести глобальные сведения vmstat и статистику по каждому размеру страницы отдельно.

vmstat -P
: Вывести статистику по каждому размеру страницы отдельно.

Для обоих опций в качестве аргумента необходимо указывать список размеров страниц (через запятую) или ключевое слово **all** для вывода сведений для всех поддерживаемых размеров станиц с одним или несколькими кадрами. В следующем примере выведены сведения для всех размеров страниц с кадрами в системе:

```
# vmstat -P all
Конфигурация системы: mem=4096MB
разм.            память                      страница
----- ------------------------- ------------------------------------
           разм.  дост. свободно   re    pi    po    fr     sr    cy
   4K   542846   202832   329649     0     0     0     0      0     0
  64K    31379      961    30484     0     0     0     0      0     0
```

Мониторинг системного размера страниц с помощью команды svmon

С помощью команды **svmon** можно просмотреть размер страницы в системе.

Команду **svmon** можно расширить для отображения постраничной статистики. Например, для отображения глобальной статистики о каждом размере страницы можно использовать опцию **-G** с командой **svmon**:

```
# svmon -G
               разм.     занято      своб.      закр.     вирт.
память      8208384    5714226    2494158     453170    5674818
пр. подкачки 262144      20653

               раб.       пост.       клнт.
pin          453170          0           0
занято      5674818        110       39298
```

```
разм.стр.   разм.пула   занято   пр.пдкчк.      закр.      вирт.
s   4 КБ          -    5379122      20653     380338    5339714
m  64 КБ          -      20944          0       4552      20944
```

Дополнительная информация приведена в описании команды **svmon** в книге *Справочник по командам, том 5*.

Замечания по работе с памятью для больших страниц

При оценке влияния на производительность приложения больших страниц памяти необходимо учесть требования к памяти рабочей части программы.

Работа с большими страницами может увеличить общие требования к памяти вследствие фрагментации памяти. Командами **svmon** и **vmstat** можно просмотреть использование памяти и определить, увеличивается ли занятый объем при работе с большими страницами.

Рекомендуется использовать размер страницы по умолчанию и предоставить операционной системе решать, когда использовать другой размер (например, 64 KB). Если рабочий набор приложения часто используется, и требуются области размером 64 КБ, тогда можно установить размер страницы 64 КБ, т.к. память не будет расходоваться впустую.

Непрерывная оптимизация памяти

При непрерывной оптимизации памяти программа объединяет страницы памяти и увеличивает размер страницы в динамическом режиме.

Поддержка динамических страниц переменного размера

Поддержка динамических страниц переменного размера (DVPSS) в AIX версии 6.1 состоит в использовании страниц размером 4 Кб, когда обращения к памяти происходят редко, и страниц размером 64 Кб в местах частого обращения к памяти. Переключение размера страниц происходит автоматически, без участия пользователя.

DVPSS позволяет POWER6 использовать страницы разного размера в одном сегменте. Архитектура размещает страницы разного размера различным образом. Однако POWER6 поддерживает только два размера страниц - 4 Кб и 64 Кб.

По умолчанию размер страницы составляет 4 Кб, размер трансляции - 4 Кб. Трансляции происходят до тех пор, пока не произойдет обращение ко всем 16 4-килобайтным страницам (64 Кб). После обращения ко всем 16 страницам происходит проверка того, что они все находятся в одинаковом состоянии - наличие защиты от чтения/записи, защиты от выполнения, ключа защиты памяти и состояние защиты ввода-вывода). Если состояния одинаковые, то трансляции, размером 4 КБ, заменяются на 64 КБ.

Агент непрерывной оптимизации программы (CPOagent)

Недостаток метода DVPPS в операционной системе состоит в том, что для перехода на размер страницы 64 Кб система должна обратиться ко всем 16 страницам, размером 4 Кб. CPOagent позволяет снять это ограничение и объединять страницы в процессе работы программы. Эта функция относится к AIX 6.1 Technology уровня 6 и выше.

CPOagent расположен в каталоге:

```
usr/lib/perf/CPOagent
```

Синтаксис

CPOagent [-f *файл-конфигурации*]

Флаг

Элемент	Описание
-f	Изменяет имя файла конфигурации по умолчанию. Если эта опция не задана, используется файл `/usr/lib/perf/CPOagent.cf`.

По умолчанию, CPOagent не запускается. Пользователь root должен запустить CPOagent явно. После запуска CPOagent выполняется в фоновом режиме и анализирует возможность оптимизации каждого процесса путем его переключения на страницы большего размера. Агент сравнивает объем памяти и процессорное время, используемые процессом, с пороговым значением.

Примечание: CPOagent может увеличить размер страницы до 64 Кб.

Файл конфигурации CPOagent

Сразу после запуска CPOagent считывает файл конфигурации и анализирует его. Файл конфигурации - это файл stanza с полями:

```
TCPU=<n1>
TMEM=<n2>
PATI=<n3>
PATM=<n4>
PPTS=<n5>
TOPM=<n6>
PFLR=<c>
```

Описание полей конфигурации:

Поля	Описание
TCPU	Задает ограничение процессорного времени, которое может использовать один процесс (в процентах) По умолчанию: 25 Минимально: 10 Максимально: 100
TMEM	Задает ограничение памяти, используемой процессом (в Мб) По умолчанию: 1 Минимально: 1
PATI	Задает периодичность анализа размера страницы (PATI), в минутах. Временной интервал, с которым агент анализирует каждый процесс и определяет, нужно ли увеличить размер страницы для этого процесса. По умолчанию: 15 Минимально: 5 Максимально: 60
PATM	Задает периодичность анализа страниц (PATM), в секундах. Временной интервал, с которым собирается статистика по страницам и принимается решение: стоит ли объединить соседние страницы и перейти к большему размеру. По умолчанию: 30 Минимально: 5 Максимально: 180
PPTS	Количество страниц для анализа на предмет объединения. Указывает количество страниц, к которым должно было произойти обращение, чтобы их объединили. По умолчанию: 4 Минимально: 4

Поля	Описание
TOPM	Количество процессов, больше всех использующих процессорное время, в которых следует объединить страницы и увеличить размер страниц. По умолчанию: 2 Минимально: 1
PFLR	Задает маску имен процессов, которые должен анализировать CPOagent на предмет возможности оптимизации. Переменная содержит символы подстановки для фильтрации всех процессов.

Преимущества использования CPOagent

Преимущества механизма CPOagent включают:

- Приложения можно настраивать в прозрачном режиме. При этом сами приложения изменять не требуется.
- Увеличение размера страницы происходит при выполнении ряда условий, указываемых администратором в файле `CPOagent.cf`, в зависимости от рабочей нагрузки на процессор и доступных ресурсов. Использование сценариев позволяет оптимизировать управление размером страниц. При этом поддержка динамических страниц переменного размера настраивается для каждого приложения отдельно.

Пример сценария

Пусть в системе выполняется приложение `StressEngine`. Оно использует большие объемы памяти и большую долю процессорного времени. Без CPOagent приложение `StressEngine` не сможет перейти в режим поддержки динамических страниц переменного размера, пока не обратится ко всем 16 страницам в одном из своих сегментов, и все страницы не перейдут в одинаковое состояние. Размер страницы можно просмотреть в отчете о процессе с помощью команды **svmon**.

```
# svmon -P 8454254 -O pgsz=on,unit=MB,segment=on
Ед. изм: Мб
--------------------------------------------------------------------------------
     Pid Command          Inuse      Pin     Pgsp  Virtual
 8454254 StressEngine   157.87     42.3        0   157.84

     Разм.Стр.     Занято      Закр.   Пр.пдкчк. Виртуальн.
     s    4 Кб               64.2       0.02          0       64.2
     m   64 KB               93.7       42.3          0       93.7

    Vsid      Esid Тип  Описание                PSize  Inuse   Pin Pgsp Virtual
  86f49b         2 раб. проц. прив.                 s   64.1  0.02    0    64.1
    9000         d раб. данные общ. библ.           m   47.9     0    0    47.9
    8002         0 дерево раб. проц.                m   45.8  42.3    0    45.8
                   children=939b1c, 0
  80fdc3         f раб. данные общ. библ.           s   0.09     0    0    0.09
  85fd37         1 clnt code,/dev/hd1:10            s   0.02     0    -       -
```

Пусть запущен CPOagent, а файл `CPOagent.cf` выглядит следующим образом:

```
-------------------------------------------
TCPU=25
TMEM=50
PATI=15
PATM=30
PPTS=4
TOPM=2
PFLR=Stress*
-------------------------------------------
```

Согласно файлу конфигурации, CPOagent работает с интервалом 15 минут (PATI =15). Через каждые 15 минут он проверяет, сколько памяти и процессорного времени использует процесс. Первые 2 процесса (TOPM =2), в название которых входит строка Stress (PFLR = Stress*), использование процессора превышает 25% (TCPU = 25), а использование памяти - 50 Мб (TMEM = 50), являются претендентами на объединение

смежных страниц и увеличение размера для последующих страниц. В каждом из процессов проверяются произвольные 4 страницы (PPTS = 4), а затем запускается алгоритм объединения страниц и увеличения размера страницы. Кроме того, статистика по страницам обновляется каждые 30 секунд (PATM =30), и на ее основании выбираются страницы - кандидаты на объединение. Таким образом, после запуска CPOagent нет необходимости ждать, пока ко всем 16 в сегменте произойдет обращение. CPOagent определяет необходимость объединения страниц в приложении на основании конфигурации `CPOagent.cf` и том, сколько приложению требуется памяти и времени процессора. Объединенные страницы включаются в отчет о процессе, который можно просмотреть с помощью команды **svmon**.

```
# svmon -P 8454254 -O pgsz=on,unit=MB,segment=on
Ед. изм: Мб
-------------------------------------------------------------------------------
     Pid Command          Inuse      Pin     Pgsp  Virtual
 8454254 StressEngine    157.87     42.3        0   157.84

     Разм.Стр.     Занято      Закр.   Пр.пдкчк. Виртуальн.
     s    4 Кб               64.2       0.02          0       64.2
     m   64 KB               93.7       42.3          0       93.7

    Vsid      Esid Тип  Описание                PSize  Inuse   Pin Pgsp Virtual
  86f49b         2 раб. проц. прив.                 sm   64.1  0.02    0    64.1
    9000         d раб. данные общ. библ.           m   47.9     0    0    47.9
    8002         0 дерево раб. проц.                m   45.8  42.3    0    45.8
                   children=939b1c, 0
  80fdc3         f раб. данные общ. библ.           sm   0.09     0    0    0.09
  85fd37         1 clnt code,/dev/hd1:10            s   0.02      0    -       -
```

Выгрузка прерванных нитей VMM

Инфраструктура выгрузки прерванных нитей VMM (VTIOL) позволяет VMM передавать обработку службы `iodone()` нитям ядра.

Функция VTIOL позволяет сократить вероятность снижения производительности нити с высоким приоритетом в результате выполнения процесса процесса `iodone()`. Функция VTIOL обрабатывает службу `iodone()` с помощью фоновых нитей вместо прерывания процесса с высоким приоритетом. VMM использует несколько эвристических методов для определения необходимости выгрузки обработки службы `iodone()`. Например, для выгрузки доступны отдельные операции ввода-вывода без нитей `waiter`, такие как фоновые операции записи и операции упреждающего чтения. Явным образом применяемые нити `waiter` могут указывать, что операции ввода-вывода должны выполняться с более высоким приоритетом. В таких случаях операции ввода-вывода не выгружаются и обрабатываются на уровне прерываний.

Изменение приоритетов выгрузки прерванных нитей VMM

Инфраструктура изменения приоритетов выгрузки прерванных нитей VMM (VTIOLR) предлагает усовершенствования функции VTIOL.

Инфраструктура VTIOLR позволяет операционной системе AIX определить порядок обработки выгруженных операций ввода-вывода VMM с учетом конкретных критериев. Например, если небольшая операция `read()` добавляется в очередь после большой операции ввода-вывода из процесса `sync()` файловой системы, то операция `read()` может рассматриваться как активное задание, а процесс `sync()` - как фоновое задание. В этом случае функция VTIOLR может изменить приоритет операции `read()` и выполнить операцию ввода-вывода перед обработкой других буферов `iodone()`. После изменения приоритета буферы `iodone()` обрабатываются путем выгрузки в набор фоновых нитей. Таким образом, операция ввода-вывода выполняется фоновыми нитями, а не на уровне прерываний.

Производительность логических томов и дискового ввода-вывода

В этой главе основное внимание уделено производительности логических томов и локальных дисков.

Если вы не знакомы с принципами организации групп томов, логических и физических томов, а также логических и физических разделов в операционной системе, то ознакомьтесь с разделом Управление дисками - Обзор.

Выбор количества и типов жестких дисков, а также размера и места размещения пространств подкачки и логических томов на этих дисках является важной частью процедуры, выполняемой перед началом установки, и влияет на производительность системы. Различные рекомендации по планированию конфигурации дисков перед началом установки приведены в разделе Рекомендации по выбору конфигурации дисков перед началом установки.

Понятия, связанные с данным:

“Рекомендации перед установкой дисков” на стр. 98
По умолчанию алгоритм определения и расширения логических томов выбирает оптимальные значения параметров. Однако производительность работы подсистемы ввода-вывода будет выше, если при определении размера и размещении логических томов программа установки будет учитывать предполагаемый объем данных и требования рабочей схемы.

Отслеживание дискового ввода-вывода

Что следует принимать во внимание при сборе информации о дисковом вводе-выводе.

- Поиск наиболее интенсивно используемых файлов, файловых систем и логических томов:
  - Можно ли лучше расположить или распределить интенсивно используемые файловые системы по нескольким физическим накопителям? (**lslv**, **iostat**, **filemon**)
  - Расположены ли интенсивно используемые файлы в локальной или удаленной системе? (**filemon**)
  - Не связана ли основная доля дискового ввода-вывода с подкачкой? (**vmstat**, **filemon**)
  - Достаточен ли размер кэша для хранения всех обновляемых процессами страниц? (**vmstat**, **svmon**)
  - Какую долю дискового ввода-вывода составляют синхронные операции (без кэширования)?
- Определение фрагментации файлов:
  - Насколько фрагментированы интенсивно используемые файлы? (**fileplace**)
- Поиск наиболее часто используемого физического тома:
  - Не является ли применяемый накопитель или контроллер узким местом для производительности системы? (**iostat**, **filemon**)

Создание резервной конфигурации

Перед внесением серьезных изменений в конфигурацию дисков, а также перед настройкой параметров дисковой подсистемы рекомендуется сохранить информацию об исходном состоянии системы, включая данные о конфигурации и производительности.

Получение информации о времени ожидания ввода-вывода

В AIX версии 6.1 и выше изменен алгоритм вычисления процессорного времени, затраченного на ожидание дискового ввода-вывода (время wio).

Незанятый процессор считается выполняющим ввод-вывод, если существует операция ввода-вывода, запущенная на этом процессоре.

Кроме того, ожидание ввода-вывода в файл NFS рассматривается как время ожидания ввода-вывода.

Оценка производительности дисков командой iostat

Начните оценку с запуска команды **iostat** во время максимальной нагрузки на систему или работы важного приложения, задержки ввода-вывода для которого необходимо свести к минимуму.

Следующий сценарий запускает команду **iostat** в фоновом режиме на время копирования в большого файла:

```
# iostat 5 3 >io.out &
# cp big1 /dev/null
```

Операционная система AIX поддерживает ведение хронологии работы диска. Если хронология дисковых операций ввода-вывода выключена, при вводе команды **iostat** выдается следующее сообщение:

```
Отсутствует хронология работы диска с момента загрузки.

Это не влияет на интервальную статистику дисковых операций ввода-вывода.
```

Для включения хронологии дисковых операций ввода-вывода введите в командной строке `smit chgsys`, а затем выберите **true** в поле **Постоянное ведение хронологии ввода-вывода диска**.

Следующий пример сохраняет три отчета в файле `io.out`:

```
# iostat 5 3 >io.out &
# cp big /dev/null

System configuration: lcpu=4 drives=1 ent=0.50 paths=1 vdisks=1

tty:      tin         tout    avg-cpu: % user % sys % idle % iowait physc % entc
          0.0          0.0      0.1      0.6    99.2   0.1    0.0     1.2

Диски :       % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk0           0.4       5.6       0.6         28         0

tty:      tin         tout    avg-cpu: % user % sys % idle % iowait physc % entc
          0.0          0.0      0.1      1.4    98.4   0.2     0.0    2.4

Диски :       % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk0           6.0     123.2      10.6        4       612

tty:      tin         tout    avg-cpu: % user % sys % idle % iowait physc % entc
          2.6          2.6     0.1        0.5   99.4    0.0    0.0     1.1

Диски :       % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk0           0.0       0.0       0.0          0         0
```

Первый отчет представляет собой статистику с момента последней загрузки и показывает равномерность (или, в данном случае, неравномерность) распределения ввода-вывода по жестким дискам. Жесткий диск hdisk1 почти все время простаивал, в то время как жесткий диск hdisk2 обрабатывал около 63 процентов всего ввода-вывода (согласно значениям в столбцах `Kb_read` и `Kb_wrtn`).

Второй отчет содержит информацию, собранную за первые 5 секунд выполнения команды **cp**. С момента запуска команды **cp** прошло 2.6 секунд. В результате 2.5 секунды интенсивного ввода-вывода в сочетании с 2.5 секундами простоя дали среднее значение `% iowait`, равное 39.5 процентов. Более короткий интервал измерения позволил бы получить более точные значения. Этот пример показывает, что при анализе отчетов следует учитывать, что все указанные значения являются средними за интервал измерения.

Понятия, связанные с данным:

“Команда iostat” на стр. 107
Команда **iostat** позволяет быстро обнаружить неполадки дискового ввода-вывода, приводящие к снижению производительности.

Ссылки, связанные с данной:

“Непрерывное отслеживание производительности системы командой iostat” на стр. 14
Команда **iostat** позволяет получить информацию об использовании дисков и CPU.

Информация о терминалах:

Два столбца с информацией о терминалах (*tin* and *tout*) в выводе команды **iostat** показывают число символов, прочитанных и записанных всеми терминалами.

Это значение учитывает как физические, так и псевдотерминалы. Физическими терминалами считаются устройства, подключенные к асинхронному порту. Псевдотерминалы представляют собой оболочки, сеансы **telnet** и окна **aixterm**.

Поскольку чтение и запись символов занимает ресурсы процессора, обратите внимание на зависимость между активностью терминалов и загрузкой процессоров. Если такая зависимость есть, попытайтесь повысить эффективность работы подсистемы TTY. Возможные действия включают модификацию прикладных программ, изменение параметров порта терминала при передаче файлов, а также модернизацию асинхронного адаптера.

Сведения о процессоре:

В столбцах с информацией о процессоре (`% user`, `% sys`, `% idle` и `% iowait`) указывается распределение времени процессора.

В выводе команды **vmstat** эта информация выводится в столбцах `us`, `sy`, `id` и `wa`. Дополнительная информация приведена в разделе “Команда vmstat” на стр. 104. Кроме того, ознакомьтесь с информацией о новом способе вычисления значения `%iowait`, приведенной в разделе “Получение информации о времени ожидания ввода-вывода” на стр. 179.

Если в системе работает только одно приложение, то высокий процент времени ожидания ввода-вывода, скорее всего, связан с реальной нагрузкой. В системах с несколькими процессорами во время ожидания ввода-вывода одним из процессоров остальные будут работать. В этом случае величина `% iowait` будет невелика или равна нулю, поскольку работающие процессоры "скрывают" часть времени ожидания. Таким образом, несмотря на низкое значение `% iowait`, ввод-вывод может существенно влиять на производительность приложения.

Если команда **iostat** показывает, что производительность приложений не ограничена ресурсами процессоров, и величина `% iowait` больше 20 процентов, то низкая производительность связана с дисковым или другим вводом-выводом. Такая ситуация может возникнуть при массовой подкачке страниц из-за недостатка физической памяти. Кроме того, она может быть связана с несбалансированностью нагрузки на диски, фрагментацией данных или активных областей. Команда **iostat** выводит всю необходимую информацию для случая несбалансированной нагрузки. Информацию о файловых системах и логических томах, относящихся к логическим ресурсам, можно получить с помощью команд **filemon** и **fileplace**.

Отчет о дисках:

Отчет о дисках содержит информацию о быстродействии физических дисковых накопителей.

Если вы считаете, что низкая производительность связана с дисковым вводом-выводом, изучите отчет команды **iostat**. Для исключения информации о терминалах и процессорах укажите опцию **-d**. Кроме того, вы можете получить информацию только об отдельных дисках, указав их имена.

Не забывайте, что первый отчет содержит статистику работы системы со времени запуска.

Диски: Имена физических томов. Эти имена начинаются со строки `hdisk` или `cd`, за которой следует число. Если в параметрах команды **iostat** указаны физические тома, будут показаны только эти тома.

% tm_act
Доля времени в процентах, в течение которого физический диск был активен (интенсивность использования диска), или, другими словами, суммарное время выполнения запросов к диску. Устройство считается активным во время передачи данных и выполнения команд, таких как поиск на диске. Доля времени, в течение которого диск активен, прямо пропорциональна объему передаваемых данных и обратно пропорциональна его производительности. По мере увеличения интенсивности использования диска быстродействие снижается и время ответа увеличивается. Обычно, если диск занят больше 70 процентов времени, то время ожидания выполнения запроса к диску превышает собственно время выполнения запроса, и большая часть процессов UNIX, выполняющих ввод-вывод, блокируется. Обратите внимание на ситуации, когда одни устройства простаивают, в то время как другие перегружены. Равномерное распределение нагрузки по всем дискам позволит повысить производительность работы приложений. Обратите внимание, что страничный обмен также создает нагрузку на диски.

Kbps Средняя скорость передачи данных между системой и устройством (Кб/с). Это значение равно сумме `Kb_read` и `Kb_wrtn`, поделенной на длительность интервала измерения (в секундах).

tps Частота операций передачи данных между диском и системой, в операциях в секунду. Операция передачи - это запрос на ввод-вывод, переданный физическому диску через драйвер устройства. Несколько логических запросов при передаче устройству могут быть объединены в один. Объем данных, передаваемых за одну операцию, не ограничен.

Kb_read
Суммарный объем данных (в КБ), считанных с физического тома за интервал измерения.

Kb_wrtn
Суммарный объем данных (в КБ), записанных на физический том за интервал измерения.

Каждая из перечисленных выше переменных может принимать любые значения, зависящие от характеристик приложения, конфигурации системы, типов физических дисков и адаптеров. Поэтому при анализе данных нужно попытаться найти зависимости и взаимосвязи. Наиболее явно связаны нагрузка на диск (`%tm_act`) и скорость передачи данных (`tps`).

Для того чтобы сделать правильные выводы из отчетов команды iostat, необходимо знать способ доступа к данным (последовательный, произвольный или смешанный), применяемый приложением, а также типы дисковых накопителей и адаптеров в системе. Например, если приложение выполняет чтение или запись последовательно, то скорость передачи данных (`Kbps`) при высокой загрузке диска (`%tm_act`) также должна быть высокой. Значения `Kb_read` и `Kb_wrtn` показывают объем данных, считанных и записанных приложением. Однако, эти значения не позволяют определить способ доступа к данным.

Обычно высокая нагрузка на диск (`%tm_act`) не свидетельствует о неправильной работе, если она сопровождается высокой скоростью передачи (`Kbps`). Однако высокая нагрузка на диск при низкой скорости передачи свидетельствует о фрагментации логического тома, файловой системы или файла.

При рассмотрении вопросов, связанных с производительностью дисков, логических томов и файловых систем можно прийти к заключению о том, что чем больше дисков в системе, тем выше ее быстродействие. Однако, это не всегда так, поскольку все передаваемые между системой и диском данные проходят через дисковый адаптер. Этот адаптер также может стать узким местом. Если каждый дисковый накопитель подключен к отдельному адаптеру, и все интенсивно используемые файловые системы находятся на разных физических томах, то увеличение числа дисков действительно позволяет повысить производительность системы. При этом степень повышения производительности будет зависеть от применяемого способа доступа.

Для того чтобы узнать, насколько загружен адаптер, сложите скорости передачи данных (`Kbps`), указанные в отчете команды **iostat**, для всех дисков, подключенных к этому адаптеру. Сумма всех значений `Kbps` должна быть меньше максимальной скорости передачи данных, поддерживаемой адаптером. В большинстве случаев рекомендуется использовать 70 процентов пропускной способности адаптера. В операционной системе AIX эту информацию можно просмотреть с помощью опции **-a** или **-A**.

Оценка производительности дисков командой vmstat

Команда **iostat** позволяет проверить, действительно ли низкая производительность системы связана с неэффективным выполнением операций дискового ввода-вывода.

Проанализировав столбец *wa* команды **vmstat**, можно прийти к аналогичному выводу (более подробная информация приведена в разделе “Команда vmstat” на стр. 104. Ниже перечислены другие источники полезной информации о дисковом вводе-выводе:

- Столбец `disk xfer` в выводе команды **vmstat**

  Для просмотра статистической информации о логических дисках (до 4 дисков) введите следующую команду:

```
# vmstat hdisk0 hdisk1 1 8
kthr     memory             page              faults        cpu     disk xfer
----  ----------  ----------------------- ------------ ----------- ------
r  b   avm   fre  re  pi  po  fr  sr  cy  in   sy  cs  us sy id wa  1 2 3 4
0  0  3456 27743   0   0   0   0   0   0 131  149  28  0  1 99  0  0 0
0  0  3456 27743   0   0   0   0   0   0 131   77  30  0  1 99  0  0 0
1  0  3498 27152   0   0   0   0   0   0 153 1088  35  1 10 87  2  0 11
0  1  3499 26543   0   0   0   0   0   0 199 1530  38  1 19  0 80  0 59
0  1  3499 25406   0   0   0   0   0   0 187 2472  38  2 26  0 72  0 53
0  0  3456 24329   0   0   0   0   0   0 178 1301  37  2 12 20 66  0 42
0  0  3456 24329   0   0   0   0   0   0 124   58  19  0  0 99  0  0 0
0  0  3456 24329   0   0   0   0   0   0 123   58  23  0  0 99  0  0 0
```

В столбцах *disk xfer* указывается число операций передачи данных для указанных физических томов за интервал измерения. В отчете команды могут быть указаны имена от одного до четырех физических томов. Число операций выводится для каждого из заданных устройств в порядке их перечисления. Это значение отражает число запросов к физическому устройству. Оно не связано напрямую с объемом считанных или записанных данных. Несколько логических запросов могут объединяться в один физический запрос.

- Столбец *in* вывода команды **vmstat**

  В этом столбце показана частота аппаратных прерываний (единиц в секунду) за интервал измерения. Прерывания отправляются при возникновении таких событий, как завершение выполнения запроса диском или истечение очередных 10 миллисекунд, измеряемых таймером. Последнее прерывание отправляется 100 раз в секунду, поэтому значение в поле *in* всегда больше 100. Команда **vmstat** позволяет получить более подробную информацию о прерываниях.

- Вывод команды **vmstat -i**

  Команда с опцией *-i* позволяет узнать число прерываний, полученных каждым устройствам с момента запуска системы. Если в команде указан интервал сбора информации и, при необходимости, число отчетов, то статистика с момента запуска системы будет выведена только в первом разделе; в остальных разделах вывода будет приведена статистика за очередной интервал.

```
# vmstat -i 1 2
приоритет уровень тип    число модуль(обработчик)
    0       0   аппаратный   0 i_misc_pwr(a868c)
    0       1   аппаратный   0 i_scu(a8680)
    0       2   аппаратный   0 i_epow(954e0)
    0       2   аппаратный   0 /etc/drivers/ascsiddpin(189acd4)
    1       2   аппаратный 194 /etc/drivers/rsdd(1941354)
    3      10   аппаратный 10589024 /etc/drivers/mpsdd(1977a88)
    3      14   аппаратный 101947 /etc/drivers/ascsiddpin(189ab8c)
    5      62   аппаратный 61336129 clock(952c4)
   10      63   аппаратный 13769 i_softoff(9527c)
приоритет уровень тип    число модуль(обработчик)
    0       0   аппаратный   0 i_misc_pwr(a868c)
    0       1   аппаратный   0 i_scu(a8680)
    0       2   аппаратный   0 i_epow(954e0)
    0       2   аппаратный   0 /etc/drivers/ascsiddpin(189acd4)
    1       2   аппаратный   0 /etc/drivers/rsdd(1941354)
    3      10   аппаратный  25 /etc/drivers/mpsdd(1977a88)
    3      14   аппаратный   0 /etc/drivers/ascsiddpin(189ab8c)
    5      62   аппаратный 105 clock(952c4)
   10      63   аппаратный   0 i_softoff(9527c)
```

Примечание: Вывод этой команды зависит от системы, ее аппаратной и программной конфигурации (например, прерывания от таймера могут быть не показаны в выводе команды **vmstat -i**, хотя они и учитываются в столбце *in* вывода команды **vmstat**). Обратите внимание на большие значения в столбце *count*; постарайтесь выяснить причину, по которой соответствующему модулю пришлось отправить большое число прерываний.

Оценка производительности дисков командой sar

Команда **sar** - это стандартная утилита UNIX для получения статистических данных о системе.

В команде **sar** предусмотрено большое число опций, позволяющих получить информацию об очередях, подкачке, терминалах и других устройствах. Команда **sar -d** выдает статистику ввода-вывода в режиме реального времени:

```
# sar -d 3 3

AIX konark 3 4 0002506F4C00    08/26/99

12:09:50     device    %busy    avque    r+w/s   blks/s   avwait   avserv

12:09:53     hdisk0      1      0.0        0        5      0.0      0.0
             hdisk1      0      0.0        0        1      0.0      0.0
                cd0      0      0.0        0        0      0.0      0.0

12:09:56     hdisk0      0      0.0        0        0      0.0      0.0
             hdisk1      0      0.0        0        1      0.0      0.0
                cd0      0      0.0        0        0      0.0      0.0

12:09:59     hdisk0      1      0.0        1        4      0.0      0.0
             hdisk1      0      0.0        0        1      0.0      0.0
                cd0      0      0.0        0        0      0.0      0.0


Average      hdisk0      0      0.0        0        3      0.0      0.0
             hdisk1      0      0.0        0        1      0.0      0.0
                cd0      0      0.0        0        0      0.0      0.0
```

Ниже приведено описание полей вывода команды **sar -d**:

Загрузка (%busy)
Доля времени, в течение которого устройство было занято обработкой запросов на передачу данных. То же, что и столбец `%tm_act` в выводе команды **iostat**.

Среднее число ожидающих запросов (avque)
Среднее число запросов от адаптера, ожидающих выполнения к моменту измерения. Очередь драйвера устройств может содержать дополнительные операции ввода-вывода. Большое значение в этом поле говорит о том, что с большой долей вероятности дисковый ввод-вывод является узким местом в системе.

Частота операций передачи данных (r+w/s)
Число операций чтения и записи, выполняемых устройством в секунду. То же, что и столбец `tps` в выводе команды **iostat**.

Скорость передачи данных в блоках (blks/s)
Число блоков по 512 байт, передаваемых в секунду.

Среднее число ожидающих транзакций (avwait)
Среднее число транзакций, ожидающих выполнения (длина очереди). Среднее время (в миллисекундах) нахождения запросов на передачу данных в очереди устройства. В настоящее время этот параметр не измеряется и всегда равен 0.0.

Среднее время поиска (avserv)
Среднее время поиска в миллисекундах. Среднее время, затрачиваемое устройством на обслуживание запроса на передачу данных (включая поиск, переход к нужному сектору и собственно передачу данных). В настоящее время этот параметр не измеряется и всегда равен 0.0.

Получение информации о фрагментации логических томов с помощью команды lslv

Информацию о фрагментации логических томов можно получить с помощью команды **lslv**.

Введите команду **lslv -l** *имя-логического-тома*, как показано ниже:

```
# lslv -l hd2
hd2:/usr
PV               COPIES        IN BAND       DISTRIBUTION
hdisk0           114:000:000   22%           000:042:026:000:046
```

В столбце COPIES указано, что существует только одна копия логического тома hd2. Столбец IN BAND показывает, насколько выполняется стратегия размещения логического тома. Чем больше эта величина, тем эффективнее размещение данных. С каждым логическим томом связана своя стратегия размещения. Если операционная система не может полностью реализовать стратегию размещения, она пытается выполнить максимальное число требований. В данном примере существует всего 114 логических разделов (LP); 42 LP находятся во внешней области, 26 LP - в средней области и 46 LP - на внутреннем крае. Поскольку с данным логическим томом связана стратегия размещения в средней области, параметр in-band равен 22 процентам (26 / (42+26+46). В столбце DISTRIBUTION показано распределение физических разделов по диску в следующем формате:

```
внешний край : внешняя область : средняя область : внутренняя область : внутренний край
```

Дополнительная информация о распределении физических разделов приведена в разделе “Расположение данных на физическом томе” на стр. 202.

Получение информации о физическом размещении данных с помощью команды lslv

Если скорость работы приложений существенно зависит от скорости ввода-вывода, исследуйте физическое расположение файлов на диске и определите, нужна ли реорганизация на каком-либо уровне.

Для того чтобы получить информацию о расположении разделов логического тома hd11 на физическом томе hdisk0, введите следующую команду:

```
# lslv -p hdisk0 hd11
hdisk0:hd11:/home/op
USED  USED  USED  USED  USED  USED  USED  USED  USED  USED     1-10
USED  USED  USED  USED  USED  USED  USED                      11-17

USED  USED  USED  USED  USED  USED  USED  USED  USED  USED   18-27
USED  USED  USED  USED  USED  USED  USED                     28-34

USED  USED  USED  USED  USED  USED  USED  USED  USED  USED   35-44
USED  USED  USED  USED  USED  USED                         45-50

USED  USED  USED  USED  USED  USED  USED  USED  USED  USED    51-60
0052  0053  0054  0055  0056  0057  0058                      61-67

0059  0060  0061  0062  0063  0064  0065  0066  0067  0068    68-77
0069  0070  0071  0072  0073  0074  0075                      78-84
```

Для просмотра информации об остальной части hd11 на диске hdisk1 введите:

```
# lslv -p hdisk1 hd11
hdisk1:hd11:/home/op
0035  0036  0037  0038  0039  0040  0041  0042  0043  0044    1-10
0045  0046  0047  0048  0049  0050  0051                     11-17

USED  USED  USED  USED  USED  USED  USED  USED  USED  USED   18-27
USED  USED  USED  USED  USED  USED  USED                     28-34

USED  USED  USED  USED  USED  USED  USED  USED  USED  USED   35-44
USED  USED  USED  USED  USED  USED                         45-50

0001  0002  0003  0004  0005  0006  0007  0008  0009  0010   51-60
0011  0012  0013  0014  0015  0016  0017                     61-67

0018  0019  0020  0021  0022  0023  0024  0025  0026  0027   68-77
0028  0029  0030  0031  0032  0033  0034                     78-84
```

Пять показанных разделов представляют соответственно внешний край, внешнюю область, среднюю область, внутреннюю область и внутренний край.

- Если указано значение USED, то данный физический раздел занят логическим томом, отличным от указанного. Число задает номер логического раздела того тома, который указан в команде **lslv -p**.
- Если указано значение FREE, то физический раздел не занят никаким логическим томом. Если логические разделы не расположены на диске последовательно друг за другом, то логический том фрагментирован.
- STALE - это физический раздел, данные которого вы не можете использовать. Информацию о физических разделах типа STALE можно просмотреть с помощью команды **lspv -m**. Для того чтобы физические разделы STALE содержали ту же информацию, что и доступные физические разделы, их необходимо обновить. Этот процесс, называемый синхронизацией, может быть запущен при включении системы или во время ее работы с помощью команды **syncvg**. До записи текущей версии данных в разделы STALE запросы на чтение и запись в эти разделы не выполняются.

В предыдущем примере логический том hd11 фрагментирован на физическом томе hdisk1, поскольку первые логические разделы находятся во внутренних областях тома hdisk1, в то время как логические разделы 35-51 расположены во внешней области. Неупорядоченный доступ к логическому тому hd11 будет выполняться медленно из-за большого времени поиска. Приведенные выше отчеты также свидетельствуют о том, что на физических томах hdisk0 и hdisk1 отсутствуют свободные физические разделы.

Доступ к расположению файлов командой fileplace

Для того чтобы получить информацию о размещении на диске ранее скопированного файла big1, можно воспользоваться командой **fileplace**. Команда **fileplace** показывает расположение блоков файла на логическом томе, либо на одном или нескольких физических томах.

Для того чтобы узнать, установлена ли команда **fileplace**, введите следующую команду:

```
# lslpp -lI perfagent.tools
```

Затем:

```
# fileplace -pv big1

Файл: big1  Размер: 3554273 байт  Том: /dev/hd10
Размер блока: 4096  Размер фрагмента: 4096  Число фрагментов: 868  Сжатие: нет
I-узел: 19  Режим: -rwxr-xr-x  Владелец: hoetzel  Группа: system

  Физический адрес (зеркальная копия 1)                       Логические фрагменты
  ----------------------------------                          ----------------
  0001584-0001591  hdisk0     8 фраг.    32768 байт,   0.9%     0001040-0001047
  0001624-0001671  hdisk0    48 фраг.   196608 байт,   5.5%     0001080-0001127
  0001728-0002539  hdisk0   812 фраг.  3325952 байт,  93.5%     0001184-0001995

  868 фрагментов занимают объем 956:  эффективность = 90.8%
  3 фрагмента из 868 возможных:  компактность размещения = 99.8%
```

Такой отчет команды говорит о том, что файл фрагментирован очень слабо, поскольку содержит несколько небольших разрывов. Таким образом, расположение файла big1 практически не влияет на время его последовательного чтения. Более того, поскольку создание файла размером 3.5 МБ не привело к его существенной фрагментации, мы можем считать, что соответствующая файловая система не фрагментирована.

Некоторые части файла могут быть не связаны с блоками тома. При чтении эти области заполняются файловой системой нулями. Такие области называются неразмещенными логическими блоками. Номинальный размер файла с такими областями будет больше фактического размера, занимаемого им на диске (команда **ls -l** показывает номинальный размер файла, а команда **du** - фактическое число занятых блоков).

Команда **fileplace** считывает список блоков файла с логического тома. Если файл новый, эта информация может быть еще не записана на диск. В этом случае для записи этой информации на диск введите команду **sync**. Кроме того, команда **fileplace** не показывает файлы NFS, расположенные на удаленном сервере (для этого необходимо запустить эту команду на сервере).

Примечание: Если при создании файла его записи были в произвольном порядке распределены по диску, то в выводе команды **fileplace** будут показаны только те страницы, которые содержат записи файла. Файловая система не заполняет автоматически промежуточные страницы при создании файла. Однако при последовательном чтении такого файла (например, командой **cp** или **tar**) пространство между записями заполняется двоичными нулями. В результате файл, скопированный командой **cp**, займет больше места, чем исходный, хотя будет содержать те же данные.

Эффективность использования пространства и компактность размещения:

Большая эффективность использования пространства означает, что фрагментация файлов невелика, а последовательный доступ к ним выполняется быстро. Высокая компактность размещения означает, что при размещении файлы разбиваются на небольшое число фрагментов, что также повышает скорость последовательного доступа.

Эффективность использования пространства =
Общее число фрагментов, занятых для хранения файла / Физический адрес последнего фрагмента - Физический адрес первого фрагмента + 1)

Компактность размещения =
(Общее число фрагментов - Число групп фрагментов + 1) / Общее число фрагментов

Если компактность размещения или эффективность использования пространства уменьшается, рекомендуется реорганизовать логический том с помощью команды **reorgvg** (за дополнительной информацией обратитесь к разделу “Реорганизация логических томов” на стр. 208). Информация о реорганизации файловых систем приведена в разделе “Реорганизация файловой системы” на стр. 244.

В этом примере Физический адрес последнего фрагмента - Физический адрес первого фрагмента + 1 = 0002539 - 0001584 + 1 = 956 фрагментов; всего занято 8 + 48 + 812 = 868 фрагментов; эффективность использования пространства равна 868 / 956 (90.8 процентов); компактность размещения равна (868 - 3+ 1) / 868 = 99.8 процентов.

Поскольку общее число фрагментов файла не учитывает дополнительные блоки, ссылки на которые хранятся в основных блоках, в то время как физические адреса их учитывают, эффективность использования дискового пространства никогда не достигает 100 процентов для файла больше 32 КБ, даже если он расположен в последовательных фрагментах.

Оценка общего объема ввода-вывода пространства подкачки с помощью команды vmstat

Команда **vmstat** позволяет узнать, какая доля операций ввода-вывода связана со страничным обменом.

Операции с пространством подкачки обычно не упорядочены и оперируют с одной страницей каждая. Два приведенных ниже примера иллюстрируют интенсивность страничного обмена при компиляции программы, написанной на языке C, в системе, объем памяти которой был искусственно уменьшен с помощью команды **rmss**. В столбцах *pi* и *po* (число загруженных и выгруженных страниц по 4096 байт) указано количество операций подкачки и выгрузки страниц, выполненных за 5-секундный интервал. Первый отчет (информация с момента последнего запуска системы) был удален. Обратите внимание, что работа с областями подкачки характеризуется всплесками активности.

```
# vmstat 5 8
нити     память              страница          ошибки        cpu
----- ----------- ------------------------ ------------ -----------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
 0  1 72379   434   0   0   0   0    2   0 376  192 478  9  3 87  1
 0  1 72379   391   0   8   0   0    0   0 631 2967 775 10  1 83  6
```

```
 0  1 72379   391   0   0   0   0    0   0 625 2672 790  5  3 92  0
 0  1 72379   175   0   7   0   0    0   0 721 3215 868  8  4 72 16
 2  1 71384   877   0  12  13  44  150   0 662 3049 853  7 12 40 41
 0  2 71929   127   0  35  30 182  666   0 709 2838 977 15 13  0 71
 0  1 71938   122   0   0   8  32  122   0 608 3332 787 10  4 75 11
 0  1 71938   122   0   0   0   3   12   0 611 2834 733  5  3 75 17
```

Приведенные ниже отчеты команды **vmstat -s**, созданные до и после компиляции, показывают общий объем подкачки. Помните, что об истинной интенсивности подкачки свидетельствует число загруженных и выгруженных страниц пространства подкачки. Когда говорят о числе переданных страниц вообще, имеют ввиду сумму ввода-вывода подкачки и обычного файлового ввода-вывода, выполняемого через механизм страничного обмена. Из отчета были удалены строки, не относящиеся к содержимому данного раздела.

# vmstat -s # перед	**# vmstat -s # после**
6602 page ins 3948 page outs **544 paging space page ins 1923 paging space page outs** 0 total reclaims	7022 page ins 4146 page outs **689 paging space page ins 2032 paging space page outs** 0 total reclaims

Тот факт, что при компиляции выполнялось больше операций записи в пространство подкачки, чем операций чтения, говорит о том, что система была перегружена. Некоторые страницы повторно оказывались в пространстве подкачки, поскольку были выгружены до завершения работы с ними.

Оценка общего объема дискового ввода-вывода с помощью команды vmstat

Описанный выше способ может применяться для оценки объема дискового ввода-вывода, создаваемого определенной программой.

Если система не занята выполнением других задач, то команды

```
# vmstat -s >statout
# testpgm
# sync
# vmstat -s >> statout
# egrep "ins|outs" statout
```

позволят узнать общее число обращений к диску до и после выполнения программы, например:

```
     5698 страниц прочитано
     5012 страниц записано
        0 страниц прочитано из области подкачки
       32 страниц записано в область подкачки
     6671 страниц прочитано
     5268 страниц записано
        8 страниц прочитано из области подкачки
      225 страниц записано в область подкачки
```

За время работы этой команды (компиляция большого проекта на C) система прочитала 981 страниц (8 из области подкачки) и записала 449 страницы (193 в область подкачки).

Подробный анализ ввода-вывода командой filemon

Команда **filemon** применяет программу трассировки для получения подробной информации об интенсивности ввода-вывода на различных уровнях файловой системы, включая логическую файловую систему, сегменты виртуальной памяти (VM), Администратор логических томов (LVM) и физические диски.

Данные собираются командой **filemon** для тех уровней, которые указаны в опции **-O**, либо для всех уровней. По умолчанию данные собираются для уровней VM, LVM и физических дисков. Могут создаваться краткие или подробные отчеты. Поскольку команда **filemon** вызывает программу трассировки, она может запускаться только пользователем root или членами группы system.

Для того чтобы определить, установлена ли команда **filemon**, введите следующую команду:

```
# lslpp -lI perfagent.tools
```

Трассировка запускается командой **filemon**, приостанавливается подкомандой **trcoff**, возобновляется подкомандой **trcon**. Команда **filemon** записывает отчет в стандартный вывод сразу после завершения трассировки.

Примечание: Если не указан флаг **-u**, то данные будут собраны только о тех файлах, которые были открыты после запуска команды **filemon**.

Вместо выполнения трассировки команда **filemon** может считывать данные трассировки ввода-вывода из указанного файла. В этом случае отчет команды **filemon** будет содержать информацию об операциях ввода-вывода в системе, зафиксированных в файле. Автономная обработка файла трассировки может выполняться в другой системе. Кроме того, она позволяет разделить сбор данных и их обработку.

Вначале файл трассировки должен быть обработан командой **trcrpt -r**. Вывод команды должен быть перенаправлен в другой файл:

```
# gennames > gennames.out
# trcrpt -r   trace.out > trace.rpt
```

Преобразованный файл трассировки передается команде **filemon** для создания отчета об операциях ввода-вывода, зарегистрированных в исходном файле трассировки:

```
# filemon -i trace.rpt -n gennames.out | pg
```

В этом примере команда **filemon** считывает события трассировки файловой системы из файла `trace.rpt`. Поскольку данные трассировки уже находятся в файле, команда **filemon** не переходит в фоновый режим. После чтения всего файла создаются отчеты об уровнях виртуальной памяти, логических и физических томах, которые передаются в стандартный вывод (в данном случае - команде **pg**).

Если команда **trace** была запущена с флагом **-C all**, команду **trcrpt** также следует запустить с флагом **-C all** (см. “Создание отчета на основе вывода команды trace -C” на стр. 398).

Ниже приведен пример применения команды **filemon**:

```
# filemon -o fm.out -O all; cp /smit.log /dev/null ; trcstop
```

Отчет, полученный командой в системе, не выполнявшей других заданий, выглядел следующим образом:

```
вторник 19 августа 1999 г. 11.30.49
Система: AIX texmex  Узел: 4  Компьютер: 000691854C00

Интервал измерения: 0.369 секунды
Использование CPU:  9.0%

Наиболее активные файлы
------------------------------------------------------------------------
  #MBs  #opns   #rds   #wrs  файл                 том:i-узел
------------------------------------------------------------------------
   0.1      1     14      0  smit.log              /dev/hd4:858
   0.0      1      0     13  null
   0.0      2      4      0  ksh.cat               /dev/hd2:16872
   0.0      1      2      0  cmdtrace.cat          /dev/hd2:16739

Наиболее активные сегменты
------------------------------------------------------------------------
  #MBs  #rpgs  #wpgs   ИД    тип                   том:i-узел
------------------------------------------------------------------------
   0.1     13      0   5e93  ???
   0.0      2      0   22ed  ???
   0.0      1      0   5c77  постоянный

Наиболее активные логические тома
------------------------------------------------------------------------
  util  #rblk  #wblk   Кб/с   том                  описание
------------------------------------------------------------------------
```

```
  0.06    112      0  151.9  /dev/hd4            /
  0.04     16      0   21.7  /dev/hd2            /usr

Наиболее активные физические тома
------------------------------------------------------------------------
  util  #rblk  #wblk   Кб/с   том                описание
------------------------------------------------------------------------
  0.10    128      0  173.6  /dev/hdisk0         НД



------------------------------------------------------------------------
Подробная статистика работы с файлами
------------------------------------------------------------------------

Файл: /smit.log  том: /dev/hd4 (/)  I-узел: 858
операций открытия:      1
всего передано байт:    57344
операций чтения:        14      (0 ошибок)
  прочитано (байт):     ср   4096.0 мин    4096 макс   4096 откл     0.0
  время чтения (мс):    ср    1.709 мин   0.002 макс 19.996 откл   5.092

Файл: /dev/null
операций открытия:      1
всего передано байт:    50600
операций записи:        13      (0 ошибок)
  записано  (байт):     ср   3892.3 мин    1448 макс   4096 откл   705.6
  время записи (мс):    ср    0.007 мин   0.003 макс  0.022 откл   0.006

Файл: /usr/lib/nls/msg/en_US/ksh.cat  том: /dev/hd2 (/usr)  I-узел: 16872
операций открытия:      2
всего передано байт:    16384
операций чтения:        4       (0 ошибок)
  прочитано (байт):     ср   4096.0 мин    4096 макс   4096 откл     0.0
  время чтения (мс):    ср    0.042 мин   0.015 макс  0.070 откл   0.025
операций поиска:        10

Файл: /usr/lib/nls/msg/en_US/cmdtrace.cat  том: /dev/hd2 (/usr)  I-узел: 16739
операций открытия:      1
всего передано байт:    8192
операций чтения:        2       (0 ошибок)
  прочитано (байт):     ср   4096.0 мин    4096 макс   4096 откл     0.0
  время чтения (мс):    ср    0.062 мин   0.049 макс  0.075 откл   0.013
операций поиска:        8

------------------------------------------------------------------------
Подробная статистика работы с виртуальной памятью (в страницах по 4096 байт)
------------------------------------------------------------------------

Сегмент: 5e93  тип: ???
флаги сегмента
операций чтения:        13       (0 ошибок)
  время чтения (мс):    ср    1.979 мин   0.957 макс  5.970 откл   1.310
  последовательностей:  1
  длина последоват.:    ср     13.0 мин      13 макс     13 откл     0.0

Сегмент: 22ed  тип: ???
флаги сегмента          I-узел
операций чтения:        2       (0 ошибок)
  время чтения (мс):    ср    8.102 мин   7.786 макс  8.418 откл   0.316
  последовательностей:  2
  длина последоват.:    ср      1.0 мин       1 макс      1 откл     0.0

Сегмент: 5c77  тип: постоянный
флаги сегмента:         pers defer
операций чтения:        1       (0 ошибок)
```

```
  время чтения (мс):    сред 13.810 мин  13.810 макс 13.810 откл  0.000
  последовательностей:  1
  длина последоват.:    ср      1.0 мин       1 макс      1 откл     0.0

------------------------------------------------------------------------
Подробная статистика работы с логическими томами (в блоках по 512 байт)
------------------------------------------------------------------------

Том: /dev/hd4  описание: /
операций чтения:        5       (0 ошибок)
  прочитано (блоков):   ср     22.4 мин       8 макс     40 откл    12.8
  время чтения (мс):    ср    4.847 мин   0.938 макс 13.792 откл   4.819
  последовательностей:  3
  длина последоват.:    ср     37.3 мин       8 макс     64 откл    22.9
операций поиска:        3       (60.0%)
  размер поиска (блоков):нач    6344,
                        ср     40.0 мин       8 макс     72 откл    32.0
период запросов (мс):   ср   70.473 мин   0.224 макс331.020 откл 130.364
поток данных:           151.9 Кб/с
загрузка:               0.06

Том: /dev/hd2  описание: /usr
операций чтения:        2       (0 ошибок)
  прочитано (блоков):   ср      8.0 мин       8 макс      8 откл     0.0
  время чтения (мс):    ср    8.078 мин   7.769 макс  8.387 откл   0.309
  последовательностей:  2
  длина последоват.:    ср      8.0 мин       8 макс     8 откл    0.0
операций поиска:        2       (100.0%)
  размер поиска (блоков):нач  608672,
                        ср     16.0 мин      16 макс     16 откл     0.0
период запросов (мс):   ср  162.160 мин   8.497 макс331.020 откл 153.663
поток данных:           21.7 Кб/с
загрузка:               0.04

------------------------------------------------------------------------
Подробная статистика работы с физическими томами (в блоках по 512 байт)
------------------------------------------------------------------------

Том: /dev/hdisk0  описание: N/A
операций чтения:        7       (0 ошибок)
  прочитано (блоков):   ср     18.3 мин       8 макс     40 откл    12.6
  время чтения (мс):    ср    5.723 мин   0.905 макс 20.448 откл   6.567
  последовательностей:  5
  длина последоват.:    ср     25.6 мин       8 макс     64 откл    22.9
операций поиска:        5       (71.4%)
  размер поиска (блоков):нач  4233888,
                        ср  171086.0 мин       8 макс 684248 откл 296274.2
  размер поиска (%):     нач  48.03665,
                        ср  1.94110 мин 0.00009 макс7.76331 откл 3.36145
период запросов (мс):   ср   50.340 мин   0.226 макс331.020 откл 108.483
поток данных:           173.6 Кб/с
загрузка:           0.10
```

Вызов команды **filemon** в реальных системах приведет к созданию отчетов намного большего объема и может потребовать увеличения размера буфера трассировки. Команда **filemon** требует большого количества ресурсов памяти и процессора, что может негативно сказаться на производительности системы. Перед вызовом **filemon** в рабочей системе необходимо проверить ее в тестовой среде. Кроме того, рекомендуется выполнять автономную обработку данных трассировки, а в многопроцессорных системах - указывать в команде **trace** флаг **-C all**.

Примечание: Хотя команда **filemon** вычисляет минимальное, максимальное и среднее значение, а также среднеквадратичное отклонение, результаты работы команды не следует использовать для построения доверительных интервалов и других статистических оценок. В общем случае распределение значений не является ни случайным, ни симметричным.

Общие отчеты команды filemon:

В общих отчетах перечислены файлы, сегменты, логические и физические тома, наиболее интенсивно использовавшиеся за интервал измерения.

Эта информация показана в начале вывода команды **filemon**. По умолчанию отчеты о логических файлах и виртуальной памяти ограничены 20 файлами и сегментами, наиболее активными с точки зрения количества переданных данных. Если указан флаг **-v**, выводится информация обо всех файлах и сегментах. Строки отчета расположены в порядке уменьшения значений показателей.

Наиболее активные файлы:

Команда **filemon** создаёт отчёты со списком наиболее интенсивно используемых файлов на различных уровнях файловой системы, включая логическую файловую систему, сегменты виртуальной памяти (VM), Администратор логических томов (LVM) и физические диски. В этом разделе описаны заголовки столбцов отчёта.

#MBs Суммарный объем данных в мегабайтах, переданных для файла за интервал измерения. Строки упорядочены по убыванию этого значения.

#opns Число операций открытия файла за интервал измерения.

#rds Число операций чтения файла.

#wrs Число операций записи в файл.

файл Имя файла (полное имя файла приводится в подробном отчете).

том:I-узел
Логический том, на котором находится файл, и номер I-узла файла в связанной с ним файловой системе. С помощью этого значения можно связать файл с соответствующим постоянным сегментом, показанным в подробном отчете о сегментах VM. Для временных файлов, созданных и удаленных за время выполнения, это поле может быть пустым.

Наиболее активные файлы - это `smit.log` на логическом томе hd4 и устройство **null**. Для управления экраном приложение использует базу данных terminfo, поэтому файлы `ksh.cat` и `cmdtrace.cat` также постоянно используются. При каждом выводе сообщения на экран оболочка обращается к этим каталогам для получения данных.

Для идентификации неизвестных файлов преобразуйте имя логического тома, `/dev/hd1`, в точку монтирования файловой системы, `/home` и найдите файл с помощью команды **find** или **ncheck**:

```
# find / -inum 858 -print
/smit.log
```

или

```
# ncheck -i 858 /
/:
858     /smit.log
```

Наиболее активные сегменты:

Команда **filemon** создаёт отчёты со списком наиболее активных сегментов на различных уровнях файловой системы, включая логическую файловую систему, сегменты виртуальной памяти (VM), Администратор логических томов (LVM) и физические диски. В этом разделе описаны заголовки столбцов отчёта.

#MBs Суммарный объем данных в мегабайтах, переданных для сегмента за интервал измерения. Строки упорядочены по убыванию этого значения.

#rpgs Число страниц по 4 КБ, считанных в сегмент с диска (загрузка страниц).

#wpgs Число страниц по 4 КБ, записанных из сегмента на диск (выгрузка страниц).

#segid Идентификатор сегмента памяти VMM.

segtype
Тип сегмента: рабочий сегмент, постоянный сегмент (локальный файл), клиентский сегмент (удаленный файл), сегмент таблицы страниц, системный сегмент или специальный постоянный сегмент, содержащий данные файловой системы (протокол, корневой каталог, .inode, .inodemap, .inodex, .inodexmap, .indirect или .diskmap).

volume:inode
Для постоянных сегментов - имя логического тома, содержащего связанный с ним файл, и номер I-узла этого файла. С помощью этого значения можно связать постоянный сегмент с файлом, указанным в подробном отчете со статистикой файлов. Для непостоянных сегментов это значение не указывается.

Во время работы команды с помощью инструмента анализа виртуальной памяти **svmon** можно получить дополнительную информацию о сегменте по его идентификатору (segid): **svmon -D** *ИД-сегмента*. Более подробная информация приведена в разделе Команда svmon.

Строка `???` в этом примере означает, что система не может определить тип сегмента. Дополнительную информацию можно получить с помощью команды **svmon**.

Наиболее активные логические разделы:

Команда **filemon** создаёт отчёты со списком наиболее активно используемых логических разделах на различных уровнях файловой системы, включая логическую файловую систему, сегменты виртуальной памяти (VM), Администратор логических томов (LVM) и физические диски. В этом разделе описаны заголовки столбцов отчёта.

util Интенсивность использования логического тома.

#rblk Число блоков по 512 байт, считанных с логического тома.

#wblk Число блоков по 512 байт, записанных на логический том.

КБ/с Средняя скорость передачи данных в КБ/с.

том Имя логического тома.

описание
Точка монтирования файловой системы или тип логического тома (пространство подкачки, журнал jfs, загрузочный том или дамп системы). Например, для логического тома `/dev/hd2` указана точка монтирования `/usr`; `/dev/hd6` - это пространство подкачки, а `/dev/hd8` - журнал JFS. Кроме того, в этом поле может быть указано слово *compressed*. Это означает, что все данные автоматически сжимаются с помощью алгоритма Лемпеля-Зива (Lempel-Zev) перед записью на диск и автоматически распаковуются при чтении с него (дополнительная информация приведена в разделе “Сжатие JFS” на стр. 244).

Интенсивность использования указывается в долях от максимума: 0,06 означает, что в течение интервала измерения том был загружен на 6 процентов.

Наиболее активные физические разделы:

Команда **filemon** создаёт отчёты со списком наиболее активно используемых физических разделах на различных уровнях файловой системы, включая логическую файловую систему, сегменты виртуальной памяти (VM), Администратор логических томов (LVM) и физические диски. В этом разделе описаны заголовки столбцов отчёта.

util Интенсивность использования физического тома.

Примечание: Запросы ввода-вывода для логического тома начинают обрабатываться раньше и завершают обрабатываться позже, чем для физического тома. В связи с этим показатель использования логического тома будет больше, чем аналогичный показатель физического тома.

#rblk Число блоков по 512 байт, считанных с физического тома.

#wblk Число блоков по 512 байт, записанных на физический том.

КБ/с Средняя скорость передачи данных в КБ/с.

том Имя физического тома

описание
Описание типа физического тома, например Мультимедийный дисковод CD-ROM с интерфейсом SCSI или Дисковый накопитель с 16-разрядным интерфейсом SCSI.

Интенсивность использования тома измеряется в процентах от максимального значения: 0,10 означает, что в течение интервала измерения том был загружен на 10 процентов.

Наиболее активные файлы, упорядоченные по процессу:

Команда **filemon** создает отчеты, упорядоченные по процессам, со списком наиболее интенсивно используемых файлов на различных уровнях файловой системы, включая логическую файловую систему, сегменты виртуальной памяти (VM), Администратор логических томов (LVM) и физические диски.

#MBS Общий объем данных в мегабайтах, переданных через файл. Строки упорядочены по убыванию этого значения.

#opns Число операций открытия файла за интервал сбора данных.

#rds Число системных вызовов чтения, выполненных для файла.

#wrs Число системных вызовов записи, выполненных для файла.

файл Имя файла; полное имя файла указано в подробном отчете.

PID ИД процесса, открывшего файл.

Процесс
ИД процесса, открывшего файл.

TID ИД нити, открывшей файл.

Наиболее активные файлы, упорядоченные по нити:

Команда **filemon** создает отчеты, упорядоченные по нитям, со списком наиболее интенсивно используемых файлов на различных уровнях файловой системы, включая логическую файловую систему, сегменты виртуальной памяти (VM), Администратор логических томов (LVM) и физические диски.

#MBS Общий объем данных в мегабайтах, переданных через файл. Строки упорядочены по убыванию этого значения.

#opns Число операций открытия файла за интервал сбора данных.

#rds Число системных вызовов чтения, выполненных для файла.

#wrs Число системных вызовов записи, выполненных для файла.

файл Имя файла; полное имя файла указано в подробном отчете.

PID ИД процесса, открывшего файл.

Процесс
ИД процесса, открывшего файл.

TID ИД нити, открывшей файл.

Подробные отчеты команды filemon:

Подробные отчеты содержат дополнительную информацию об объектах, статистика для которых приведена в общих отчетах.

Подробные отчеты содержат по одной записи для каждого файла, сегмента или тома. Поля записей всех четырех подробных отчетов описаны ниже. Некоторые поля описывают одно значение; другие содержат информацию о распределении целой серии значений. Например, для всех отслеживаемых запросов чтения и записи сохраняется время ответа. На основании полученных значений вычисляется среднее значение, минимум, максимум и стандартное отклонение. Стандартное отклонение показывает, как сильно отдельное время ответа отклоняется от среднего значения. Примерно две трети значений времени ответа отличаются от среднего не больше, чем на стандартное отклонение (`ср - откл` или `ср + откл`). Если бы время ответа варьировалось сильнее, то стандартное отклонение от среднего значения было бы больше.

Подробная статистика работы с файлами:

Подробная статистика выводится для каждого файла, упомянутого в отчете *Наиболее активные файлы*.

С помощью раздела *Наиболее активные файлы* можно определить применявшийся способ доступа к файлу. Кроме суммарного числа переданных байт, операций открытия, чтения, записи и поиска, выводится объем считываемых и записываемых данных, а также время выполнения этих операций.

Файл Имя файла. В большинстве случаев указывается полное имя.

том Имя логического тома или файловой системы, в которой хранится файл.

I-узел Номер I-узла файла в файловой системе.

операций открытия
Число операций открытия файла за интервал сбора данных.

всего передано байт
Общее число байт, считанных из файла или записанных в файл.

чтение Число операций чтения данных из файла.

прочитано (байт)
Объем прочитанных данных в байтах (`ср/мин/макс/откл`).

время чтения (мс)
Время выполнения операций чтения в мс (`ср/мин/макс/откл`).

запись Число операций записи в файл.

записано (байт)
Объем записанных данных.

время записи (мс)
Время выполнения операций записи.

операций поиска
Число вызовов функции **lseek()**.

Объем считанных и записанных данных позволяет определить эффективность чтения и записи информации приложением. Наиболее оптимальным является значение, кратное размеру страницы (4 КБ).

Подробная статистика сегментом виртуальной машины:

В списке `Наиболее активные сегменты` показана подробная статистика для всех сегментов виртуальной машины.

Подробная статистика выводится для каждого сегмента, упомянутого в отчете Наиболее активные сегменты.

Сегмент
Внутренний идентификатор сегмента в операционной системе.

тип сегмента
Тип содержимого сегмента.

флаги сегмента
Различные атрибуты сегмента.

том Для постоянных сегментов - имя логического тома, в котором находится соответствующий файл.

I-узел Для постоянных сегментов - номер I-узла соответствующего файла.

чтение Число страниц по 4096 байт, считанных в сегмент с диска (загрузка страницы).

время чтения (мс)
Время выполнения операций чтения в мс (ср/мин/макс/откл).

последовательностей
Число последовательностей чтения. Последовательность - это несколько страниц, считанных подряд. Число последовательностей чтения характеризует упорядоченность доступа к файлу.

длина последовательности
Информация о числе страниц в последовательности.

запись Число страниц, записанных на диск из сегмента (выгрузка страницы).

время записи (мс)
Время выполнения операций записи.

последовательностей
Число последовательностей записи. Последовательность - это несколько страниц, записанных подряд.

длина последовательности
Параметры распределения числа страниц в последовательности.

Сравнивая число операций и последовательностей чтения, можно узнать применявшийся способ доступа к файлу (последовательный или неупорядоченный). Например, если число последовательностей близко к числу операций чтения, доступ к файлу можно считать неупорядоченным. С другой стороны, если число последовательностей много меньше числа операций чтения, и их средняя длина велика, доступ к файлу можно считать последовательным. То же самое относится и к операциям записи.

Подробная статистика физического или логического тома:

Подробная статистика выводится для каждого тома, упомянутого в отчете Наиболее активные логические тома или Наиболее активные физические тома.

Кроме числа операций чтения и записи выводятся параметры распределения времени выполнения этих операций, начальная и средняя длина поиска.

ТОМ Имя тома.

описание
Описание тома. (Информация о содержимом логического тома или типе физического тома.)

чтение Число операций чтения, выполненных над томом.

прочитано (блоков)
Объем прочитанных данных в блоках (ср/мин/макс/откл).

время чтения (мс)
Время выполнения операций чтения в мс (ср/мин/макс/откл).

последовательностей
Число последовательностей чтения. Последовательность - это несколько блоков (по 512 байт), считанных подряд. Число последовательностей чтения характеризует упорядоченность доступа.

длина последовательности
Информация о числе блоков в последовательности.

запись Число операций записи, выполненных над томом.

записано (блоков)
Объем записанных данных.

время записи (мс)
Время выполнения операций записи.

последовательностей
Число последовательностей записи. Последовательность - это несколько блоков (по 512 байт), записанных подряд.

длина последовательности
Информация о числе блоков в последовательности.

операций поиска
Число операций поиска, предшествующих запросу на чтение или запись; также показан процент операций, перед которыми выполнялся поиск.

размер поиска (блоков)
Информация о размере поиска в блоках по 512 байт. Кроме обычной статистики (ср/мин/макс/откл) отдельно показан размер начальной операции поиска (если считать, что головка чтения/записи первоначально находилась над блоком 0). Размер начального поиска может быть очень большим, поэтому он выводится отдельно и не учитывается в общей статистике.

размер поиска (цилиндров)
Информация о размере поиска в цилиндрах диска (только для физических томов).

период запросов
Характеризует величину интервала времени (в миллисекундах) между последовательными запросами на чтение или запись данных (ср/мин/макс/откл). Это значение определяет частоту обращений к тому.

производительность
Общая пропускная способность тома в Кб/с.

использование
Доля времени, в течение которого том был занят. Записи упорядочены по убыванию этого значения.

Большое время поиска приводит к увеличению времени ответа и снижению производительности приложений. Сравнивая количество операций чтения и число последовательностей можно определить способ доступа. Это относится и к операциям записи.

Подробная статистика работы с файлами, упорядоченная по процессу:

Подробная статистика, упорядоченная по процессу, выводится для каждого файла, упомянутого в отчете Наиболее активные файлы.

ИД процесса
ИД процесса, открывшего файл.

Имя Полное имя открытого файла.

ИД нити
ИД нити, открывшей файл.

Число операций поиска
Число операций поиска.

Число операций чтения
Число операций чтения.

Число ошибок чтения
Число ошибок чтения.

Число операций записи
Число операций записи.

Число ошибок записи
Число ошибок записи.

Прочитано байт
Чисто прочитанных байт.

минимальное
Минимальное число одновременно прочитанных байт.

среднее
Среднее число одновременно прочитанных байт.

максимальное
Максимальное число одновременно прочитанных байт.

Записано байт
Чисто записанных байт.

минимальное
Минимальное число одновременно записанных байт.

среднее
Среднее число одновременно записанных байт.

максимальное
Максимальное число одновременно записанных байт.

Время чтения
Время выполнения операций чтения.

Время записи
Время выполнения операций записи.

Подробная статистика работы с файлами, упорядоченная по нити:

Подробная статистика, упорядоченная по нити, выводится для каждого файла, упомянутого в отчете `Наиболее активные файлы`.

ИД нити
ИД нити, открывшей файл.

Имя Полное имя открытого файла.

ИД процесса
ИД процесса, открывшего файл.

Число операций поиска
Число операций поиска.

Число операций чтения
Число операций чтения.

Число ошибок чтения
Число ошибок чтения.

Число операций записи
Число операций записи.

Число ошибок записи
Число ошибок записи.

Прочитано байт
Чисто записанных байт.

минимальное
Минимальное число одновременно прочитанных байт.

среднее
Среднее число одновременно прочитанных байт.

максимальное
Максимальное число одновременно прочитанных байт.

Записано байт
Чисто записанных байт.

минимальное
Минимальное число одновременно записанных байт.

среднее
Среднее число одновременно записанных байт.

максимальное
Максимальное число одновременно записанных байт.

Время чтения
Время выполнения операций чтения.

Время записи
Время выполнения операций записи.

Рекомендации по работе с командой filemon:

Ниже приведены рекомендации по применению команды **filemon**.

- Файл `/etc/inittab` всегда находится в числе активных. В файле `/etc/inittab` перечислены работающие демоны, поэтому он регулярно сравнивается со списком запущенных процессов.
- Файл `/etc/passwd` также обычно находится в числе активных. Он применяется для проверки прав доступа к файлам и каталогам.
- Большое время поиска увеличивает время ответа и уменьшает производительность системы.
- Если для большинства операций чтения и записи требуется предварительный поиск, причиной может быть фрагментация файлов или размещение на одном физическом диске нескольких интенсивно используемых файловых систем. Однако в случае электронной обработки транзакций (OLTP) или СУБД предварительный поиск выполняется практически всегда.
- Если число операций чтения и записи близко к числу последовательностей, доступ к физическому диску в большинстве случаев неупорядоченный, а не последовательный. Последовательность - это несколько страниц, записанных или считанных подряд. Число страниц в последовательности указано в поле `длина последовательности`. Для неупорядоченного доступа к файлу также требуется много операций поиска. В этом случае путем анализа вывода команды **filemon** невозможно определить, фрагментирован ли файл. Для получения дополнительной информации нужно вызвать команду **fileplace**.
- Для удаленных файлов в столбце `том:I-узел` показано имя удаленной системы.

Поскольку на выполнение команды **filemon** затрачивается часть процессорного времени, применяйте ее только при необходимости. При анализе вывода команды учитывайте влияние этой команды на производительность системы. Тесты показывают, что в среде с большой нагрузкой на процессор:

- При небольшом объеме ввода-вывода команда **filemon** замедлила компиляцию большого проекта на один процент.
- При большом объеме ввода-вывода команда **filemon** замедлила работу программы, записывающей данные на диск, на пять процентов.

Обзор показателей дискового ввода-вывода

Обычно большое значение `% iowait` указывает на ошибку в приложении, недостаток памяти или неправильную настройку подсистемы ввода-вывода. Например, приложение может считывать большой объем лишних данных, которые впоследствии не обрабатываются. Для устранения узкого места в подсистеме ввода-вывода важно понять причину его возникновения.

Работа приложения зависит от эффективности дискового ввода-вывода во многих ситуациях, для каждой из которых можно придумать собственное решение. Ниже перечислены некоторые действия, позволяющие повысить эффективность дискового ввода-вывода:

- Уменьшение числа активных логических томов и файловых систем, расположенных на одном физическом диске. Основная идея состоит в равномерном распределении нагрузки по всем дискам.
- Распределение логического тома по нескольким физическим дискам. Это особенно полезно при одновременном доступе к нескольким файлам.
- Создание нескольких журналов JFS для группы томов, расположенных в разных файловых системах (желательно на устройствах с кэшированием записи). Это рекомендуется делать для приложений, которые создают, изменяют и удаляют большое количество файлов, включая временные.
- Если в выводе команды **iostat** указано, что операции ввода-вывода распределены по дисковым накопителям системы неравномерно, причем один из дисков загружен более чем на 70-80 процентов, то для уменьшения фрагментации рекомендуется реорганизовать файловые системы, например, путем резервного копирования и восстановления. Фрагментация приводит к увеличению числа операций поиска, на выполнение которых может затрачиваться большая часть времени.
- Если фоновые программы выполняют большое число операций дискового ввода-вывода, мешая работе интерактивных программ, то установите для них ограничение ввода-вывода.
- Если небольшой набор файлов считывается несколько раз, то увеличьте объем оперативной памяти. Это позволит повысить эффективность ввода-вывода за счет буферизации этих файлов.
- Если применяется преимущественно неупорядоченный доступ, то для повышения производительности рекомендуется увеличить число дисковых накопителей и распределить активные файлы по большему числу устройств.
- Если приложения обращаются к файлам, расположенным на нескольких дисках, причем применяется преимущественно последовательный способ доступа, то для повышения производительности рекомендуется увеличить число дисковых адаптеров. Кроме того, для хранения таких файлов могут быть созданы логические тома с чередованием данных.
- Применение устройств с кэшированием записи.
- Применение асинхронного ввода-вывода.

Контроль производительности LVM с помощью команды lvmstat

С помощью команды **lvmstat** можно определить, какие области и разделы логического тома используются наиболее интенсивно.

Для просмотра статистики по обращениям к накопителям с помощью команды **lvmstat** нужно включить сбор статистических данных для логического тома или группы томов.

Для того чтобы включить сбор статистики **lvmstat** для отдельного логического тома, выполните следующую команду:

```
# lvmstat -l имя-логического-тома -e
```

Для того чтобы отключить сбор статистики **lvmstat** для отдельного логического тома, выполните следующую команду:

```
# lvmstat -l имя-логического-тома -d
```

Для того чтобы включить сбор статистики **lvmstat** для всех логических томов в группе томов, выполните следующую команду:

```
#
lvmstat -v имя-логического-тома -e
```

Для того чтобы отключить сбор статистики **lvmstat** для всех логических томов в группе томов, выполните следующую команду:

```
#
lvmstat -v имя-логического-тома -d
```

Если в команде **lvmstat** не будет указан интервал, то статистика будет приведена для всех разделов логического тома. Если будет указан интервал в секундах, команда **lvmstat** выдаст только статистику по разделам, к которым были обращения за указанный период времени. Ниже приведен пример вывода команды **lvmstat**:

```
# lvmstat -l lv00 1

Log_part  mirror#  iocnt   Kb_read   Kb_wrtn      Kbps
       1       1   65536     32768         0      0.02
       2       1   53718     26859         0      0.01

Log_part  mirror#  iocnt   Kb_read   Kb_wrtn      Kbps
       2       1    5420      2710         0  14263.16

Log_part  mirror#  iocnt   Kb_read   Kb_wrtn      Kbps
       2       1    5419      2709         0  15052.78

Log_part  mirror#  iocnt   Kb_read   Kb_wrtn      Kbps
       3       1    4449      2224         0  13903.12
       2       1     979       489         0   3059.38

Log_part  mirror#  iocnt   Kb_read   Kb_wrtn      Kbps
       3       1    5424      2712         0  12914
```

С помощью флага **-c** можно ограничить число разделов, для которых будет выдавать статистику команда **lvmstat**. Флаг **-c** указывает, сколько наиболее интенсивно используемых разделов должно быть показано в статистике. Ниже приведен пример команды **lvmstat** с флагом **-c**:

```
# lvmstat -l lv00 -c 5
```

Эта команда выдаст сведения по пяти наиболее активным разделам.

Если не будет указано число итераций, команда **lvmstat** будет выдавать результаты до тех пор, пока она не будет прервана. В противном случае команда **lvmstat** выполнит указанное число итераций.

Если команда **lvmstat** продемонстрирует, что только некоторые разделы используются интенсивно, их можно разнести на разные жесткие диски с помощью команды **lvmstat**. Команда **lvmstat** позволяет переносить отдельные разделы на другие жесткие диски. Инструкции по работе с командой **lvmstat** приведены в разделе migratelp Command книги *Справочник по командам, том 3*.

Дополнительные сведения и описание команды **lvmstat** приведены в разделе lvmstat Command книги *Справочник по командам, том 3*.

Атрибуты логических томов, влияющие на производительность

При создании логических томов следует учитывать множество факторов, которые могут влиять на производительность. В зависимости от выбора тех или иных вариантов вы можете заполнить поля ввода в меню **mklv** команды **smitty**.

Расположение данных на физическом томе

Стратегия размещения данных на физическом томе определяет правила выбора физических разделов на физическом томе. Существует пять возможных вариантов: размещение на внешнем крае, во внешней области, в средней области, во внутренней области и на внутреннем крае.

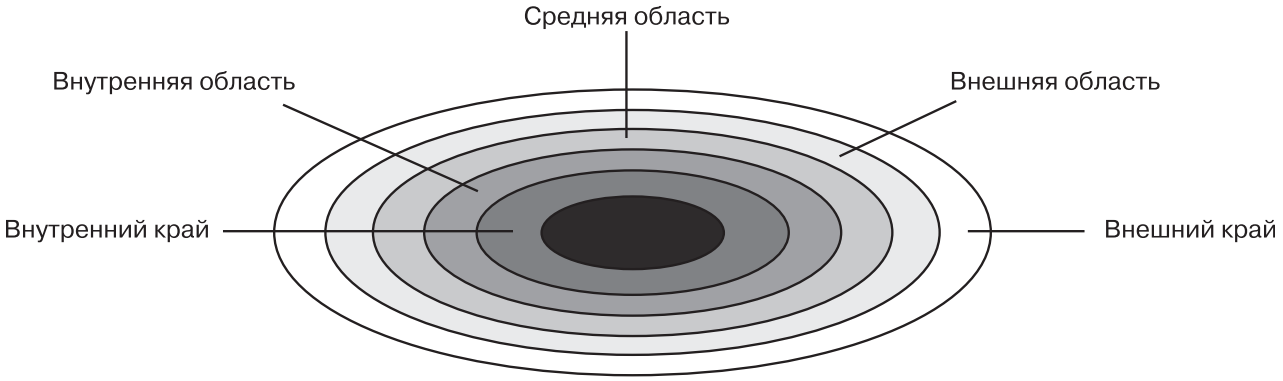


Рисунок 17. Стратегия размещения данных на физическом томе. На рисунке показаны области для размещения данных на физическом томе, или диске. Диск разделен на дорожки, первые из которых расположены на внешнем крае диска, а последние - на внутреннем. Поскольку данные считываются путем прокручивания дорожки под считывающей головкой, данные, записанные ближе внутреннему краю, будут считываться быстрее. Отчасти это связано с тем, что тратится меньше времени на перемещение считывающей головки. Чем ближе к внутреннему краю диска, тем плотнее записаны данные. Следовательно, для их считывания требуется переместить головку диска на меньшее расстояние. В итоге общее время выполнения операций чтения уменьшается.

Физические разделы нумеруются последовательно, начиная с единицы, от внешнего края к внутреннему.

Возможно размещение данных на внешнем и внутреннем краях физического тома. В этом случае на позиционирование считывающей головки затрачивается очень большое время, что приводит к значительному увеличению времени ответа в приложениях, работающих с этими разделами. Дорожки, расположенные на внешних краях дисков, разбиваются на большее число секторов, поэтому последовательные операции ввода-вывода на них выполняются быстрее.

Возможно также размещение данных во внутренней и внешней средних областях. Такие стратегии в среднем обеспечивают довольно низкое время позиционирования считывающей головки. Эта стратегия может применяться для большинства разделов физического тома.

Кроме того, данные могут размещаться в средней части физического тома. В этом случае на позиционирование считывающей головки затрачивается меньше всего времени. В результате все обращения к этим разделам обрабатываются очень быстро. Эта стратегия может применяться только для ограниченного числа разделов физического тома.

Если в системе велика интенсивность подкачки, то логический том с пространством подкачки рекомендуется разместить в средней области диска. В то же время, загрузочный логический том и логический том дампа применяются редко, поэтому их можно разместить на краях физического тома.

В общем случае, чем больше число обращений к логическому тому (общее число обращений или число обращений какого-то важного приложения), тем ближе к средней области диска должны быть расположены физические разделы этого логического тома.

Диапазон физических томов

Стратегия распределения данных по физическим томам применяется для выбора физических устройств, на которых будут размещены физические разделы логического тома. При этом может быть задействовано максимальное или минимальное число дисков.

Строгая стратегия распределения с максимальным числом дисков (диапазон=максимум, строгая=да)

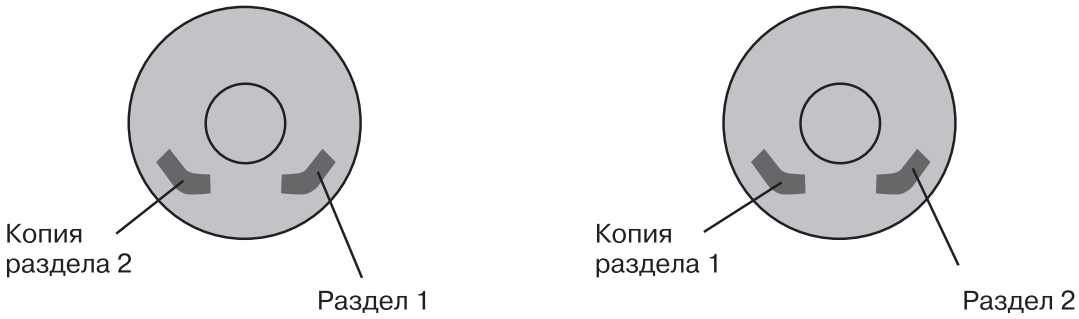


Рисунок 18. Стратегия распределения данных по физическим томам. На данном рисунке показано 2 физических тома. Один из них содержит раздел 1 и копию раздела 2. Второй содержит раздел 2 и копию раздела 1. В данном случае применяется стратегия размещения с максимальным числом дисков (Range=maximum), при этом на каждом диске хранится только одна копия логического тома (Strict=y).

Значение minimum определяет число физических томов, применяемых для размещения физических разделов. Обычно эта стратегия применяется для обеспечения надежности и повышения доступности логических томов, копии которых не создаются. Существует два варианта применения опции minimum:

- Без создания копий: Опция minimum указывает, что все физические разделы логического тома должны быть размещены на одном физическом томе. Если требуется разместить данные на нескольких томах, то применяется минимальное число томов.
- С созданием копий: Опция minimum указывает, что число физических томов должно совпадать с числом копий. Если требуется разместить данные на большем числе томов, то применяется минимальное число томов, на которых можно разместить все физические разделы. Во всех случаях при выборе числа томов учитываются значения других параметров, например, опция strict.

Указанные правила применяются при расширении и копировании логического тома. Например, если выбрана опция minimum и разрешено создание копий, то для определения числа физических томов оценивается текущее размещение логического тома.

Опция maximum определяет число физических томов, применяемых для размещения физических разделов. Если выбрана эта опция, то физические разделы логического тома распределяются по максимальному числу дисков. Эта опция применяется для повышения производительности в том случае, когда создаются копии логических томов. Если логический том, существующий в единственном экземпляре, будет распределен по нескольким физическим томам, то потеря одного физического раздела этого тома приведет к повреждению всего логического тома.

Максимальное число физических томов, выделяемых для размещения

Задает максимальное число физических томов, на которых может быть размещен логический том.

Значение этого параметра не должно превосходить общего числа физических томов в группе томов и должно быть больше единицы. Эта опция связана с “Диапазон физических томов”.

Согласование зеркальных копий

При выполнении обычных операций ввода-вывода LVM обеспечивает согласованность данных, хранящихся в зеркальных копиях логического тома.

При записи данных на логический том LVM отправляет запросы на запись данных во все зеркальные копии. Ошибка может возникнуть в том случае, если сбой системы произойдет до того, как данные будут записаны во все зеркальные копии. Если для логического тома включена функция согласования зеркальных копий (MWC), то LVM сохраняет дополнительную информацию, необходимую для восстановления несогласованных зеркальных копий. Такая функция должна быть включена для большинства логических томов с зеркальной защитой. Если после повторного подключения группы томов хранящиеся в ней данные не применяются, согласование зеркальных копий можно выключить.

Запись MWC занимает один сектор диска. В нее записываются идентификаторы логических разделов, которые окажутся несогласованными в случае аварийного завершения работы системы. На основании этой информации система восстанавливает согласованность логических разделов после повторной активизации группы томов.

Примечание: Поскольку сектор, в котором расположена запись MWC, расположен на внешнем крае диска, то для повышения производительности зеркальную копию такого логического тома также рекомендуется разместить на внешнем крае.

Начиная с версии 5, в AIX реализован дополнительный режим обеспечения целостности при записи на накопители с зеркальной защитой. Этот режим называется пассивной целостностью зеркальных дисков. По умолчанию применяется активная MWC. В режиме активной целостности восстановление данных происходит с минимальными затратами времени при первой загрузке системы после сбоя. Однако такой подход приводит к снижению производительности дисковой подсистемы при записи, особенно при работе с большим количеством мелких файлов. В режиме пассивной MWC снижения производительности нет, но при перезагрузке после сбоя нужно полностью синхронизировать поврежденную группу томов с помощью команды **syncvg -f**. Для этого требуется отключить автоматическое включение групп томов.

Пассивная MWC выигрывает не только за счет отсутствия снижения производительности, но и за счет автоматической синхронизации логических томов по мере обращения к разделам дисковой памяти. Администратору не требуется вручную синхронизировать логические тома и отключать режим автоматического включения групп томов. Недостаток пассивной MWC заключается в снижении скорости операций чтения вплоть до окончания синхронизации всех разделов.

Нужную опцию обеспечения целостности при записи на диски с зеркальной защитой можно указать в SMIT при создании и изменении логических томов. Данная опция имеет смысл только для логических томов с зеркальной защитой.

Информация, связанная с данной:

Стратегия согласованности зеркальной записи для логического тома

Выделить отдельный PV каждой копии логического раздела

Указывает, включена ли стратегия strict.

Если она включена, то все копии логического раздела размещаются на разных физических томах. Эта опция связана с “Диапазон физических томов” на стр. 203.

Перемещать логический том при реорганизации

Указывает, можно ли перемещать логический том во время реорганизации.

В частности, логические тома с чередованием данных перемещать нельзя (соответствующая опция установлена по умолчанию). В зависимости от конфигурации системы, вы можете разрешить перемещать некоторые логические тома.

Стратегии планирования чтения и записи копий логических томов
Для логического тома могут быть установлены различные стратегии планирования.

Для логических томов с несколькими копиями предусмотрены следующие стратегии планирования:

- Стратегия *параллельного доступа* позволяет равномерно распределить операции чтения по дискам. При получении запроса на чтение данных система проверяет, свободна ли основная копия. Если да, то данные считываются из основной копии. Если нет, то система пытается считать данные из дополнительной копии. Если эта копия свободна, то данные считываются из дополнительной копии. В противном случае данные считываются из копии с минимальным числом необработанных запросов на ввод-вывод. Запись данных выполняется одновременно во все копии.
- В стратегии *параллельная запись/последовательное чтение* данные всегда считываются из основной копии. Запись данных выполняется одновременно во все копии.
- Стратегия *параллельная запись/карусельное чтение* аналогична стратегии параллельного чтения и записи за исключением того, что в ней сразу выбирается копия с минимальным числом необработанных операций ввода-вывода. Другими словами, в этой стратегии не проверяется, что свободна основная копия. В результате все копии используются в равной степени, даже если в системе всегда выполняется не более одного запроса на чтение-запись. Запись данных выполняется одновременно во все копии.
- В стратегии *последовательного чтения и записи* все данные считываются из основной копии. Запись выполняется последовательно: вначале записывается основная копия, затем - дополнительная.

Если данные существуют в системе в единственном экземпляре, то драйвер логического тома преобразует адрес, указанный в запросе на чтение или запись, в физический адрес, а затем вызывает драйвер физического устройства для обработки запроса. В этом случае при выполнении операций записи отслеживаются все перемещения поврежденных блоков. Кроме того, процессу возвращается информация обо всех ошибках, возникающих во время чтения.

Стратегии планирования для логических томов с зеркальными копиями позволяют считывать данные с таких логических томов так же эффективно, как и с томов без зеркальной защиты. К таким стратегиям относится стратегия параллельной записи с параллельным чтением или карусельным распределением запросов на чтение. Запись на тома с зеркальной защитой выполняется медленнее, чем на обычные тома с тем же числом дисков.

Включение проверки записи
Указывает, нужно ли проверять, что данные были записаны на логический том, путем считывания этих данных.

Включение этой опции приводит к снижению производительности.

Размер полосы
Размер блоков чередования равен произведению *размера полосы операций* и числа дисков данных в массиве. Размер полосы операций может представлять собой любую степень двойки от 4 КБ до 128 МБ.

Для размещения логического тома с чередованием данных требуется как минимум два физических тома. Число разделов, входящих в состав логического тома, должно быть кратно числу дисков, применяемых для размещения тома. Дополнительная информация приведена в разделе “Настройка логических томов с чередованием данных” на стр. 209.

Настройка производительности LVM с помощью команды lvmo
С помощью команды **lvmo** можно изменять число физических буферов (pbuf) LVM для отдельных групп томов.

В команде **lvmo** предусмотрены следующие параметры:

pv_pbuf_count
Число pbuf, добавляемых при добавлении физического тома в группу томов.

max_vg_pbuf_count
Максимальное допустимое число pbuf для группы томов. Новое значение вступает в силу при следующем включении группы томов.

global_pbuf_count
Минимальное число pbuf, добавляемых при добавлении физического тома в любую группу томов. Для изменения этого значения применяется команда **ioo**.

aio_cache_pbuf_count
Текущее общее число физических буферов (pbuf), доступных логическому тому aio_cache в группе томов. Максимальное число **aio_cache_pbuf_count**, которое может быть выделено группе томов, задается параметром **max_vg_pbuf_count**.

В следующем примере команда **lvmo -a** показывает текущие значения настраиваемых параметров группы томов rootvg.

```
# lvmo -a

vgname = rootvg
pv_pbuf_count = 256
total_vg_pbufs = 768
max_vg_pbuf_count = 8192
pervg_blocked_io_count = 0
global_pbuf_count = 256
global_blocked_io_count = 20
aio_cache_pbuf_count = 512
```

Для просмотра текущих параметров другой группы томов выполните следующую команду:

```
lvmo -v <группа-томов> -a
```

Для изменения значения переменной укажите ее в команде **lvmo** со знаком равенства:

Примечание: В следующем примере параметру *pv_pbuf_count* для группы томов redvg присвоено значение 257.

```
# lvmo -v redvg -o pv_pbuf_count=257

vgname = redvg
pv_pbuf_count = 257
total_vg_pbufs = 257
max_vg_pbuf_count = 263168
pervg_blocked_io_count = 0
global_pbuf_count = 256
global_blocked_io_count = 20
```

Примечание: Слишком большие значения pbuf могут отрицательно повлиять на производительность системы или вызвать непредсказуемое поведение сервера.

Информация, связанная с данной:

Команда lvmo

Рекомендации по работе с физическими томами

Что следует принимать во внимание при работе с физическими томами.

Скорость обработки обращений к диску главным образом зависит от способа доступа приложения к диску. Много независимых операций ввода-вывода небольшого объема называются неупорядоченным доступом; небольшое число операций большого объема - последовательным. В случае неупорядоченного доступа запросы будут обрабатываться быстрее, если в системе будет установлено много дисков небольшого размера. В случае последовательного доступа должно применяться небольшое число дисков с высокой скоростью передачи данных, либо логические тома с чередованием данных, размещенные на нескольких дисках.

Имя группы томов

Для повышения производительности и упрощения администрирования системы группа томов по умолчанию, rootvg, должна по возможности состоять из одного физического тома, на который первоначально была установлена операционная система.

Рекомендуется не размещать в группе томов rootvg пользовательские данные, для того чтобы их не приходилось восстанавливать после обновления, повторной установки или сбоя операционной системы. Кроме того, в этом случае вы сможете быстрее обновлять и повторно устанавливать операционную систему, так как никакие другие данные менять не нужно.

Пользовательские данные должны храниться на других дисках, входящих в отдельные группы томов. Хранение пользовательских данных в отдельной группе томов упрощает процедуру экспорта данных в другие системы.

Файловую систему и ее протокол рекомендуется разместить на разных дисках, если в протокол часто заносятся записи о выполняемых операциях. Для повышения производительности логические тома протоколов (протоколов JFS и баз данных) рекомендуется размещать на устройствах с кэшем (таких как твердотельные диски или массивы дисков с кэшем записи).

Понятия, связанные с данным:

“Реорганизация протоколов файловой системы и логических томов протоколов” на стр. 255
Для поддержания согласованной структуры файловой системы в JFS или расширенном JFS (JFS2) ведется протокол, аналогичный журналу базы данных. В этот протокол заносится информация обо всех операциях, выполненных над метаданными файловой системы. К метаданным относится суперблок, i-узлы, косвенные ссылки на данные и каталоги.

Влияние зеркальной защиты корневой группы томов на производительность

Если установлена зеркальная защита, то все данные должны быть записаны на все копии логического тома. Как правило, такая операция записи выполняется дольше по сравнению с логическим томом без зеркальной защиты.

Хотя зеркальные копии часто создаются для защиты пользовательских данных, в частности, баз данных, они редко применяются для защиты системных томов.

Кроме того, зеркальная защита создает дополнительную нагрузку на процессор, так как для выполнения двух операций ввода-вывода требуется обработать больше команд, чем для выполнения одной операции. Для того чтобы понять, какие трудности могут возникнуть при настройке зеркальной защиты для корневой группы томов, нужно знать, какие данные содержит эта группа томов.

Логические тома, входящие в корневую группу томов, которая содержит файлы из каталога `/` и часто используемый каталог `/usr/bin`, в котором расположены многие исполняемые программы. Данные в этом каталоге редко изменяются. В пространство подкачки данные записываются только в том случае, если физической памяти системы не хватает для выполнения текущих операций. Время от времени подкачка выполняется во всех системах, однако большое число операций подкачки приводит к увеличению времени ответа. Для решения этой проблемы обычно требуется увеличить объем оперативной памяти системы.

Многие приложения записывают данные в файловые системы `/tmp` и `/var`. Например, каталог `/tmp` часто применяется приложениями (например, компиляторами) для хранения временных файлов. В каталог `/var` записываются файлы, присвоенные очереди почтовой программы и очереди печати. Каталог **jfslog** при обычной работе применяется только для записи. Из оставшихся файловых систем во время обычной работы системы применяется только каталог `/home`. Довольно часто домашние каталоги пользователей размещаются в других файловых системах. Это упрощает работу с корневой группой томов.

Для того чтобы установить зеркальную защиту для корневой группы томов, необходимо скопировать каждый логический том с помощью команды **mklvcopy** или скопировать всю группу томов с помощью команды **mirrorvg**.

По умолчанию команда **mirrorvg** устанавливает стратегию планирования, в которой операции чтения и записи выполняются параллельно, и выключает функцию проверки записи для всех логических томов. Для пространства подкачки функция согласования зеркальных копий выключается. Для всех остальных логических томов эта функция включается. Для того чтобы сократить время перемещения головки диска от логического тома к записи о согласованности зеркальных копий, разместите все логические тома, на которые часто записываются данные, как можно ближе к внешнему краю диска.

Зеркальная защита корневой группы томов не оказывает значительное влияние на производительность системы, если включена зеркальная защита пространства подкачки; снижение производительности прямо пропорционально интенсивности подкачки. Если пространство подкачки расположено в корневой группе томов, и интенсивность подкачки велика, то не рекомендуется устанавливать зеркальную защиту для корневой группы томов.

Подводя итог, можно сказать, что зеркальную защиту для корневой группы томов имеет смысл устанавливать в том случае, если в системе редко выполняется подкачка.

Реорганизация логических томов

Если логический том сильно фрагментирован, то его можно реорганизовать с помощью команды **reorgvg** (или **smitty reorgvg**), чтобы оно соответствовал заданным стратегиями размещения данных.

Указанная команда изменяет размещение физических разделов в группе томов в соответствии со свойствами логического тома. Если в команде заданы имена логических томов, то наивысший приоритет присваивается первому из указанных томов. Перед вызовом этой команды убедитесь, что группа томов подключена, и в ней есть свободные разделы. Реорганизация выполняется только в том случае, если для логических томов установлен флаг, разрешающий перемещение тома. Если такой флаг не установлен, то логический том игнорируется.

Стратегии, связанные с логическими томами, должны определяться их назначением. При этом руководствуйтесь следующими рекомендациями:

- Логические тома, которые используются чаще всего, нужно размещать на разных физических томах.
- Логические тома, которые используются чаще всего, рекомендуется распределить по нескольким физическим томам.
- Разделы логических томов, которые используются чаще всего, нужно разместить в средней области дисков. Это не относится к логическим томам, для которых включена опция проверки согласованности зеркальных копий.
- Логические тома, которые используются реже всего, рекомендуется размещать на краях дисков (за исключением случая последовательного доступа к тому).
- Данные логического тома должны находиться в смежных разделах диска.
- Создавайте логические тома максимального размера.
- Логические тома, которые используются чаще всего, следует размещать рядом друг с другом.
- Файлы, считываемые или записываемые последовательно, следует размещать на крае диска.

Рекомендации по повышению производительности

Высокая производительность при работе с логическими томами часто достигается за счет снижения доступности томов. Решите, что важнее - высокая производительность или высокая доступность томов.

Ниже приведены некоторые рекомендации по настройке производительности с помощью команд SMIT:

- Если в системе в основном выполняются операции чтения, то для повышения производительности работы с логическими томами рекомендуется установить зеркальную защиту и выбрать стратегию планирования с параллельными операциями чтения и записи. В этом случае данные будут считываться из той копии, которая используется реже всего. В то же время, зеркальная защита снижает скорость выполнения

операций записи, так как в этом случае требуется записать данные в несколько копий и обновить запись о согласованности зеркальных копий. Для того чтобы каждая копия размещалась на отдельном физическом томе, выберите стратегию strict.

- Выключите опцию проверки записи. Если создается несколько копий, то выключите опцию согласования зеркальных копий.
- В общем случае, логические тома, которые используются чаще всего, следует размещать в средней области диска. В этом случае время позиционирования считывающей головки будет минимальным. Однако из этого правила есть некоторые исключения:
  - На дорожки, расположенные на краях диска, помещается больше данных, поэтому их чтение выполняется быстрее. Если при работе с логическим томом применяется последовательный способ доступа, то рекомендуется разместить этот том на краю диска.
  - Другим исключением являются логические тома, для которых включена опция проверки согласованности зеркальных копий (MWCC). Поскольку сектор MWCC расположен на краю диска, то для повышения производительности логический том с зеркальной защитой также рекомендуется разместить на краю диска.
- Логические тома, к которым приложения обращаются часто или одновременно, нужно размещать на диске рядом друг с другом. В этом случае компактность ссылок важнее того, в какой области диска будут расположены логические тома.
- Логические тома, которые используются довольно часто, рекомендуется помещать во внешнюю область, а редко используемые тома - на внешний край диска.
- Если будет применяться стратегия размещения с максимальным числом дисков, то операции чтения и записи будут распределены по физическим томам.

Рекомендации по повышению доступности логических томов

Ниже приведены некоторые рекомендации по повышению доступности логических томов с помощью команд SMIT:

- Создавайте три копии логического тома (две зеркальных копии)
- Включите опцию проверки записи
- Установите стратегию размещения с минимальным числом дисков (число физических томов совпадает с числом зеркальных копий)
- Установите стратегию планирования с последовательными операциями чтения и записи
- Выберите стратегию strict (все зеркальные копии хранятся на разных физических томах)
- Добавляйте в группу томов не менее трех физических томов
- Размещайте зеркальные копии на физических томах, подключенных к разным шинам, адаптерам и источникам питания

Если в группе томов будет хотя бы три тома, то в случае повреждения одного из физических томов целостность группы томов будет сохранена. Если копии будут размещены на дисках, подключенных к разным адаптерам, шинам и источникам питания, то в случае сбоя одного из устройств всегда будет доступна зеркальная копия, расположенная на другом физическом томе.

Настройка логических томов с чередованием данных

Чередование данных - это технология распределения данных по нескольким дискам в рамках одного логического тома для обеспечения параллельного доступа к данным. Главное преимущество чередования данных - значительное повышение производительности при последовательном чтении или записи больших объемов данных, однако оно также позволяет уменьшить время произвольного доступа.

Простой пример показан на следующем рисунке.

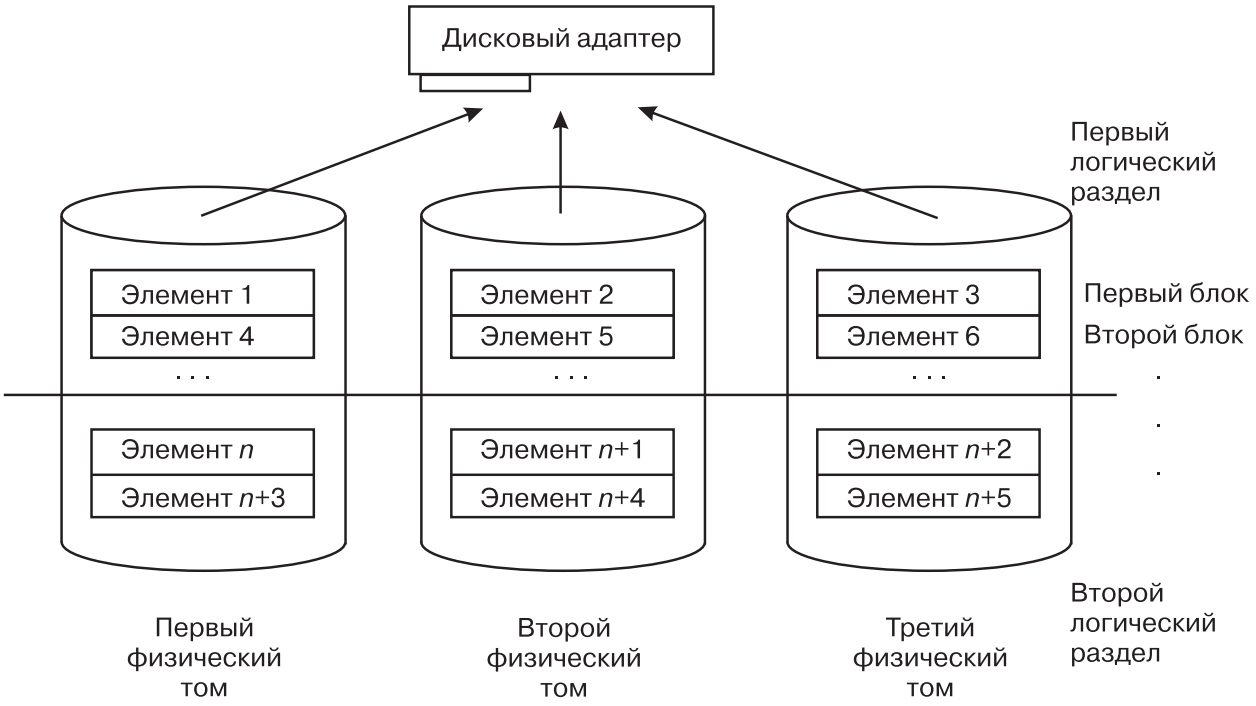


Рисунок 19. Логический том /dev/lvs0 с чередованием данных. На этом рисунке показано три физических тома, или диска. Каждый диск поделен на два логических тома. Первый логический том диска 1 содержит блоки данных 1 и 4. Логический том 1 диска 2 содержит блоки данных 2 и 5, а логический том 1 диска 3 содержит блоки данных 3 и 5. Второй логический том диска 1 содержит блоки данных n и n+3. Второй логический том диска 2 содержит блоки данных n+1 и n+4. Второй логический том диска 3 содержит блоки данных n+2 и n+5.

В обычном логическом томе адреса данных соответствуют порядку следования блоков в физических разделах. В логическом томе с чередованием данных адреса данных соответствуют порядку следования блоков чередования. Полный набор с чередованием данных состоит из нескольких блоков чередования, по одному на каждом из физических дисков, входящих в логический том. Соответствие между считываемым или записываемым блоком и физическими блоками данных на физических дисках устанавливается LVM. Если в обработке запроса на чтение или запись участвует несколько физических дисков, необходимые операции ввода-вывода выполняются на всех дисках одновременно.

Рассмотрим пример, в котором гипотетическая группа томов lvs0, состоящая из шести разделов по 2 МБ с длиной чередования в 64 КБ, содержит файловую систему JFS. Если приложение последовательно считывает большой файл, а функция упреждающего чтения каждый раз считывает максимальное число страниц, каждый запрос на чтение генерирует два или три запроса на операции ввода-вывода для каждого диска, поэтому всего считывается восемь страниц (в предположении, что файл не фрагментирован). Операции чтения выполняются в порядке, установленном драйвером дисков. Затем считанные данные собираются вместе и передаются в приложение.

Хотя у всех дисков величина начальной задержки различна и зависит от расположения считывающей головки в момент начала операции, через некоторое время скорость чтения для всех трех дисков будет близка к максимальной.

Планирование логического тома с чередованием данных

Для того чтобы создать логический том с чередованием данных, необходимо принять во внимание определенные факторы.

При определении логического тома с чередованием данных следует указать:

диски Не меньше двух физических дисков. Во время операций последовательного ввода-вывода с этими

дисками должно выполняться минимальное число прочих операций. Для некоторых сочетаний адаптеров и дисков необходимо распределять нагрузку на логический том с чередованием данных между несколькими адаптерами.

размер блока чередования
Хотя эта величина может быть любой степенью двойки в пределах от 4 КБ до 128 КБ, при ее выборе следует учитывать работу последовательного упреждающего чтения, так как именно с этой функцией связана большая часть операций чтения. Следует стремиться к тому, чтобы каждая операция опережающего чтения приводила к выполнению хотя бы одной операции ввода-вывода на каждом диске, в идеале - чтобы на всех дисках выполнялось одинаковое количество операций (см. предыдущий рисунок).

размер Число физических разделов, входящих в состав логического тома. Это число должно быть кратно числу входящих в состав тома физических дисков.

атрибуты
Операционная система AIX поддерживает зеркальную защиту, в результате чего значение атрибута числа копий может быть больше 1.

Настройка для ввода-вывода логических томов с чередованием данных

Чередование данных увеличивает скорость последовательного и неупорядоченного доступа к дискам.

Для получения максимальной производительности следуйте следующим рекомендациям:

- Распределите логический том по максимальному количеству физических томов.
- Задействуйте при этом максимальное количество адаптеров.
- Создайте для логических томов с чередованием данных отдельную группу томов.
- Установите размер блоков чередования равным 64 КБ.
- Укажите в параметре *minpgahead* значение 2 с помощью команды **ioo**. См. раздел “Настройка производительности последовательного чтения” на стр. 246.
- С помощью команды **ioo** укажите в параметре *maxpgahead* значение, в 16 раз превышающее число дисков. При этом упреждающее чтение будет выполняться блоками, размер которых равен размеру блока чередования (64 КБ), умноженному на число дисков, и в результате при каждой операции упреждающего чтения будет выполняться одна операция чтения для каждого диска.
- Запросы ввода-вывода: 64 КБ, умноженные на число дисков. Это значение равно значению параметра *maxpgahead*.
- С помощью команды **ioo** измените значение *maxfree* в соответствии с новым значением *maxpgahead* (*maxfree* = *minfree* + *maxpgahead*). См. раздел “Значения для параметров minfree и maxfree” на стр. 154.
- Работайте с буферами ввода-вывода, выровненными по границе 64 байт. Если в логический том входят физические диски, подключенные к нескольким адаптерам, буферы ввода-вывода должны быть выровнены по границе 64 байт. При этом LVM не потребуется выполнять операции ввода-вывода для нескольких дисков последовательно. Следующий код создает буфер, выровненный по границе 64 байт:

```
char *buffer;
buffer = malloc(MAXBLKSIZE+64);
buffer = ((int)buffer + 64) & ~0x3f;
```

Если логические тома с чередованием данных находятся на логических томах с прямым доступом, и на них записывается объем данных более 1,125 МБ, то увеличение параметра *lvm_bufcnt* с помощью команды **ioo**позволяет повысить производительность записи. См. раздел “Тонкая настройка буферов файловой системы” на стр. 252.

Данный пример относится к логическому тому JFS с чередованием данных. То же самое относится и к расширенному JFS, за исключением того, что в расширенном JFS поддерживаются расширенные аналоги параметров **ioo**.

Кроме того, не рекомендуется смешивать на одном физическом томе логические тома с чередованием и без чередования данных. Все физические тома, на которых размещается набор логических томов с чередованием данных, должны быть одного размера.

Влияние зеркальных логических томов с чередованием данных на производительность

В AIX для логических томов с чередованием данных можно установить зеркальную защиту.

Это позволяет легко организовать высокопроизводительную резервную память. Измерения показывают, что скорость чтения и записи после установки зеркальной защиты не меняется; однако для этого необходимо вдвое больше дисков.

Запись через файловую систему позволяет достичь большей производительности за счет кэширования данных. Операции прямого вывода выполняются с меньшей скоростью. Поскольку прямая запись выполняется синхронно, управление будет возвращено в вызывающую программу только после выполнения всех операций записи. При операциях с большими блоками данных быстродействие увеличивается. Кроме того, на производительность влияет согласованность зеркальных копий (MWC).

Чередование данных в сочетании с зеркальной защитой позволяют создать резервную память с очень низким временем доступа.

Использование прямого дискового ввода-вывода

Некоторые приложения, такие как базы данных, не используют при работе файловую систему, поскольку самостоятельно выполняют такие действия, как ведение протоколов, трассировка данных и кэширование. Производительность таких приложений обычно превышает производительность линейного ввода-вывода, используемого вместо файлового, поскольку исключаются операции лишнего копирования данных в памяти, регистрации и блокирования I-узлов.

Для выполнения линейного ввода-вывода приложения должны работать со специальными символьными файлами `/dev/rlv*`. Применение специальных блочных файлов `/dev/lv*` невозможно, поскольку при работе с ними операции ввода-вывода разбиваются на блоки по 4 КБ. Интерфейсы `/dev/rhdisk*` и `/dev/hdisk*` линейного ввода-вывода на диск не должны использоваться, поскольку это вызывает снижение производительности и возможное рассогласование данных.

Использование вызовов sync и fsync

Если файл открыт с опцией O_SYNC или O_DSYNC, то перед выходом из каждой функции записи данные будут принудительно записываться на диск. Если при выполнении записи потребуется выделить дополнительное дисковое пространство (например, при расширении файла), то будет создана соответствующая запись в протоколе JFS.

Принудительная синхронизация содержимого оперативной памяти и данных, хранящихся на диске, выполняется в следующих случаях:

- Приложение вызывает функцию **fsync()** для некоторого файла. При этом все страницы памяти файла, содержащие обновленные данные, будут записаны на диск. Управление возвращается программе только после завершения работы **fsync()**.
- Приложение вызывает функцию **sync()**. При этом будет запланирована запись на диск всех страниц памяти файла, содержащих обновленные данные. Возврат управления программе не означает, что запрос **sync()** уже обработан.
- Пользователь может ввести команду **sync**, которая, в свою очередь, вызовет функцию **sync()**. В этом случае появление приглашения (или переход к следующей команде сценария оболочки) не означает, что запись уже выполнена.
- Демон `/usr/sbin/syncd` запускает функцию **sync()** через равные интервалы времени - обычно через 60 секунд. Благодаря этому система не накапливает больших объемов данных в энергозависимой памяти.

Операция sync расходует незначительный объем ресурсов CPU. Помимо этого, у нее есть следующие преимущества:

- Компактная запись данных.
- Запись по крайней мере 28 КБ системных данных, даже если с момента предыдущего вызова **sync** не было выполнено ни одной операции ввода-вывода.
- Ускорение записи данных на диск за счет отключения алгоритма отложенной записи. Это свойство особенно важно для тех программ, в которых после каждой операции записи выполняется операция **fsync()**.
- Создание при вызове **sync()** или **fsync()** записей в протоколе JFS о том, что измененные данные сохранены на диске.

Настройка ограничений для очереди диска и адаптера SCSI

Операционная система позволяет ограничить число ожидающих запросов адаптера SCSI на обмен данными с определенной шиной или диском SCSI. Такие ограничения позволяют воспользоваться аппаратными функциями обработки группы запросов, обеспечивая эффективную работу алгоритмов оптимизации поиска в драйверах устройств.

Для устройств, выпущенных не фирмой IBM, иногда нужно изменить ограничения для очередей, поскольку по умолчанию выбираются наиболее консервативные значения. Ниже описаны ситуации, в которых следует изменить значения по умолчанию, а также рекомендуемые значения.

Дисковые накопители независимых производителей

Для дисковых устройств фирмы IBM число одновременно ожидающих запросов по умолчанию равно 3. Это значение было выбрано в результате исследований и его нельзя изменить явно. Для дисковых устройств других производителей длина очереди по умолчанию равна 1. Если устройство поддерживает буферизацию нескольких запросов, то следует соответствующим образом изменить в системе описание этого устройства.

Ниже приведен пример вывода команды **lsattr**, в котором перечислены значения параметров по умолчанию для дискового устройства независимого производителя:

```
# lsattr -D -c disk -s scsi -t osdisk
pvid          нет  Идентификатор физического тома   Ложь
clr_q         нет  При ошибке устройство очищает очередь
q_err         да   Применяется бит QERR
q_type        нет  Тип очереди
queue_depth   1    Длина очереди
reassign_to   120  Тайм-аут повторной инициализации
rw_timeout    30   Тайм-аут чтения/записи
start_timeout 60   Тайм-аут запуска устройства
```

Для изменения этих параметров вы можете воспользоваться интерфейсом SMIT (команда **smitty chgdsk**) или командой **chdev**. Ниже приведен пример команды, устанавливающей размер очереди для диска SCSI hdisk5 независимого производителя, равным 3:

```
# chdev -l hdisk5 -a q_type=simple -a queue_depth=3
```

Дисковые подсистемы независимых производителей

Так же как и дисковые устройства, дисковые подсистемы независимых производителей относятся к классу disk, подклассу SCSI и типу `osdisk` ("прочие SCSI устройства").

Операционная система воспринимает дисковую подсистему как одно большое дисковое устройство. Поскольку в действительности дисковая подсистема состоит из нескольких физических дисковых устройств, каждое из которых может обрабатывать группы запросов, то для эффективного использования физических устройств следует установить достаточно большую длину очереди. Например, если hdisk7 - это дисковая подсистема из восьми дисков, то можно внести следующие изменения в ее параметры:

```
# chdev -l hdisk7 -a q_type=simple -a queue_depth=24
```

Если дисковая подсистема подключена через шину адаптера SCSI-2 Fast/Wide, то может потребоваться изменить ограничение на количество ожидающих запросов и для этой шины.

Расширение конфигурации

К сожалению, повышение производительности путем настройки возможно только до определенного предела. После этого возникает вопрос о том, какое дополнительное аппаратное обеспечение наиболее эффективно повысит производительность системы. Этот вопрос особенно сложен для систем, сильно зависящих от производительности дисковой подсистемы, так как в таком случае необходимо учесть большое число переменных.

Ниже перечислены изменения, которые могут повысить скорость выполнения программ, зависящих от дисковой подсистемы:

- Добавление дисковых накопителей и распределение данных между ними. Это позволяет выполнять операции дискового ввода-вывода параллельно.
- Приобретение высокоскоростных дисков в качестве дополнения или замены существующих.
- Добавление дисковых адаптеров для подключения существующих и/или новых дисков.
- Установка дополнительной оперативной памяти и увеличение параметров VMM *minperm* и *maxperm* для повышения эффективности использования кэша, расположенного в оперативной памяти.

Для получения более подробных рекомендаций, учитывающих особенности конфигурации и задачи, воспользуйтесь программой оценки объема необходимых ресурсов, например, BEST/1.

Использование RAID

Резервный массив независимых дисков (RAID) представляет собой способ повышения коэффициента доступности данных путем применения массивов дисков и различных алгоритмов распределения данных по дискам.

Дисковый массив - это группа дисковых устройств, совместное применение которых позволяет обеспечить более высокую производительность операций ввода-вывода и скорость передачи данных, чем при использовании одиночных устройств большого объема. Массив представляет собой набор дисков, управляемых специальным контроллером (контроллером массива), который обеспечивает правильное распределение данных по отдельным дискам. Данные каждого файла записываются не на один диск, а разбиваются на сегменты и распределяются по нескольким дискам массива.

Кроме того, массивы обеспечивают резервирование данных, обеспечивая их сохранение даже в случае сбоя одного из дисков. В зависимости от уровня RAID, данные могут сохраняться на дисках с зеркальным копированием или с чередованием данных.

Массив может включать в себя подмассивы. В зависимости от настройки, подсистема может содержать один или несколько подмассивов, называемых также логическими устройствами (LUN). Каждый LUN характеризуется собственным набором параметров (уровень RAID, размер логического блока, логического устройства и т.п.). С точки зрения операционной системы каждый подмассив представляет собой отдельный диск с собственным именем.

Алгоритмы поддержки RAID могут быть реализованы на уровне файловой системы или на уровне драйвера диска (последний подход применяется в RAID 0 и RAID 1). Выполнять такие алгоритмы может встроенный микропроцессор аппаратного адаптера RAID. Аппаратные адаптеры RAID как правило обеспечивают более высокую скорость работы, чем программные средства поддержки RAID, поскольку встроенный процессор уменьшает нагрузку на центральный процессор. Кроме того, в аппаратных адаптерах могут быть реализованы аппаратные схемы передачи и обработки данных.

Опции RAID, поддерживаемые LVM

В AIX LVM поддерживает три опции RAID.

Элемент	Описание
RAID 0	Чередование
RAID 1	Зеркальная защита
RAID 10 или 0+1	Зеркальная защита и чередование

Использование кэша быстрой записи

Кэш быстрой записи (FWC) - это дополнительный энергонезависимый кэш, дублирующий обычный кэш адаптера. FWC отслеживает операции, при которых данные не были записаны на диск.

Кэш быстрой записи позволяет существенно сократить время отклика при выполнении операций записи. Однако следует помнить о том, что нельзя допускать переполнения кэша, когда запросы на запись поступают быстрее, чем кэш может из сохранять. Это отрицательно влияет на максимальную частоту выполнения операций ввода-вывода, поскольку требует выполнения дополнительных операций адаптером для определения того, находятся ли передаваемые данные в кэше.

Кэш быстрой записи обеспечивает существенное повышение производительности при выполнении определенных классов задач, например, при копировании базы данных на новый набор дисков. Если кэш быстрой записи используется несколькими адаптерами, то выигрыш в производительности будет еще больше.

Благодаря перечисленным ниже особенностям протокола JFS, кэш FWC уменьшает нагрузку, связанную с ведением протокола:

1. Ведение протоколов JFS требует выполнения большого числа операций записи. В кэш FWC заносятся только измененные данные.
2. Операции записи выполняются часто и небольшими блоками. Поскольку объем кэша невелик, то максимальная производительность достигается при объединении запросов адаптером и выводе данных большими блоками. Обработка запросов на ввод-вывод больших блоков данных выполняется быстрее, так как для записи всех данных требуется меньше оборотов диска.
3. Размер протоколов обычно невелик по сравнению с размером кэша, поэтому регистрация не приводит к частой перезаписи кэша. Таким образом, повысить производительность за счет перезаписи данных, находящихся в кэше, не удается. Несмотря на то, что применение других типов контроллеров с кэшем записи также позволяет повысить эффективность регистрации, в этом разделе рассматриваются только вопросы производительности, связанные с кэшем FWC.

Если производительность дисковой подсистемы ограничена пропускной способностью отдельного дискового устройства, то одним из решений может быть распределение данных логического тома по нескольким устройствам RAID 5. Размер блоков чередования составляет 64 КБ, умноженный на число дисков данных RAID 5. Если адаптер настроен на применение RAID 5, то запись блоков данных, больших или равных по размеру сегменту чередования, выполняется без обращения к кэшу. Именно поэтому операции записи блоков по 128 КБ в массив 2+p с кэшем FWC выполняется медленней, чем запись блоков по 127 КБ и с той же скоростью, что запись блоков по 128 КБ в массив 2+p без кэша FWC. Такой подход позволяет избежать частой "промывки" кэша при выполнении последовательного ввода-вывода больших объемов данных.

Функции быстрого переключения при сбое ввода-вывода устройств Fibre Channel

AIX поддерживает функцию быстрого переключения при сбое ввода-вывода устройств Fibre Channel после событий связи в коммутируемой среде.

Если драйвер адаптера FC обнаруживает событие связи, например, потерянное соединение между устройством хранения и коммутатором, то драйвер ожидает в течение короткого промежутка времени (приблизительно 15 секунд) стабилизации работы маршрутизатора. Если в этот момент драйвер адаптера

FC обнаруживает, что устройство не подключено к маршрутизатору, он начинает отказывать всем операциям ввода-вывода. Любые новые попытки ввода-вывода или последующие повторы неудачных операций ввода-вывода отменяются адаптером немедленно, до тех пор, пока устройство не будет снова обнаружено драйвером адаптера в сети.

>Функция быстрого переключения управляется новым атрибутом устройств **fscsi** - **fc_err_recov**. По умолчанию этому атрибуту присвоено значение `delayed_fail` - поведение при сбое ввода-вывода, используемое в предыдущих версиях AIX. Для включения функции быстрого переключения при сбое ввода-вывода задайте для этого атрибута значение `fast_fail`, как показано в следующем примере:

```
chdev -l fscsi0 -a fc_err_recov=fast_fail
```

В этом примере экземпляр устройства **fscsi** - `fscsi0`. Логическая схема быстрого переключения при сбое вступает в силу, когда драйвер адаптера получает от коммутатора уведомление RSCN о происшедшем событии, связанном с соединением и касающемся удаленного порта устройства для хранения данных.

Функция быстрого переключения при сбое ввода-вывода полезна при использовании ПО с альтернативными путями. Задав для атрибута **fc_err_recov** значение `fast_fail`, можно уменьшить время сбоя ввода-вывода при потере соединения между устройством хранения и коммутатором. Это уменьшит время переключения на альтернативные пути при сбое.

В конфигурациях с одним путем, особенно конфигурациях с одним путем к устройству подкачки, рекомендуется использовать значение по умолчанию `delayed_fail`.

Для работы функции быстрого переключения при сбое ввода-вывода необходимо выполнение следующих условий:

- Коммутируемая среда. Она не поддерживается в среде кольца с арбитражной логикой, включая внешнее кольцо.
- Встроенное ПО адаптера FC 6227, уровень 3.22A или выше.
- Встроенное ПО адаптера FC 6228, уровень 3.82A или выше.
- Встроенное ПО адаптера FC 6239, всех уровней.
- Все последующие выпуски адаптеров FC поддерживают быстрый сбой ввода-вывода.

Если какое-либо из этих требований не выполняется, то устройством **fscsi** составляется протокол ошибки типа `INFO`, указывающий на несоблюдение одного из описанных выше условий и сообщающий о том, что функция быстрого переключения при сбое ввода-вывода не работает.

Отдельные устройства FC поддерживают включение и выключение быстрого переключения при сбое ввода-вывода в состоянии **Доступно**. Для того чтобы проверить, поддерживает ли устройство динамическое отслеживание, выполните команду **lsattr**. Функцию быстрого переключения при сбое ввода-вывода можно изменить на поддерживающих устройствах без повторной настройки устройства или перезапуска соединения. Изменение следует запрашивать, когда маршрутизатор сети хранения данных (SAN) находится в стабильном состоянии. Запрос не выполняется, если во время его обработки в SAN активно восстановление после ошибок.

Информация, связанная с данной:

Команда lsattr

Функции динамического отслеживания Fibre Channel

AIX поддерживает динамическое отслеживание устройств Fibre Channel.

В предыдущих версиях AIX пользователь должен был удалять конфигурацию запоминающих устройств и экземпляров адаптеров FC перед изменением параметров системной сети (SAN), которые могли привести к изменению ИД N_Port (ИД SCSI) портов удаленного хранилища.

Если включено динамическое отслеживание устройств FC, то драйвер адаптера обнаруживает изменения ИД N_Port устройства Fibre Channel. После этого драйвер адаптера FC перенаправляет трафик, предназначенный для этого устройства, на новый адрес, при условии, что устройство активно. События, которые могут изменить ИД N_Port, включают один из следующих сценариев:

- Переключение кабеля, соединяющего коммутатор и запоминающее устройство, из одного порта коммутатора в другой.
- Соединение двух различных коммутаторов с помощью межкоммутаторного канала (ISL).
- Перезагрузка коммутатора.

Динамическое отслеживание FC управляется атрибутом `dyntrk` устройства **fscsi**. По умолчанию для этого атрибута указывается значение `no`. Для включения динамического отслеживания устройств FC задайте для этого атрибута значение `dyntrk=yes`, как показано в следующем примере.

```
chdev -l fscsi0 -a dyntrk=yes
```

В этом примере экземпляр устройства **fscsi** - `fscsi0`. Логика динамического мониторинга вызывается, если драйвер адаптера получает указание от коммутатора, что имело место событие соединения, с использованием порта удаленного запоминающего устройства.

Для поддержки динамического мониторинга требуется следующая конфигурация:

- Коммутируемая среда. Он не поддерживается в среде кольца с арбитражной логикой, включая внешнее кольцо.
- Встроенное ПО адаптера FC 6227, уровень 3.22A или выше.
- Встроенное ПО адаптера FC 6228, уровень 3.82A или выше.
- Встроенное ПО адаптера FC 6239, всех уровней.
- Все последующие выпуски адаптеров FC поддерживают быстрый сбой ввода-вывода.
- Устройства глобального имени (имени порта) и имен узлов должны оставаться постоянными, и устройство глобального имени должно быть уникальным. Изменение глобального имени или имени узла доступного или активного устройства может вызвать ошибки ввода-вывода. Также, каждый экземпляр запоминающего устройства FC должен иметь атрибуты **world_wide_name** и **node_name**. Обновленные наборы файлов, содержащие атрибут **sn_location** (см. следующий пункт) также должны быть обновлены для того, чтобы содержать оба этих атрибута.
- Запоминающее устройство должно предоставлять надежный способ извлечения уникального серийного номера для каждого LUN. Драйверы устройств FC AIX не могут автоматически определить расположение серийного номера. Метод извлечения серийного номера должен быть задан явно поставщиком устройства для поддержки динамического отслеживания таких устройств. Данная информация передается драйверам с помощью атрибута администратора объектных данных (ODM) **sn_location** для каждого запоминающего устройства. Если драйвер диска или магнитной ленты определяет, что атрибут администратора объектных данных (ODM) **sn_location** отсутствует, создается протокол ошибки типа `INFO` и динамическое отслеживание не включается.

  Примечание: При выполнении команды **lsattr** для жесткого диска атрибут **sn_location** может не отображаться. Т. е. имя атрибута не отображается, даже если оно существует в ODM.
- Драйверы FC могут отслеживать устройства в *зоне SAN*, если значения ИД N_Port в зоне стабильны в течение 15 секунд. *Зона SAN* - это зона, доступная адаптера шины одиночного хоста. Если кабели не переустанавливаются, или ИД N_Port продолжают изменятся после окончания 15-ти секунд, могут возникнуть ошибки ввода-вывода.
- Устройства не отслеживаются на разных адаптерах шины хоста. Устройства могут отлеживаться, если они остаются видимыми из того же адаптера шины хоста, к которому они были первоначально подключены.

  Например, если устройство A перемещается из одного места в другое в маршрутизаторе A, подключенном к адаптер шины хоста A (это значит, что его N_Port на маршрутизаторе A изменяется), то устройство продолжает отслеживаться без вмешательства пользователя и продолжает ввод-вывод.

Однако, если устройство A видимо из адаптера шины хоста A, но не видимо из адаптера шины хоста B, и устройство A перемещается из маршрутизатора, соединенного с адаптером шины хоста A в маршрутизатор, подключенный к адаптеру шины хоста B, устройство A станет недоступно в маршрутизаторе A и в маршрутизаторе B. Потребуется вмешательство пользователя для того, чтобы оно стало доступно в маршрутизаторе B. Необходимо выполнить команду **cfgmgr**. Экземпляр устройства AIX в маршрутизаторе A недоступен, и необходимо создать экземпляр устройства в маршрутизаторе B. Это устройство необходимо добавить вручную к группам томов, экземплярам устройств с альтернативными путями и т.д. Эта процедура выполняется аналогично удалению устройства из маршрутизатора A и добавлению его в маршрутизатор B.

- Нельзя выполнять динамическое отслеживание устройств дампа FC во время выполнения дампа памяти системы AIX. Также, динамическое отслеживание не поддерживается при загрузке или выполнении команды **cfgmgr**. Изменения в SAN нельзя выполнять во время любой из этих операций.
- После отслеживания устройства в ODM может содержаться неактуальная информация. Возможная причина: ИД SCSI в ODM не соответствуют фактическим ИД SCSI в SAN. ODM остается в этом состоянии до выполнения команды **cfgmgr** вручную или до перезагрузки системы (в том случае, если все драйверы, включая целевые драйверы FC SCSI других фирм, поддерживают динамическое отслеживание). Если команда **cfgmgr** выполняется вручную, то ее необходимо выполнить на всех участвующих устройствах **fscsi**. Для этого можно выполнить команду **cfgmgr** без опций или **cfgmgr** на каждом устройстве **fscsi**.

  Примечание: Выполнение **cfgmgr** во время выполнения рабочей среды для рекалибровки ИД SCSI может не обновить ИД SCSI в администраторе объектных данных (ODM) для устройства хранения. Это справедливо, если устройство памяти открыто (группы томов включены). Команду **cfgmgr** необходимо выполнять на устройствах, которые не открыты, или для рекалибровки идентификаторов устройств SCSI необходимо перезапустить систему. Неактуальные идентификаторы устройств SCSI в администраторе объектных данных (ODM) не вызывают ошибок драйверов FC, и для нормальной работы рекалибровка не обязательна. В то же время, все приложения, которые подключаются к драйверу адаптера напрямую с помощью команд **ioctl** и при этом обращаются к значениям ИД SCSI, указанным в ODM, нуждаются в обновлении (см. следующий пункт), во избежание использования устаревших ИД SCSI.
- Все приложения и расширения ядра, соединяющиеся с этим драйвером адаптера FC с помощью вызовов **ioctl** или напрямую с точками входа драйвера FC, должны поддерживать API **ioctl** и **scsi_buf** драйвера адаптера FC версии 1 для правильной работы с динамическим отслеживанием FC. Приложения и расширения ядра могут не работать правильно или выдавать ошибки во время событий динамического отслеживания, если это условие не выполняется. Если драйвер адаптера FC обнаруживает приложение или расширение ядра, не поддерживающее версию 1 API **ioctl** и **scsi_buf**, создается протокол ошибки типа `INFO`, и динамическое отслеживание не включается для целевого устройства.

  Информация о независимых поставщиках программного обеспечения, занимающихся разработкой расширений ядра или приложений, взаимодействующих со стеком драйвера Fibre Channel AIX, содержится в разделе Обязательные команды ioctl для FCP, iSCSI и драйвера виртуального адаптера клиента SCSI (Required FCP, iSCSI, and Virtual SCSI Client Adapter Device Driver ioctl Commands). Сведения об изменениях, необходимых для поддержки функции динамического отслеживания, содержатся в разделе Общие сведения о структуре scsi_buf.
- Даже при включенном динамическом отслеживании, пользователи должны производить изменения в SAN, такие как перемещение кабелей и установка соединений ISL только при активных окнах обслуживания. Изменять SAN в рабочем режиме не рекомендуется, потому что интервалы времени для выполнения изменений слишком малы. Например, кабели, не установленные заново правильно, вызывают ошибку ввода-вывода. Выполнение этих операций в промежутки, когда потоки данных отсутствуют или незначительны, позволяет свести к минимуму вероятность сбоев ввода-вывода.
- Динамическое отслеживание в разделах AIX предоставляет программное обеспечение для конкретной цели - восстановления событий конфигурации из SAN. События настройки выполняются во время операций перемещения LPAR. При подготовке в событию переноса или во время его стратегия AIX используется для предотвращения простоев в обслуживании. Поэтому динамическое отслеживание для адаптеров клиента FC всегда включено и не может быть выключено.

Базовые драйверы дисков, магнитных лент и FastT AIX FC SCSI поддерживают динамическое отслеживание. Запоминающие устройства IBM ESS, EMC Symmetrix, и HDS поддерживают динамическое отслеживание в случае, если вендор поставляет наборы файлов администратора объектных данных (ODM) вместе с необходимыми атрибутами **sn_location** и **node_name**. Обратитесь к вендору устройства хранения, если вы не знаете, поддерживает ли набор файлов администратора объектных данных (ODM) динамическое отслеживание.

Если входы администратора объектных данных (ODM), соответствующие вендору, не используются для запоминающего устройства, но подсистемы хранения ESS, Symmetrix или HDS настроены с сообщением `MPIO Other FC SCSI Disk`, то для устройств с данной конфигурацией поддерживается динамическое отслеживание. Это отменяет потребность в атрибуте **sn_location**. Все текущие модули управления путями AIX, поставляемые вместе с базовым вариантом AIX,поддерживают динамическое отслеживание.

Накопитель на магнитной ленте STK, использующий стандартный драйвер устройства AIX, также поддерживает динамическое отслеживание, если набор файлов STK содержит необходимые атрибуты **sn_location** и **node_name**.

Примечание: Изменения SAN, затрагивающие накопитель на магнитной ленте должны производиться при отсутствии операций ввода-вывода. Одна ошибка ввода-вывода накопителя на магнитной ленте может вызвать сбой резервного копирования.

Устройства, настройка которых была выполнена с указаниями `Другой диск FC SCSI` или `Другой накопитель на магнитной ленте FC SCSI`, не поддерживают функцию динамического отслеживания.

Отдельные устройства FC поддерживают включение и выключение динамического отслеживания в состоянии **Доступно**. Для того чтобы проверить, поддерживает ли устройство динамическое отслеживание, выполните команду **lsattr**. Динамическое отслеживание можно изменить на поддерживающих устройствах без повторной настройки устройства или перезапуска соединения. Изменение следует запрашивать, когда маршрутизатор сети хранения данных (SAN) находится в стабильном состоянии. Запрос не выполняется, если во время его обработки в SAN активно восстановление после ошибок. Запросы на изменение не выполняются, если связанные устройства, такие как диски и накопители на магнитной ленте, не поддерживают изменения.

Информация, связанная с данной:

Обязательные адаптеры FCP, iSCSI, и Virtual SCSI Client, Общие сведения о структуре scsi_buf

Команда lsattr

Взаимодействие быстрого переключения и динамического отслеживания ошибок ввода-вывода

Хотя взаимодействие быстрого переключения и динамического отслеживания ошибок ввода-вывода устройств FC - это технически две разные функции, включение одной из них вызывает изменение интерпретации другой в определенных ситуациях. В следующей таблице приведены различные поведения драйверов FC в зависимости от комбинаций этих параметров.

dyntrk	fc_err_recov	Поведение драйвера
нет	`delayed_fail`	Значение по умолчанию. Данное поведение существовало в предыдущих версиях AIX. Драйверы FC не восстанавливаются при изменении идентификатора устройства SCSI, а также ввод-вывод продолжается дольше без ошибок при обрыве связи между удаленным портом хранения и коммутатором. Это может быть предпочтительно в ситуациях с одним путем, если не требуется динамическое отслеживание.

dyntrk	fc_err_recov	Поведение драйвера
нет	fast_fail	Если драйвер получает уведомление об изменении зарегистрированного состояния (RSCN) от коммутатора, это может означать разрыв соединения между удаленным портом хранения и коммутатором. После 15-секундной задержки, драйверы FC запрашивают состояние устройства на маршрутизаторе. Если ответа нет, адаптер производит сброс вводов-выводов. Если устройство не подключено к маршрутизатору, повторные попытки и операции ввода-вывода вызовут ошибку. Если драйверы FC обнаруживают, что изменился идентификатор устройства SCSI, то драйверы не восстанавливаются и операция ввода-вывода завершается с ошибкой PERM.
да	delayed_fail	Если драйвер получает уведомление об изменении зарегистрированного состояния (RSCN) от коммутатора, это может означать разрыв соединения между удаленным портом хранения и коммутатором. После 15-секундной задержки, драйверы FC запрашивают состояние устройства на маршрутизаторе. Если ответа нет, адаптер производит сброс вводов-выводов. Если устройство не подключено к маршрутизатору, повторные попытки и операции ввода-вывода вызовут ошибку, хотя драйвер хранения (диск, лента, FastT) может внести задержку 2-5 секунд для повторных попыток ввода-вывода. Если драйверы FC обнаруживают, что изменился идентификатор устройства SCSI, то драйверы устройства FC перенаправляют трафик на новый идентификатор устройства SCSI.
да	fast_fail	Если драйвер получает уведомление об изменении зарегистрированного состояния (RSCN) от коммутатора, это может означать разрыв соединения между удаленным портом хранения и коммутатором. После 15-секундной задержки, драйверы FC запрашивают состояние устройства на маршрутизаторе. Если ответа нет, адаптер производит сброс вводов-выводов. Если устройство не подключено к маршрутизатору, повторные попытки и операции ввода-вывода вызовут ошибку. Драйвер устройства хранения (диск, лента, FastT) не будет делать задержек между попытками. Если драйверы FC обнаруживают, что изменился идентификатор устройства SCSI, то драйверы устройства FC перенаправляют трафик на новый идентификатор устройства SCSI.

Если динамическое отслеживание выключено, настройкиdelayed_fail и fast_fail атрибута **fc_err_recov** имеют большое значение. При включенном динамическом отслеживании настройка атрибута **fc_err_recov** менее важна. Причина этого состоит в том, что существует частичное совпадение стратегий динамического отслеживания и быстрого переключения после сбоев. Т.о., включение динамического отслеживания частично включает логику быстрого переключения.

Процедура восстановления при недоступности устройства на маршрутизаторе одна и та же для обоих настроек fc_err_recov при включенном динамическом отслеживании. Разница заключается в том, что драйверы могут делать задержки между попытками ввода-вывода, если для **fc_err_recov** установлено значение delayed_fail. Это ведет к дополнительному увеличению промежутка времени до сбоя ввода-вывода, в зависимости от длительности задержки и количества повторных попыток перед окончательным сбоем ввода-вывода. Тем не менее, при напряженном потоке ввода-вывода разница между значениями delayed_fail и fast_fail может быть более заметной.

Администратор SAN может использовать различные комбинации для получения наилучшей производительности.

Модульный ввод-вывод

Библиотека модульного ввода-вывода позволяет анализировать и настраивать ввод-вывод на уровне приложения для получения оптимальной производительности.

Обычно приложения предоставляют минимум возможностей по оптимизации производительности своих операций ввода-вывода. Это вынуждает полагаться на механизмы операционной системы по настройке ввода-вывода. Чаще всего в данной системе работает несколько приложений с противоречивыми требованиями для получения высокой производительности ввода-вывода, поэтому приходится находить компромисс между ними. Библиотека модульного ввода-вывода позволяет оптимизировать ввод-вывод для каждого приложения отдельно.

Рекомендации и замечания

Работа с MIO может быть очень полезна, однако при этом необходимо выполнять определенные рекомендации.

Преимущества

- Простота реализации MIO заметно упрощает анализ ввода-вывода приложений.
- MIO позволяет кэшировать ввод-вывод на уровне приложений: можно оптимизировать системный вызов ввода-вывода, а затем системные прерывания.
- Кэш pf можно настроить для файла или группы файлов, что обеспечивает большую гибкость, чем кэш операционной системы.
- MIO можно использовать для ввода-вывода приложений, работающих параллельно, причем часть из них можно связать с MIO и настроить на работу с кэшем pf и прозрачным вводом-выводом в обход стандартного кэша JFS и JFS2. Приложения, связанные с MIO, не будут загружать кэш операционной системы, который будет доступен другим приложениям.
- Кэш MIO особенно полезен для последовательного доступа к большим файлам.

Замечания

- Неверная конфигурация кэша библиотеки MIO может привести к снижению производительности. Для того чтобы избежать этого, сначала проанализируйте потребности ввода-вывода своего приложения, затем подберите параметры модуля, которые наилучшим образом отвечают задаче и позволяют повысить быстродействие. Ниже приведены примеры неверного использования MIO:
  - Для приложений, обращающихся к файлам, меньшим по размеру чем объем памяти операционной системы, опция **direct** модуля **pf** приведет к снижению производительности.
  - Быстродействие может снизиться при доступе к файлам в случайном порядке.
- Кэш MIO размещается подсистемой malloc в адресном пространстве приложения, поэтому не следует делать размер кэша большим, чем объем памяти ОС, так как в этом случае будет задействовано пространство подкачки. Это может привести к снижению производительности или сбою операционной системы.

Архитектура MIO

Библиотека модульного ввода-вывода (MIO) состоит из пяти модулей ввода-вывода, с возможностью вызова их для каждого файла отдельно.

Доступные модули:

- Модуль *mio* обеспечивает взаимодействие с программой пользователя.
- Модуль *pf* обеспечивает предварительную загрузку данных.
- Модуль *trace* обеспечивает сбор статистики.
- Модуль *recov* обеспечивает анализ ошибки ввода-вывода и повтор действий в случае сбоя.
- Модуль *aix* обеспечивает взаимодействие с операционной системой.

Модули по умолчанию - *mio* и *aix*; остальные модули являются необязательными.

Оптимизация ввода-вывода и модуль pf

Модуль *pf* - это кэш в пользовательском пространстве, использующий простой механизм обращения к последним использовавшимся страницам (LRU, Last Recently Used). Модуль *pf* также отслеживает обращения к страницам кэша для прогноза будущих запросов к данным файла и вызывает команды **aio_read** для предварительной загрузки данных в кэш.

Часто приложения считывают последовательно большие (размером в десятки гигабайт) файлы. Таким приложениям кэш операционной системы помогает очень мало. Большие буферы операционной системы в данном случае бесполезны, потому что повторное использование данных не выполняется. Библиотека MIO в

таких ситуациях вызывает модуль *pf*, который обнаруживает режим последовательного доступа и заранее асинхронно считывает в кэш меньшего размера необходимые данные. Размер кэша *pf* должен быть достаточным для обеспечения опережающего чтения. Кроме того, модуль *pf* может применять прозрачный ввод-вывод во избежание лишних операций копирования из памяти в системный буфер, тем самым освобождая системный буфер для других операций. Эксперименты с файловой системой JFS или JFS2 в AIX показали эффективность прозрачного ввода-вывода модуля *pf* для последовательного чтения больших файлов.

Реализация MIO

Модульный ввод-вывод может быть реализован одним из трёх способов: перенаправление компоновкой с **libtkio**, перенаправление включением **libmio.h** и явное использование вызовов функций MIO.

Любой вариант реализации несложный, но рекомендуется вариант компоновки с **libtkio**.

Перенаправление компоновкой с библиотекой tkio

Библиотека Trap Kernel I/O (**tkio**) поставляется с пакетом **libmio** и предназначена для упрощения реализации оптимизации модульного ввода-вывода в приложении.

Для реализации модульного ввода-вывода можно с помощью переменной среды *TKIO_ALTLIB* настроить базовый ввод-вывод на уровне ядра таким образом, чтобы стало возможным замещать стандартные вызовы вызовами из другой библиотеки: `setenv TKIO_ALTLIB "libmio.a(get_mio_ptrs.so)"`

При этом стандартный общий объект заменяется на общий объект MIO (*get_mio_ptrs.so*), возвращающий структуру с указателями на все функции ввода-вывода MIO. Загрузка выполняется однажды, для первого системного вызова, а затем все вызовы ввода-вывода перенаправляются библиотекой libtkio на библиотеку MIO.

Этот способ реализации модульного ввода-вывода рекомендуется потому, что при сбое загрузки или вызова функции **libtkio** восстановит структуру с указателями на стандартные функции ввода-вывода. Кроме того, если приложение не работает с модульным вводом-выводом, его можно отключить просто не задавая переменную *TKIO_ALTLIB*.

Перенаправление с помощью libmio.h

Это способ реализации модульного ввода-вывода требует добавления двух строк в исходный код приложения.

Реализовать MIO можно включением макрокоманды C *USE_MIO_DEFINES*, определяющего набор других макрокоманд в заголовочном файле **libmio.h**, выполняющих перенаправление вызовов ввода-вывода на библиотеку MIO. Файл **libmio.h** поставляется с пакетом **libmio** и содержит следующие операторы **#define**:

```
#define open64(a,b,c) MIO_open64(a,b,c,0)
#define close MIO_close
#define lseek64 MIO_lseek64
#define ftruncate64 MIO_ftruncate64
#define fstat64 MIO_fstat64
#define fcntl MIO_fcntl
#define ffinfo MIO_ffinfo
#define fsync MIO_fsync
#define read MIO_read
#define write MIO_write
#define aio_read64 MIO_aio_read64
#define aio_write64 MIO_aio_write64
#define aio_suspend64 MIO_aio_suspend64
#define lio_listio MIO_lio_listio
```

1. Для реализации MIO этим способом добавьте дву строки в исходный код:

   ```
   #define USE_MIO_DEFINES
   #include "libmio.h"
   ```

2. Скомпиируйте приложение.

Явный вызов процедур MIO
В реализации MIO предусмотрен прямой вызов процедур MIO.

Вместо включения заголовочного файла `libmio.h`, в котором условия **#define** перенаправляют ввод-вывод в библиотеку MIO, вы можете добавить записи **#define** непосредственно в исходный код приложения и затем перекомпилировать приложение.

Переменные среды MIO
Для контроля поведения модульного ввода-вывода предусмотрено четыре переменных среды.

MIO_STATS
MIO_STATS содержит путь к файлу, в который следует записывать диагностические сообщения и вывод модулей MIO.

Значение переменной интерпретируется как имя файла всегда, кроме двух случаев. Если оно равно stdout или stderr, то данные выводятся на стандартный вывод или стандартный вывод для ошибок соответственно. Если имя файла начинается со знака сложения (+), например, *+filename.txt*, то данные будут добавляться в конец файла без потери уж имеющихся в нём.

MIO_FILES
MIO_FILES позволяет определить, какие модули вызываются для данного файла при вызове **MIO_open64**.

Формат MIO_FILES:

```
первый-список-имён-файлов [ список-модулей ] второй-список-имён-файлов [ список-модулей ] ...
```

При вызове **MIO_open64** MIO проверяет существование переменной среды *MIO_FILES*. Если она существует, то модули, вызываемые для разных файлов, определяются на основе её значения.

MIO_FILES анализируется слева направо. Все символы, предшествующие левой скобке ([), интерпретируются как *список-имён-файлов*. *список-имён-файлов* - это список шаблонов *шаблон-имени-файла*, разделенных двоеточием (:). При открытии файлов их имена проверяются на соответствие *шаблонам имени файлов*, в последних можно использовать следующие символы подстановки:

- Звёздочка (*) соответствует любому количеству символов в имени файла или папки, в т.ч. нулю.
- Вопросительный знак (?) соответствует одному символу в имени файла или папки.
- Две звёздочки (**) соответствуют остальным символам полного имени файла.

Если *шаблон имени файла* не содержит косой черты (/), то каталог файла или папки, переданный **MIO_open64**, игнорируется и сопоставление выполняется только по собственно имени файла или папки.

Если имя файла соответствует одному из *шаблонов имён файлов* в *списке имён файлов*, то вызываются все модули из списка, указанного в прямых скобках после этого *списка имён файлов*. В противном случае анализатор переходит ко второму *списку имен файлов* и сопоставляет имя файла с его элементами. Если имя файла соответствует более чем одному шаблону, используется только первый шаблон. Если имя файла не соответствует ни одному из *шаблонов имен файлов* из всех *списков имен файлов*, то будет вызван модуль **aix**. При наличии совпадения вызываются модули из связанного списка, указанного в переменной среды *MIO_FILES*. Модули вызываются слева направо (самый левый наиболее близок к пользовательской программе, а самый правый - к операционной системе). Если список модулей не начинается с модуля **mio**, то в переменную среды добавляется префикс параметра по умолчанию модуля **mio**. Если в списке не указан модуль **aix**, то в переменную среды добавляется параметр по умолчанию модуля **aix**.

Ниже приведен простой пример обработки переменной *MIO_FILES*:

```
MIO_FILES= *.dat:*.scr [ trace ] *.f01:*.f02:*.f03 [ trace | pf | trace ]
```

Функция **MIO_open64** открывает файл `test.dat` и сравнивает его имя с шаблоном ***.dat**, в результате чего вызываются модули **mio**, **trace** и **aix**.

Функция **MIO_open64** открывает файл `test.f02` и сравнивает его имя с ***.f02** - вторым *шаблоном имени файла* из второго *списка имен файлов*, в результате чего вызываются модули **mio**, **trace**, **pf**, **trace** и **aix**.

Каждый модуль имеет свои собственные жестко запрограммированные опции по умолчанию для вызова по умолчанию в переменной среды. Их можно переопределить, указав значения в связанном списке модулей *MIO_FILES*. Следующий пример кода включает статистику для модуля **trace** и перенаправляет ее вывод в файл `my.stats`:

```
MIO_FILES= *.dat : *.scr [ trace/stats=my.stats ]
```

Как видно, опции для модуля указываются после косой черты (/). Для некоторых опций требуется указать целое или строковое значение. Если строковое значение содержит косую черту (/), его необходимо заключить в фигурные скобки {}. К целому значению опции можно добавить букву k, m, g или t, обозначающую килобайты, мегабайты, гигабайты или терабайты. Целые значения можно вводить в десятеричном, восьмеричном или шестнадцатеричном формате. Если целое значение содержит префикс 0x, то оно интерпретируется как шестнадцатеричное число. Если целое значение содержит префикс 0, то оно интерпретируется как восьмеричное число.

MIO_DEFAULTS

Переменная среды *MIO_DEFAULTS* предназначена для улучшения удобочитаемости переменной *MIO_FILES*.

Если пользователь укажет по несколько модулей для нескольких *списков имён файлов*, переменная *MIO_FILES* станет очень длинной. Вместо последовательного переопределения значений по умолчанию для большинства файлов можно задать новые значения по умолчанию с помощью переменной *MIO_DEFAULTS*. Она представляет собой список модулей (через запятую) и новых значений по умолчанию для них, например:

```
MIO_DEFAULTS = trace/events=prob.events , aix/debug
```

Теперь по умолчанию при вызове модуля **trace** трассировка событий работы с двоичными данными включена и её вывод сохраняется в файле `prob.events`, а при вызове модуля **aix** будет включена отладка (**debug**).

MIO_DEBUG

Переменная среды *MIO_DEBUG* предназначена для отладки MIO.

Переменная *MIO_DEFAULTS* определяет события, для которых следует сообщать отладочную информацию. Она может содержать следующие ключевые слова:

ALL Включить все возможные ключевые слова *MIO_DEBUG*.

ENV Сообщать о запросах на сопоставление с данными из переменных среды.

OPEN Сообщать о запросах на открытие посредством функции **MIO_open64**.

MODULES
Сообщать о запускаемых модулях для каждого вызова функции **MIO_open64**.

TIMESTAMP
Добавлять в начало каждой строки файла статистики системное время.

DEF Выводить таблицу определений для каждого модуля. Она выполняется для всех модулей библиотеки MIO при открытии файла.

Опции модулей

У каждого модуля MIO имеются различные опции для анализа и оптимизации производительности на уровне приложения.

Опции модуля MIO

Модуль **mio** обеспечивает взаимодействие с программой пользователя MIO и по умолчанию запускается во время выполнения.

mode Переопределять режим доступа к файлу при открытии.

Этот режим передаётся в качестве параметра системному вызову AIX open: он заменяет начальный режим, задаваемый в исходном коде.

nomode
Не переопределять режим доступа к файлу. Этот вариант используется по умолчанию.

direct Включить бит **O_DIRECT** в флаг для open.

nodirect
Исключить бит **O_DIRECT** из флага для open.

osync Включить бит **O_SYNC** в флаг для open.

noosync
Исключить бит **O_SYNC** из флага для open.

Определения опции модуля TRACE

Модуль **trace** является модулем сбора статистики для пользовательской программы MIO и не является обязательным модулем.

stats{=output_file}
Статистика вывода при закрытии: имя файла для диагностики вывода трассировки.

Если не указан *файл_вывода*, или если файлом вывода является заданный по умолчанию файл**mioout**, модуль **trace** ищет файл статистики вывода, заданный переменной среды *MIO_STATS*.

nostats Не вести статистику вывода.

events{=event_file}
Создать двоичный файл событий. Значение по умолчанию: `trace.events`.

noevents
Не создавать двоичный файл событий. Эта опция применяется по умолчанию.

bytes Статистика вывода в блоках (байтах). По умолчанию размер блока равен байту.

kbytes Статистика вывода в блоках (килобайтах).

gbytes Статистика вывода в блоках (гигабайтах).

tbytes Статистика вывода в блоках (терабайтах).

inter Промежуточная статистика вывода.

nointer
Не вести промежуточную статистику вывода. Эта опция применяется по умолчанию.

Опции модуля PF

Модуль **pf** предназначен для предварительной загрузки данных для пользовательской программы MIO и не является обязательным.

pffw Выполнять предварительную загрузку данных также в режиме записи.

nopffw Не выполнять предварительную загрузку данных в режиме записи. Этот вариант используется по умолчанию.

По умолчанию страницы не кэшируются модулем **pf** если вызов, приводящий к предварительной загрузке, поступил при записи пользователем данных в кэш. Такое поведение выбрано потому, что не имеет смысла загружать страницу перед её заменой. Однако, если последующая пользовательская запись в страницу имеет недопустимый формат (без разделителей сектора), алгоритм, помечающий

черновые секторы станет некорректным, так что потребуется синхронная запись черновых страниц, а затем синхронное кэширование всей страницы. В этом случае лучше асинхронно загрузить заменяемую страницу, вместо синхронного попадания в кэш при непопадании страницы в кэш.

release Освобождать страницы глобального кэша при прекращении использования файла глобального кэша. Этот вариант используется по умолчанию.

norelease
Не освобождать страницы глобального кэша при прекращении использования файла глобального кэша.

Опции **release** и **norelease** определяют, должен ли освобождаться глобальный кэш при прекращении использования его файлов. По умолчанию он закрывается и освобождается. Если глобальный кэш открывается и закрывается несколько раз, это влечёт за собой фрагментацию памяти. Опция **norelease** указывает, что кэш не должен закрываться даже при прекращении использования его файлов.

private
Использовать частный кэш. Это означает, что кэш может использоваться только открывшим его файлом.

global Использовать глобальный кэш. Это означает, что кэш может использоваться несколькими файлами. Этот вариант используется по умолчанию.

Использование глобального кэша обычно лучше по нескольким причинам. Объём памяти известен, т.к. известно количество открытых кэшей и их размер. При использовании частных кэшей число активных кэшей в данных момент определить нельзя. Возможно открытие до 256 глобальных кэшей. По умолчанию опция global равна нулю, что означает открытие одного глобального кэша. Каждый открытый глобальный кэш можно присвоить определённой группе файлов.

асинхронный
Использовать асинхронные вызовы дочернего модуля. Этот вариант используется по умолчанию.

synchronous
Использовать синхронные вызовы дочернего модуля.

noasynchronous
То же, что и synchronous.

Опции **asynchronous**, **synchronous** и **noasynchronous** определяют, следует ли использовать асинхронный ввод-вывод для загрузки данных кэша с файловой системы. Может использоваться для отладки или при выключенном aio.

direct Использовать прозрачный ввод-вывод.

nodirect
Не использовать прозрачный ввод-вывод. Этот вариант используется по умолчанию.

Опции **direct** и **nodirect** определяют, следует ли включать бит O_DIRECT в флаги открытия (oflags) файла. Кэш **pf** поддерживает прозрачный ввод-вывод. Страницы кэша при этом выравниваются на 4K и генерируются запросы, гарантирующие использование прозрачного ввода-вывода.

bytes Выводить статистику в байтах. Этот вариант используется по умолчанию.

kbytes Выводить статистику в килобайтах.

mbytes
Выводить статистику в мегабайтах.

gbytes Выводить статистику в гигабайтах.

tbytes Выводить статистику в терабайтах.

cache_size
Общий размер кэша в байтах. Указание размеров в КБ, МБ, ГБ и ТБ также допустимо. Значение по умолчанию: 64k.

page_size
Размер страницы кэша в байтах. Указание размеров в КБ, МБ, ГБ и ТБ также допустимо. Значение по умолчанию: 4k.

prefetch
Число страниц, для которых следует выполнять предварительную загрузку. Значение по умолчанию: 1.

stride Коэффициент прогресса, в страницах. Значение по умолчанию: 1.

stats{=файл}
Сохранять статистику предварительной загрузки: файл для диагностики**pf**.

Если *файл* не указан или если он равен **mioout** (значение по умолчанию) модуль **pf** использует файл, указанный в переменной среды *MIO_STATS*.

nostats Не выводить статистику предварительной загрузки.

inter Сохранять промежуточную статистику предварительной загрузки при получении пользовательского сигнала (**kill -SIGUSR1**).

nointer
Не выводить промежуточную статистику предварительной загрузки. Этот вариант используется по умолчанию.

Если указана опция **inter**, то при выполнении команды **kill -30 pf** сгенерирует промежуточную статистику.

retain Не очищать данные файла при закрытии для последующего повторного открытия.

notain Очищать данные файла при закрытии. Этот вариант используется по умолчанию.

Если указана опция retain, страницы закрываемого файла будут сохраняться в глобальном кэше для повторного использования в случае открытия файла в том же кэше. Между закрытием и открытием файл не должен изменяться. При этом страницы закрытого файла учитываются демоном LRU как страницы открытого файла.

listio Использовать механизм listio.

nolistio
Не использовать механизм listio. Этот вариант используется по умолчанию.

Обычно listio используется только для отладки.

tag={метка}
Строка, помещаемая в начало статистики.

notag Не помещать в начало статистики метку. Этот вариант используется по умолчанию.

Метка - это строка, вставляемая перед записываемыми в файл данными. Когда размер файла статистики станет большим, метки упрощают поиск нужных частей файла.

scratch
Файл является вспомогательным и должен быть удалён при закрытии.

noscratch
Файл должен быть сохранён при закрытии. Этот вариант используется по умолчанию.

Считать файл вспомогательным, и, следовательно, не выгружать и удалять его при закрытии.

passthru
Байтовый диапазон некэшируемых данных.

Указанная часть файла не будет кэшироваться. Данные записываются и читаются прозрачно через кэш. Это ранее было необходимо для многопоточных приложений, записывающих данные в файл несколькими потоками и помещающих в его начало служебную информацию.

Опции модуля RECOV

Модуль **recov** анализирует сбои ввода-вывода и пытается повторить действие. Это необязательный модуль MIO.

fullwrite
Все записи должны быть полными. Если сбой записи произошёл о причине нехватки места, модуль произведёт повтор записи. Этот вариант используется по умолчанию.

partialwrite
Записи могут быть неполными. Если сбой записи произошёл о причине нехватки места, повтор записи выполняться не будет.

stats{=файл}
Файл, в который следует записывать сообщения об ошибках.

Если *файл* не указан или если он равен **mioout** (значение по умолчанию) модуль **recov** использует файл, указанный в переменной среды *MIO_STATS*

nostats Не записывать сообщения об ошибках в файл.

command
Команда, которую следует выполнить при обнаружении ошибки записи. Значение по умолчанию: `command={ls -l}`.

open_command
Команда, которую следует выполнить при обнаружении ошибки открытия вследствие закрытия соединения. Значение по умолчанию: `open_command={echo connection refused}`.

retry Число попыток повторения действия. Значение по умолчанию: 1.

Определения опция модуля AIX

Модуль **aix** - это интерфейс MIO с операционной системой, вызываемый по умолчанию во время выполнения.

debug Печатать отладочные операторы открытия и закрытия.

nodebug
Не печатать отладочные операторы открытия и закрытия. Это значение по умолчанию.

sector_size
Размер сектора. Если он не указан, применяется размер сектора файловой системы.

notrunc
Не вызывать trunc. Это требуется для устранения неполадок с JFS O_DIRECT.

trunc Вызывать trunc. Это значение по умолчанию.

Примеры использования MIO

Доступно много сценариев, связанных с библиотекой MIO.

Пример реализации MIO посредством компоновки с libtkio

Модульный ввод-вывод можно реализовать посредством компоновки приложения с libtkio для перенаправления всех вызовов ввода-вывода на библиотеку MIO.

Этот сценарий задаёт переменные среды MIO, компонует приложение с библиотекой Trap Kernel I/O (**tkio**) и вызывает его.

```
#!/bin/csh
#
setenv TKIO_ALTLIB "libmio.a(get_mio_ptrs.so)"

setenv MIO_STATS example.stats
setenv MIO_FILES " *.dat [ trace/stats ] "
setenv MIO_DEFAULTS " trace/kbytes "
setenv MIO_DEBUG OPEN
#
cc -o example example.c  -ltkio

#
./example file.dat
```

Пример реализации MIO посредством включения заголовочного файла libmio.h

Модульный ввод-вывод можно реализовать посредством включения заголовочного файла `libmio.h` в исходный код приложения, перенаправляющего все вызовы ввода-вывода на библиотеку MIO.

В файл `example.c` добавляются следующие две строки:

```
#define USE_MIO_DEFINES
#include "libmio.h"
```

Этот сценарий задаёт переменные среды MIO, компилирует и компонует приложение с библиотекой MIO и вызывает его.

```
#!/bin/csh
#
setenv MIO_STATS example.stats
setenv MIO_FILES " *.dat [ trace/stats ] "
setenv MIO_DEFAULTS " trace/kbytes "
setenv MIO_DEBUG OPEN
#
cc -o example example.c -lmio
#
./example file.dat
```

Файлы результатов диагностики MIO

При вызове функции **MIO_close** в файл статистики записываются диагностические данные для библиотеки MIO.

Если опция **stats** имеет значение по умолчанию `mioout`, имя этого файла определяется переменной среды *MIO_STATS*. В файле статистики могут содержаться следующие данные:

- Отладочная информация
- Диагностические данные, генерируемые модулем **trace** (если для него задана опция **stats**).

  Примечание: Для отключения генерации этих диагностических данных используйте опцию **nostats** модуля **trace**, а для отделения их от данных других модулей - **stats{=имя-файла}**.
- Диагностические данные, генерируемые модулем **pf** (если для него задана опция **stats**).

  Примечание: Для отключения генерации этих диагностических данных используйте опцию **nostats** модуля **pf**, а для отделения их от данных других модулей - **stats{=имя-файла}**.
- Данные трассировки восстановления (если для модуля **recov** задана опция **stats**).

  Примечание: Для отделения диагностических данных модуля **recov** данных других модулей используйте опцию **stats{=имя-файла}**.

Пример файла диагностики модуля трассировки:

Файл статистики модуля **trace** содержит данные об отладка и диагностике.

Элементы заголовка

- Дата
- Имя хоста
- Включен или отключен AIO
- Имя программы
- Версия библиотеки MIO
- Переменные среды

Элементы отладки

- Список всех настроек отладки
- Включены все таблицы определений модулей опции отладки DEF
- Включить запрос для MIO_open64, если задана отладка OPEN
- Активация модулей, если включена опция отладки MODULES

Элементы, свойственные модулю трассировки с разметкой

- Время, если опция отладки **TIMESTAMP**
- Закрытое или промежуточное прерывание трассировки
- Расположение модуля трассировки в списке module_list
- Имя обрабатываемого файла
- Скорость: отношение количества данных к интервалу времени; суммарное время потраченное на работу модуля трассировки
- Необходимая скорость: отношение количества данных к интервалу времени, в течение которого файл был открыт, включая время, в течение которого выполнялось открытие и закрытие файла
- Текущий (во время трассировки) размер файла и наибольший размер файла в процессе обработки
- Информация о файловой системе: тип файлов, размер сектора
- Режим и метки открытого файла
- Для каждой функции: сколько раз вызывалась данная функция, время обработки этой функции
- Для функций чтения или записи: дополнительная информация, например, требуется (необходимое пространство для чтения и записи), всего (фактический размер для записи или чтения, полученный с помощью вызова системы AIX), максимум и минимум
- Для поиска: среднее значение поправки для поиска (полное значение поправки для поиска и счетчик)
- Для функций чтения или записи: информация о приостановке, например, счетчик, время и доля времени переноса, включая приостановку, время чтения и записи
- Число обращений к странице fcntl page_info: страница

```
date
Trace on close or intermediate  :  previous module or calling program <-> next module : file name : (total transferred bytes/total time)=rate
    demand rate=rate/s=total transferred bytes/(close time-open time)
    current size=actual size of the file  max_size=max size of the file
mode=file open mode FileSystemType=file system type given by fststat(stat_b.f_vfstype) sector size=Minimum direct i/o transfer size
oflags=file open flags
open    open count          open time
fcntl      fcntl count         fcntl time
read      read count         read time   requested size      total size      minimum      maximum
aread    aread count       aread time requested size      total size      minimum      maximum
   suspend  count             time            rate
write      write count      write time  requested size       total size      minimum      maximum
seek      seek count        seek time      average seek delta
size
page    fcntl page_info count
```

Пример

```
MIO statistics file : Tue May 10 14:14:08 2005
hostname=host1  : with Legacy aio available
Program=/mio/example
MIO library libmio.a 3.0.0.60  AIX 32 bit addressing built Apr 19 2005 15:08:17
MIO_INSTALL_PATH=
MIO_STATS       =example.stats
```

```
MIO_DEBUG       =OPEN
MIO_FILES       = *.dat [ trace/stats ]
MIO_DEFAULTS    = trace/kbytes

MIO_DEBUG OPEN =T

Opening file file.dat
   modules[11]=trace/stats
================================================================================

Trace close : program <-> aix : file.dat : (4800/0.04)=111538.02 kbytes/s
     demand rate=42280.91 kbytes/s=4800/(0.12-0.01))
     current size=0   max_size=1600
  mode =0640  FileSystemType=JFS sector size=4096
  oflags =0x302=RDWR  CREAT  TRUNC
  open              1     0.00
  write           100     0.02       1600       1600      16384      16384
  read            200     0.02       3200       3200      16384      16384
  seek            101     0.01 average seek delta=-48503
  fcntl             1     0.00
  trunc             1     0.01
  close             1     0.00
  size            100
================================================================================
```

Пример файла результатов диагностики модуля pf:

В файле статистики модуля **pf** содержатся отладочные и диагностические данные.

Элементы и их размещение

```
закрытие pf  для <имя файла в кэше>
закрытие pf  для глобального или частного кэша <номер глобального кэша>
<число страниц, вычисляемое по формуле размер-кэша/размер-страницы> страниц с размером <размер-страницы> Б   <размер-сектора> Б/сектор
<фактическое число страниц, для которых не выполнена предварительная загрузка из-за опции pffw (без страниц, не загруженных из-за некорректности сектора)> /
                                                              <число страниц, для которых не выполнена предварительная
                                    загрузка из-за опции pffw>  страниц для записи не
                                    загружено предварительно
<число непопаданий> не использовано из всего 242 <число страниц, для которых выполнена предварительная загрузка> загруженных страниц:
prefetch=<число страниц, для предварительной загрузки>
<число> отложенных записей
<число> синхронизаций из-за некорректный записи
<число> страниц сохранено при закрытии
<единица> передано / Число запросов
программа --> <число Б, записанных в кэш предком>/<число записей от предка> --> pf -->

                                                  <число Б, записанных от потомка кэша>/<число страниц, записанных частично>
программа <-- <число Б, прочитанных из кэша предком>/<число чтений от предка> <-- pf <--
                                                  <число Б прочитанных от потомка кэша>/<число чтений страниц от потомка>
```

Пример

```
закрытие pf для /home/user1/pthread/258/SM20182_0.SCR300
50 страниц с размером 2097152 Б 131072 Б/сектор
133/133 страниц для записи не загружено предварительно
23 не использовано из всего 242 загруженных страниц: prefetch=2
95 отложенных записей
МБ передано / Число запросов
программа --> 257/257 --> pf --> 257/131 --> aix
программа <-- 269/269 <-- pf <-- 265/133 <-- aix
```

Пример файла результатов диагностики модуля recov:

В файле статистики модуля **recov** содержатся отладочные и диагностические данные.

При сбое открытия или записи модуль **recov** добавляет сообщение об этом, содежащее следующую информацию:

- Значение опций **open_command** или **command**, определяющих команду, выполняемую при обнаружении ошибки. Опции модуля рассмотрены в разделе “Опции модуля RECOV” на стр. 228.
- Номер ошибки.

- Число повторных попыток выполнения действия.

```
15:30:00
 recov : command=ls -l file=file.dat errno=28 try=0
 recov : failure : new_ret=-1
```

Пример конфигурации MIO

Поддерживается настройка MIO на уровне приложений.

Конфигурация ОС

Приложение **alot_buf** выполняет следующие действия:

- Создаёт файл с размером 14 ГБ.
- Выполняет 140 000 последовательных записей с буфером 100 КБ.
- Выполняет последовательное чтение с буфером 100 КБ.
- Выполняет последовательное чтение от конца в обратном порядке с буфером 100 КБ.

```
# vmstat
 Конфигурация системы: lcpu=2 mem=512MB
 нити     память             страница           ошибки        cpu
 ----- ----------- ---------------------- ------------ -----------
  r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
  1  1 35520 67055   0   0   0   0    0   0 241   64  80  0  0 99  0

# ulimit -a
 время, с              не ограничено
 файл, блоков          не ограничено
 данные, КБ            131072
 стек, КБ              32768
 память, КБ            32768
 дамп памяти, блоков   2097151
 нефайлы, дескрипторов 2000


# df -k /mio
ФС         Блоков по 1024  Свободно %Исп. Инд.исп. %И.исп. Точка монтирования
/dev/fslv02      15728640  15715508    1%      231      1% /mio

# lslv fslv02
ЛОГИЧЕСКИЙ ТОМ:      fslv02               ГРУППА ТОМА:         mio_vg
ИД ЛТ:               000b998d00004c00000000f17e5f50dd.2 ПРАВА ДОСТУПА:     чтение/запись
СОСТОЯНИЕ ГРУППЫ Т.: активная/завершённая СОСТОЯНИЕ ЛТ:        открыт/синхронизирован
ТИП:                 jfs2                 ПРОВЕРКА ЗАПИСИ:     выкл.
МАКС. ЛОГ. РАЗДЕЛОВ: 512                  РАЗМЕР ФИЗ. РАЗДЕЛА: 32 МБ
КОПИЙ:               1                    ПЛАНИРОВАНИЕ:        параллельное
ЧИСЛО ЛР:            480                  ЧИСЛО ФР:        480
УСТАРЕВШИХ ФР:       0                    СТРАТЕГИЯ ВВ:    перемещаемый
ВНЕШНЯЯ СТРАТЕГИЯ:   минимум              ПЕРЕМЕЩАЕМЫЙ:    да
ВНУТРЕННЯЯ СТРАТЕГИЯ:середина             ВЕРХНЯЯ ГРАНИЦА: 32
ТОЧКА МОНТИРОВАНИЯ: /mio                  МЕТКА:           /mio
СОГЛАСОВАННОСТЬ ЗЕРКАЛЬНОЙ ЗАПИСИ: включено/АКТИВНО
КАЖДАЯ КОПИЯ ЛОГИЧЕСКОГО РАЗДЕЛА НА ОТДЕЛЬНОМ ФИЗИЧЕСКОМ ТОМЕ?: да
СОХРАНЕНИЕ СОСТОЯНИЯ ВВОДА-ВЫВОДА:     НЕТ
```

Конфигурация MIO для анализа приложения /mio/alot_buf

```
setenv MIO_DEBUG " OPEN MODULES TIMESTAMP"
setenv MIO_FILES "* [ trace/stats/kbytes ]"
setenv MIO_STATS mio_analyze.stats

time /mio/alot_buf
```

Примечание: Файл с результатами диагностики для отладочных (**debug**) и трассировочных (**trace**) данных - `mio_analyze.stats`. Все значения указываются в килобайтах.

Примечание: Команда **time** включает указание времени выполнения команды.

Результат анализа

- Время выполнения - 28:06.
- Результаты диагностики MIO сохраняются в файле `mio_analyse.stats`.

```
Файл статистики MIO: Чтв Май 26 17:32:22 2005
hostname=miohost  : доступен устаревший асинхронный ввод-вывод
Program=/mio/alot_buf
MIO library libmio.a 3.0.0.60 AIX 64 bit addressing built Apr 19 2005 15:07:35
MIO_INSTALL_PATH=
MIO_STATS       =mio_analyze.stats
MIO_DEBUG       = MATCH OPEN MODULES TIMESTAMP
MIO_FILES       =* [ trace/stats/kbytes ]
MIO_DEFAULTS    =

MIO_DEBUG OPEN =T
MIO_DEBUG MODULES =T
MIO_DEBUG TIMESTAMP =T

17:32:22
Открытие файла test.dat
   modules[18]=trace/stats/kbytes
   trace/stats={mioout}/noevents/kbytes/nointer
   aix/nodebug/trunc/sector_size=0/einprogress=60
================================================================================

18:00:28
Трассировка закрытия : программа <-> aix : test.dat : (42000000/1513.95)=27741.92 КБ/с
     частота запросов=24912.42 КБ/с=42000000/(1685.92-0.01))
     текущий размер=14000000   макс. размер=14000000
  режим =0640  ФС=JFS2 размер сектора=4096
  флаги открытия =0x302=RDWR  CREAT  TRUNC
  open              1     0.01
  write        140000   238.16   14000000   14000000     102400     102400
  read         280000  1275.79   28000000   28000000     102400     102400
  seek         140003    11.45 средняя дельта поиска=-307192
  fcntl             2     0.00
  close             1     0.00
  size         140000
================================================================================
```

Примечание:

- 140 000 записей 102 400 Б.
- 280 000 чтений 102 400 Б.
- скорость 27 741.92 КБ/с.

Конфигурация MIO для повышения производительности ввода-вывода

```
setenv MIO_FILES "* [ trace/stats/kbytes | pf/cache=100m/page=2m/pref=4/stats/direct | trace/stats/kbytes ]"
setenv MIO_DEBUG "OPEN MODULES TIMESTAMP"
setenv MIO_STATS mio_pf.stats

time /mio/alot_buf
```

- Ввод-вывод приложения можно выполнять с помощью набора модулей `trace| pf | trace`. Так можно получить производительность кэша **pf** по отношению к приложению и производительность операционной системы по отношению к кэшу **pf**.
- Размер глобального кэша **pf** - 100 МБ. Размер одной страницы - 2 МБ. Число страниц, для которых выполняется предварительная загрузка - 4. Кэш **pf** использует асинхронный прозрачный ввод-вывод.
- Файл с результатами диагностики для отладочных (**debug**), трассировочных (**trace**) и данных модуля **pf** - `mio_pf.stats`. Все значения указываются в килобайтах.

Результат теста на производительность

- Время выполнения - 15:41.
- Результаты диагностики MIO сохраняются в файле `mio_pf.stats`.

```
Файл статистики MIO: Чтв Май 26 17:10:12 2005
hostname=uriage  : доступен устаревший асинхронный ввод-вывод
Program=/mio/alot_buf
MIO library libmio.a 3.0.0.60 AIX 64 bit addressing built Apr 19 2005 15:07:35
MIO_INSTALL_PATH=
MIO_STATS       =mio_fs.stats
MIO_DEBUG       = MATCH OPEN MODULES TIMESTAMP
MIO_FILES       =* [ trace/stats/kbytes | pf/cache=100m/page=2m/pref=4/stats/direct | trace/stats/kbytes ]
MIO_DEFAULTS    =

MIO_DEBUG OPEN =T
MIO_DEBUG MODULES =T
MIO_DEBUG TIMESTAMP =T

17:10:12
Открытие файла test.dat
   modules[79]=trace/stats/kbytes|pf/cache=100m/page=2m/pref=4/stats/direct|trace/stats/kbytes
   trace/stats={mioout}/noevents/kbytes/nointer
 pf/nopffw/release/global=0/asynchronous/direct/bytes/cache_size=100m/page_size=2m/prefetch=4/st
ride=1/stats={mioout}/nointer/noretain/nolistio/notag/noscratch/passthru={0:0}
   trace/stats={mioout}/noevents/kbytes/nointer
   aix/nodebug/trunc/sector_size=0/einprogress=60
================================================================================

17:25:53
Трассировка закрытия : pf <-> aix : test.dat : (41897728/619.76)=67603.08 КБ/с
     частота запросов=44527.71 КБ/с=41897728/(940.95-0.01))
     текущий размер=14000000   макс. размер=14000000
  режим =0640  ФС=JFS2 размер сектора=4096
  флаги открытия =0x8000302=RDWR  CREAT  TRUNC  DIRECT
  open              1     0.01
  ill form          0   сбой выравнивания памяти 0
  write             1     0.21       1920       1920    1966080    1966080
  awrite         6835     0.20   13998080   13998080    2097152    2097152
   suspend       6835   219.01    63855.82 КБ/с
  read              3     1.72       6144       6144    2097152    2097152
  aread         13619     1.02   27891584   27891584    1966080    2097152
   suspend      13619   397.59    69972.07 КБ/с
  seek          20458     0.00 средняя дельта поиска=-2097036
  fcntl             5     0.00
  fstat             2     0.00
  close             1     0.00
  size           6836

17:25:53
закрытие pf для test.dat
   50 страниц с размером 2097152 Б 4096 Б/сектор
   6840/6840 страниц для записи не загружено предварительно
   7 не использовано из всего 20459 загруженных страниц: prefetch=4
         6835 отложенных записей
   Б передано / Число запросов
   программа -->   14336000000/140000 --> pf --> 14336000000/6836   --> aix
   программа <--   28672000000/280000 <-- pf <-- 28567273472/13622  <-- aix

17:25:53
закрытие pf для глобального кэша 0
   50 страниц с размером 2097152 Б 4096 Б/сектор
   6840/6840 страниц для записи не загружено предварительно
   7 не использовано из всего 20459 загруженных страниц: prefetch=0
         6835 отложенных записей
   Б передано / Число запросов
   программа -->   14336000000/140000 --> pf --> 14336000000/6836   --> aix
   программа <--   28672000000/280000 <-- pf <-- 28567273472/13622  <-- aix

17:25:53
Трассировка закрытия : программа <-> pf : test.dat : (42000000/772.63)=54359.71 КБ/с
     частота запросов=44636.36 КБ/с=42000000/(940.95-0.01))
```

```
    текущий размер=14000000   макс. размер=14000000
  режим =0640  ФС=JFS2 размер сектора=4096
  флаги открытия =0x302=RDWR  CREAT  TRUNC
  open              1     0.01
  write        140000   288.88   14000000   14000000     102400     102400
  read         280000   483.75   28000000   28000000     102400     102400
  seek         140003    13.17 средняя дельта поиска=-307192
  fcntl             2     0.00
  close             1     0.00
  size         140000
================================================================================
```

Примечание: Программа выполняет 140 000 записей 102 400 байт и 280 000 чтений 102 400 байт, но модуль pf выполняет 6 836 записей (из которых 6 835 - асинхронные) 2 097 152 байт и 13 622 чтений (из которых 13 619 - асинхронные) 2 097 152 байт. Скорость - 54 359.71 КБ/с.

Производительность файловой системы

Настройка файловых систем, поддерживаемых в AIX, существенно влияет на общую производительность системы. Эта операция может занять много времени.

Базовая информация о файловых системах приведена в разделе *Управление операционной системой и устройствами.*

Типы файловых систем

В AIX поддерживаются как локальные, так и удаленные файловые системы.

К локальным относятся следующие файловые системы:

- Журналируемая файловая система
- Расширенная журналируемая файловая система
- Файловая система на компакт-диске
- Файловая система на RAM-диске

К удаленным относятся следующие файловые системы:

- Сетевая файловая система
- Базовая параллельная файловая система

Журналируемая файловая система (JFS)

Журналируемые файловые системы позволяют быстро восстанавливать данные в случае сбоя. Это достигается за счет ведения журнала изменений файлов.

При каждом изменении файла система записывает определенные метаданные в зарезервированную область файловой системы. Фактическая операция записи выполняется только после изменения метаданных в журнале.

Т.к. журналируемая файловая система была создана для работы с 32-разрядным ядром в предыдущих выпусках AIX, она не работает оптимально в среде 64-разрядного ядра. Однако, JFS все еще можно использовать в AIX 6.1 и более поздних версиях сред 64-разрядного ядра.

Расширенная JFS

Расширенная JFS, или JFS2, - это еще одна внутренняя файловая система AIX.

Расширенная JFS применяется по умолчанию в средах с 64-разрядным ядром. Из-за ограничений адресного пространства 32–разрядного ядра применение расширенной JFS в средах с таким ядром не рекомендуется.

Поддержка наборов данных интегрируется в JFS2 как компонент операционной системы AIX. Набор данных - это единичный элемент администрирования данных. Он состоит из дерева каталога и по крайней мере одного корневого каталога. Под администрированием понимается создание наборов данных, создание полных копий наборов данных (реплик) и управление ими параллельно на разных серверах или перемещение набора данных на другой сервер. Набор данных может существовать как часть смонтированной файловой системы. То есть экземпляр смонтированной файловой системы может содержать несколько наборов данных. Для включения поддержки наборов данных в JFS2 используется команда **mkfs -o dm=on**. По умолчанию поддержка наборов данных выключена. Экземпляром JFS2 с включенной поддержкой наборов данных можно управлять с помощью Администратора обслуживания наборов данных (DSM).

Различия между JFS и расширенной JFS

В этом разделе перечислены различия между JFS и расширенной JFS (JFS2).

Таблица 4. Функциональные различия между JFS и расширенным JFS

Функция	**JFS**	**Расширенный JFS**
Оптимизация	32-разрядное ядро	64-разрядное ядро
Максимальный размер файловой системы	32 терабайт	4 Пб **Примечание:** Это - ограничение архитектуры. AIX в настоящее время поддерживает только до 16 терабайт.
Максимальный размер файла	64 Гб	4 Пб **Примечание:** Это - ограничение архитектуры. AIX в настоящее время поддерживает только до 16 терабайт.
Количество I-узлов	Устанавливается при создании файловой системы	Изменяется динамически, ограничено объемом свободной памяти на диске
Поддержка больших файлов	Можно включить при монтировании	По умолчанию
Динамическая дефрагментация	Да	Да
namefs	Да	Да
DMAPI	Нет	Да
Сжатие	Да	Нет
Квоты	Да	Да
Отложенное обновление	Да	Нет
Поддержка прозрачного ввода-вывода	Да	Да

Примечание:

- С помощью команды **mksysb** нельзя создать в 32-разрядной системе копию 64-разрядной системы с файловой системой JFS2.
- В отличие от JFS, в расширенном JFS нельзя применять API **link()** при работе с двоичными каталогами. Из-за этого некоторые программы, корректно работающие с файловыми системами JFS, могут неправильно работать с файловыми системами JFS2.

Ведение журнала:

Перед записью данных JFS заносит в журнал метаданные, что снижает производительность записи.

Один из способов повышения производительности заключается в отключении журнала изменений JFS с помощью опции монтирования **nointegrity**. Учтите, что повышение производительности достигается за счет нарушения целостности метаданных. В связи с этим данную опцию следует применять с большой осторожностью, поскольку файловую систему, которая монтировалась с этой опцией, нельзя восстановить после сбоя системы.

В расширенном JFS отключить журнал нельзя. В расширенном JFS применяются усовершенствованные алгоритмы обработки журналов, и поэтому издержки не так велики, как при работе с JFS.

Организация каталога:

Индексный узел (I-узел) - это структура данных, в которой хранятся все сведения о файле или каталоге. При поиске файла система ищет нужный I-узел по имени файла в каталоге.

Поскольку поиск файлов выполняется очень часто, для обеспечения высокой производительности системы очень важна эффективность механизма поиска.

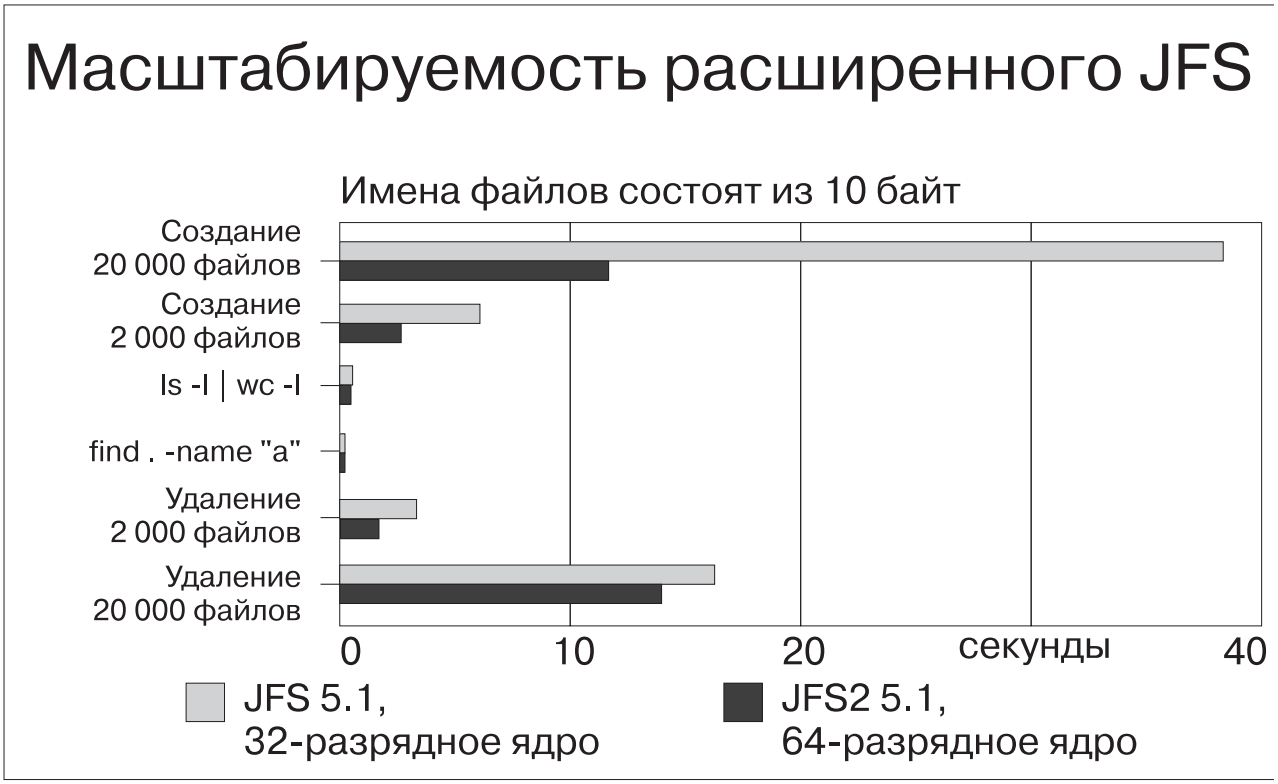
В JFS применяется линейная организация каталогов и линейный поиск. В расширенном JFS применяется организация в виде двоичного дерева, что позволяет существенно повысить скорость доступа к файлам.

Масштабирование:

Главное преимущество расширенного JFS по сравнению с обычным JFS заключается в возможности масштабирования.

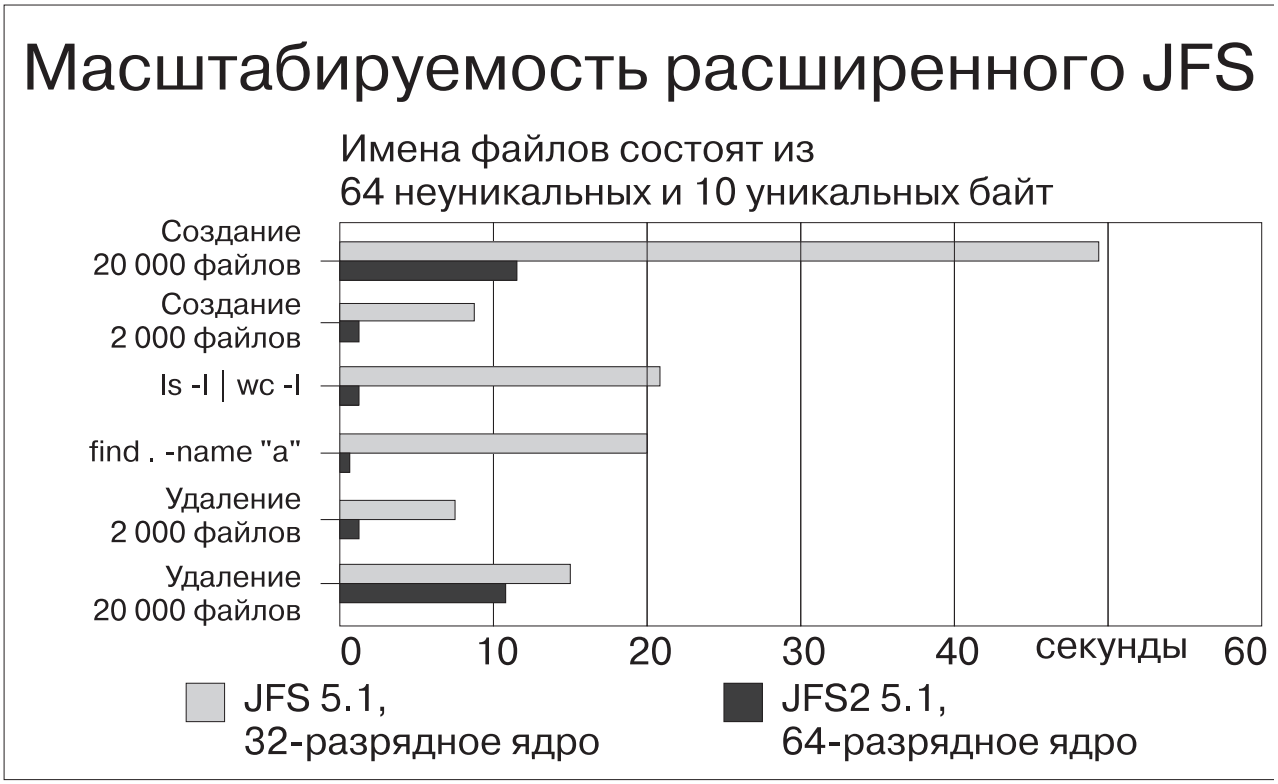
В расширенном JFS существенно увеличен максимальный размер файла. Максимальный размер файла в JFS составляет 64 ГБ. В Расширенном JFS AIX в настоящее время поддерживает файлы размером до 16 терабайт, хотя архитектура файловой системы предусматривает возможность хранения файлов размером до 5 петабайт.

Еще одно преимущество заключается в повышении производительности при работе с большим числом файлов. На следующей схеме показано, за счет чего в расширенном JFS достигается более высокая производительность.



В данном примере создаются и удаляются файлы с уникальными именами длиной 10 байт, а также выполняется поиск этих файлов в каталогах. Пример иллюстрирует высокую скорость создания и удаления файлов в расширенном JFS (по сравнению с JFS). Скорость поиска в каталогах в данном случае примерно одинакова.

Следующий пример иллюстрирует преимущества расширенного JFS при выполнении операций создания, удаления и поиска при работе с файлами с похожими именами. В этом примере длина имен файлов составляет 74 байта, причем первые 64 байта имен у всех файлов совпадают. Результаты данного теста приведены на следующем рисунке:



Кроме того, в JFS и расширенном JFS поддерживается кэширование длинных имен файлов (длиной до 32 символов). Это увеличивает производительность выполнения таких операций, как **ls** и **find**, над каталогами с большим числом файлом с длинными именами.

Файловая система на компакт-диске

Это файловая система, которая хранится на компакт-диске и доступна только для чтения.

AIX поддерживает несколько типов файловых систем на CD-ROM. Это описано в разделе Типы файловых систем в *Управление операционной системой и устройствами*.

Файловая система RAM

Диск RAM - это виртуальный жесткий диск, хранящийся в оперативной памяти.

Такие диски обладают существенно более высокой производительностью, чем физические жесткие диски, и обычно применяются для повышения скорости обработки временных файлов при высокой нагрузке на подсистему ввода-вывода. Максимальный объем файловой системы RAM ограничен объемом оперативной памяти. С файловой системой, созданной на диске RAM, можно работать как с обычной файловой системой. Не рекомендуется долгосрочно хранить данные на дисках RAM, так как все содержимое таких дисков теряется в случае сбоя, а также при перезагрузке системы.

Сетевая файловая система (NFS)

Сетевая файловая система, или NFS, - это распределенная файловая система, предоставляющая доступ к файлам и каталогам, хранящимся в удаленных системах, обычными средствами для работы с локальными файлами. Например, обычными командами операционной системы можно создавать, удалять, читать и записывать файлы, а также изменять их атрибуты.

Сведения о настройке NFS и другую информацию вы можете найти в разделе Производительность NFS.

Система имен файлов (NameFS)

Система имен файлов содержит функции монтирования файл-на-файл и каталог-на-каталог (также называемое слабое монтирование), которые позволяют монтировать подкаталог или файловую систему в другом месте в области имен файлов, что позволяет иметь доступ к файлу с помощью двух различных путей.

Эта функция также полезна при изменении атрибутов монтирования для некоторых каталогов. Например, если файлы в определенном каталоге требуют поддержку прозрачного ввода-вывода, но он не поддерживается файловой системой, то этот каталог можно смонтировать с помощью **Системы имен файлов** с флагом **-o dio** (при условии, что тип файловой системы, содержащей объект, поддерживает **dio**).

Система имен файлов - это логическая структура. Она существует только во время монтирования и служит для группировки файлов для логических целей. Все операции с объектами в Системе имен файлов выполняются физической файловой системой, и функции и семантика применяется, как если бы Система имен файлов не существовала.

Система имен файлов создается монтированием одного имени файла (`path1`) поверх другого (`path2`). Объекты `path1` и `path2` должны быть файлами или каталогами, и их типы должны совпадать. При доступе к файлу, указанному в `path2` происходит доступ к `path1`. При доступе к каталогу, указанному в `path2/<pathname>` происходит доступ к `path1/<pathname>`. Программный интерфейс системы имен файлов выполняется стандартным файловым интерфейсом системных вызовов. Доступ к системе имен файлов осуществляется указанием **gfstype** в структуре**vmount**, переданной в системный вызов `vmount()`. Пользовательский интерфейс системы имен файлов - это команда **mount**. Команда **mount** распознает тип **vfs** *Систем имен файлов* как допустимую опцию для флага **-v**.

Примечание: Систему имен файлов нельзя экспортировать с помощью NFS.

Базовая параллельная файловая система

Распараллеленные файловые системы или GPFS - это высокопроизводительные файловые системы, распределенные по нескольким общим жестким дискам и применяемые для обеспечения быстрого доступа к данным для всех узлов кластера. Для работы с ними применяются стандартные интерфейсы UNIX или AIX.

В файловых системах GPFS данные распределяются по нескольким жестким дискам, причем предусмотрены возможности дублирования данных и средства восстановления после сбоев.

Информация, связанная с данной:

GPFS on AIX Clusters: High Performance File System Administration Simplified

Возможные источники снижения производительности в JFS и расширенном JFS

Потенциальные источники снижения производительности в JFS и расширенном JFS.

Влияние протокола файловой системы на производительность файловой системы

Операции записи несколько замедляются за счет того, что перед фактической записью данных сведения об изменениях заносятся в журнал метаданных.

Сведения о том, как устранить этот фактор снижения производительности, приведены в разделе “Производительность логических томов и дискового ввода-вывода” на стр. 178.

Сжатие и фрагментация

В целях экономии дисковой памяти в журналируемых файловых системах могут применяться сжатие и фрагментация данных.

В среднем сжатие позволяет сэкономить примерно половину дисковой памяти. Однако и сжатие, и фрагментация могут быть причинами снижения производительности дисковых операций. Подробные сведения об этом приведены в разделе “Производительность логических томов и дискового ввода-вывода” на стр. 178.

В целях повышения производительности в файловых системах JFS и JFS2 предусмотрены средства динамической дефрагментации. В процессе дефрагментации файловые системы можно монтировать и использовать без ограничений.

Улучшение производительности файловой системы

В этом разделе обсуждаются способы повышения производительности файловых систем в AIX.

Последовательное упреждающее чтение

Наблюдая за тем, в какой последовательности программа обрабатывает компоненты файла при последовательном доступе, VMM заранее считывает нужные страницы в память.

Если программа последовательно обращается к двум расположенным одна за другой страницам файла, VMM предполагает, что программа и в дальнейшем будет последовательно обрабатывать файл, и выполняет дополнительные операции последовательного считывания файла. Эти операции чтения выполняются одновременно с программой, в результате чего данные становятся доступны программе раньше, чем если бы VMM начинал операции ввода-вывода только после того, как программа обратится к следующей странице.

В JFS количество страниц, считываемых заранее, определяется двумя параметрами VMM:

minpgahead
Количество страниц, считываемых с опережением в случае, когда VMM впервые обнаружит признаки последовательного доступа к файлу.

Если программа продолжает последовательно обрабатывать файл, в следующий раз будет считано с опережением 2**minpgahead* страниц, затем - 4**minpgahead* и т.д., пока это количество не достигнет значения *maxpgahead*.

maxpgahead
Максимальное количество страниц, которые VMM считывает из файла с опережением.

В расширенном JFS количество страниц, считываемых заранее, определяется следующими параметрами VMM:

j2_minPageReadAhead
Количество страниц, считываемых с опережением в случае, когда VMM впервые обнаружит признаки последовательного доступа к файлу.

Если программа продолжает последовательно обрабатывать файл, в следующий раз будет считано с опережением 2**j2_minPageReadAhead* страниц, затем 4**j2_minPageReadAhead* страниц и так далее, пока это количество не достигнет значения, равного *j2_maxPageReadAhead*.

j2_maxPageReadAhead
Максимальное количество страниц, которые VMM считывает с опережением из файла с последовательным доступом.

Последовательная отложенная запись

Существуют два алгоритма отложенной записи: последовательный и произвольный.

Код поддержки файловых систем AIX делит каждый файл на кластеры длиной 16 КБ (в JFS) и 128 КБ (в расширенном JFS). Это позволяет решить следующие задачи:

- Увеличить производительность записи
- Ограничить число изменяемых страниц файла в памяти

- Снизить дополнительную нагрузку в системе
- Минимизировать фрагментацию диска

Страницы, относящиеся к данному разделу, не записываются на диск до того, как программа запишет первый байт следующего раздела длиной 16 КБ. В этот момент система принудительно записывает на диск четыре ожидающих выгрузки страницы из первого раздела. Страницы с данными остаются в памяти до тех пор, пока не поступит повторный запрос к соответствующим страницам физической памяти. Для обработки этого запроса не потребуется выполнять дополнительных операций ввода-вывода. Если программа повторно обращается к любой из страниц до этого момента, то операции ввода-вывода не выполняются.

Если в памяти находится большое количество уже обработанных страниц, и повторно к ним не обращаются, то демон sync записывает их на диск, что может привести к резкому возрастанию интенсивности операций с диском. Для более равномерного распределения операций ввода-вывода можно включить механизм отложенной записи, указав системе, сколько страниц может храниться в памяти без записи на диск. Пороговое значение для числа страниц, хранящихся в памяти без записи на диск, задается отдельно для каждого файла.

Размер раздела и пороговое значение числа страниц для отложенной записи можно изменить с помощью команды **ioo**. Дополнительная информация приведена в разделе “Настройка производительности последовательной отложенной записи” на стр. 247.

Файлы, размещенные в памяти, и отложенная запись

Обычные файлы автоматически преобразуются в сегменты памяти. Это означает, что при обращении к файлу не применяются буферы ядра и процедуры ввода-вывода блоков, что позволяет файлам использовать больший объем памяти при ее достаточном количестве (Кэш файла может выйти за пределы объявленной области буферов ядра).

Файлы можно преобразовать явно с помощью функции **shmat()** или **mmap()**, однако при этом для кэша не будет доступен дополнительный объем памяти. Приложения, явно преобразующие файлы с помощью функций **shmat()** и **mmap()**, и обращающиеся к ним по адресу, а не с помощью функций **read()** и **write()**, могут не указывать длинные пути к файлам в системных вызовах, однако не могут использовать системную функцию отложенной записи.

Если приложение не применяет функцию **write()**, то измененные страницы накапливаются в памяти, а затем алгоритм замены страниц VMM или демон **sync** записывает их на диск случайным образом. В результате выполняется много небольших операций записи на диск, что снижает эффективность использования ресурсов процессора и диска, а также повышает степень фрагментации, что в будущем негативно скажется на скорости чтения файла.

Механизм быстрого освобождения страниц

Быстрое освобождение страниц - это механизм для файловых систем JFS и расширенной JFS, с помощью которого страницы освобождаются сразу после записи на постоянный носитель (при записи) или передачи приложению (при чтении). Это позволяет увеличить масштабируемость при последовательном вводе-выводе больших файлов, повторный доступ к страницам которых в ближайшее время не требуется.

При записи большого файла без механизма быстрого освобождения страниц запись будет выполняться очень быстро, пока список свободных страниц не будет исчерпан. Как только число свободных страниц уменьшится до значения параметра *minfree*, VMM начнет удаление наиболее давно использовавшихся (LRU) страниц из памяти. Во время этого процесса VMM захватывает блокировку, применяемую также для записи. Конкуренция за эту блокировку может привести к существенному снижению производительности.

Для того чтобы включить функцию быстрого освобождения страниц, укажите соответствующий флаг в команде **mount**: rbr (быстрое освобождение при последовательном чтении), rbw (быстрое освобождение при последовательной записи), rbrw (быстрое освобождение при последовательном чтении и записи).

Побочный эффект применения механизма быстрого освобождения страниц состоит в увеличении нагрузки на процессор при той же производительности ввода-вывода. Это связано с работой по освобождению страниц, обычно выполняемой с задержкой демоном LRU. Учтите также, что все обращения к файловой системе будут связаны с дисковым вводом-выводом, поскольку страницы не кэшируются VMM.

Для применения быстрого освобождения страниц для NFS можно использовать команду **mount -o rbr**.

Поддержка прозрачного ввода-вывода

И в JFS, и в расширенном JFS возможен прозрачный доступ к файлам.

При прозрачном способе доступа к файлам кэш не используется, а данные напрямую передаются с диска в пользовательский буфер и обратно. Советы по настройке приведены в разделе “Настройка прозрачного ввода-вывода” на стр. 254

Отложенная запись

В JFS предусмотрена возможность записи результата нескольких операций за одно физическое обращение к диску (отложенная запись). Применение режима отложенной записи позволяет сократить число обращений к диску для часто перезаписываемых файлов.

Для включения режима отложенной записи нужно открыть файл с флагом отложенной записи (**O_DEFER**). Фактически отложенная запись представляет собой разновидность кэширования и ускоряет чтение и запись данных из других процессов.

При работе с файлом, открытым в режиме отложенной записи, фактическое обращение к диску и сохранение результатов операций выполняется только после того, как процесс выполнит команду **fsync**. Учтите, что операции синхронной записи (т.е. операции записи в файл, открытый с флагом **O_SYNC**) выполняются немедленно, независимо от того, был ли при создании файла указан флаг **O_DEFER**.

Примечание: Данный режим не поддерживается в расширенном JFS.

Поддержка параллельного ввода-вывода

В расширенном JFS поддерживается параллельный доступ к файлам.

Как и при прозрачном вводе-выводе, при этом способе доступа к файлам кэш не используется, а данные напрямую передаются с диска в пользовательский буфер и обратно. Кроме того, обходится блокировка I-узла, что позволяет нескольким нитям параллельно считывать и записывать один и тот же файл.

Примечание: В JFS этот режим не поддерживается.

Атрибуты файловой системы, влияющие на производительность

Чем дольше используется файловая система, тем более фрагментированной она становится. При динамическом выделении ресурсов блоки файлов размещаются все менее компактно, а логически последовательные файлы и логические тома физически становятся фрагментированными.

Ниже приведен список результатов обращения к файлам фрагментированного логического тома:

- Последовательный доступ фактически перестает быть таковым
- Неупорядоченный доступ требует больше времени
- Время доступа в основном определяется временем поиска

После загрузки файла в память все перечисленные эффекты пропадают. Кроме перечисленных особенностей, производительность файловой системы зависит от следующих физических факторов:

- Типы применяемых дисков и количество адаптеров
- Объем памяти, выделенной для буферизации файлов
- Соотношение числа обращений к локальным и удаленным файлам

- Число обращений к файлу и применяемый приложением способ доступа

JFS позволяет изменять размер фрагмента файловой системы для более эффективного использования памяти с дополнительным разбиением блоков размером 4 КБ на более мелкие фрагменты. Количеством i-узлов файловой системы можно управлять с помощью параметра, задающего количество байт на каждый i-узел (NBPI). Для файловых систем с размером фрагмента менее 4 КБ возможно сжатие данных. Размер фрагмента и сжатие данных значительно влияют на производительность. Эти атрибуты файловой системы обсуждаются в следующих разделах:

- “Размер фрагмента JFS”
- “Сжатие JFS” на стр. 244

Размер фрагмента JFS

В JFS размер выделяемого фрагмента файловой системы может быть меньше 4 КБ.

Размер фрагмента задается при создании файловой системы. Допустимые значения: 512, 1024, 2048 и 4096 байт. Значение по умолчанию равно 4096 байт. Файлы, размер которых меньше размера фрагмента, записываются в один фрагмент.

Для размещения файлов, размер которых меньше 4096 байт, выделяется минимально необходимое количество смежных фрагментов. Для хранения файлов размером от 4096 байт до 32 КБ (включительно) применяется один или несколько целых блоков по 4 КБ и несколько фрагментов меньшего размера. Например, файл размером 5632 байт займет целый блок размером 4 КБ, на который будет ссылаться первый указатель i-узла. При размере фрагмента 512 байт для первого блока размером 4 КБ будет выделено восемь фрагментов. Оставшиеся 1,5 КБ файла займут три фрагмента, на которые будет ссылаться второй указатель i-узла. Для файлов размером больше 32 КБ дисковое пространство выделяется блоками по 4 КБ, и указатели i-узла ссылаются именно на блоки по 4 КБ.

Независимо от размера фрагмента, размер блока всегда составляет 4096 байт. Однако в файловой системе, размер фрагмента которой менее 4096 байт, вместо полного блока можно выделить нужное количество расположенных друг за другом фрагментов общим объемом 4096 байт. Такая область не обязательно выровнена по границе 4096 байт.

Файловая система пытается выделять место для файлов непрерывными цепочками фрагментов. При этом файлы размещаются в пределах логического тома так, чтобы интервалы между файлами и степень фрагментации были минимальными.

В файловых системах с небольшим размером фрагмента фрагментация является главной причиной снижения производительности. При наличии большого количества маленьких файлов, разбросанных по всему логическому тому, может оказаться невозможным записать большой файл одним куском или несколькими участками, расположенными близко друг к другу. В этом случае обращение к большим файлам будет очень медленным. В худшем случае из-за фрагментации нельзя выделить место для файла, даже если суммарный объем свободных фрагментов больше размера файла.

Кроме фрагментации, применение фрагментов малого размера приводит также к увеличению числа операций ввода-вывода. Файл размером 4 КБ, хранящийся в одном фрагменте того же размера, можно прочитать или записать, выполнив всего одну дисковую операцию ввода-вывода. Если же применяются фрагменты по 512 байт, то для этого файла будет выделено восемь фрагментов и для полного считывания или записи файла потребуется выполнить несколько дополнительных операций ввода-вывода (позиционирование головки, поиск и передача данных). Таким образом, при работе с файловыми системами с размером фрагмента 4 КБ число операций дискового ввода-вывода может быть существенно меньше, чем при использовании файловых систем с меньшим размером фрагмента.

Если администратор принял решение создать файловую систему с небольшим размером фрагмента, он должен разработать стратегию дефрагментации этой файловой системы с помощью утилиты **defragfs**. При этом следует учесть объем ресурсов, который будет затрачиваться на выполнение команды **defragfs**. Обратитесь к разделу “Дефрагментация файловой системы” на стр. 246.

Сжатие JFS

Все данные могут перед записью на диск автоматически сжиматься с помощью алгоритма Lempel-Zev (LZ), а при чтении - автоматически разворачиваться. Алгоритм LZ основан на замене встречающихся последовательностей символов на указатель на первое вхождение последовательности. Применение этого алгоритма позволяет сэкономить в среднем около 50 процентов дискового пространства.

Данные файловых систем сжимаются на уровне отдельных логических блоков. Сжатие данных в более крупных блоках (например, во всей совокупности логических блоков выбранного файла) привело бы к менее эффективному использованию дискового пространства. Сжатие отдельных логических блоков файла позволяет существенно ускорить выполнение операций неупорядоченного поиска и обновления данных.

Алгоритм сжимает данные блоками по 4096 байт (1 страница) и сохраняет сжатые данные в виде минимально необходимой последовательности непрерывных участков. Очевидно, что если в файловой системе используются непрерывные участки дискового пространства по 4 КБ, сжатие данных не приведет к экономии дискового пространства. Таким образом, при использовании сжатия данных в файловой системе должны применяться фрагменты размером менее 4096 байт.

Несмотря на то, что сжатие данных в общем случае позволяет более эффективно использовать место на диске, в файловой системе все же может оставаться неиспользуемое пространство:

- В связи с тем, что степень сжатия блоков размером 4096 байт заранее неизвестна, файловая система резервирует для данных полный блок. Ненужные участки диска освобождаются после сжатия, однако такая процедура предварительного выделения памяти может привести к более раннему появлению сообщений о нехватке места.
- Свободное место необходимо для работы команды **defragfs**.

Помимо увеличения числа операций ввода-вывода и фрагментации свободного пространства, при использовании файловых систем со сжатием данных следует также учитывать следующие факторы, влияющие на производительность:

- Низкая скорость работы с файловой системой может быть прямым следствием использования алгоритмов сжатия/разворачивания данных. Если время сжатия и разворачивания данных сравнительно велико, то применение функций сжатия может оказаться нецелесообразным, особенно в коммерческой среде, когда данные должны предоставляться очень быстро.
- При первом изменении для всех логических блоков файловой системы со сжатием выделяется по 4096 байт дискового пространства; после записи логического блока на диск это пространство перераспределяется. Такое перераспределение требует затрат дополнительных ресурсов, которых можно избежать в файловых системах без сжатия данных.
- Для сжатия каждого байта данных необходимо в среднем 50 циклов CPU, а для разворачивания - в среднем 10 циклов. Таким образом, сжатие данных увеличивает нагрузку на процессор.
- Процесс сжатия JFS (jfsc) kproc работает с фиксированным приоритетом, равным 30, поэтому ресурсы CPU, на котором работает kproc, во время сжатия могут оказаться недоступными для процессов, работающих с меньшим приоритетом.

Реорганизация файловой системы

Для того чтобы уменьшить степень фрагментации файловой системы, выполните следующие действия:

- Скопируйте файлы на резервный носитель
- Повторно создайте файловую систему с помощью команды **mkfs** *файловая-система*, или удалите все данные из файловой системы
- Скопируйте файлы в файловую систему

В ходе этой процедуры файлы копируются последовательно, в результате чего степень фрагментации файловой системы снижается. Дополнительная информация приведена в следующих разделах:

- “Реорганизация файловой системы” на стр. 245
- “Дефрагментация файловой системы” на стр. 246

Реорганизация файловой системы

В этом разделе описаны действия, необходимые для выполнения реорганизации файловой системы.

В следующем примере в системе есть логический том, содержащий файловую систему *hd11* (точка монтирования - /home/op). Для того чтобы реорганизовать файловую систему *hd11*, выполните следующие действия:

1. Создайте пофайловую резервную копию файловой системы. При резервном копировании файловой системы по i-узлам команда **restore** сохранит исходное размещение файлов. Выполните следующие команды:

   ```
   # cd /home/op
   # find . -print | backup -ivf/tmp/op.backup
   ```

   Эта команда создаст в другой файловой системе файл, содержащий все файлы из исходной файловой системы. Если в системе недостаточно дисковой памяти, то создайте резервную копию файловой системы на магнитной ленте.

2. Выполните следующие команды:

   ```
   # cd /
   # unmount /home/op
   ```

   Если некоторые процессы работают с каталогом /home/op или его подкаталогами, то перед вызовом команды **unmount** завершите работу этих процессов.

3. Восстановите файловую систему на логическом томе /home/op, вызвав следующую команду:

   ```
   # mkfs /dev/hd11
   ```

   Перед уничтожением существующей файловой системы будет показано приглашение для подтверждения операции. Имя файловой системы не изменится.

4. Восстановите исходную конфигурацию, вызвав следующую команду (каталог /home/op будет пустым):

   ```
   # mount /dev/hd11 /home/op
   # cd /home/op
   ```

5. Восстановите все данные, вызвав следующую команду:

   ```
   # restore -xvf/tmp/op.backup >/dev/null
   ```

   Стандартный вывод команды перенаправлен на устройство /dev/null, чтобы на экран не выводились имена восстанавливаемых файлов. Это позволяет сэкономить некоторое время при выполнении операции.

6. Снова проанализируйте размещение файла большого размера (за дополнительной информацией обратитесь к разделу Оценка размещения файла с помощью команды fileplace). Для этого введите следующую команду:

   ```
   # fileplace -piv big1
   ```

Теперь этот файл размещен в смежных фрагментах диска:

```
Файл: big1  Размер: 3554273 байт  Том: /dev/hd11
Размер блока: 4096  Размер фрагмента: 4096  Число фрагментов: 868  Сжатие: нет
I-узел: 8290  Режим: -rwxr-xr-x  Владелец: hoetzel  Группа: system

НЕЯВНЫЙ БЛОК: 60307

  Физический адрес (зеркальная копия 1)                       Логические фрагменты

  ---------------------------------                           ----------------

  0060299-0060306  hdisk1     8 фраг.    32768 байт,   0.9%     0008555-0008562
  0060308-0061167  hdisk1   860 фраг.  3522560 байт,   99.1%    0008564-0009423

  868 фрагментов занимают объем 869:  эффективность = 99.9%
  2 фрагмента из 868 возможных:  компактность размещения = 99.9%
```

Опция **-i** команды **fileplace** указывает, что блок, размещенный между первыми восьмью блоками файла и остальными блоками файла содержит ссылки на остальные блоки. Другими словами, информация из этого блока дополняет данные из i-узла. Такой блок всегда создается, когда размер файла превышает восемь блоков.

Реорганизация логических томов не выполняется, если они содержат временную информацию (например, `/tmp`), либо не содержат файловой системы (например, содержат протокол). Корневая файловая система редко изменяется, поэтому, как правило, ее не требуется реорганизовывать. Реорганизацию корневой файловой системы можно выполнить только во время установки или обслуживания. То же самое относится и к файловой системе `/usr`, так как многие из ее файлов требуются для работы системы.

Дефрагментация файловой системы

Если при создании файловой системы был задан размер фрагмента меньше 4 КБ, то через некоторое время нужно узнать, сколько в системе существует отдельных неиспользуемых фрагментов. Если число таких фрагментов велико, в файловой системе нет непрерывных участков памяти для размещения файлов.

Для того чтобы объединить отдельные неиспользуемые фрагменты, введите команду **smitty dejfs** или **smitty dejfs2**, либо команду **defragfs**. Для выполнения дефрагментации требуется некоторый объем дисковой памяти. Кроме того, при монтировании файловой системы должно быть установлено разрешение на чтение и запись.

Тонкая настройка производительности файловой системы

Описаны различные аспекты настройки производительности файловой системы.

Настройка производительности последовательного чтения

Функция упреждающего чтения VMM, позволяет повысить производительность программ, применяющих последовательный способ доступа при работе с большими файлами.

Функция упреждающего последовательного чтения VMM рассмотрена в “Последовательное упреждающее чтение” на стр. 240.

Ниже приведен типичный пример упреждающей записи.

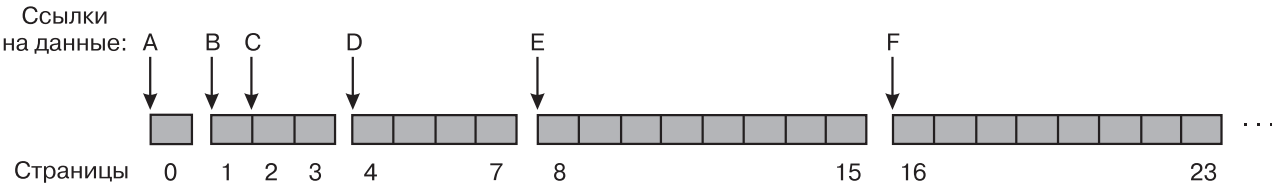


Рисунок 20. Пример последовательного упреждающего чтения. На этом рисунке показана группа блоков, изображающих сегменты страниц файла. Сегменты содержат страницы 0, 1-3, 4-7, 8-15 и 16-23. Принцип работы алгоритма упреждающего чтения описан ниже.

В данном случае значение *minpgahead* равно 2, а значение *maxpgahead* равно 8 (значения по умолчанию). Программа последовательно обрабатывает файл. На рисунке показаны только те ссылки (от A до F), которые имеют отношение к операции упреждающего чтения. Последовательность действий:

A При первом обращении к файлу считывается первая страница файла (страница номер 0). В этот момент VMM не делает никаких предположений относительно способа доступа к файлу.

B Если после этого программа, не запрашивая других страниц файла, обращается к первому байту следующей страницы 1, то VMM считает, что способ доступа последовательный. Тогда VMM планирует чтение еще двух страниц, 2 и 3 (число страниц определяется значением *minpgahead*). Таким образом, на этапе B считывается 3 страницы.

C Когда программа обращается к первому байту первой страницы, прочитанной заранее (страницы 2), VMM удваивает число страниц, считываемых с упреждением, и планирует чтение уже 4 страниц (с 4 по 7).

D Когда программа обращается к первому байту первой страницы, прочитанной заранее (страницы 4), то VMM снова удваивает число страниц, считываемых с упреждением, и планирует чтение 8 страниц (с 8 по 15).

E Когда программа обращается к первому байту первой страницы, прочитанной с упреждением (страницы 8), VMM определяет, что число страниц, читаемых с упреждением, достигло максимума (*maxpgahead*), и планирует чтение страниц с 16 по 23.

F При каждом следующем обращении программы к первому байту предыдущей группы страниц, прочитанных заранее, VMM считывает с упреждением *maxpgahead* страниц. Таким образом файл прочитывается до конца.

Если программа прекращает последовательную обработку файла и обращается не к следующей по порядку странице, процесс последовательного упреждающего чтения прекращается. Если VMM обнаружит, что программа вернулась к последовательному чтению страниц, то функция упреждающего чтения будет снова включена, и в первый раз будет считано *minpgahead* страниц.

Значения *minpgahead* и *maxpgahead* можно изменить с помощью опций **-r** и **-R** команды **ioo**. При этом следует помнить следующее:

- Рекомендуется указывать в качестве значений параметров степени двойки. Установка других значений может привести к снижению производительности системы или сбою в работе некоторых функций.
  - Поскольку в алгоритме число страниц все время удваивается, все значения должны быть степенью двойки.
  - Некоторые драйверы дисков не поддерживают значения *maxpgahead* больше 16 (чтение с упреждением более чем 64 КБ данных). В этом случае всегда считывается по 64 КБ данных.
  - В системах, где скорость последовательного чтения данных с логических томов с чередованием данных имеет первостепенное значение, можно указать более высокие значения *maxpgahead*.
- Для того чтобы отключить функцию последовательного чтения, в параметрах *minpgahead* и *maxpgahead* следует указать значение 0. В результате производительность может снизиться. При выполнении некоторых операций ввода-вывода с неупорядоченным доступом к данным выключение функции может положительно сказаться на производительности. Тем не менее, если считывается большой объем данных, то рекомендуется включить алгоритм упреждающего чтения.
- Максимальная производительность операций последовательного ввода-вывода на файловых системах без чередования данных достигается при значениях 8-16 параметра *maxpgahead*.
- Число страниц, считываемых заранее, быстро увеличивается от значения *minpgahead* до значения *maxpgahead*, поэтому для большинства файлов не имеет смысла увеличивать значение *minpgahead*.
- Число заранее считываемых страниц можно регулировать отдельно для JFS и расширенного JFS. Для JFS нужно воспользоваться параметрами *minpgahead* и *maxpgahead*, а для расширенного JFS - параметрами *j2_minPageReadAhead* и *j2_maxPageReadAhead*.

Настройка производительности последовательной отложенной записи

Алгоритм отложенной записи асинхронно выгружает измененные страницы из оперативной памяти на диск после достижения порогового значения, не дожидаясь команды демона **syncd**.

За счет этого уменьшается число страниц в памяти, ожидающих выгрузки, снижается общая нагрузка на систему и уменьшается фрагментация диска. Существует два вида отложенной записи: для последовательного и для неупорядоченного доступа.

Последовательная отложенная запись:

Если были изменены все 4 страницы кластера, то при изменении страницы в другом кластере планируется выгрузка исходного кластера на диск. Если функция отложенной записи не применяется, то страницы остаются в памяти до вызова демона **syncd**, что приводит к резкому увеличению нагрузки на диск и фрагментации файла.

По умолчанию файл JFS делится на разделы размером по 16 КБ или 4 страницы. Такие разделы называются *кластерами*.

Вы можете изменить число кластеров, применяемых VMM в качестве порогового значения. Значение по умолчанию равно единице. Для того чтобы отложенная запись выполнялась позднее, увеличьте значение *numclust* с помощью команды **ioo -o** *numclust*.

Для расширенного JFS команда **ioo -o** *j2_nPagesPerWriteBehindCluster* задает количество страниц, а не количество кластеров. Для расширенного JFS число страниц по умолчанию равно 32, а размер страницы по умолчанию - 128 КБ.

Произвольная отложенная запись:

В алгоритме отложенной записи предусмотрена возможность ограничения числа измененных страниц файла в памяти, при достижении которого планируется выгрузка измененных страниц на диск.

Некоторые приложения выполняют большое число операций ввода-вывода с неупорядоченным доступом к данным. Такой способ доступа несовместим с алгоритмом отложенной записи, поэтому все измененные страницы остаются в памяти до вызова демона **syncd**. Если в оперативной памяти находится много страниц, измененных приложением, то при вызове функции **sync()** демоном **syncd** на диск будет одновременно выгружено большое число страниц.

Для JFS это пороговое значение можно изменить с помощью команды **ioo** с параметром *maxrandwrt*. Значение по умолчанию, равное нулю, указывает, что функция произвольной отложенной записи отключена. Если этому параметру будет присвоено значение 128, то все измененные страницы файла, начиная со 129, будут выгружаться на диск. Первая группа страниц будут выгружена после вызова **sync()**.

Для расширенного JFS управление отложенной записью при прямом доступе к данным осуществляется с помощью опций **j2_nRandomCluster** (флаг **-z**) и **j2_maxRandomWrite** (флаг **-J**) команды **ioo**. По умолчанию значения этих опций равны 0. Опция **j2_maxRandomWrite** расширенного JFS равносильна опции *maxrandwrt* JFS. Эта опция задает максимальное количество измененных страниц файла в оперативной памяти. Опция **j2_nRandomCluster** указывает, при каком количестве кластеров между двумя последовательными операциями записи эти операции будут считаться операциями с неупорядоченным доступом.

Настройка производительности асинхронного дискового ввода-вывода

При выполнении операции синхронного ввода-вывода управление будет возвращено приложению только после завершения операции. Операции асинхронного ввода-вывода выполняются в фоновом режиме параллельно с приложением. За счет этого сокращается время выполнения приложения. Фоновое выполнение операций ввода-вывода увеличивает производительность таких приложений, как базы данных и файловые серверы.

Для выполнения асинхронного ввода-вывода предназначены функции **aio_read()**, **aio_write()** и **lio_listio()** (а также их 64-разрядные версии). Эти функции возвращают управление приложению, как только запрос помещается в очередь. В итоге приложение может продолжать работу одновременно с выполнением операции дискового ввода-вывода.

Для каждого запроса на асинхронный ввод-вывод в адресном пространстве приложения создается управляющий блок. Этот блок содержит управляющую информацию и информацию о состоянии запроса. После выполнения операции ввода-вывода этот блок может быть использован для обработки следующего запроса.

Приложение пользователя может задать следующие способы его оповещения при завершении операции ввода-вывода:

- Приложение может периодически считывать информацию о состоянии операции ввода-вывода.
- Система может отправить приложению уведомление о выполнении операции ввода-вывода.
- Приложение может приостановить выполнение основных операций до завершения операции ввода-вывода.

Каждая операция ввода-вывода управляется отдельным процессом ядра (kproc). Как правило, этот процесс не может обрабатывать другие запросы из очереди до завершения обработки текущего запроса. значение переменной `minservers` по умолчанию - 3, а `maxservers` - 30. По умолчанию значение атрибута `maxservers` равно числу асинхронных процессов ввода-вывода на один процессор. Чтобы определить максимальное число асинхронных процессов ввода-вывода kprocs в системе AIX, нужно умножить значение `maxservers` на количество процессоров.

Все переменные асинхронного ввода-вывода имеют текущее значение, значение по умолчанию, минимальное и максимальное значения, которые можно просматривать с помощью команды **ioo**. Только текущее значение можно изменять с помощью команды **ioo**. Остальные три значения изменять нельзя. Они служат для сообщения пользователю границ переменной. Текущее значение можно менять, а также сделать его постоянным вне зависимости от перезапусков системы. Если в системе редко запускаются приложения, выполняющие асинхронный ввод-вывод, то рекомендуется оставить значения по умолчанию.

Важно отметить, что `minservers` и `maxservers` являются переменными препроцессора. Обе эти переменные являются динамическими, но изменение их значений не вызывает синхронного изменения числа доступных серверов в системе. Если увеличивается значение `minservers`, фактическое число серверов увеличивается в соответствии с числом параллельных запросов ввода-вывода. Значение `minservers` является ограничением. В отличие от этого, число доступных серверов уменьшается после их остановки в нормальном режиме при уменьшении значения `minservers`. Если число запросов на асинхронный ввод-вывод велико, увеличьте значение `maxservers` таким образом, чтобы оно примерно совпадало с числом одновременно выполняемых запросов. Как правило, значение `minservers` изменять не нужно, так как расширение ядра, отвечающее за асинхронный ввод-вывод, автоматически создает дополнительные серверы по мере необходимости.

Примечание: Процессы асинхронного ввода-вывода, выполняемые для логического тома с прямым доступом или файлов, открытых в режиме CIO, не используют процессы сервера процессора. Параметры `maxservers` и `minservers` не влияют на выполнение таких операций.

Просмотрите информацию об использовании процессора серверами асинхронного ввода-вывода. Если серверы в равной мере используют процессор, то все серверы заняты. В этом случае рекомендуется увеличить число серверов. Для того чтобы получить информацию об использовании процессора отдельными серверами, запустите команду **pstat -a**. Кроме того, можно вызвать команду **ps -k**. В выводе этой команды все серверы асинхронного ввода-вывода указаны под именем `kproc`.

Если система должна быстро обрабатывать большое число операций асинхронного ввода-вывода, однако число одновременно выполняемых операций невелико, то рекомендуется присвоить параметру `maxservers` значение не ниже 10*(число дисков, к которым обращаются в асинхронном режиме).

Примечание: Для вступления в силу значений переменных `minservers` или`maxservers` перезагрузка не требуется. Значение переменной `minservers` должно быть задано для наилучшей производительности с учетом средней загрузки.

Значение переменной `minservers` не может превышать `maxservers`.

Дополнительная информация о значениях переменных асинхронного ввода-вывода приведена в разделе Изменение значений переменных для асинхронного ввода-вывода.

Настройка производительности синхронизации файлов

Синхронизацию файлов можно выполнить несколькими способами.

Файловые операции ввода-вывода JFS, которые не накапливаются последовательно в памяти при соблюдении определенных условий:

- Список свободных страниц сокращается *minfree*, и осуществляется замена страницы.
- Демон **syncd** вызвал команду выгрузки страниц на диск.
- Вызвана команда **sync**.
- Функция произвольной отложенной записи выгружает страницы на диск, так как достигнут порог выгрузки.

При накапливании множества страниц до выполнения одного из этих условий, когда страницы выгружены демоном **syncd**, блокировка i-узла получена и блокирована до тех пор, пока все измененные страницы будут записаны на диск. В течение этого времени попытки потоков получить доступ к этому файлу блокируются из-за недоступности блокировки i-узла. Команда **fuser** также блокируется для файла, так как для предоставления требуемой информации ей требуется блокировка i-узла. Демон **syncd** выгрузит все измененные страницы файла. Если в системе с большим объемом оперативной памяти накопится много измененных страниц, то при выполнении демона **syncd** резко возрастет число операций ввода-вывода.

В AIX можно настроить опцию **sync_release_ilock**. Команда **ioo** с опцией **-o sync_release_ilock=1** позволяет разрешить освобождение блокировки i-узла во время выгрузки измененных страниц файла. Эта опция позволяет уменьшить время ответа при обращении к файлу во время работы функции **sync()**.

Для того чтобы уменьшить негативный эффект, связанный с блокировкой i-узла, увеличьте частоту, с которой демон **syncd** вызывает функцию sync. Измените `/sbin/rc.boot`, который запускает демона **syncd**. Для того чтобы внесенные изменения вступили в силу, перезагрузите систему. Для того чтобы изменить это значение в работающей системе, завершите работу демона **syncd** и перезапустите его, указав новый интервал запуска функции.

Кроме того, вы можете изменить параметры функции отложенной записи, применяемой при произвольном доступе, с помощью команды **ioo** (за дополнительной информацией обратитесь к разделу “Настройка производительности последовательной отложенной записи” на стр. 247).

Переменные синхронизации JFS2

Операция синхронизации файловой системы не может быть эффективна при случайной операции ввода-вывода большого файла. При выполнении синхронизации блокируются все операции чтения и записи в файл из пользовательских программ. Для многих измененных страниц в файле требуется значительное количество времени для завершения записи на диск. В таких случаях могут использоваться следующие параметры настройки JFS2:

- `j2_syncPageCount`: Ограничивает количество измененных страниц, которые планируется записать в файл в один проход при синхронизации. При установке этой переменной файловая система записывает указанное количество страниц без блокировки ввода-вывода остальной части файла. Вызов синхронизации повторяет операции записи до тех пор, пока все измененные страницы не будут записаны.
- `j2_syncPageLimit`: Переопределяет параметр `j2_syncPageCount` при достижении порогового значения. Этот параметр обеспечивает выполнение операции синхронизации для файла. Переменные сохраняются стандартным способом.

Переменные обрабатываются с помощью команды **ioo**.

Переменные `j2_syncPageCount` и `j2_syncPageLimit` добавляются в список значений, которые управляются командой **ioo**.

Можно использовать флаг -o для отображения или изменения значения и флаг -h для просмотра справки.

```
# ioo -h j2_syncPageCount
```

Задает максимальное количество измененных страниц файла, которые записываются на диск, путем синхронизации системного вызова за одну операцию.

Значения: Значение по умолчанию: 0 Диапазон: 0-65536

Тип: Динамический

Единица измерения: Страницы 4 Кб

Настройка: При запуске приложения, которое использует кэширование файловой системы и большое число случайных операций записи, необходимо настроить этот параметр во избежании приостановки приложения во время операций синхронизации. Значения должны быть в интервале от 256 до 1024. Значение по умолчанию равно нулю, что задает обычную синхронизацию записи всех измененных страниц за один вызов. Задание небольших значений переменных приводит к увеличению времени синхронизации и сокращению времени отклика приложения. При задании больших значений время на отклик увеличивается, а время синхронизации уменьшается.

```
# ioo -h j2_syncPageLimit
```

Переменная j2_syncPageCount, задающая максимальное количество системных вызовов синхронизации, используется для ограничения количества записываемых страниц, что повышает производительность операции синхронизации.

Значения: Значение по умолчанию: 256 Диапазон: 16-65536

Тип: Динамический

Единица измерения: Число

Настройка: Имеет значение, если переменная j2_syncPageCount задана и может быть увеличено, если изменение j2_syncPageCount не дает должного эффекта. Допустимые значения должны быть в интервале от 250 до 8000. Переменная j2_syncPageLimit игнорируется, если значение переменной j2_syncPageCount равно 0.

Эта переменная может быть задана, если значение переменной j2_syncPageCount отлично от 0, и может быть увеличена таким образом, чтобы воздействие от изменения переменной j2_syncPageCount не привело к увеличению времени отклика приложения.

Значения должны быть в интервале от 1 до 8000. Оптимальные значения для этих переменных зависят от объема памяти и пропускной способности ввода/вывода. В качестве нейтрального можно использовать значение 256 для обеих переменных.

Период синхронизации и параллелизм JFS2

Синхронизацией файловой системы управляет демон sync (**syncd**). Настраиваемые параметры JFS2 позволяют файловой системе обрабатывать синхронизацию без применения команды **syncd**.

Обработчик синхронизации JFS2 распределяет процесс синхронизации таким образом, что все кэшированные данные не записываются на диск одновременно. Одновременно синхронизация может выполняться только для одной файловой системы. В каждой файловой системе запуск следующей операции синхронизации планируется после завершения предыдущей операции. Кроме того, можно увеличить число нитей, обрабатывающих операцию синхронизации нескольких файловых систем.

Следующие настраиваемые параметры применяются для синхронизации файловой системы JFS2:

j2_syncByVFS:
Разрешает применение обработчика синхронизации JFS2 и задает интервал между операциями синхронизации для каждой файловой системы.

j2_syncConcurrency:
Задает число нитей, обрабатывающих синхронизацию файловой системы. Это значение задает число файловых систем для параллельного выполнения операции синхронизации. В отдельной файловой системе операцию синхронизации выполняет только одна нить.

Команда **ioo** управляет параметрами ввода-вывода. Дополнительная информация приведена в описании команды **ioo**.

```
# ioo -h j2_syncByVFS
```

Назначение: задает число секунд ожидания между системными вызовами синхронизации в файловой системе JFS2. Это значение заменяет значение, указанное в команде **syncd**.

Значения: значение по умолчанию: 0, диапазон: 0-86400.

Тип: динамический

Единицы измерения: секунды.

Тонкая настройка: задает число секунд между итерациями процесса синхронизации. Обработчик синхронизации JFS2 ожидает в течение времени, указанного в параметре `j2_syncByVFS`, перед вызовом процедуры `syncvfs` для файловых систем JFS2. Значение 0 указывает, что следует использовать обычную обработку syncd. Отличное от нуля значение, переопределяет время, указанное в команде **syncd**, и разрешает применение обработчика синхронизации JFS2.

```
# ioo -h j2_syncConcurrency
```

Назначение: Задает число потоков для операций синхронизации JFS2.

Значения: значение по умолчанию: 1, диапазон: 1-128.

Тип: динамический

Единица измерения: число

Тонкая настройка: демон синхронизации запускает операции синхронизации параллельно для числа файловых систем, указанного в параметре `j2_syncConcurrency`. Это значение применяется только в том случае, если параметр `j2_syncByVFS` содержит значение, отличное от нуля.

Информация, связанная с данной:

Команда ioo

Тонкая настройка буферов файловой системы

Следующие параметры **ioo** и **vmstat -v** можно применять при выявлении узких мест буфера ввода-вывода и при тонкой настройке дисковых операций ввода-вывода:

Число операций ввода-вывода, заблокированных из-за нехватки буферов

Команда **vmstat -v** позволяет просмотреть количество операций ввода-вывода, заблокированных из-за нехватки буферов в различных компонентах ядра. Ниже приведен пример части вывода команды **vmstat -v**:

```
...
            0 блокированных операций ввода-вывода из области подкачки без psbuf
            2740 блокированных операций ввода-вывода файловых систем без fsbuf
            0 блокированных операций ввода-вывода для области подкачки внешних файловых систем без fsbuf

...
```

Значения числа блокированных операций ввода-вывода из области подкачки без psbuf и блокированных операций ввода-вывода файловых систем без fsbuf увеличиваются в случае, если объект bufstruct недоступен, и VMM помещает нить в список ожидания VMM. Значение числа блокированных операций ввода-вывода для области подкачки для внешних файловых систем увеличивается, если объект bufstruct или Расширенная файловая система JFS недоступны.

Параметр numfsbufs

Если к файловой системе одновременно поступает большое число обращений, либо из файловой системы последовательно считываются или записываются большие объемы данных, то операции ввода-вывода в этой файловой системе могут выполняться медленно из-за нехватки структур bufstruct. Число структур bufstruct, создаваемых для каждой файловой системы (параметр *numfsbufs*, можно увеличить с помощью команды **ioo**). Новое значение вступает в силу только при монтировании файловой системы. Следовательно, для применения нового значения нужно размонтировать и снова смонтировать файловую систему. По умолчанию значение параметра *numfsbufs* равно 93.

Параметр j2_nBufferPerPagerDevice

Примечание: Если vmstat -v указывает на нехватку структур buffstructs для JFS, то перед внесением изменений в параметр *j2_nBufferPerPagerDevice* необходимо выключить переменную *j2_dynamicBufferPreallocation*.

В расширенном JFS параметр *j2_nBufferPerPagerDevice* задает количество структур bufstruct. По умолчанию в файловой системе JFS2 создается 512 структур bufstruct. Число структур bufstructs, создаваемых для каждого расширенного JFS (*j2_nBufferPerPagerDevice*), можно изменить с помощью команды **ioo**. Значение применяется только при монтировании файловых систем.

Параметр lvm_bufcnt

Если приложение записывает большие объемы данных сразу на диск, минуя файловую систему, то операции ввода-вывода могут выполняться медленно из-за задержек, возникающих на уровне LVM. Это особенно характерно для случае чтения или записи большого объема данных при работе с диском с высокой производительностью. При возникновении задержек на уровне LVM рекомендуется увеличить значение параметра *lvm_bufcnt* с помощью команды **ioo**. Это значение задает число буферов uphysio. Изменение вступает в силу немедленно. По умолчанию число буферов uphysio равно 9. Поскольку LVM считывает и записывает данные блоками по 128 КБ, и значение *lvm_bufcnt* по умолчанию равно 9, то за одну операцию может быть записано 9*128 КБ данных. Если при операциях ввода-вывода объем передаваемых данных превышает 9*128 КБ, то увеличение значения *lvm_bufcnt* может привести к повышению производительности системы.

Параметр pd_npages

Параметр *pd_npages* задает число страниц, которые будут удалены из памяти за один раз при удалении файла. Изменять значение этого параметра имеет смысл только для приложений реального времени, удаляющих файлы. Уменьшение значения параметра *pd_npages* позволяет сократить время ответа в таких приложениях за счет того, что до возобновления работы нити или процесса система успеет удалить меньшее число страниц. По умолчанию значение этого параметра равно максимальному размеру файла, поделенному на размер страницы (4096). Если максимальный размер файла составляет 2 ГБ, то значение *pd_npages* по умолчанию равно 524288.

Параметр v_pinshm

Если параметр *v_pinshm* равен 1, и при вызове функции **shmget()** был задан флаг SHM_PIN, то страницы в общем сегменте памяти будут закреплены. Значение по умолчанию равно 0.

В приложении может быть предусмотрен параметр, указывающий, нужно ли задавать флаг SHM_PIN (например, в Oracle 8.1.5 и выше для этого предусмотрен параметр *lock_sga*). Не рекомендуется закреплять большое число страниц, поскольку это уменьшает доступную системе память. Однако, закрепление страниц позволяет повысить эффективность выполнения асинхронных операций ввода-вывода при работе с общими сегментами памяти (для закрепления буфера в памяти расширение асинхронного ввода-вывода ядра не требуется).

Настройка прозрачного ввода-вывода

Основное преимущество прозрачного ввода-вывода состоит в том, что на выполнение операций чтения и записи файлов затрачивается меньше процессорного времени. Это связано с тем, что данные не копируются из кэша файлов VMM в пользовательский буфер.

Обычно при обращении к файлу данные передаются между буфером приложения и носителем через VMM. Часть оперативной памяти применяется VMM для кэширования. Если число попаданий в кэш файлов велико, то кэширование значительно повысить производительность ввода-вывода. Однако если число попаданий невелико, либо приложение выполняет большое число операций ввод-вывода, то применение кэша не ускоряет обработку запросов на ввод-вывод.

Если число попаданий в кэш невелико, то при выполнении большинства запросов на чтение происходит обращение к диску. Операции записи, как правило, выполняются быстрее с применением кэша. Однако если при открытии файла была указана опция O_SYNC или O_DSYNC (см. “Использование вызовов sync и fsync” на стр. 212), то при выполнении операций записи в обычном режиме происходит обращение к диску. Приложения могут указывать эту опцию для устранения лишнего копирования данных.

Кроме того, применение прозрачного ввода-вывода позволяет повысить эффективность использования кэша для других файлов. При кэшировании считываемых или записываемых данных из кэша могут быть удалены другие файлы. Если разработчик приложения точно знает, к каким файлам невыгодно обращаться через кэш, то только эти файлы могут открываться с опцией O_DIRECT.

Для того чтобы операции прозрачного ввода-вывода выполнялись быстро, запрос на ввод-вывод должен соответствовать типу применяемой файловой системы. Функции **finfo()** и **ffinfo()** позволяют узнать смещение указателя, размер и способ выравнивания адресов, применяемые в файловых системе с фиксированным размером блока, фрагментированных файловых системах и файловых системах для больших файлов (прозрачный ввод-вывод не поддерживается для файловых систем со сжатием данных). Эта информация содержится в структуре **diocapbuf**, описанной в файле `/usr/include/sys/finfo.h`.

Если поступило сразу несколько запросов на открытие файла, но не во всех из них была задана опция O_DIRECT, то во избежание конфликтов для файла устанавливается обычный режим кэшированного ввода-вывода. Аналогично, если файл размещен в памяти с помощью системного вызова **shmat()** или **mmap()**, то для него будет установлен режим кэшированного ввода-вывода. После того как файл будет закрыт последним приложением, открывшим файл в обычном режиме ввода-вывода, файловая система установит для файла режим прозрачного ввода-вывода (с помощью функции **close()**, **munmap()** или **shmdt()**). На переключение из обычного режима в режим прозрачного ввода-вывода затрачивается много ресурсов системы, так как все измененные страницы файла, находящиеся в оперативной памяти, должны быть выгружены на диск.

Для выполнения операций в режиме прозрачного ввода-вывода требуется значительно меньше процессорного времени, чем для выполнения операций в обычном режиме. Режим прозрачного ввода-вывода позволяет повысить эффективность работы приложений, если в уровень попаданий в кэш в обычном режиме мал. Ценность прозрачного ввода-вывода растет с увеличением разрыва между скоростями работы процессора и памяти.

Кандидатами на применение прозрачного ввода-вывода прежде всего будут программы, выполняющие большое число операций дискового ввода-вывода, но ограниченные ресурсами процессора. Например, это могут быть программы, выполняющие большое число операций ввода-вывода с последовательным доступом к данным. Режим прозрачного ввода-вывода не рекомендуется применять для приложений, считывающих и записывающих небольшие объемы данных, так как в этом режиме не применяются алгоритмы упреждающего чтения и отложенной записи. Режим прозрачного ввода-вывода может применяться и в приложениях, работающих с логическими томами с чередованием данных.

Производительность чтения при использовании прозрачного ввода-вывода:

Хотя применение прозрачного ввода-вывода позволяет снизить нагрузку на процессор, операции ввода-вывода в таком режиме обычно выполняются дольше, особенно если считываются и записываются небольшие объемы данных. Это связано с тем, что результаты выполнения запросов не заносятся в кэш.

В режиме прозрачного ввода-вывода данные считываются с диска в синхронном режиме, тогда как в случае кэшированного ввода-вывода необходимые данные могут быть найдены в кэше. Следовательно, режим прозрачного ввода-вывода неэффективно применять для чтения данных, которые в обычном режиме с большой вероятностью находились бы в кэше. Кроме того, в режиме прозрачного ввода-вывода не применяется алгоритм упреждающего чтения VMM, так как запросы на ввод-вывод не передаются VMM. Алгоритм упреждающего чтения позволяет значительно повысить эффективность последовательного доступа к файлам, так как VMM заранее считывает необходимые страницы с диска и загружает их в оперативную память. Компенсировать отсутствие функции упреждающего чтения можно следующими способами:

- Увеличить объем данных, считываемых за один раз (объем данных должен составлять не менее 128 КБ)
- Выполнять упреждающее чтение в асинхронном режиме с помощью нескольких нитей
- Воспользоваться функциями асинхронного ввода-вывода, например, **aio_read()** и **lio_listio()**

Производительность записи при использовании прозрачного ввода-вывода:

В режиме прозрачного ввода-вывода запросы на запись не передаются VMM. Вместо этого сразу же происходит обращение к диску. Это приводит к значительному увеличению времени выполнения операций записи, так как в режиме кэшированного ввода-вывода данные записываются в оперативную память, а позднее выгружаются на диск операцией *sync*, или при выполнении *отложенной записи*.

Поскольку данные, записываемые в режиме прозрачного ввода-вывода, не копируются в оперативную память, при выполнении синхронизации не нужно выгружать соответствующие страницы памяти на диск. За счет этого сокращается объем работы, выполняемый демоном **syncd**.

Реорганизация протоколов файловой системы и логических томов протоколов

Для поддержания согласованной структуры файловой системы в JFS или расширенном JFS (JFS2) ведется протокол, аналогичный журналу базы данных. В этот протокол заносится информация обо всех операциях, выполненных над метаданными файловой системы. К метаданным относится суперблок, i-узлы, косвенные ссылки на данные и каталоги.

Когда страницы выгружаются из оперативной памяти на диск с помощью функции **sync()** или **fsync()**, в протокол заносятся записи о фиксации с информацией о том, что теперь данные сохранены на диске. Операции, при которых выполняются действия над протоколом:

- Создание или удаление файла.
- Вызов функции **write()** для файла, открытого с опцией O_SYNC, если во время записи был выделен новый блок диска.
- Вызов функций **fsync()** и **sync()**.
- Операции записи, при выполнении которых был создан блок первого или второго уровня со ссылками на другие блоки.

Протоколы файловых систем позволяют быстро восстановить файловые системы после сбоя системы. Если приложение в течение короткого интервала времени выполняет много операций синхронного ввода-вывода, либо часто создает и удаляет файлы, то в протокол заносится много записей. Если логический том протокола и логический том файловой системы расположены на одном диске, то из-за большого числа обращений к этому диску производительность может снизиться. В этом случае рекомендуется (особенно для сервера NFS) разместить протокол файловой системы на другом диске.

Для повышения производительности логические тома протоколов (файловых систем и баз данных) рекомендуется размещать на устройствах с кэшем быстрой записи.

В AIX предусмотрена опция **nointegrity** команды **mount**. При указании этой опции при монтировании файловой системы информация об этой файловой системе не будет заноситься в протокол JFS. Если администратор знает, что в случае сбоя системы для файловой системы достаточно вызвать команду **fsck**, то рекомендуется указать эту опцию при монтировании.

Для записи информации об операциях ввода-вывода в протокол файловой системы служит команда **filemon**. Если файловая система и устройство протокола сильно загружены, то рекомендуется разместить их на разных дисках (при условии, что в группу томов входит несколько дисков).

В группе томов можно создать несколько устройств протокола. Однако, протокол файловой системы всегда должен располагаться в той же группе томов, что и файловая система. Логический том протокола или файловой системы можно переместить на другой диск с помощью команды **migratepv**. Это можно сделать даже во время работы системы.

Понятия, связанные с данным:

“Имя группы томов” на стр. 207
Для повышения производительности и упрощения администрирования системы группа томов по умолчанию, rootvg, должна по возможности состоять из одного физического тома, на который первоначально была установлена операционная система.

Создание логических томов для протокола

Для того чтобы ресурсы системы могли использоваться параллельно, разместите логический том наиболее активно используемой файловой системы и логический том протокола этой системы на разных физических томах. Для каждой файловой системы можно создать отдельный протокол.

При создании логических томов эффективность работы с дисками изменяется. Логический том активно используемой файловой системы рекомендуется разместить на дисках с высокой производительностью (например, на дисках с кэшем быстрой записи), как описано ниже:

1. Создайте новый логический том протокола файловой системы:
   ```
   # mklv -t протокол -y логический-том группа-томов 1 физический-том
   ```
   или
   ```
   # mklv -t jfs2log -y логический-том группа-томов 1 физический-том
   ```
   или
   ```
   # smitty mklv
   ```
2. Отформатируйте протокол:
   ```
   # /usr/sbin/logform -V тип-vfs /dev/логический-том
   ```
3. Измените файл /etc/filesystems и управляющий блок логического тома (LVCB):
   ```
   # chfs -a log=/dev/логический-том /файловая-система
   ```
4. Размонтируйте, а затем вновь смонтируйте файловую систему.

Ниже описан другой способ создания протокола на отдельном томе:

- Определите группу томов, содержащую один физический том.
- Определите логический том в новой группе томов (в этом случае протокол JFS группы томов будет размещен на первом физическом томе).

- Добавьте оставшиеся физические тома в группу томов.
- Разместите на добавленных физических томах наиболее активно используемые файловые системы.

Ограничение дискового ввода-вывода

Ограничение дискового ввода-вывода позволяет предотвратить ситуацию, когда одна программа создает очень длинную очередь вывода, обработка которой занимает все ресурсы устройств ввода-вывода и увеличивает время ответа для других программ.

Функция ограничения дискового ввода-вывода устанавливает верхнюю и нижнюю границы для количества ожидающих операций ввода-вывода, относящихся к одному сегменту, или, фактически, к одному файлу. Если процесс пытается выполнить запись в файл, количество ожидающих операций ввода-вывода для которого уже находится на *верхней границе*, система приостанавливает его выполнение до тех пор, пока количество ожидающих операций не достигнет *нижней границы*. Логическая схема обработки запросов на операции ввода-вывода не изменяется. Единственный эффект состоит в увеличении задержек при выполнении большого числа операций ввода-вывода.

Для того чтобы задать верхнюю и нижнюю границы для числа ожидающих операций ввода-вывода, выберите в меню SMIT **Среда системы -> Изменить свойства операционной системы** (**smitty chgsys**) и укажите число страниц для верхней и нижней границ, а для отдельных файловых систем используйте опции монтирования **maxpout** и **minpout**.

Параметр **maxpout** задает число страниц, которое можно запланировать для файла в состоянии ввода-вывода перед остановкой нитей. Параметр **minpout** задает минимальное число запланированных страниц, при котором нити пробуждаются из приостановленного состояния. Значение по умолчанию для **maxpout** равно 8193, а для **minpout** - 4096. Для выключения ограничения ввода-вывода, задайте для этих параметров значение 0.

Изменение значений параметров **maxpout** и **minpout** для системы вступают в силу немедленно без перезагрузки системы. Измененные значения параметров **maxpout** и **minpout** перезаписывают системные параметры. Можно исключить файловую систему из системного ограничения ввода-вывода, смонтировав файловую систему и установив значения параметров **maxpout** и **minpout** равными 0. Далее приведен пример такой команды:

```
mount -o minpout=0,maxpout=0 /<файловая система>
```

Настройка параметров **maxpout** и **minpout** может предотвратить поглощение системных ресурсов какой-либо нитью, выполняющей последовательные операции записи в файл.

В следующей таблице представлено время ответа сеанса редактора **vi** на сервере IBM eServer pSeries модели 7039-651, настроенном как 4-процессорная система с процессором 1,7 ГГц, в зависимости от значений параметров **maxpout** и **minpout** при записи на диск:

Значение maxpout	Значение minpout	Размер блока dd (10 ГБ)	Время записи (с)	Производительность (МБ/с)	Комментарии vi
0	0	10000	201	49,8	После завершения dd
33	24	10000	420	23,8	Без задержки
65	32	10000	291	34,4	Без задержки
129	32	10000	312	32,1	Без задержки
129	64	10000	266	37,6	Без задержки
257	32	10000	316	31,6	Без задержки
257	64	10000	341	29,3	Без задержки
257	128	10000	223	44,8	Без задержки
513	32	10000	240	41,7	Без задержки
513	64	10000	237	42,2	Без задержки

Значение maxpout	Значение minpout	Размер блока dd (10 ГБ)	Время записи (с)	Производительность (МБ/с)	Комментарии vi
513	128	10000	220	45,5	Без задержки
513	**256**	**10000**	**206**	**48,5**	**Без задержки**
513	384	10000	206	48,5	3 - 6 секунд
769	512	10000	203	49,3	15-40 секунд, может быть больше
769	640	10000	207	48,3	Меньше 3 секунд
1025	32	10000	224	44,6	Без задержки
1025	64	10000	214	46,77	Без задержки
1025	128	10000	209	47,8	Менее 1 секунды
1025	256	10000	204	49,0	Менее 1 секунды
1025	384	10000	203	49,3	3 секунды
1025	512	10000	203	49,3	25-40 секунд, может быть больше
1025	640	10000	202	49,5	7 - 20 секунд, может быть больше
1025	768	10000	202	49,5	15 - 95 секунд, может быть больше
1025	896	10000	209	47,8	3 - 10 секунд

Диапазон наилучших значений параметров **maxpout** и **minpout** зависит от быстродействия CPU и системы ввода-вывода. Ограничение ввода-вывода эффективно с том случае, если значение параметра **maxpout** равно или превосходит значение параметра **j2_nPagesPerWriteBehindCluster**. Например, если значение параметра **maxpout** равно 64, а параметра **minpout** - 32, максимальное число страниц в состоянии ввода-вывода равно 64, и существует 2 ввода-вывода перед блокированием следующей операции записи.

Настраиваемые параметры имеют следующие значения по умолчанию:

Параметр	Значение по умолчанию
j2_nPagesPerWriteBehindCluster	32
j2_nBufferPerPagerDevice	512

Для расширенного JFS команда **ioo -o j2_nPagesPerWriteBehindCluster** задает количество страниц, запланированных на одно время. Для расширенного JFS число страниц по умолчанию равно 32, а размер страницы по умолчанию - 128 КБ. С помощью команды **ioo -o j2_nBufferPerPagerDevice** можно задать количество страниц буферных структур файловой системы. Значение по умолчанию равно 512. Для того чтобы изменения вступили в силу, следует повторно смонтировать файловую систему.

В случае расширенной JFS с помощью команды **-o remount** можно изменить значения параметров **maxpout** и **minpout** смонтированной файловой системы.

Быстродействие сети

AIX поддерживает несколько различных протоколов для взаимодействия, а также средства их мониторинга и настройки.

Настройка производительности TCP и UDP

Оптимальные значения настраиваемых параметров связи зависят от типа локальной сети и особенностей работы основных системных программ и приложений. В этом разделе описаны общие принципы настройки параметров связи для операционной системы AIX.

Ниже приведен план проверки и настройки параметров сети и рабочей схемы:

- Убедитесь, что адаптеры установлены в правильные разъемы

- Убедитесь, что установлен правильный выпуск встроенного программного обеспечения
- Убедитесь, что для адаптеров и сетевых коммутаторов правильно заданы параметры быстродействия и дуплексного режима
- Проверьте правильность выбранного размера MTU
- Настройте параметры AIX, связанные с типом сети, быстродействие и протоколом
- Другие особенности:
  - Опции разгрузки адаптера
    - Аппаратная проверка контрольной суммы TCP
    - Отправка больших пакетов TCP или повторное разбиение
  - Объединение прерываний
  - Входные нити

Установка адаптера

Производительность сети зависит от характеристик выбранного аппаратного обеспечения (например, типа адаптера) и его физического расположения в системе.

Для обеспечения максимальной производительности сетевые адаптеры следует устанавливать в разъемы шины ввода-вывода, наилучшим образом подходящие для них.

При определении наиболее подходящего разъема шины ввода-вывода необходимо принять во внимание следующее:

- Адаптеры PCI-X по сравнению с адаптерами PCI
- 64-разрядные адаптеры по сравнению с 32-разрядными адаптерами
- Поддерживаемые частоты разъемов шины (33 МГц, 50/66 МГц или 133 МГц)

С возрастанием быстродействия адаптера или его пропускной способности все большую роль начинает играть выбор разъема для установки адаптера. Например, максимальная производительность адаптеров PCI-X достигается в случае их установки в разъемы PCI-X, так как их стандартная тактовая частота составляет 133 МГц. Адаптеры PCI-X можно устанавливать и в обычные разъемы PCI, но в этом случае тактовая частота снижается до 33 МГц или 66 МГц, что может быть недостаточно для некоторых рабочих схем.

Аналогичным образом 64-разрядные адаптеры работают наиболее эффективно в 64-разрядных разъемах. 64-разрядные адаптеры также можно устанавливать в 32-разрядные разъемы, но при этом их быстродействие снижается. Адаптеры с большим MTU, такие как Gigabit Ethernet в режиме больших кадров, наиболее эффективно работают в 64-разрядных разъемах.

Кроме того, производительность может зависеть от числа адаптеров, подключенных к одной шине или мосту хоста PCI (PHB). В зависимости от модели системы или типа адаптеров число высокоскоростных адаптеров на один PHB может быть ограничено. Рекомендации по установке предусматривают распределение адаптеров между несколькими шинами PCI, а также ограничение числа адаптеров, подключенных к одой шине PCI. За дополнительной информацией о конкретных модели системы и типе адаптера обратитесь к *Справочнику по установке адаптеров PCI*.

В следующей таблице перечислены разъемы PCI и PCI-X, предусмотренные в серверах IBM System p.

Тип разъема	Код, применяемый в данном разделе
PCI, 32-разрядный, 33 МГц	A
PCI, 32-разрядный, 50/66 МГц	B
PCI, 64-разрядный, 33 МГц	C
PCI, 64-разрядный, 50/66 МГц	D
PCI-X, 32-разрядный, 33 МГц	E
PCI-X, 32-разрядный, 66 МГц	F
PCI-X, 64-разрядный, 33 МГц	G
PCI-X, 64-разрядный, 66 МГц	H
PCI-X, 64-разрядный, 133 МГц	I

Современные серверы на основе процессоров IBM Power Systems оснащаются только разъемами PCI-X. Разъемы PCI-X допускают установку адаптеров PCI.

В следующей таблице приведены примеры адаптеров и рекомендуемых для них разъемов:

Тип адаптера	Предпочитаемый тип разъема (по мере возрастания приоритета)
Адаптер Ethernet PCI II 10/100 Мбит/с (10/100 Ethernet), FC 4962	A-I
Адаптер ATM 155 Мбит/с IBM, FC 4953 или 4957	D, H и I
Адаптер 622 Мбит/с IBM PCI, FC 2946	D, G, H и I
Адаптер Gigabit Ethernet-SX PCI, FC 2969	D, G, H и I
Адаптер Ethernet PCI IBM 10/100/1000 Base-T, FC 2975	D, G, H и I
Адаптер Gigabit Ethernet-SX PCI-X (Gigabit Ethernet fibre), FC 5700	G, H и I
Адаптер PCI-X 10/100/1000 Base-TX (Gigabit Ethernet), FC 5701	G, H и I
Двухпортовый адаптер Gigabit Ethernet-SX PCI-X (Gigabit Ethernet fibre), FC 5707	G, H и I
Двухпортовый адаптер PCI-X 10/100/1000 Base-TX (Gigabit Ethernet), FC 5706	G, H и I
Адаптер 10 Gigabit-SR Ethernet PCI-X, FC 5718	I (только разъемы PCI-X 133)
Адаптер 10 Gigabit-LR Ethernet PCI-X, FC 5719	I (только разъемы PCI-X 133)

В отчете команды **lsslot -c pci** содержится следующая информация:

- Тип разъема PCI
- Быстродействие шины
- Распределение устройств по разъемам

Ниже приведен пример вывода команды **lsslot -c pci**, выполненной в двухпроцессорной системе p615 с шестью внутренними разъемами:

```
# lsslot -c pci
# Разъем    Описание                          Устройство
U0.1-P1-I1  Поддержка PCI-X, 64-разр., 133 МГц fcs0
U0.1-P1-I2  Поддержка PCI-X, 32-разр.,  66 МГц Пусто
U0.1-P1-I3  Поддержка PCI-X, 32-разр.,  66 МГц Пусто
U0.1-P1-I4  Поддержка PCI-X, 64-разр., 133 МГц fcs1
U0.1-P1-I5  Поддержка PCI-X, 64-разр., 133 МГц ent0
U0.1-P1-I6  Поддержка PCI-X, 64-разр., 133 МГц ent2
```

Для адаптеров Gigabit Ethernet информация о типе шины PCI и быстродействии указывается в конце вывода команды **entstat -d en***[номер-интерфейса]* или команды **netstat -v**. Ниже приведен пример вывода команды **netstat -v**:

```
# netstat -v

Статистика адаптера PCI-X 10/100/1000 Base-TX (14106902):
-------------------------------------------------------------------
Состояние соединения: Включено
Выбранная скорость передачи данных: Автоматическое согласование
Текущая скорость передачи данных: 1000 Мбит/с Дуплекс
Режим PCI: PCI-X (100-133)
Разрядность шины PCI: 64-разряда
```

Встроенное ПО системы

Встроенное программное обеспечение системы позволяет настраивать различные ключевые параметры адаптеров PCI, а также опции контроллеров ввода-вывода различных шин ввода-вывода и шин PCI.

В некоторых случаях встроенное программное обеспечение задает параметры, уникальные для конкретных адаптеров, такие как Таймер скрытого состояния PCI и Размер строки кэша, а для адаптеров PCI-X - Максимальное число байт для чтения из памяти (MMRBC). Эти параметры позволяют обеспечить высокую производительность адаптеров. Если из-за устаревшей версии встроенного программного обеспечения значения этих параметров указаны неправильно, то настройка программного обеспечения не позволит достичь максимальной производительности. Перед добавлением новых адаптеров в устаревшие системы рекомендуется обновить встроенное программное обеспечение.

Команда **lscfg -vp|grep -p " ROM"** позволяет просмотреть уровень встроенного программного обеспечения как платформы, так и системы. Ниже приведен пример вывода следующей команды:

```
lscfg -vp|grep -p " ROM"

     ...строки пропущены...

     Встроенное программное обеспечение системы:
        Уровень ПЗУ (изменяемый)....M2P030828
        Версия.....................RS6K
        Системная информация (YL)...U0.1-P1/Y1
      Физическое расположение: U0.1-P1/Y1

      Встроенное программное обеспечение SPCN:
        Уровень ПЗУ (изменяемый)....0000CMD02252
        Версия.....................RS6K
        Системная информация (YL)...U0.1-P1/Y3
      Физическое расположение: U0.1-P1/Y3

      Встроенное программное обеспечение SPCN:
        Уровень ПЗУ (изменяемый)....0000CMD02252
        Версия.....................RS6K
        Системная информация (YL)...U0.2-P1/Y3
      Физическое расположение: U0.2-P1/Y3

      Встроенное программное обеспечение платформы:
        Уровень ПЗУ (изменяемый)....MM030829
        Версия.....................RS6K
        Системная информация (YL)...U0.1-P1/Y2
      Физическое расположение: U0.1-P1/Y2
```

Оптимизация работы адаптера

В операционной системе AIX предусмотрено несколько процедур для оптимизации быстродействия адаптера.

Пропускную способность можно определить с помощью программ на основе сокетов для приложений, работающих с потоковыми данными по соединению TCP. Например, это может быть набор из двух программ, одна из которых вызывает **send( )** а вторая, получатель, - **recv( )**. Пропускная способность зависит от скорости передачи данных по сети, размера MTU (размера кадра), дополнительной нагрузки, такой как интервалы между кадрами, начальные биты, заголовки связи и заголовки TCP/IP. При расчете пропускной способности тактовая частота процессора считается равной одному гигагерцу. Таким образом пропускную

способность можно измерить в отдельных локальных сетях. Если информации проходит через маршрутизаторы, дополнительные узлы и удаленные соединения, то пропускная способность будет ниже.

Пропускная способность одностороннего (симплексного) потокового соединения TCP можно определить с помощью рабочей схемы, включающей отправку данных из системы A в систему B, например, с помощью протокола FTP в тесте передачи из памяти в память. За дополнительной информацией обратитесь к разделу “Команда ftp” на стр. 298. Дуплексный режим эффективнее полудуплексного, так как подтверждения TCP возвращаются независимо от передачи пакетов данных.

Примечание: В следующих таблицах значение **Скорость передачи данных** представляет собой физическую скорость передачи данных без учета дополнительной нагрузки, такой как интервалы между кадрами, начальные биты, дополнительная нагрузка пакетов ATM, заголовки связи и концевики. Эти механизмы снижают эффективную пропускную способность соединения.

В следующей таблице приведены максимальные значения пропускной способности сети, а также пропускной способности одностороннего (симплексного) потокового соединения TCP:

Таблица 5. Максимальная пропускная способность сети для симплексных потоков TCP

Тип сети	Скорость передачи потока (Мбит/с)	Пропускная способность (Мб)	Пропускная способность (МБ)
Ethernet 10 Мбит/с, полудуплексный	10	6	0,7
Ethernet 10 Мбит/с, дуплексный	10 (20 Мбит/с дуплексный)	9,48	1,13
Ethernet 100 Мбит/с, полудуплексный	100	62	7,3
Ethernet 100 Мбит/с, дуплексный	100 (200 Мбит/с дуплексный)	94,8	11,3
Ethernet 1000 Мбит/с, дуплексный, MTU 1500	1000 (2000 Мбит/с дуплексный)	948	113,0
Ethernet 1000 Мбит/с, дуплексный, MTU 9000	1000 (2000 Мбит/с дуплексный)	989	117,9
Ethernet 10 Гбит/с, дуплекс, MTU 1500 (с включенной поддержкой RFC1323)	10000	7200 (пиковое значение: 9415)[1]	858 (пиковое значение: 1122)[1]
Ethernet 10 Гбит/с, дуплекс, MTU 9000 (с включенной поддержкой RFC1323)	10000	9631 (пиковое значение: 9891)[1]	1148 (пиковое значение: 1179)[1]
FDDI, MTU 4352 (по умолчанию)	100	92	11,0
Режим асинхронной передачи (ATM) 155, MTU 1500	155	125	14,9
ATM 155, MTU 9180 (по умолчанию)	155	133	15,9
ATM 622, MTU 1500	622	364	43,4
ATM 622, MTU 9180 (по умолчанию)	622	534	63,6

[1] Значения в таблице отражают пропускную способность выделенных адаптеров в выделенных разделах. В таблице не представлена производительность адаптеров 10 Gigabit Ethernet в виртуальном адаптере Ethernet (в VIOS), в общих адаптерах Ethernet (SEA) или для общих разделов (общих LPAR), так как производительность зависит от переменных и параметров настройки, выходящих за рамки этой таблицы.

Двустороннее (дуплексное) потоковое соединение TCP позволяет одновременно передавать данные в обоих направления. Например, параллельное выполнение команды **ftp** в системе A и команды **ftp** в системе B может считаться дуплексной передачей TCP. Рабочая схема такого типа позволяет параллельно отправлять и принимать данные. Некоторые стандарты, такие как FDDI и Ethernet в полудуплексном режиме, не допускают параллельный прием и отправку данных. Дуплексная рабочая схема для таких сетей не рекомендуется. Дуплексная рабочая схема отличается от удвоенной симплексной рабочей схемы, так как

пакеты подтверждения TCP, возвращаемые получателем, передаются вместе с пакетами данных, передаваемыми в том же направлении. В следующей таблице приведены максимальные значения пропускной способности различных двусторонних (дуплексных) потоковых соединений TCP:

Таблица 6. Максимальная пропускная способность сети для дуплексных потоков TCP

Тип сети	Скорость передачи потока (Мбит/с)	Пропускная способность (Мб)	Пропускная способность (МБ)
Ethernet 10 Мбит/с, полудуплексный	10	5.8	0,7
Ethernet 10 Мбит/с, дуплексный	10 (20 Мбит/с дуплексный)	18	2,2
Ethernet 100 Мбит/с, полудуплексный	100	58	7,0
Ethernet 100 Мбит/с, дуплексный	100 (200 Мбит/с дуплексный)	177	21,1
Ethernet 1000 Мбит/с, дуплексный, MTU 1500	1000 (2000 Мбит/с дуплексный)	1811 (пиковое значение: 1667) [1]	215 (пиковое значение: 222) [1]
Ethernet 1000 Мбит/с, дуплексный, MTU 9000	1000 (2000 Мбит/с дуплексный)	1936 (пиковое значение: 1938) [1]	231 (пиковое значение: 231) [1]
Ethernet 10 Гбит/с, дуплексный, MTU 1500	10000 (20000 Мбит/с дуплексный)	14400 (пиковое значение: 18448) [1]	1716 (пиковое значение: 2200) [1]
Ethernet 10 Гбит/с, дуплексный, MTU 9000	10000 (20000 Мбит/с дуплексный)	18000 (пиковое значение: 19555) [1]	2162 (пиковое значение: 2331) [1]
FDDI, MTU 4352 (по умолчанию)	100	97	11,6
ATM 155, MTU 1500	155 (310 Мбит/с дуплексный)	180	21,5
ATM 155, MTU 9180 (по умолчанию)	155 (310 Мбит/с дуплексный)	236	28,2
ATM 622, MTU 1500	622 (1244 Мбит/с дуплексный)	476	56,7
ATM 622, MTU 9180 (по умолчанию)	622 (1244 Мбит/с дуплексный)	884	105

[1] Значения в таблице отражают пропускную способность выделенных адаптеров в выделенных разделах. В таблице не представлена производительность адаптеров 10 Gigabit Ethernet в виртуальном адаптере Ethernet (в VIOS), в общих адаптерах Ethernet (SEA) или для общих разделов (общих LPAR), так как производительность зависит от переменных и параметров настройки, выходящих за рамки этой таблицы.

Примечание:

1. Пиковые значения соответствуют максимальной пропускной способности, когда в каждом направлении выполняются несколько сеансов TCP. Остальные значения указаны для одиночных сеансов TCP. Значения одиночных сеансов различаются в зависимости от частоты процессора, конкретного адаптера и используемого типа разъема PCI.
2. Пропускная способность Ethernet 1000 Мбит/с (Gigabit Ethernet) в дуплексном режиме указана для разъемов адаптеров PCI-eXtended (PCI-X) или адаптеров PCIe. Для адаптеров PCI и адаптеров PCI-X, установленных в разъемах PCI, производительность будет ниже. Пропускная способность Ethernet 10 ГБ указана только для адаптеров PCIe.
3. Пропускная способность данных указана для протокола TCP/IP, использующего IPv4. Опция **RFC1323** включена для следующих адаптеров:
   - Для адаптеров, размер MTU которых равен 4096 или больше
   - Для адаптеров 10 Gigabit Ethernet или быстрее
4. В столбце Пропускная способность (Мбит/с) единицами измерения являются мегабиты в секунду, где мегабит - это 1000000 бит. В столбце Пропускная способность (МБ/с) единицами измерения являются мегабайты в секунду, где 1 МБ - это 1048576 байтов.

Параметры адаптера и устройства

Некоторые параметры адаптеров и устройств влияют как на правильную работу, так и на производительность системы.

Как правило для устройств AIX предусмотрены значения по умолчанию, обеспечивающие эффективную работу в большинстве случаев. Изменять эти значения не рекомендуется. Однако стратегии некоторых компаний предусматривают применение строго определенных сетевых параметров. Кроме того, изменение значений по умолчанию может быть связано с требованиями сетевого оборудования.

Параметры скорости адаптера и его режима

По умолчанию в операционной системе AIX применяется режим Auto_Negotiation, позволяющий согласовывать быстродействие и параметры дуплексного режима для обеспечения оптимальной пропускной способности. Для правильной работы режим Auto_Negotiation должен быть указан как для адаптера, так и для другой конечной точки соединения (коммутатора).

Для адаптеров Ethernet предусмотрены следующие режимы работы:

- 10_Half_Duplex
- 10_Full_Duplex
- 100_Half_Duplex
- 100_Full_Duplex
- Auto_Negotiation

Для правильной работы важно, чтобы адаптер и другая конечная точка соединения (как правило это коммутатор Ethernet или другой адаптер в двухточечной конфигурации без применения коммутаторов Ethernet) были настроены одинаковым образом. Если параметры быстродействия и дуплексного режима одной конечной точки соединения настроены вручную, соответствующие значения следует указать и для другой конечной точки. Если для одной конечной точки параметры заданы вручную, а для другой указан режим Auto_Negotiation, то как правило возникают неполадки, снижающие производительность соединения.

Так как большинство коммутаторов Ethernet по умолчанию применяют режим Auto_Negotiation, в большинстве случаев рекомендуется использовать именно его. Однако некоторые коммутаторы Ethernet 10/100 не поддерживают режим Auto_Negotiation в дуплексном режиме. Параметры быстродействия и дуплексного режима таких коммутаторов следует настраивать вручную.

Примечание: Адаптеры 10 Gigabit Ethernet не поддерживают режим Auto_Negotiation, потому что они работают только на одной скорости в оптической среде SR и LR.

Для просмотра и изменения параметров быстродействия и дуплексного режима применяются команды, уникальные для каждой модели коммутаторов Ethernet. Описание этих команд приведено в документации, поставляемой производителем коммутатора.

Для изменения параметров адаптера в операционной системе AIX предусмотрена команда **smitty devices**. Просмотреть параметры и режим согласования можно с помощью команды **netstat -v** или **entstat -d en***X*, где *X* - это номер интерфейса Ethernet. Ниже приведен фрагмент вывода команды **entstat -d en3**:

```
Статистика адаптера PCI-X 10/100/1000 Base-TX (14106902):
-------------------------------------------------------------
Состояние соединения: Включено
Выбранная скорость передачи данных: Автоматическое согласование
Текущая скорость передачи данных: 1000 Мбит/с Дуплекс
```

Параметр MTU адаптера

Размер Максимального блока передачи (MTU) должен совпадать для всех устройств, входящих в состав одной физической сети или логической сети (если применяется VLAN). Размер MTU задает максимальный размер пакетов, передаваемых по соединению.

Различные сетевые адаптеры поддерживают MTU разных размеров. Убедитесь, что размеры MTU всех устройств в сети совпадают. Например, адаптер Gigabit Ethernet не сможет работать в режиме больших пакетов с MTU 9000 байт, если для остальных адаптеров в сети указан размер MTU по умолчанию, равный 1500 байт. Адаптеры Ethernet 10/100 не поддерживают режим больших пакетов и не совместимы с этой

функцией Gigabit Ethernet. Кроме того, если один из коммутаторов Ethernet поддерживает режим больших пакетов, то его следует включить и на всех остальных коммутаторах Ethernet.

Размер MTU адаптера рекомендуется выбрать на ранних этапах настройки сети, чтобы можно было правильно настроить все устройства и коммутаторы. Кроме того, многие опции настройки AIX зависят от выбранного размера MTU.

Влияние размера MTU на производительность

Размер MTU в значительной степени влияет на производительность сети.

Большие размеры MTU позволяют операционной системе обеспечить заданную пропускную способность путем отправки меньшего числа пакетов. Большие пакеты позволяют значительно снизить нагрузку на операционную систему, если рабочая схема предусматривает отправку больших сообщений. Если рабочая схема предусматривает отправку только небольших сообщений, то увеличение размера MTU не приведет к повышению производительности.

В большинстве случаев рекомендуется применять наибольший размер MTU, поддерживаемый адаптерами и сетью. Например, для режима асинхронной передачи (ATM) размер MTU, по умолчанию равный 9180, гораздо эффективней размера MTU, равного 1500 (как правило применяется при эмуляции LAN). Если в сети применяются только адаптеры Gigabit и 10 Gigabit Ethernet, рекомендуется включить режим больших кадров. Например, большие кадры обычно применяются для взаимодействия серверов организации.

Включение режима больших кадров на Gigabit Ethernet

Режим больших кадров выбирается на уровне устройства.

Изменение размера MTU с помощью команды **ifconfig** в этом случае недопустимо. Для того чтобы изменить параметры адаптера с помощью программы SMIT, выполните следующие действия:

1. Выберите `Устройства`
2. Выберите `Средства связи`
3. Выберите `Тип адаптера`
4. Выберите `Показать или изменить параметры адаптера Ethernet`
5. Измените значение опции `Передавать большие пакеты` с `нет` на `да`

Ниже приведен пример меню SMIT:

```
                                Показать или изменить параметры адаптера Ethernet

Введите или выберите значения в полях ввода.
После внесения изменений нажмите клавишу Enter.

                                                [Поля ввода]
  Адаптер Ethernet                               ent0
  Описание                                       Адаптер PCI-X 10/100/1000 Base-TX (14106902)
  Состояние                                      Доступен
  Расположение                                   1H-08
  Размер очереди приема                         [1024]  +#
  Размер очереди передачи                       [512]   +#
  Размер очереди передачи программного обеспечения  [8192]  +#
  Передавать большие кадры                       да +
  Включить аппаратное разбиение на фрагменты     да +
  Включить аппаратную проверку контрольной суммы  да +
  Пропускная способность                         Auto_Negotiation +
  Включить альтернативный адрес ETHERNET         нет +
  Альтернативный адрес ETHERNET                 [0x000000000000] +
  Применить изменение только для базы данных     нет +

F1=Справка          F2=Обновить         F3=Отмена           F4=Список
Esc+5=Сбросить      Esc+6=Команда       Esc+7=Изменить      Esc+8=Изображение
Esc+9=Оболочка      Esc+0=Выход         Enter=Выполнить
```

Настройка производительности с помощью команды no

Сетевая опция или команда **no** позволяет просматривать и изменять глобальные сетевые опции, а также управлять ими.

Ниже перечислены опции команды **no**, позволяющие изменить параметры настройки:

Опция Определение

-a Показывает все переменные и их текущие значения.

-d [*переменная***]**
Восстанавливает значение по умолчанию для указанной переменной.

-D Восстанавливает значения по умолчанию для всех переменных.

-o *переменная***=[новое значение]**
Показывает значение указанной переменной, либо задает для нее новое значение.

-h [*переменная***]**
Выдает справочную информацию по указанной переменной, если она задана. В противном случае выдает справочную информацию по команде **no**.

-r Применяется вместе с опцией **-o** и позволяет указать в файле `nextboot` постоянное значение параметра типа перезагрузки.

-p Применяется вместе с опцией **-o** и позволяет сделать динамическую переменную из файла `nextboot` постоянной.

-L *[переменная]*
Применяется вместе с опцией **-o** для просмотра списка характеристик одной или всех переменных.

Ниже приведен пример вывода команды **no**:

```
NAME                      CUR    DEF    BOOT   MIN    MAX    UNIT           TYPE  DEPENDENCIES

--------------------------------------------------------------------------------------------------

Общие параметры сети

--------------------------------------------------------------------------------------------------
sockthresh                85     85     85     0      100    %_of_thewall      D
--------------------------------------------------------------------------------------------------
fasttimo                  200    200    200    50     200    миллисекунд       D
--------------------------------------------------------------------------------------------------
inet_stack_size           16     16     16     1             килобайт          R
--------------------------------------------------------------------------------------------------
...строки пропущены....


где:
CUR = текущее значение

DEF = значение по умолчанию

BOOT = значение опции перезагрузки

MIN = минимальное значение

MAX = максимальное значение

UNIT = единицы измерений

TYPE = тип параметра: D (динамический), S (статический), R (перезагрузка), B
(загрузка), M (монтирование),
                      I (дополняющий) или C (соединение)

DEPENDENCIES = построчный список зависимых настраиваемых параметров
```

Некоторые атрибуты сети можно изменить в любой момент. Другие атрибуты изменяются во время загрузки. Их нужно изменить до загрузки расширения ядра **netinet**.

Примечание: Изменения, внесенные с помощью команды **no**, действуют до следующей загрузки системы. Затем всем параметрам присваиваются новые значения. Опции **-r** и **-p** команды **no** позволяют сделать

динамические изменения постоянными, изменив файл nextboot. Для того чтобы параметры следующей загрузки вступили в силу, необходима перезагрузка системы.

Дополнительные сведения о команде **no** приведены в разделе The no Command книги *Справочник по командам, том 4*.

Режим fastpath для обратно передаваемых данных TCP

Опция протокола TCP fastpath используется для ускорения доставки обратно передаваемых данных.

Настраиваемый сетевой параметр tcp_fastlo позволяет сократить маршрут доставки обратно передаваемых данных TCP во всем стеке TCP/IP (протокол и интерфейс) и таким образом повысить быстродействие.

Этот параметр не требует изменения приложений. Если параметр активирован, обратно передаваемые данные TCP обрабатываются подобно доменам в UNIX.

Второй параметр, tcp_fastlo_crosswpar, позволяет передавать данные TCP напрямую между рабочими разделами (wpar). Если параметр tcp_fastlo_crosswpar используется, также должен быть включен параметр tcp_fastlo.

Для разрешения прямой доставки обратно передаваемых данных TCP введите команду **no**:

```
# no -o tcp_fastlo=1
```

Эта опция является динамической и действительна для последующих соединений TCP.

Для разрешения прямой доставки обратно передаваемых данных TCP между рабочими разделами (wpar) введите команду **no**:

```
# no -o tcp_fastlo_crosswpar=1
```

Примечание: По умолчанию параметры tcp_fastlo и tcp_fastlo_crosswpar выключены (установлены в 0). Они зарезервированы для последующих выпусков AIX.

По обратно передаваемым данным TCP, которые были доставлены в прямом режиме, ведется отдельная статистика, которую можно просмотреть с помощью команды **netstat**, когда открыто соединение TCP. Эти данные не включаются в интерфейс обратной передачи. Однако режим быстрой пересылки обратной передаваемых данных TCP для установки соединений использует протокол TCP/IP и устройство обратной передачи данных, поэтому пакеты таких данных включаются в общее количество переданной информации.

Уменьшение числа прерываний

Обработка прерываний занимает значительное число рабочих циклов процессора.

Для обработки прерывания системе необходимо сохранить параметры текущего состояния, определить источник прерывания, выполнить текущие служебные операции и вызвать обработчик прерываний соответствующего драйвера устройства. Драйвер, как правило, выполняет дополнительные операции высокого уровня, такие как чтение из реестра состояния прерывания адаптера, которое выполняется медленнее, чем аппаратные операции, а также блокировка SMP, получение и освобождение буферов, и т. д.

В большинстве драйверов устройств AIX не применяются прерывания, сообщающие о завершении передачи, что позволяет избежать возникновения прерываний при передаче пакетов. Завершение передачи, как правило, обрабатывается при следующей операции передачи данных, что позволяет не создавать прерывания для отдельных операций передачи. С помощью таких команд, как **netstat -v**, **entstat**, **atmstat** и **fddistat** можно просмотреть состояние счетчика переданных и полученных пакетов, а также количество прерываний для получаемых и отправляемых данных. Статистические данные наглядно демонстрируют отсутствие прерываний для операций передачи данных. Некоторые адаптеры и драйверы других фирм могут не поддерживать данное соглашение.

Включение сторожевой нити для адаптеров LAN

Если *применяются нити*, драйвер помещает принятый пакет в очередь нити, а затем нить вызывает уровень IP, TCP или сокета.

По умолчанию драйверы напрямую вызывают уровень IP, который обращается к уровню сокета в режиме прерывания. За счет этого сокращается число команд, но возрастает время обработки прерывания. В многопроцессорной системе один процессор может не справиться с обработкой пакетов, принимаемых быстрым адаптером. Такая нить может выполняться другими, менее загруженными процессорами. В некоторых случаях, например, при высокой частоте входящих пакетов, применение нитей позволяет повысить пропускную способность системы благодаря параллельному применению нескольких процессоров.

Функция обработки с помощью нитей имеет некоторые недостатки: при низкой загрузке ее применение приводит к увеличению задержек и повышенной загрузке процессора, поскольку пакет необходимо поместить в нить, а нить необходимо обработать.

Примечание: Эта функция не поддерживается в однопроцессорных системах, поскольку ее применение в таких системах приведет к снижению производительности за счет увеличения числа команд.

Эта функция действует только при приеме данных. Ее можно включить для интерфейса с помощью команды **ifconfig** (**ifconfig** *интерфейс* **нить** или **ifconfig** *интерфейс хост* **up нить**).

Для выключения этой функции вызовите команду **ifconfig** *интерфейс* **-thread**.

```
# ifconfig en0 thread

# ifconfig en0
en0: flags=5e080863,e0<UP,BROADCAST,NOTRAILERS,RUNNING,SIMPLEX,MULTICAST,GROUPRT,64BIT,CHECKSUM_OFFLOAD,PSEG,THREAD,CHAIN>
        inet 192.1.0.1 netmask 0xffffff00 broadcast 192.1.0.255

# ifconfig en0 -thread

# ifconfig en0
en0: flags=5e080863,c0<UP,BROADCAST,NOTRAILERS,RUNNING,SIMPLEX,MULTICAST,GROUPRT,64BIT,CHECKSUM_OFFLOAD,PSEG,THREAD,CHAIN>
        inet 192.1.0.1 netmask 0xffffff00 broadcast 192.1.0.255
```

Команда **netstat -s** также позволяет просмотреть некоторые счетчики и определить число пакетов, обработанных с помощью нитей, а также узнать, были ли какие-либо входящие пакеты отброшены нитями. Ниже приведен пример вывода команды **netstat -s**:

```
# netstat -s| grep hread

        352 пакета обработано нитями
        0 пакетов отброшено нитями
```

Ниже приведены некоторые рекомендации по применению нитей:

- Для применения этой функции число процессоров должно быть больше числа адаптеров. Рекомендуется, чтобы число процессоров было как минимум в два раза больше числа адаптеров.
- В системах с мощными процессорами эффект от применения этой функции меньше. Она в первую очередь предназначена для компьютеров с низкопроизводительными процессорами.
- Эта функция позволяет повысить производительность в том случае, когда система получает большое число пакетов. В сетях с MTU=1500 эта функция будет работать более эффективно, чем в сетях с MTU=9000 (большие кадры), так как в сетях с меньшим размером MTU число пакетов будет больше.

  Нити работают эффективнее, если их очередь все время заполнена, и им не нужно переходить в состояние ожидания получения ввода. В этом случае будут сэкономлены ресурсы, затрачиваемые драйвером на активизацию нитей.
- Нити уменьшают объем процессорного времени, затрачиваемого на обработку прерываний. В результате процессор быстрее возобновляет выполнение пользовательских задач.
- Если число пакетов недостаточно велико, для того чтобы очередь нити никогда не пустовала, то производительность может снизиться на 10 процентов. В этом случае в среднем 10 процентов процессорного времени будет затрачиваться на планирование запуска нитей.

Сетевые опции, связанные с интерфейсом

Набор параметров интерфейса (ISNO) позволяет пользователю изменить свойства отдельного интерфейса IP, влияющие на его производительность.

Значения этих параметров переопределяют значения параметров системы, заданных с помощью команды **no**. Применение таких параметров можно разрешить или запретить с помощью опции **use_isno** команды **no**. По умолчанию параметры отдельных интерфейсов настраивать разрешено. Опция, запрещающая применение ISNO, может применяться системным администратором в качестве средства диагностики для обнаружения ошибок в конфигурации.

Обратите внимание, что значения параметров ISNO, установленные для сокета, можно узнать (с помощью функции **getsockopt()**) только после установления соединения TCP. Интерфейс, с которым будет связан сокет, выбирается только при установлении соединения, поэтому до этого момента для сокета указываются значения по умолчанию, установленные с помощью команды **no**. Значения ISNO применяются для сокета после установления соединения TCP и определения сетевого интерфейса.

К параметрам сетевого интерфейса были добавлены следующие параметры, допустимые только для соединений TCP (не UDP):

- **rfc1323**
- **tcp_nodelay**
- **tcp_sendspace**
- **tcp_recvspace**
- **tcp_mssdflt**

Значения этих параметров переопределяют соответствующие общесистемные параметры, задаваемые командой **no**. Эти параметры поддерживаются всеми основными интерфейсами TCP/IP (Token-Ring, FDDI, 10/100 Ethernet и Gigabit Ethernet), за исключением интерфейса IP **css** # коммутатора SP. Пользователи коммутатора SP могут задать необходимые параметры на уровне системы командой **no**, а затем изменить значения для других сетевых интерфейсов с помощью параметров ISNO.

Данные параметры устанавливаются для интерфейса TCP/IP (например, en0 или tr0), а не для сетевого адаптера (ent0 или tok0).

AIX задает значения по умолчанию для интерфейсов Gigabit Ethernet как для MTU 1500, так и для режима больших кадров (MTU 9000). При настройке интерфейса с помощью меню **SMIT tcpip** для получения удовлетворительной производительности с использованием опций ISNO рекомендуется указывать значения по умолчанию.

Для адаптеров Ethernet 10/100 и Token-Ring опции ISNO по умолчанию системой не задаются, так как для их настройки подходят глобальные значения по умолчанию команды **no**. Однако, при необходимости, глобальные значения по умолчанию атрибутов ISNO можно переопределить.

Ниже приведен пример значений ISNO по умолчанию для параметров **tcp_sendspace** и **tcp_recvspace** Gigabit Ethernet, работающего в режиме MTU 1500:

```
# ifconfig en0
en0: flags=5e080863,c0<UP,BROADCAST,NOTRAILERS,RUNNING,SIMPLEX,MULTICAST,GROUPRT,64BIT,CHECKSUM_OFFLOAD,PSEG,CHAIN>
        inet 10.0.0.1 netmask 0xffffff00 broadcast 192.0.0.255
        tcp_sendspace 131072 tcp_recvspace 65536
```

В режиме больших кадров для параметров **tcp_sendspace**, **tcp_recvspace** и **rfc1323** задаются следующие значения ISNO по умолчанию:

```
 # ifconfig en0
en0: flags=5e080863,c0<UP,BROADCAST,NOTRAILERS,RUNNING,SIMPLEX,MULTICAST,GROUPRT,64BIT,CHECKSUM_OFFLOAD,PSEG,CHAIN>
       inet 192.0.0.1 netmask 0xffffff00 broadcast 192.0.0.255
       tcp_sendspace 262144 tcp_recvspace 131072 rfc1323 1
```

Используйте следующие параметры для включения **rfc1323**, если размер MTU - 4096 байт или более, и задания значений **tcp_sendspace** и **tcp_recvspace** как минимум 128 КБ для высокоскоростного адаптера (гигабит или более). Для супервысокоскоростного адаптера необходимо задать 256 КБ. Пустое значение означает, что опция не задана, и будет унаследовано глобальное значение Нет.

Интерфейс	Speed	MTU	tcp_sendspace	tcp_recvspace	rfc1323	tcp_nodelay	tcp_mssdflt
lo0 (loopback)	нд	16896	131072	131072	1		
Ethernet	10 или 100 (М-бит)						
Ethernet	1000 (Гигабит)	1500	131072	65536	1		
Ethernet	1000 (Гигабит)	9000	262144	131072	1		
Ethernet	10 GigE	1500	262144	262144	1		
Ethernet	10 GigE	9000	262144	262144	1		
Ether Channel	Настраивается на основании значений скорости/MTU лежащих в основе интерфейсов.						
Виртуальный Ethernet	нд	any	262144	262144	1		
InfiniBand	нд	2044	131072	131072	1		

Опции ISNO можно задать с помощью следующих инструментов:

- SMIT
- Команда **chdev**
- Команда **ifconfig**

Программа SMIT и команда **chdev** вносят изменения в базу данных ODM, поэтому новые значения становятся постоянными. Команда **ifconfig** изменяет значения только в оперативной памяти, поэтому после перезагрузки восстанавливаются значения, сохраненные в ODM.

Изменение опций ISNO с помощью программы SMIT:

Изменить опции ISNO с помощью утилиты SMIT.

Выполните следующую команду:

```
# smitty tcpip
```

1. Выберите опцию Дополнительная настройка.
2. Выберите опцию Сетевые интерфейсы.
3. Выберите Выбор сетевых интерфейсов.
4. Выберите Показать или изменить характеристики сетевого интерфейса.
5. Выберите интерфейс с помощью курсора. Например, en0.

Появится следующее меню:

```
Показать или изменить стандартный интерфейс Ethernet

Введите или выберите значения в полях ввода.
После внесения изменений нажмите клавишу Enter.

                                                [Поля ввода]
  Имя сетевого интерфейса                        en0
  IP-адрес (десятичный формат с точками)        [192.0.0.1]
  Маска подсети (шестн. или десят. формат с точками) [255.255.255.0]
  Текущее состояние                             Включен                  +
  Применять протокол преобразования адресов (ARP)?          да                        +
  Адрес оповещения (десятичный формат с точками)  []
  Сетевые опции, связанные с интерфейсом
    ('NULL' отменяет выбор опции)
    rfc1323                                     []
```

```
    tcp_mssdflt                                 []
    tcp_nodelay                                 []
    tcp_recvspace                               []
    tcp_sendspace                               []

F1=Справка          F2=Обновить         F3=Отмена           F4=Список
Esc+5=Сбросить      Esc+6=Команда       Esc+7=Изменить      Esc+8=Изображение
Esc+9=Оболочка      Esc+0=Выход         Enter=Выполнить
```

Обратите внимание, что хотя значения по умолчанию ISNO заданы, они не отображаются. В качестве примера переопределите значение по умолчанию параметра **tcp_sendspace**, указав для него меньшее значение 65536.

Снова включите интерфейс с помощью команды **smitty tcpip** и выберите опцию Минимальная конфигурация и запуск. Затем выберите en0 и просмотрите значения по умолчанию, заданные при первоначальной настройке интерфейса.

Для просмотра опций ISNO воспользуйтесь командой **ifconfig**. В выводе будет указано, что значение атрибута **tcp_sendspace** равно 65536. Пример приведен ниже:

```
# ifconfig en0
en0: flags=5e080863,c0<UP,BROADCAST,NOTRAILERS,RUNNING,SIMPLEX,MULTICAST,GROUPRT,64BIT,CHECKSUM_OFFLOAD,PSEG,CHAIN>
          inet 192.0.0.1 netmask 0xffffff00 broadcast 192.0.0.255
          tcp_sendspace 65536 tcp_recvspace 65536
```

Вывод команды **lsattr** также позволяет убедиться, что значение по умолчанию этого атрибута было переопределено:

```
# lsattr -E -l en0
alias4                      Псевдоним IPv4, включая маску подсети      True
alias6                      Псевдоним IPv6, включая длину префикса     True
arp           включен       Протокол преобразования адресов (ARP)      True
authority                   Пользователи с правами доступа             True
broadcast                   Адрес оповещения                           True
mtu           1500          Макс. размер пакета IP для устройства      True
netaddr       192.0.0.1     IP-адрес                                   True
netaddr6                    IP-адрес IPv6                              True
netmask       255.255.255.0 Маска подсети                              True
prefixlen                   Длина префикса для адреса IPv6             True
remmtu        576           Макс. размер пакета IP для удал. сетей     True
rfc1323                     Включить/выключить масштаб. окна RFC 1323  True
security      нет           Уровень защиты                             True
state         работает      Текущее состояние интерфейса               True
tcp_mssdflt                 Задать максимальный размер сегмента TCP    True
tcp_nodelay                 Включить/выключить опцию TCP_NODELAY       True
tcp_recvspace               Задать размер буфера сокета для приема     True
tcp_sendspace 65536         Задать размер буфера сокета для отправки   True
```

Изменение опций ISNO с помощью команд chdev и ifconfig:

Ниже перечислены команды, которые служат для проверки правильности настройки системы и поддерживаемых интерфейсов, а также для настройки параметров интерфейсов.

- Убедитесь, что опция **use_isno** включена, с помощью следующей команды:

  ```
  # no -a | grep isno
  use_isno = 1
  ```

- Убедитесь, что интерфейс поддерживает новые параметры ISNO, с помощью команды **lsattr -El**:

  ```
  # lsattr -E -l en0 -H
  атрибут             значение                          задается пользователем
  :
  rfc1323                     Включить/выключить масштаб. окна RFC 1323  True
  tcp_mssdflt                 Задать максимальный размер сегмента TCP    True
  ```

```
tcp_nodelay                 Включить/выключить опцию TCP_NODELAY       True
tcp_recvspace               Задать размер буфера сокета для приема     True
tcp_sendspace               Задать размер буфера сокета для отправки   True
```

- Для того чтобы изменить значения параметров интерфейса, вызовите команду **ifconfig** или **chdev**. Команда **ifconfig** временно изменяет значения параметров (может применяться при тестировании). Команда **chdev** изменяет запись ODM, поэтому новое значение вступает в силу после следующей загрузки системы.

  Например, для того чтобы присвоить параметрам **tcp_recvspace** и **tcp_sendspace** значение 64 КБ и включить опцию **tcp_nodelay**, вызовите одну из следующих команд:

  ```
  # ifconfig en0 tcp_recvspace 65536 tcp_sendspace 65536 tcp_nodelay 1
  ```

  или

  ```
  # chdev -l en0 -a tcp_recvspace=65536 -a tcp_sendspace=65536 -a tcp_nodelay=1
  ```

- Значения параметров можно просмотреть с помощью команды **ifconfig** или **lsattr**:

  ```
  # ifconfig en0
  en0: flags=5e080863,c0<UP,BROADCAST,NOTRAILERS,RUNNING,SIMPLEX,MULTICAST,GROUPRT,64BIT,CHECKSUM_OFFLOAD,PSEG,CHAIN>
          inet 9.19.161.100 netmask 0xffffff00 broadcast 9.19.161.255
          tcp_sendspace 65536 tcp_recvspace 65536 tcp_nodelay 1
  ```

  или

  ```
  # lsattr -El en0
  rfc1323                     Включить/выключить масштаб. окна RFC 1323  True
  tcp_mssdflt                 Задать максимальный размер сегмента TCP    True
  tcp_nodelay                 Включить/выключить опцию TCP_NODELAY       True
  tcp_recvspace 65536         Задать размер буфера сокета для приема     True
  tcp_sendspace 65536         Задать размер буфера сокета для отправки   True
  ```

Настройка рабочей схемы TCP

Некоторые параметры операционной системы AIX существенно влияют на производительность TCP.

Протокол управления передачей (TCP) применяется многими приложениями, включая **ftp** и **rcp**.

Примечание: Команда **no -o** в случае изменения параметров, связанных с соединениями TCP/IP, выдает предупреждение о том, что изменения будут применяться только для соединений, устанавливаемых после внесения этих изменений. Кроме того, после внесения изменений команда **no -o** перезапускает демон **inetd**, что может повлиять на работу процессов, обслуживаемых демоном **inetd**.

Настройка потоковой рабочей схемы TCP:

Потоковая рабочая схема предназначена для передачи больших объемов информации из одной конечной системы в другую. В качестве примеров потоковой рабочей схемы можно привести передачу файлов, а также резервное копирование или восстановление данных. Наиболее важной характеристикой в данном случае является пропускная способность. Однако в некоторых случаях рекомендуется обратить внимание на задержку при взаимодействии двух конечных систем.

Ниже приведены основные параметры, влияющие на производительность TCP в потоковых приложениях:

- **tcp_recvspace**
- **tcp_sendspace**
- **rfc1323**
- **вычисление MTU маршрута**
- **tcp_nodelayack**
- **sb_max**
- Опции адаптера, такие как аппаратная проверка контрольной суммы и отправка больших объемов данных TCP

В приведенной ниже таблице перечислены рекомендуемые значения параметров, позволяющие получить оптимальную производительность для разных типов адаптеров и размеров MTU:

Device	Speed	Размер MTU	tcp_sendspace	tcp_recvspace	sb_max [1]	rfc1323
Token Ring	4 или 16 Мбит/с	1492	16384	16384	32768	0
Ethernet	10 Мбит/с	1500	16384	16384	32768	0
Ethernet	100 Мбит/с	1500	16384	16384	65536	0
Ethernet	Gigabit	1500	131072	65536	131072	0
Ethernet	Gigabit	9000	131072	65535	262144	0
Ethernet	Gigabit	9000	262144	131072[2]	524288	1
Ethernet	10 Гигабит	1500	131072	65536	131072	0
Ethernet	10 Гигабит	9000	262144	131072	262144	1
ATM	155 Мбит/с	1500	16384	16384	131072	0
ATM	155 Мбит/с	9180	65535	65535[3]	131072	0
ATM	155 Мбит/с	65527	655360	655360[4]	1310720	1
FDDI	100 Мбит/с	4352	45056	45056	90012	0
Fibre-Channel	2 Гбит/с	65280	655360	655360	1310720	1

(1) Для параметра *sb_max* рекомендуется указывать значение по умолчанию, равное 1048576. Значения, перечисленные в таблице, представляют собой минимально допустимые значения параметра *sb_max*.

(2) При использовании rfc1323 данные опции позволяют дополнительно увеличить производительность системы в режиме больших кадров Gigabit Ethernet.

(3) При некоторых размерах буферов приема и передачи TCP пропускная способность может стать очень низкой (около 1 Мбит/с или меньше). Для того чтобы эта проблема не возникала, размер *tcp_sendspace* должен как минимум в три раза превосходить размер MTU, а также должен быть не меньше значения *tcp_recvspace* целевой системы.

(4) Размер окна TCP должен быть не более чем 16-разрядным значением. Другими словами, максимальный размер окна может составлять 65536 байт. В результате при работе с адаптерами с большими значениями MTU (например, 32 КБ или 64 КБ) эффективность передачи данных по потоковому соединению TCP будет очень низкой. Например, если размер MTU устройства равен 64 КБ, и параметр *tcp_recvspace* равен 64 КБ, то окно будет закрыто после отправки первого из пакетов. Следующий пакет будет отправлен только после получения подтверждения от целевой системы. Эту проблему можно решить следующими способами:

- Включить опцию *rfc1323*, позволяющую увеличить размер окна TCP более чем до 64 КБ. В этом случае можно установить большое значение *tcp_recvspace*, например, в десять раз превосходящее размер MTU. Это позволит TCP передавать данные потоком, что приведет к повышению производительности.
- Уменьшить размер MTU адаптера. Например, с помощью команды **ifconfig at0 mtu 16384** можно установить размер MTU в сети ATM равным 16 КБ. При этом TCP вычислит меньшее значение MSS. Если размер MTU равен 16 КБ, а размер окна - 64 КБ, то TCP может отправить four пакета, не дожидаясь получения подтверждения.

Ниже приведены общие рекомендации по настройке потоковой рабочей схемы TCP:

- Размер буферов приема и передачи TCP должен быть как минимум в 10 раз больше размера MTU.
- Если размер MTU превышает 8 КБ, то для увеличения допустимых размеров буферов приема TCP следует применять опцию *rfc1323*.
- Для высокопроизводительных адаптеров большой размер буферов приема и отправки позволяет увеличить производительность.
- Для высокопроизводительных адаптеров значение параметра *tcp_sendspace* должно быть в два раза больше значения *tcp_recvspace*.
- *rfc1323* для интерфейса lo0 имеет значение по умолчанию. Значение по умолчанию размера MTU для **lo0** больше, чем 1500, и, поэтому, переменные *tcp_sendspace* и *tcp_recvspace* имеют значение 128K.

Примерами приложений TCP, производительность которых может значительно повыситься после настройки параметров *tcp_sendspace* и *tcp_recvspace*, могут служить команды **ftp** и **rcp**.

Переменная tcp_recvspace:

Параметр *tcp_recvspace* указывает, сколько байт данных принимающая система может поместить в буфер ядра очереди принимающих сокетов.

Параметр *tcp_recvspace* также применяется для определения размера окна TCP - ограничения на количество байт, отправляемых получателю во избежание переполнения приемного буфера. Параметр *tcp_recvspace* оказывает ключевое влияние на производительность протокола TCP, поскольку протокол TCP должен обеспечивать полную загрузку сетевого конвейера. Если TCP не обеспечивает полную загрузку конвейера, производительность падает.

Параметр *tcp_recvspace* можно задать следующими способами:

- Системным вызовом **setsockopt()** из программы
- Командой **no -o tcp_recvspace=[*значение*]**
- Параметром ISNO *tcp_recvspace*

В общем рекомендуется, чтобы значение *tcp_recvspace* было по крайней мере в 10 раз меньше, чем значение MTU. Чтобы определить значение *tcp_recvspace*, нужно разделить на 8 значение *bandwidth-delay product*, определяемое по следующей формуле:

```
bandwidth-delay product = capacity(bits)= bandwidth(bits/second) x round-trip time (seconds)
```

Разделив значение capacity на 8, можно получить хорошую оценку оптимального размера окна TCP, обеспечивающего полную загрузку сетевого конвейера. Чем больше время оборота пакета и быстрее сеть, тем больше значение *bandwidth-delay product* и, как следствие, больше окно TCP. Примером может служить сеть на 100 Мбит с временем оборота 0,2 мс. Значение *bandwidth-delay product* можно вычислить по формуле, приведенной выше:

```
bandwidth-delay product = 100000000 x 0,0002 = 20000
20000/8 = 2500
```

В данном примере окно TCP должно быть не менее 2500 байт. В локальных сетях на 100 Мбит и 1 Гбит рекомендуется, чтобы значения переменных *tcp_recvspace* и *tcp_sendspace* были по крайней мере в 2 или 3 раза больше вычисленного таким образом значения *bandwidth-delay product*.

Параметр tcp_sendspace:

Параметр *tcp_sendspace* указывает ограничение на объем данных отправляющего сообщения в буфере ядра. Превышение этого ограничения приводит к блокировке запроса на отправку данных.

Данные приложения размещаются в буфере передачи сокета TCP-socket перед их отправкой. Они хранятся в структурах mbuf и кластерах в области памяти ядра. По умолчанию размер буфера задается переменной *tcp_sendspace*. Его можно изменить вызовом процедуры **setsockopt()**.

Значение *tcp_sendspace* должно быть не меньше значения *tcp_recvspace*, а для высокоскоростных адаптеров значение *tcp_sendspace* должно по крайней мере вдвое превышать значение *tcp_recvspace*.

Если для сокета установлена опция O_NDELAY или O_NONBLOCK (ввод-вывод выполняется без блокировки приложения), то в случае переполнения буфера передачи приложению будет отправлен код ошибки EWOULDBLOCK/EAGAIN, однако оно не будет переведено в состояние ожидания. В приложении должен быть предусмотрен обработчик таких ошибок (рекомендуется повторять попытки передачи данных через небольшой промежуток времени).

Параметр rfc1323:

Параметр *rfc1323* позволяет включить масштабирования окна TCP.

Возможность масштабирования определяется в ходе согласования параметров TCP, поэтому она должна быть включена на обоих концах соединения TCP. По умолчанию размер окна TCP не может превышать 65536 байт (64 КБ), однако он может быть увеличен, если переменной *rfc1323* присвоено значение 1. Если переменной *tcp_recvspace* нужно присвоить значение, превышающее 65536, присвойте переменной *rfc1323* значение 1 на обоих концах соединения. Если переменная *rfc1323* не будет задана хотя бы на одном из концов соединения, в такой ситуации будет считаться, что переменной *tcp_recvspace* присвоено значение 65536. Если применяется эта опция, к заголовку протокола TCP добавляются 12 байт за счет области данных, что может привести к незначительному снижению производительности адаптеров с малыми MTU.

Для адаптеров с большим MTU (например, 32 или 64 КБ) настоятельно рекомендуется указать эту опцию, так как в противном случае один пакет будет занимать все окно TCP. В результате протоколу TCP не удастся организовать поточную передачу пакетов, поскольку придется ждать подтверждения получения каждого отдельного пакета от получателя, и только после этого отправлять следующий пакет. Если вы включите опцию *rfc1323* с помощью команды **no -o rfc1323=1**, то для размера окна TCP можно будет установить любое значение вплоть до 4 ГБ. Если параметр *rfc1323* равен 1, то можно значительно увеличить значение параметра *tcp_recvspace*, например, присвоить ему значение, в 10 раз превосходящее значение MTU.

Если отправляющая и принимающая системы не поддерживают опцию *rfc1323*, то для повышения пропускной способности адаптеров с большим MTU можно уменьшить MTU. Например, вместо MTU 65536, при котором в окне помещается только один пакет, можно использовать MTU 16384, и тогда при *tcp_recvspace*=65536 в окно передачи будут помещаться четыре пакета. Однако на всех узлах сети должно применяться одно и то же значение MTU.

Вычисление MTU маршрута TCP:

По умолчанию в операционной системе AIX вычисление MTU маршрута выполняется. С помощью этой опции стек протокола определяет минимальный размер MTU во всех промежуточных сетях между двумя хостами, управляемыми сетевой опцией **tcp_pmtu_discover=1**.

В реализации функции поиска путей TCP MTU применяются не сообщения ICMP ECHO, а непосредственно пакеты TCP. Расширение ядра TCP/IP поддерживает таблицу PMTU, в которой хранится информация о поиске путей MTU. При создании соединения TCP с целевым узлом в таблицу PMTU добавляется запись для этого целевого узла. Значение PMTU соответствует значению MTU интерфейса исходящей связи.

В заголовке IP отправляемых пакетов TCP установлен бит отключения фрагментирования (DF). При передаче пакета TCP на сетевой маршрутизатор, значение MTU которого меньше размера этого пакета TCP, на исходный узел отправляется сообщение ICMP с информацией о том, что пересылка сообщения невозможна, так как его нельзя разбить на фрагменты. Если маршрутизатор, отправляющий сообщение об ошибке, соответствует RFC 1191, сообщение об ошибке ICMP содержит значение MTU сети. В противном случае для повторной передачи пакета TCP необходимо задать меньшее значение размера MTU из таблицы известных значений MTU расширения ядра AIX TCP/IP. При этом табличное значение PMTU для целевого узла заменяется на новое, меньшее значение MTU и пакет TCP отправляется еще раз. При создании последующих соединений TCP с этим целевым узлом применяется новое значение PMTU.

Команда **pmtu** позволяет просматривать и удалять записи PMTU. Ниже приведен пример выполнения команды **pmtu**:

```
# вывод pmtu

   цлв           шлз           Иф    pmtu     refcnt   redisc_t    exp

 --------------------------------------------------------------------------
```

```
10.10.1.3    10.10.1.5        en1    1500        2          9       0

10.10.2.5    10.10.2.33       en0    1500        1          0       0
```

Поскольку записи PMTU `refcnt` со значением 0 не применяются, система автоматически удаляет их, что позволяет предотвратить создание таблиц PMTU слишком большого размера. Такие записи удаляются через *pmtu_expire* минут после того, как записи `refcnt` присваивается значение 0. По умолчанию параметру сети *pmtu_expire* присвоено значение 10 минут. Для того чтобы предотвратить удаление записей PMTU, можно присвоить записи *pmtu_expire* значение 0.

В данной реализации поиска путей TCP MTU можно не применять создание копий маршрутов, что позволяет поддерживать относительно небольшой размер таблицы маршрутизации и упрощает управление ей.

Переменная tcp_nodelayack:

В режиме *tcp_nodelayack* протокол TCP отправляет подтверждения немедленно, а не со стандартной задержкой в 200 мс. Немедленная отправка подтверждений немного увеличивает нагрузку, однако в определенных ситуациях позволяет существенно повысить производительность.

Известны случаи снижения производительности из-за задержки в 200 мс при отправке подтверждений TCP, поскольку отправитель ждет подтверждения от получателя, а получатель ждет следующие данные от отправителя. Это может отрицательно сказаться на пропускной способности. Если есть основания полагать, что в вашей сети именно такая ситуация, рекомендуется включить режим *tcp_nodelayack* и проверить, повышается ли производительность в этом режиме. Если она не повышается, нужно отключить режим *tcp_nodelayack*.

Настройка sb_max:

Параметр *sb_max* задает максимальное число буферов сокетов, которые могут находиться в очереди отдельного сокета. Фактически это ограничение на объем памяти, занятой буферами, образующими очередь в сокет отправителя или получателя.

Это значение вычисляется исходя из размера буфера, а не размера его содержимого.

Если драйвер устройства поместит 100 байт данных в буфер размером 2048 байт, будет занято 2048 байт памяти. Обычно драйвер устройства выбирает буфер приема, размер которого достаточен для хранения пакета максимального размера. При этом какой-то объем памяти буфера остается неиспользованным. Однако, применение буферов меньшего размера приводит к увеличению процессорного времени, затрачиваемого на копирование данных в буферы.

Примечание: В AIX переменной *sb_max* по умолчанию присвоено довольно большое значение 1048576. В документе Задача настройки потоков TCP приведены рекомендуемые значения *sb_max* для данного параметра.

Аппаратная проверка контрольной суммы TCP:

В режиме аппаратной проверки контрольной суммы TCP адаптер самостоятельно вычисляет контрольные суммы отправляемых и принимаемых пакетов TCP, снимая эту нагрузку с процессора AIX.

Отдача от этого режима зависит от размера пакетов. Для пакетов малого размера этот режим не дает практически никакого выигрыша; ощутимый выигрыш достигается на больших пакетах. Применение этого режима на адаптерах PCI-X GigE позволяет разгрузить процессор на 5% при MTU 1500 и на 15% при MTU 9000 (сверхбольшие кадры).

В системах с тактовой частотой процессора в 400 МГц и выше пропускная способность TCP в потоковом режиме с MTU 1500 будет ниже в режиме аппаратной проверки контрольных сумм, поскольку в таких системах процессор будет подсчитывать контрольные суммы быстрее, чем адаптеры Gigabit Ethernet PCI FC

2969 и FC 2975. Поэтому по умолчанию для данных адаптеров этот режим выключен. Однако со сверхбольшими кадрами эти адаптеры успевают вычислять контрольные суммы со скоростью прохождения данных, не снижая пропускной способности.

Адаптеры PCI-X Gigabit Ethernet работают в режиме аппаратной проверки контрольных сумм без снижения пропускной способности, и поскольку этот режим позволяет разгрузить процессор системы, он включен для этих адаптеров по умолчанию.

Дробления больших сообщений TCP:

В режиме аппаратного дробления пакетов TCP протокол AIX может создавать сообщения TCP длиной до 64 КБ. Адаптер отправляет сообщение единым пакетом как на уровне протокола IP, так и на уровне драйвера устройства Ethernet.

Затем адаптер разбивает сообщение на несколько кадров TCP для отправки на физическом уровне. Пакеты TCP отправляются либо кадрами по 1500 байт (при значении MTU, равном 1500), либо кадрами размером до 9000 байт (если значение MTU составляет 9000) (кадры большого размера).

Без опции аппаратного дробления пакетов TCP для отправки 64 КБ данных при использовании пакетов размером 1500 байт потребуется 44 вызова стека. В режиме аппаратного дробления выполняется один вызов стека, что снижает нагрузку на систему и ее процессор. Затем адаптер Ethernet выполняет разбиение данных на пакеты размером MTU (обычно 1500 байт). Отдача от этого режима зависит от среднего размера отправляемых пакетов TCP. Например, для адаптеров PCI-eXtended (PCI-X) Gigabit Ethernet с MTU 1500 нагрузка на процессор может снизиться на 60-75 %. При работе со сверхбольшими кадрами (MTU 9000) выигрыш становится меньше, поскольку система в любом случае отправляет большие кадры. Например, обычным является снижение нагрузки на процессор на 40 %.

Режим аппаратного дробления пакетов включен по умолчанию для адаптеров Ethernet, поддерживающих работу в выделенном режиме. Эта опция повышает производительность адаптеров 10 Gigabit Ethernet и более быстрых для задач, управляющих потоками данных (таких как FTP, RCP, резервное копирование на магнитной ленте и аналогичные приложения перемещения больших объемов данных). Устройства виртуального адаптера Ethernet и общего адаптера Ethernet (SEA) являются исключением - в них эта опция по умолчанию отключена из-за проблем со взаимодействием с операционными системами Linux или IBM i. Включить функцию отправки больших пакетов, а также другие функции управления производительностью можно в AIX и в средах виртуального адаптера Ethernet или SEA.

Опция отправки больших пакетов - это атрибут устройства, указанный как large_send. Этот атрибут можно просмотреть при выполнении следующей команды (X - это номер устройства):

```
lsattr -E -l entX
```

Опции разгрузки адаптера:

В некоторых адаптерах предусмотрен ряд параметров, позволяющих переносить нагрузку с системы AIX на адаптер.

Таблица 7. Адаптеры, их параметры и значения по умолчанию для системы

Тип адаптера	Код продукта	Аппаратная проверка контрольной суммы TCP	Значение по умолчанию	Отправка больших пакетов TCP	Значение по умолчанию
GigE, PCI, SX и TX	2969, 2975	Да	ВЫКЛ	Да	ВЫКЛ
GigE, PCI-X, SX и TX	5700, 5701	Да	ВКЛ	Да	ВКЛ
GigE, двухпортовый PCI-X, TX и SX	5706, 5707	Да	ВКЛ	Да	ВКЛ
10 GigE PCI-X LR и SR	5718, 5719	Да	ВКЛ	Да	ВКЛ
10/100 Ethernet	4962	Да	ВКЛ	Да	ВЫКЛ
ATM 155, UTP и MMF	4953, 4957	Да (только передача)	ВКЛ	Нет	Нет
ATM 622, MMF	2946	Да	ВКЛ	Нет	Нет

Настройка диалоговой рабочей схемы запросов и ответов TCP

Диалоговая рабочая схема TCP предусматривает передачу данных в обоих направлениях.

В качестве примеров диалоговой рабочей схемы можно привести приложения вызова удаленных процедур (RPC), приложения клиент-сервер, такие как запросы Web-браузера к Web-серверу, файловые системы NFS (применяющие TCP в качестве протокола передачи данных) и протокол управления блокировками базы данных. Как правило это небольшие сообщения, в ответ на которые отправляются большие ответы, или наоборот.

Основной характеристикой рабочей схемы такого типа является задержка при подтверждении приема. Как правило запросы и ответы представляют собой небольшие сообщения, поэтому пропускная способность сети в данном случае большой роли не играет.

В основном задержка определяется аппаратным обеспечением. Например, время подтверждения приема может зависеть от типа сети, типа и производительности коммутаторов и маршрутизаторов, мощности процессоров конечных систем, задержек в работе адаптеров и шин.

Опции настройки, предназначенные для сокращения задержек (оптимальное время ответа), как правило приводят к возрастанию нагрузки на CPU за счет отправки большего числа пакетов, обработки большего числа прерываний и т.д. Это классические компромиссы, позволяющие повысить производительность.

Ниже описаны основные параметры настройки диалоговых приложений:

- *tcp_nodelay* и *tcp_nagle_limit*
- *tcp_nodelayack*
- Параметры объединения прерываний адаптера

Примечание: В некоторых случаях диалоговая рабочая схема предусматривает передачу больших объемов данных в одном направлении. Для такой рабочей схемы требуется настройка как задержки, так и потоков данных.

Опции tcp_nodelay и tcp_nagle_limit:

В AIX режим TCP_NODELAY по умолчанию отключен, что может заметно снизить производительность в системах "запрос-ответ", когда запрос может быть очень маленьким и важна быстрота получения ответа. В протоколе TCP задержка при отправке подтверждения применяется для оптимизации нагрузки. Стандартная задержка составляет 200 мс.

В большинстве реализаций TCP применяется алгоритм Нэгла, согласно которому у соединения TCP может быть только один неподтвержденный сегмент. Поэтому протокол TCP не отправляет следующий пакет до тех пор, пока не будет получено подтверждение доставки предыдущего, либо пока не удастся собрать больше данных и отправить полноразмерный сегмент.

Если приложение работает по схеме "запрос-ответ", рекомендуется включить режим TCP_NODELAY с помощью системного вызова **setsockopt()**. Например, алгоритм Нэгла по умолчанию отключен для утилит **telnet** и **rlogin**, протокола NFS и web-серверов. Однако некоторые приложения не отключают алгоритм Нэгла, что приводит к существенному снижению производительности с определенной зависимостью от значения MTU и размера блоков данных, записываемых в сокет.

Для приложений, не включающих режим TCP_NODELAY, рекомендуется отключить алгоритм Нэгла одним из следующих способов:

- **tcp_nagle_limit**
- Опция ISNO **tcp_nodelay**
- **tcp_nodelayack**
- **fasttimo**
- Слияние прерываний адаптера

Опция tcp_nagle_limit:

Глобальной опции *tcp_nagle_limit* по умолчанию присвоено значение 65536.

Протокол TCP отключает алгоритм Нэгла для сегментов, размер которых равен этому значению или превышает его, поэтому эта опция позволяет задать порог отключения алгоритма Нэгла. Например, для полного отключения алгоритма Нэгла достаточно присвоить опции *tcp_nagle_limit* значение 1. Если нужно разрешить протоколу TCP объединять пакеты размером 256 байт или больше, нужно присвоить параметру *tcp_nagle_limit* значение 256.

Опция ISNO tcp_nodelay:

На уровне интерфейса можно включить режим TCP_NODELAY с помощью опции ISNO *tcp_nodelay*.

Если параметру *tcp_nodelay* будет присвоено значение 1, алгоритм Нэгла будет отключен, и протокол TCP не будет дожидаться подтверждения доставки пакетов.

Опция tcp_nodelayack:

С помощью опции *tcp_nodelayack* можно отключить задержку отправки подтверждения, которая обычно составляет 200 мс.

Отключение задержки может ускорить отправку подтверждения и, как следствие, получение следующего пакета.

Опция fasttimo:

С помощью опции **fasttimo** можно сократить задержку при отправке подтверждения с 200 мс (стандартное значение) до 100 или 50 мс.

Поскольку TCP использует этот таймер для всех соединений TCP, изменение этого значения приводит к увеличению нагрузки на систему, поскольку увеличивается частота сканирования всех соединений TCP. Рекомендуется пользоваться другими опциями из числа указанных выше, а параметр **fasttimo** оставить на крайний случай.

Объединение прерываний:

Для снижения количества прерываний пакеты собираются в группу, для которых генерируется только одно прерывание. Такая процедура называется *слиянием прерываний*.

Прерывания при получении данных, как правило, уведомляют центральный процессор о том, что пакеты помещены в очередь ввода устройства. В случае, если применяется стандартный алгоритм обработки прерываний адаптера, это может привести к отправке прерываний для каждого входящего пакета. При увеличении скорости приема пакетов перед завершением работы и освобождением прерывания драйвер завершает в прерывании обработку пакета и проверяет, пуста ли очередь входящих пакетов. Если она содержит пакеты, драйвер начинает обрабатывать несколько пакетов с помощью одного прерывания, что позволяет повысить эффективность работы системы в условиях возросшей нагрузки.

Тем не менее, некоторые адаптеры обладают дополнительными функциями, позволяющими расширить возможности управления при создании прерываний. Это часто называют объединением прерываний или изменением логики обработки прерываний, что позволяет одновременно принимать несколько пакетов и создавать одно прерывание для обработки нескольких пакетов. При получении первого пакета запускается таймер, и прерывание будет сформировано только после через *n* микросекунд или после получения *m* пакетов. Конкретные методы определяются моделью адаптера и функциями, которые драйвер предоставляет пользователю.

При малой загрузке применение функции объединения прерываний приводит к появлению задержки при приеме пакетов. Пакет будет находиться в памяти узла, но узел уведомляется об этом лишь через некоторое время. Тем не менее, при обработке большого числа пакетов эффективность работы системы остается высокой благодаря тому, что для нескольких пакетов создается одно прерывание, что позволяет снизить нагрузку на центральный процессор.

Для адаптеров AIX, поддерживающих изменение логики обработки прерываний, необходимо установить значения, соответствующие среднему уровню, чтобы снизить нагрузку, вызванную прерываниями, без значительного увеличения задержек. Если в системе запущены приложения, для работы которых необходима минимальная задержка, эти параметры необходимо отключить, либо присвоить им значения, позволяющие создавать большее количество прерываний в секунду.

Адаптеры Gigabit Ethernet поддерживают изменение логики обработки прерываний. Адаптеры Gigabit Ethernet FC 2969 и FC 2975 позволяют задавать значения задержки и счетчика буферов. При получении первого пакета адаптер запускает таймер, и прерывание создается либо после истечения заданного значения времени, либо после того, как будут задействованы *n* буферов хоста.

В адаптерах Gigabit EthernetFC 5700, FC 5701, FC 5706 и FC 5707 применяется метод ограничения количества прерываний. При этом прерывания создаются с определенной частотой, что позволяет объединять пакеты в зависимости от времени их получения. По умолчанию в секунду создается 10000 прерываний. Для того чтобы снизить нагрузку по обработке прерываний, можно установить минимальное значение, равное 2000. Если необходимо обеспечить малое время задержки и ответа системы, скорость создания прерываний можно установить равной 20000 в секунду. Если присвоить этому параметру значение 0, данная функция будет отключена.

Адаптеры 10 Gigabit Ethernet PCI-X (FC 5718 и 5719) поддерживают опцию слияния прерываний (*rx_int_delay*) с дискретностью 0.82 микросекунды. Фактическая задержка определяется умножением 0.82 на значение, заданное в параметре *rx_int_delay*. Эта опция выключена по умолчанию (*rx_int_delay*=0), потому что в тестах было обнаружено, что при больших входных скоростях слияние прерываний не увеличивает производительность.

Таблица 8. Параметры адаптеров 10 Gigabit Ethernet PCI-X

Тип адаптера	Код продукта	Атрибут ODM	Значение по умолчанию	Диапазон
10 Gigabit Ethernet PCI-X (LR или SR)	5718, 5719	rx_int_delay	0	0-512

Настройка UDP

Протокол пользовательских дейтаграмм (UDP) применяется сетевой файловой системой (NFS), сервером имен (named), упрощенным протоколом передачи файлов (TFTP) и другими протоколами специального назначения.

Так как UDP является протоколом дейтаграмм, то при отправке дейтаграммы все сообщение должно быть скопировано в область памяти ядра за одну атомарную операцию. Получается дейтаграмма также в виде одного сообщения с помощью системного вызова **recv** или **recvfrom**. Значения параметров *udp_sendspace* и *udp_recvspace* должны соответствовать требованиям буферизации для отдельных сокетов.

Максимальный размер дейтаграммы UDP составляет 64 КБ за вычетом размера заголовка UDP (8 байт) и размера заголовка IP (20 байт для IPv4 или 40 байт для IPv6).

Ниже перечислены параметры, влияющие на производительность UDP:

- *udp_sendspace*
- *udp_recvspace*
- Объединение пакетов UDP в цепочку
- Опции адаптера, такие как объединение прерываний

Настройка udp_sendspace:

Присвойте параметру udp_sendspace значение, заведомо превышающее размер максимальной отправляемой дейтаграммы UDP.

Для простоты рекомендуется присвоить этому параметру значение 65536, которое наверняка больше максимального возможного пакета UDP. Нет смысла присваивать этому параметру еще большее значение.

Настройка udp_recvspace:

Параметр *udp_recvspace* задает объем памяти для хранения входящих данных в очереди сокета UDP. По достижении ограничения *udp_recvspace* сокет перестает принимать входящие пакеты.

Статистику по отброшенным пакетам можно получить с помощью команды **netstat -p udp** (столбец `переполнение буфера сокета`). Дополнительная информация приведена в раз делеКоманда netstat in *Справочник по командам, том 4*.

Параметру *udp_recvspace* рекомендуется присвоить достаточно большое значение, поскольку одновременно могут поступить и попасть в очередь много дейтаграмм UDP. Кроме того, многие приложения UDP пользуются определенным сокетом для приема пакетов. Один и тот же сокет используется для приема данных от всех клиентов сервера. Поэтому буфер приема должен быть достаточно большим для хранения дейтаграмм, которые могут поступить от нескольких клиентов одновременно. Если это значение будет недостаточно большим, входящие пакеты начнут отбрасываться, и отправителю придется посылать их повторно. Это может отрицательно сказаться на производительности.

При настройке *udp_recvspace* следует учитывать, что подсистема связи подсчитывает объем памяти, занятый буферами, а не их содержимым. Например, дейтаграмма размером 8 КБ может быть разбита на 6 пакетов, для размещения которых потребовалось 6 буферов приема. В сети Ethernet размер буфера составляет 2048 байт. Следовательно, общий объем буферов сокета, содержащих дейтаграмму размером 8 КБ, составляет `6*2048=12288 байт`

Следовательно, при выборе значения *udp_recvspace* нужно также учитывать размер буферов приема. Этот размер зависит от размера дейтаграммы и типа драйвера устройства. Для размещения дейтаграммы размером 64 байта потребуется буфер размером 2 КБ.

Кроме того, нужно учесть число дейтаграмм, которые могут быть помещены в очередь сокета. Например, сервер NFS получает пакеты UDP от всех клиентов через стандартный сокет. Если размер очереди сокета может достигать 30 пакетов, и размер дейтаграммы NFS составляет 8 КБ, то параметру *udp_recvspace* нужно присвоить значение 30 * 12288 = 368640. В NFS версии 3 поддерживаются дейтаграммы размером до 32 КБ.

Вначале параметру *udp_recvspace* рекомендуется присвоить значение, в десять раз превышающее значение *udp_sendspace*, так как UDP не может отправить пакет приложению до тех пор, пока не был получен предыдущий пакет. Кроме того, хост может получать пакеты сразу от нескольких узлов сети. Такое значение обеспечивает некоторый запас памяти: пакеты начнут отбрасываться, только когда длина очереди достигнет 10. Для приложений, выполняющих большое число параллельных операций и применяющих UDP, это значение должно быть еще больше.

Примечание: Значение *sb_max*, устанавливающее ограничение на размер всех буферов сокета, должно быть как минимум в два раза больше размера максимального из буферов приема и передачи TCP и UDP.

Объединение пакетов UDP в цепочку:

Если размер дейтаграммы UDP превышает размер MTU, то протокол IP разбивает дейтаграммы на фрагменты, соответствующие размеру MTU. Интерфейсы Ethernet поддерживают функцию объединения пакетов UPD в цепочку. По умолчанию в операционной системе AIX эта функция включена.

С помощью объединения пакетов UDP в цепочку IP может создать цепочку, состоящую из всех фрагментов, и передать ее драйверу устройства Ethernet в одном вызове. Такой подход позволяет повысить производительность за счет уменьшения числа вызовов ARP и различных уровней интерфейса и драйвера. При этом в среде SMP уменьшается число вызовов **lock** и **unlock**. Кроме того, повышается эффективность кэширования циклов исходного кода. Указанные изменения приводят к снижению нагрузки на CPU отправляющей системы.

Значение опции объединения пакетов UDP в цепочку можно просмотреть с помощью команды **ifconfig**. В приведенном ниже примере показан вывод команды **ifconfig** для интерфейса `en0`, где флаг `CHAIN` указывает на то, что объединение пакетов в цепочку применяется:

```
# ifconfig en0
en0: flags=5e080863,80<UP,BROADCAST,NOTRAILERS,RUNNING,SIMPLEX,MULTICAST,GROUPRT,64BIT,CHECKSUM_OFFLOAD,PSEG,CHAIN>
         inet 192.1.6.1 netmask 0xffffff00 broadcast 192.1.6.255
         tcp_sendspace 65536 tcp_recvspace 65536 tcp_nodelay 1
```

Для того чтобы выключить объединение пакетов в цепочку, выполните следующую команду:

```
# ifconfig en0 -pktchain

# ifconfig en0
en0: flags=5e080863,80<UP,BROADCAST,NOTRAILERS,RUNNING,SIMPLEX,MULTICAST,GROUPRT,64BIT,CHECKSUM_OFFLOAD,PSEG>
         inet 192.1.6.1 netmask 0xffffff00 broadcast 192.1.6.255
         tcp_sendspace 65536 tcp_recvspace 65536 tcp_nodelay 1
```

Для того чтобы повторно включить объединение пакетов в цепочку, выполните следующую команду:

```
# ifconfig en0 pktchain

# ifconfig en0
en0: flags=5e080863,80<UP,BROADCAST,NOTRAILERS,RUNNING,SIMPLEX,MULTICAST,GROUPRT,64BIT,CHECKSUM_OFFLOAD,PSEG,CHAIN>
         inet 192.1.6.1 netmask 0xffffff00 broadcast 192.1.6.255
         tcp_sendspace 65536 tcp_recvspace 65536 tcp_nodelay 1
```

Объединение прерываний:

Для снижения количества прерываний пакеты собираются в группу, для которых генерируется только одно прерывание. Такая процедура называется *слиянием прерываний*.

Прерывания при получении данных, как правило, уведомляют центральный процессор о том, что пакеты помещены в очередь ввода устройства. В случае, если применяется стандартный алгоритм обработки прерываний адаптера, это может привести к отправке прерываний для каждого входящего пакета. При увеличении скорости приема пакетов перед завершением работы и освобождением прерывания драйвер завершает в прерывании обработку пакета и проверяет, пуста ли очередь входящих пакетов. Если она содержит пакеты, драйвер начинает обрабатывать несколько пакетов с помощью одного прерывания, что позволяет повысить эффективность работы системы в условиях возросшей нагрузки.

Тем не менее, некоторые адаптеры обладают дополнительными функциями, позволяющими расширить возможности управления при создании прерываний. Это часто называют объединением прерываний или изменением логики обработки прерываний, что позволяет одновременно принимать несколько пакетов и создавать одно прерывание для обработки нескольких пакетов. При получении первого пакета запускается таймер, и прерывание будет сформировано только после через *n* микросекунд или после получения *m* пакетов. Конкретные методы определяются моделью адаптера и функциями, которые драйвер предоставляет пользователю.

При малой загрузке применение функции объединения прерываний приводит к появлению задержки при приеме пакетов. Пакет будет находиться в памяти узла, но узел уведомляется об этом лишь через некоторое время. Тем не менее, при обработке большого числа пакетов эффективность работы системы остается высокой благодаря тому, что для нескольких пакетов создается одно прерывание, что позволяет снизить нагрузку на центральный процессор.

Для адаптеров AIX, поддерживающих изменение логики обработки прерываний, необходимо установить значения, соответствующие среднему уровню, чтобы снизить нагрузку, вызванную прерываниями, без значительного увеличения задержек. Если в системе запущены приложения, для работы которых необходима минимальная задержка, эти параметры необходимо отключить, либо присвоить им значения, позволяющие создавать большее количество прерываний в секунду.

Адаптеры Gigabit Ethernet поддерживают изменение логики обработки прерываний. Адаптеры Gigabit Ethernet FC 2969 и FC 2975 позволяют задавать значения задержки и счетчика буферов. При получении первого пакета адаптер запускает таймер, и прерывание создается либо после истечения заданного значения времени, либо после того, как будут задействованы *n* буферов хоста.

В адаптерах Gigabit EthernetFC 5700, FC 5701, FC 5706 и FC 5707 применяется метод ограничения количества прерываний. При этом прерывания создаются с определенной частотой, что позволяет объединять пакеты в зависимости от времени их получения. По умолчанию в секунду создается 10000 прерываний. Для того чтобы снизить нагрузку по обработке прерываний, можно установить минимальное значение, равное 2000. Если необходимо обеспечить малое время задержки и ответа системы, скорость создания прерываний можно установить равной 20000 в секунду. Если присвоить этому параметру значение 0, данная функция будет отключена.

Адаптеры 10 Gigabit Ethernet PCI-X (FC 5718 и 5719) поддерживают опцию слияния прерываний (*rx_int_delay*) с дискретностью 0.82 микросекунды. Фактическая задержка определяется умножением 0.82 на значение, заданное в параметре *rx_int_delay*. Эта опция выключена по умолчанию (*rx_int_delay*=0), потому что в тестах было обнаружено, что при больших входных скоростях слияние прерываний не увеличивает производительность.

Таблица 9. Параметры адаптеров 10 Gigabit Ethernet PCI-X

Тип адаптера	Код продукта	Атрибут ODM	Значение по умолчанию	Диапазон
10 Gigabit Ethernet PCI-X (LR или SR)	5718, 5719	rx_int_delay	0	0-512

Настройка ресурсов адаптера

Поскольку в системе могут использоваться разные модели адаптеров и различные виды драйверов, очень сложно описать все типы атрибутов адаптеров. Ниже приведены сведения об общих атрибутах, которые поддерживаются большинством сетевых адаптеров и драйверов, и от значений которых может зависеть производительность системы.

Большинство драйверов связи позволяют изменить параметры, связанные с приемом и отправкой данных. Обычно эти параметры задают ограничения на размер очередей приема и передачи, и, кроме того, могут задавать число и размер буферов и прочих ресурсов. Эти параметры ограничивают число буферов или пакетов в очереди передачи и число доступных буферов приема. Необходимо установить такой размер очередей, чтобы даже при максимальной нагрузке система справлялась с обработкой входящих и исходящих пакетов.

Ниже приведены некоторые рекомендации:

- Просмотреть подробные сведения о ресурсах адаптера и возможных ошибках можно с помощью перечисленных ниже команд. Конкретная команда определяется моделью адаптера.
  - **netstat -v**
  - **entstat**
  - **atmstat**
  - **fddistat**
  - **tokstat**
- Для отслеживания отчетов системного протокола ошибок можно воспользоваться командами **errpt** и **errpt -a**.
- Параметры следует изменять только в одном из следующих случаев:
  - В системе возникли признаки нехватки ресурсов.
  - В системе есть переполненные очереди.
  - Анализ производительности говорит о необходимости тонкой настройки.

Очереди передачи:

Некоторые драйверы устройств позволяют задать ограничение на размер *очереди передачи*.

В зависимости от драйвера и адаптера, могут быть заданы ограничения на размер как аппаратной, так и программной очереди. У некоторых драйверов есть только аппаратная очередь, у других - как программная, так и аппаратная. Некоторые драйверы управляют аппаратной очередью самостоятельно и позволяют задать ограничение только для программной очереди. Обычно драйвер устройства помещает пакет непосредственно в аппаратную очередь адаптера. Если быстродействие CPU выше пропускной способности сети, пакеты для отправки будут быстро накапливаться. Это же относится к системам SMP. При этом аппаратная очередь может переполниться.

Для того чтобы этого избежать, некоторые драйверы после заполнения аппаратной очереди создают дополнительную программную очередь, в которую и помещаются следующие пакеты. После достижения максимального размера программной очереди все новые пакеты будут отбрасываться. Это приведет к снижению производительности, так как протоколу верхнего уровня потребуется повторно отправлять эти пакеты. Тем не менее, в определенных случаях пакеты необходимо отбрасывать, так как слишком большой доступный объем памяти может привести к отправке устаревших пакетов.

Таблица 10. Возможные размеры очереди передачи адаптера PCI:

Тип адаптера	Код продукта	Атрибут ODM	Значение по умолчанию	Диапазон
Адаптер Ethernet PCI IBM 10/100 Мбит/с	2968	tx_que_size	8192	16-16384
10/100 Mbps Ethernet Adapter II	4962	tx_que_sz	8192	512-16384
Gigabit Ethernet PCI (SX или TX)	2969, 2975	tx_que_size	8192	512-16384
Gigabit Ethernet PCI (SX или TX)	5700, 5701, 5706, 5707	tx_que_sz	8192	512-16384
10 Gigabit Ethernet PCI-X (LR или SR)	5718, 5719	tx_que_sz	8192	512-16384
ATM 155 (MMF или UTP)	4953, 4957	sw_txq_size	2048	50-16384
ATM 622 (MMF)	2946	sw_txq_size	2048	128-32768
FDDI	2741, 2742, 2743	tx_queue_size	256	3-2048

При увеличении ограничения на размер аппаратной очереди возрастет объем физической памяти, требуемой для получения данных, так как увеличится число управляющих блоков и буферов. Это ограничение рекомендуется увеличивать только в случае крайней необходимости, либо в системах с большим объемом памяти. Увеличение максимального размера программной очереди передачи не приводит к расходу дополнительной памяти, так как в эту очередь помещаются только те пакеты, которые уже размещены протоколами верхнего уровня.

Дескрипторы передачи:

Некоторые драйверы позволяют выполнять тонкую настройку размера кольца передачи и числа дескрипторов передачи.

Аппаратная очередь передачи устанавливает максимальное число буферов, которые можно одновременно применять при работе с очередями адаптера. Как правило, один дескриптор указывает на один буфер; сообщение при этом может быть отправлено в несколько буферов. Многие драйверы не позволяют изменять значения этих параметров.

Тип адаптера	Код продукта	Атрибут ODM	Значение по умолчанию	Диапазон
Gigabit Ethernet PCI-X, SX или TX	5700, 5701, 5706, 507	txdesc_que_sz	512	128-1024, кратно 128

Ресурсы приема:

Некоторые адаптеры позволяют изменить объем ресурсов, выделяемых для приема пакетов из сети. Это может быть число (и даже размер) буферов приема, число дескрипторов приема DNA.

В некоторых драйверах может применяться несколько пулов буферов приема с буферами разных размеров, тонкая настройка которых может быть необходима для разной загрузки системы. В некоторых драйверах управление этими ресурсами выполняется внутренним образом, что не позволяет пользователю вносить изменения в эти значения.

Объем ресурсов, выделяемых для приема пакетов, должен быть достаточным для обработки пиковых нагрузок в сети. Получаемые пакеты помещаются драйвером сетевого интерфейса в очередь приема. Если список или кольцо дескрипторов приема переполнены, либо в системе нет доступных буферов, пакеты теряются, что приводит к необходимости повторной передачи этих пакетов отправителем. Параметры очереди приема можно настроить с помощью утилиты SMIT или команды **chdev** (за дополнительной информацией обращайтесь к разделу “Изменение параметров сети” на стр. 287). Максимальный размер очереди различен для каждого типа адаптера связи; его можно просмотреть с помощью клавиш **F4** или **Список** инструмента SMIT.

Таблица 11. Возможные размеры очереди приема адаптера PCI

Тип адаптера	Код продукта	Атрибут ODM	Значение по умолчанию	Диапазон
Адаптер Ethernet PCI IBM 10/100 Мбит/с	2968	rx_que_size rx_buf_pool_size	256 384	16, 32 ,64, 128, 26 16-2048
10/100 Mbps Ethernet PCI Adapter II	4962	rx_desc_que_sz rxbuf_pool_sz	512 1024	100-1024 512-2048
Gigabit Ethernet PCI (SX или TX)	2969, 2975	rx_queue_size	512	512 (fixed)
Gigabit Ethernet PCI-X (SX или TX)	5700, 5701, 5706, 5707,5717, 5768, 5271, 5274, 5767 и 5281	rxbuf_pool_sz rxdesc_que_sz	2048 1024	512-16384,1 128-3840,128
10 Gigabit PCI-X (SR или LR)	5718, 5719	rxdesc_que_sz rxbuf_pool_sz	1024 2048	128-1024, кратно 128 512-2048
ATM 155 (MMF или UTP)	4953, 4957	rx_buf4k_min	x60	x60-x200 (96-512)
ATM 622 (MMF)	2946	rx_buf4k_min rx_buf4k_max	256 [2] 0 [1]	0-4096 0-14000
FDDI	2741, 2742, 2743	RX_buffer_cnt	42	1-512

Примечание:

1. Атрибут **rx_buf4k_max** адаптера ATM задает максимальное число буферов в пуле буферов приема. Если это значение равно 0, драйвер выбирает число в зависимости от объема памяти системы (например, **rx_buf4k_max= thewall** * 6 / 320), но максимально допустимое значение для адаптера ATM 155 равно 9500, а для адаптера ATM 622 - 16360. Если буферы не используются их число уменьшается (до значения **rx_buf4k_min**).
2. Атрибут **rx_buf4k_min** адаптера ATM задает минимальное допустимое значение для числа свободных буферов пула. Драйвер пытается поддерживать в пуле указанное количество свободных буферов. Величина пула может быть расширена до значения **rx_buf4k_max**.

Команды просмотра и изменения атрибутов устройства:

Просмотреть верхнее ограничение на количество пакетов в очереди передачи, а также число ошибок *нет ресурса* или *нет буфера* можно с помощью нескольких утилит.

Можно воспользоваться как командой **netstat -v**, так и непосредственно статистическими утилитами адаптеров (**entstat** для адаптеров Ethernet, **tokstat** для адаптеров Token-Ring, **fddistat** для адаптеров FDDI, **atmstat** для адаптеров ATM, и т.д.).

Пример вывода команды **entstat** приведен в разделе “Статистика работы адаптера” на стр. 326. Кроме того, эти значения можно узнать с помощью утилиты **netstat -i**. Если в выводе этой команды значение *Oerrs* для интерфейса отлично от нуля, то, скорее всего, очередь вывода была переполнена.

Просмотр настроек сетевого адаптера:

Для просмотра конфигурации адаптера вызовите команду **lsattr -E -l** *имя-адаптера* или команду SMIT (**smitty commodev**).

С каждым адаптером связан свой набор переменных. Вот только некоторые примеры параметров, задающих длину очереди передачи: *sw_txq_size*, *tx_que_size*, *xmt_que_size*. Параметры длины очереди приема и/или пула буферов приема могут называться *rec_que_size*, *rx_que_size* или *rv_buf4k_min*.

Ниже приведен вывод команды **lsattr -E -l atm0** для адаптера 622 Мб/с IBM PCI. Здесь значение *sw_txq_size* равно 2048, а число буферов приема *rv_buf4K_min* - 256.

```
# lsattr -E -l atm0
adapter_clock  0           Предоставление таймера SONET                 Истина
alt_addr       0x0         Альтернативный адрес ATM MAC (12 шестн.цифр) Истина
busintr        99          Уровень прерываний шины                      Ложь
interface_type 0           Интерфейс Sonet или SDH                      Истина
intr_priority  3           Приоритет прерывания                         Ложь
max_vc         1024        Максимальное число  необходимых VC          Истина
min_vc         64          Минимальное число поддерживаемых VC          Истина
regmem         0xe0008000  Адрес шины памяти регистров адаптера         Ложь
rx_buf4k_max   0           Макс. число связанных буферов приема по 4 КБ. Истина
rx_buf4k_max   0           Мин. число связанных буферов приема по 4 КБ.  Истина
rx_checksum    yes         Проверять аппаратную контрольную сумму приема Истина
rx_dma_mem     0x4000000   Диапазон адресов памяти шины приема           Ложь
sw_txq_size    2048        Размер программной очереди передачи          Истина
tx_dma_mem     0x2000000   Диапазон адресов памяти шины передачи        Ложь
uni_vers       auto_detect Версия SVC UNI Version                       Истина
use_alt_addr   нет         Разрешить альтернативный адрес ATM MAC       Истина
virtmem        0xe0000000  Адрес шины памяти виртуальной памяти адаптера Ложь
```

Ниже в качестве примера приведены параметры адаптера Gigabit Ethernet PCI-X, выданные командой **lsattr -E -l ent0**. В данном примере значение атрибута *tx_que_size* равно 8192, атрибута *rxbuf_pool_sz* - 2048, а атрибута *rx_que_size* - 1024.

```
# lsattr -E -l ent0

alt_addr       0x000000000000   Альтернативный адрес Ethernet                Истина
busintr        163              Уровень прерываний шины                      Ложь
busmem         0xc0080000       Адрес памяти шины                            Ложь
chksum_offload yes              Проверять аппаратные конт. суммы приема и передачи Ложь
compat_mode    no               Обратная совместимость Gigabit Ethernet      Ложь
copy_bytes     2048             Копировать пакеты с разным числом байтов     Истина
flow_ctrl      yes              Применять управление потоком приема и передачи Истина
intr_priority  3                Приоритет прерывания               Ложь
intr_rate      10000            Максимальная частота прерываний адаптера     Истина
jumbo_frames   no               Передавать большие кадры                     Истина
large_send     yes              Повторное аппаратное разбиение на сегменты TX TCP  Истина
media_speed    Auto_Negotiation Быстродействие линии связи                   Истина
rom_mem        0xc0040000       Адрес памяти ROM                             Ложь
rx_hog         1000             Макс. число буферов rcv для прерывания rcv   Истина
rxbuf_pool_sz  2048             Пул буферов приема, значение 2 * rxdesc_que_sz Истина
rxdesc_que_sz  1024             Размер очереди дескриптора приема            Истина
slih_hog       10               Максимальное количество событий для прерывания Истина
tx_que_sz      8192             Размер программной очереди передачи          Истина
txdesc_que_sz  512              Размер очереди дескриптора TX                Истина
use_alt_addr   no               Разрешить альтернативный адрес Ethernet      Истина
```

Изменение параметров сети

Для изменения параметров работы сети рекомендуется применять команду **smitty**.

Выбрать конкретный тип устройства можно с помощью команды **smitty commodev**. После запуска команды на экране появится список, в котором необходимо выбрать устройство. Ниже приведен пример изменения параметров адаптера Ethernet с помощью команды **smitty commodev**:

```
                        Показать или изменить параметры адаптера Ethernet

Введите или выберите значения в полях ввода.
После внесения изменений нажмите клавишу Enter.

                                                [Поля ввода]
  Адаптер Ethernet                              ent2
  Описание                                      Адаптер PCI-X 10/100/1000 Base-TX (14106902)
  Состояние                                     Доступен
  Расположение                                  1V-08
  Размер очереди приема                        [1024]  +#
  Размер очереди передачи                      [512]   +#
  Размер очереди передачи программного обеспечения  [8192]  +#
  Передавать большие кадры                      нет +
  Включить аппаратное разбиение на фрагменты    да +
  Включить аппаратную проверку контрольной суммы  да +
```

```
  Пропускная способность                          Auto_Negotiation +
  Включить альтернативный адрес ETHERNET          нет +
  Альтернативный адрес ETHERNET                  [0x000000000000] +
  Применить изменение только для базы данных      нет +

F1=Справка          F2=Обновить         F3=Отмена           F4=Список
Esc+5=Сбросить      Esc+6=Команда       Esc+7=Изменить      Esc+8=Изображение
Esc+9=Оболочка      Esc+0=Выход         Enter=Выполнить
```

Для изменения значений параметров выполните следующие действия:

1. Отключите интерфейс с помощью следующей команды:

   ```
   # ifconfig en0 detach
   ```

   где *en0* задает имя адаптера.
2. Просмотрите параметры адаптера с помощью SMIT. Выберите **Устройства -> Связь ->** *тип-адаптера* **-> Изменить/Показать...**
3. Поместите курсор на поле, значение в котором нужно изменить, и нажмите **F4**. Появится диапазон или список допустимых значений.
4. Выберите нужное значение и нажмите Enter. Значение параметра будет обновлено в базе данных ODM.
5. Подключите адаптер с помощью следующей команды:

   ```
   # ifconfig en0 имя-хоста up
   ```

Кроме того, значения параметров можно изменить с помощью следующей команды:

```
# chdev -l [имя-интерфейса] -a [имя-атрибута]=значение
```

Например, для того чтобы присвоить параметру **tx_que_size** интерфейса en0 значение 128, нужно вызвать приведенную ниже последовательность команд. Обратите внимание, что этот драйвер поддерживает очереди передачи только четырех размеров, поэтому для изменения этого параметра удобнее воспользоваться командой SMIT.

```
# ifconfig en0 detach
# chdev -l ent0 -a tx_que_size=128
# ifconfig en0 имя-хоста up
```

Настройка максимального размера сегмента TCP

Максимальный размер пакетов, отправляемых протоколом TCP, в значительной степени влияет на пропускную способность, так как эффективность работы возрастает с увеличением размера пакетов.

Протокол TCP задает этот максимальный размер, который также называется Максимальный размер сегмента (MSS), для каждого соединения TCP. В случае сетей с непосредственным подключением TCP рассчитывает MSS на основе размера MTU сетевого интерфейса, из которого затем вычитаются размеры заголовков протокола. Например, для Ethernet, MTU которого равен 1500, после вычитания 20 байт заголовка IPv4 и 20 байт заголовка TCP размер MSS будет равен 1460.

Протокол TCP позволяет конечным системам в ходе создания соединения согласовывать оптимальное значение максимального размера сегмента (MSS). Каждая сторона указывает предлагаемый размер MSS в поле опций заголовка пакета TCP. Будет принято наименьшее из двух значений. Если конечная система не указывает MSS, то применяется значение 536 байт, не позволяющее получить высокую производительность.

Проблема заключается в том, что каждой конечной системе TCP известен только MTU сети, к которой она подключена. Размеры MTU других промежуточных сетей системе не известны. Таким образом, протокол TCP знает правильное значение MSS только в том случае, если обе конечные системы расположены в одной сети. В связи с этим способ оповещения о MSS, применяемый в TCP, зависит от конфигурации сети, что позволяет избежать фрагментации пакетов IP, передаваемых по сетям с меньшим размером MTU.

При выборе значения MSS, предлагаемого уровнем TCP при настройке соединения, учитывается взаимное расположение систем: в одной физической сети (адреса сети систем совпадают) или в разных (удаленных) сетях.

Хосты в одной сети:

Если целевая система подключена к той же сети, что и исходная система, то TCP выбирает значение MSS с учетом текущего значения MTU, установленного для сетевого интерфейса.

```
TCP MSS = MTU - размер заголовка TCP - размер заголовка IP.
```

Размер заголовка TCP составляет 20 байт, размер заголовка IPv4 составляет 20 байт, размер заголовка IPv6 составляет 40 байт.

Поскольку это максимальный размер MSS, при котором уровень IP не выполняет фрагментацию пакета, это значение является оптимальным. Следовательно, в локальных сетях изменять значение MSS не нужно.

Хосты в различных сетях:

Если целевая система расположена в удаленной сети, то стек TCP данной операционной системы предлагает установить MSS, определяемый описанным ниже способом.

Способ зависит от того, выполняется вычисление MTU маршрута или нет. Если вычисление MTU маршрута не выполняется (**tcp_pmtu_discover=0**), то TCP определяет применяемый MSS следующим образом:

1. Если размер MTU для этого маршрута указан с помощью команды **route add**, то MSS вычисляется в соответствии с заданным размером MTU.
2. Если для применяемого сетевого интерфейса определен параметр ISNO **tcp_mssdflt**, то в качестве MSS используется значение **tcp_mssdflt**.
3. Если не указано ни одно из этих значений, то протокол TCP применяет глобальное значение параметра **no tcp_mssdflt** Значение по умолчанию в этом случае составляет 1460 байт.

Вычисление MTU маршрута TCP:

По умолчанию в операционной системе AIX вычисление MTU маршрута выполняется. С помощью этой опции стек протокола определяет минимальный размер MTU во всех промежуточных сетях между двумя хостами, управляемыми сетевой опцией **tcp_pmtu_discover=1**.

В реализации функции поиска путей TCP MTU применяются не сообщения ICMP ECHO, а непосредственно пакеты TCP. Расширение ядра TCP/IP поддерживает таблицу PMTU, в которой хранится информация о поиске путей MTU. При создании соединения TCP с целевым узлом в таблицу PMTU добавляется запись для этого целевого узла. Значение PMTU соответствует значению MTU интерфейса исходящей связи.

В заголовке IP отправляемых пакетов TCP установлен бит отключения фрагментирования (DF). При передаче пакета TCP на сетевой маршрутизатор, значение MTU которого меньше размера этого пакета TCP, на исходный узел отправляется сообщение ICMP с информацией о том, что пересылка сообщения невозможна, так как его нельзя разбить на фрагменты. Если маршрутизатор, отправляющий сообщение об ошибке, соответствует RFC 1191, сообщение об ошибке ICMP содержит значение MTU сети. В противном случае для повторной передачи пакета TCP необходимо задать меньшее значение размера MTU из таблицы известных значений MTU расширения ядра AIX TCP/IP. При этом табличное значение PMTU для целевого узла заменяется на новое, меньшее значение MTU и пакет TCP отправляется еще раз. При создании последующих соединений TCP с этим целевым узлом применяется новое значение PMTU.

Команда **pmtu** позволяет просматривать и удалять записи PMTU. Ниже приведен пример выполнения команды **pmtu**:

```
# вывод pmtu

   цлв           шлз           Иф    pmtu     refcnt   redisc_t    exp

 --------------------------------------------------------------------------
```

```
10.10.1.3    10.10.1.5        en1    1500        2          9       0

10.10.2.5    10.10.2.33       en0    1500        1          0       0
```

Поскольку записи PMTU `refcnt` со значением 0 не применяются, система автоматически удаляет их, что позволяет предотвратить создание таблиц PMTU слишком большого размера. Такие записи удаляются через *pmtu_expire* минут после того, как записи `refcnt` присваивается значение 0. По умолчанию параметру сети *pmtu_expire* присвоено значение 10 минут. Для того чтобы предотвратить удаление записей PMTU, можно присвоить записи *pmtu_expire* значение 0.

В данной реализации поиска путей TCP MTU можно не применять создание копий маршрутов, что позволяет поддерживать относительно небольшой размер таблицы маршрутизации и упрощает управление ей.

Статические маршруты:

Для того чтобы переопределить значение по умолчанию для параметра MSS (1460 байт), можно задать статический маршрут к удаленной сети.

Размер MTU для этого маршрута задается в опции **-mtu** команды **route**. Укажите в этом параметре фактический минимальный MTU маршрута. На его основе будет автоматически вычислен оптимальный размер MSS. Например, следующая команда задает для пути к сети 192.3.3 размер MTU по умолчанию, равный 1500, а также хост по умолчанию для обращения к шлюзу - `en0host2`:

```
# route add -net 192.1.0 jack -mtu 1500
1500 net 192.3.3: gateway en0host2
```

Команда **netstat -r** показывает таблицу маршрутизации, в которой указан размер PMTU, равный 1500 байт. В соответствии с данным размером MTU вычисляется MSS. Ниже приведен пример вывода команды **netstat -r**:

```
# netstat -r
Таблицы маршрутизации
Пункт назн.       Шлюз          Флаги     Ссылк.  Исп.   Инт.   PMTU Exp Groups

Дерево маршрутов для группы протоколов 2 (Internet):
default          res101141         UGc       0        0  en4     -   -
ausdns01.srv.ibm res101141         UGHW      8       40  en4  1500   -
10.1.14.0        server1           UHSb      0        0  en4     -   -  =>
10.1.14/24       server1           U         5     4043  en4     -   -
server1          loopback          UGHS      0      125  lo0     -   -
10.1.14.255      server1           UHSb      0        0  en4     -   -
127/8            loopback          U         2  1451769  lo0     -   -
192.1.0.0        en0host1          UHSb      0        0  en0     -   -  =>
192.1.0/24       en0host1          U         4       13  en0     -   -
en0host1         loopback          UGHS      0        2  lo0     -   -
192.1.0.255      en0host1          UHSb      0        0  en0     -   -
192.1.1/24       en0host2          UGc       0        0  en0     -   -
en1host1         en0host2          UGHW      1   143474  en0  1500   -
192.3.3/24       en0host2          UGc       0        0  en0  1500   -
192.6.0/24       en0host2          UGc       0        0  en0     -   -

Дерево маршрутов для группы протоколов 24 (Internet v6):
loopbackv6       loopbackv6        UH        0        0  lo0 16896   -
```

Примечание: Команда **netstat -r** не позволяет просмотреть значение PMTU. Сделать это можно с помощью команды **pmtu display**. При добавлении маршрута до конечного узла с помощью команды **route add** и установке значения MTU в таблице PMTU создается запись PMTU для этого конечного узла.

Такой способ выбора оптимального MSS для конкретных маршрутов между сетями рекомендуется применять в среде с постоянной конфигурацией. Ниже перечислены недостатки такого подхода:

- Такой способ неприменим для динамической маршрутизации.

- Неэффективен при росте числа удаленных сетей.
- Статический маршрут должен быть определен в обеих конечных системах.

Применение параметра tcp_mssdflt команды no:

С помощью опции *tcp_mssdflt* можно задать максимальный размер пакета для соединений с удаленными сетями.

Глобальная опция *tcp_mssdflt* команды **no** применяется при создании соединений со всеми сетями. Однако, если сетевой интерфейс поддерживает ISNO, то для него опцию *tcp_mssdflt* можно указать отдельно. Это значение переопределяет глобальное значение команды **no** для маршрутов, проходящих через данную сеть.

Опция *tcp_mssdflt* представляет собой размер MSS протокола TCP, который задает размер данных TCP. Для вычисления размера MSS следует вычесть из требуемого размера MTU 40 байт (20 для заголовка IP и 20 для заголовка TCP). Настройка других опций протокола не требуется, так как она автоматически выполняется протоколом TCP, если применяются другие опции, такие как *rfc1323*.

В сетях с MTU, превышающим значение по умолчанию, этот способ позволяет избежать задания MSS отдельно для этой сети. Однако у такого способа вычисления MSS есть и некоторые недостатки:

- Увеличение значения по умолчанию может привести к фрагментации пакета на маршрутизаторах IP, если целевая система в действительности находится за пределами рассматриваемой среды и размеры MTU промежуточных сетей неизвестны.
- Значения параметра *tcp_mssdflt* на исходном и целевом хосте должны совпадать.

Примечание: Начиная с версии AIX, опцию *tcp_mssdflt* можно применять только в случае, если опция *tcp_pmtu_discover* равна 0.

Создание подсетей и опция subnetsarelocal команды no:

Опция *subnetsarelocal* команды **no** позволяет управлять способом определения локальных (подключенных к той же сети) и удаленных конечных систем.

Подсетями называются физические сети, которым присвоен общий адрес сети. С помощью опции *subnetsarelocal* можно изменить системную переменную, указывающую, каким образом должны рассматриваться подсети: как локальные или как удаленные сети. Если будет вызвана команда **no -o subnetsarelocal=1** (значение по умолчанию), то хост A из подсети 1 будет считать, что хост B из подсети 2 расположен в той же физической сети.

В результате при установлении соединения между хостами A и B значение MSS будет выбираться исходя из того, что они расположены в одной сети. Каждый из хостов предложит значение MSS, вычисленное на основании MTU своего сетевого интерфейса, и будет пытаться установить оптимальное значение MSS.

Ниже перечислены преимущества такого подхода:

- Не требуются статические связывания; согласование MSS происходит автоматически.
- Этот метод не отменяет и не заменяет согласование MSS TCP, что позволяет выбрать оптимальное значение даже в том случае, если MTU смежных подсетей немного отличаются.

У такого подхода есть и некоторые недостатки:

- Возможна фрагментация пакета на маршрутизаторе IP, если две сети с большими значениями MTU соединяются через сеть с меньшим MTU. Такая ситуация проиллюстрирована на следующем рисунке.

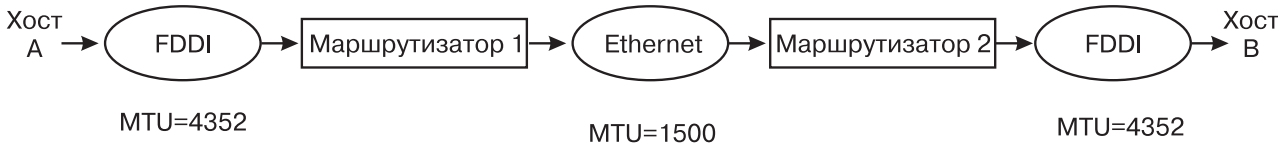


Рисунок 21. Фрагментация при передаче данных между подсетями. На этом рисунке показан маршрут передачи данных от хоста A. Он проходит через сеть FDDI с MTU=4352, маршрутизатор 1 и сеть Ethernet с MTU=1500. После этого данные передаются через маршрутизатор 2 в сеть FDDI с MTU=4352, в которой расположен целевой хост B.

- В данном примере хосты A и B устанавливают соединение с MTU 4352. Пакет, отправленный хостом A хосту B, будет разбит на фрагменты на маршрутизаторе 1 и собран на маршрутизаторе 2. При передаче данных от B к A маршрутизаторы меняются ролями.
- Как исходный, так и целевой хост должны рассматривать подсети как локальные сети.

Примечание: Если параметру *tcp_pmtu_discover* присвоено значение 1, то при вычислении значения MSS применяется MTU интерфейса исходящей связи. Значение параметра *subnetsarelocal* учитывается только в случае, если параметр сети *tcp_pmtu_discover* равен 0.

Рекомендации по настройке производительности протокола IP

В этом разделе описаны рекомендации по настройке производительности протокола IP.

На уровне IP предусмотрен только параметр *ipqmaxlen*, который задает размер очереди приема IP. Интерфейсы, как правило, не применяют очередь данных. Если пакеты поступают очень быстро, то очередь приема IP может переполниться. Для того чтобы узнать число переполнений, вызовите команду **netstat -s** или **netstat -p ip** (поле `число переполнений ipintrq`).

Если значение больше нуля, значит очередь переполнялась. В этом случае установите максимальный размер очереди с помощью команды **no**. Например:

```
# no -o ipqmaxlen=100
```

В данном примере очередь может содержать до 100 пакетов. Оптимальное значение нужно выбрать исходя из максимальной скорости поступления пакетов. Если такой информации нет, увеличьте длину очереди исходя из числа переполнений. При увеличении длины очереди дополнительная память не расходуется. Однако увеличение размера очереди может привести к увеличению времени, которое затрачивается на обработку прерываний, так как уровню IP потребуется обработать больше пакетов из очереди. Это может замедлить работу других процессов, которым также требуется CPU. Таким образом, вы можете сократить число отбрасываемых пакетов за счет уменьшения доли процессорного времени, доступного другим процессам. Если другие приложения также должны выполняться быстро, не рекомендуется значительно увеличивать значение *ipqmaxlen*.

Настройка производительности пула mbuf

Сетевая подсистема управляет памятью с помощью специальной утилиты, в которой основную роль играет структура данных, называемая mbuf.

Пулы mbuf в основном применяются для хранения в ядре данных, получаемых и отправляемых по сети. При правильно выбранном размере пулов mbuf производительность сети значительно увеличивается. В свою очередь, неправильная настройка пулов mbuf может снизить производительность сети и системы. Верхнее ограничение размера пула mbuf, которое задается с помощью параметра *thewall*, автоматически определяется операционной системой в соответствии с объемом доступной памяти системы. Настраивать верхнее ограничение размера пула mbuf может только системный администратор.

Параметр thewall

Сетевой параметр *thewall* задает верхнее ограничение размера буферов сетевого ядра.

Система автоматически присваивает параметру *thewall* максимальное возможное значение. Как правило изменять его не нужно. Уменьшение этого значения приведет к уменьшению объема памяти, выделяемой системой для сетевых буферов, что может снизить производительность. Так как система применяет только необходимое число буферов, то в случае небольшой нагрузки на сетевую подсистему фактическое число буферов будет значительно меньше значения параметра *thewall*.

Единицей измерения для *thewall* служит 1 КБ, так что значение 1 048 576 байт означает 1024 МБ или 1 ГБ оперативной памяти.

Ограничения ресурсов mbuf

AIX 6.1 позволяет выделить для буферов mbuf до 65 ГБ оперативной памяти, состоящей из 260 сегментов по 256 МБ.

Значение параметра **thewall** составляет меньшее из следующих двух значений: 65 ГБ или половина объема оперативной памяти системы.

Параметр maxmbuf

Значение параметра *maxmbuf* ограничивает объем памяти, используемой подсистемой связи.

Кроме того, параметр *maxmbuf* позволяет снизить ограничение *thewall*. Для того чтобы просмотреть текущее значение параметра *maxmbuf*, введите команду **lsattr -E -l sys0**. Если значение *maxmbuf* больше 0, то значение *maxmbuf* применяется независимо от значения *thewall*.

Значение параметра *maxmbuf* по умолчанию равно 0. Нулевое значение *maxmbuf* указывает, что применяется параметр *thewall*. Для того чтобы изменить значение параметра *maxmbuf*, введите команду **chdev** или **smitty**.

Параметры sockthresh и strthresh

Параметры *sockthresh* и *strthresh* задают верхние ограничения для открытия новых сокетов, соединений TCP, а также для создания новых потоковых ресурсов. Такой подход позволяет обеспечить существующие сеансы и соединения ресурсами, необходимыми для продолжения работы.

Параметр *sockthresh* задает ограничение на использование памяти. Число сокетов не может превышать значение, указанное в параметре *sockthresh*. По умолчанию значение параметра *sockthresh* составляет 85%. Если общий объем выделенной памяти достигнет 85% от значения, указанного в параметре *thewall* или *maxmbuf*, то создание новых сокетов будет запрещено. При этом системные вызовы **socket()** и **socketpair()** будут возвращать значение *ENOBUFS* до тех пор, пока использование памяти не снизится ниже 85%.

Аналогичным образом параметр *strthresh* ограничивает объем памяти mbuf, используемой для потоковых ресурсов. Значение *strthresh* по умолчанию равно 85%. Подсистемы async и TTY работают в потоковых средах. Если общий объем выделенной памяти достигнет 85% от значения параметра *thewall*, то дополнительная память потоковым ресурсам выделяться не будет. При этом вызовы для открытия потоков, передачи модулей и записи в потоковые устройства будут возвращать значение *ENOSR*.

Для изменения значений параметров *sockthresh* и *strthresh* применяется команда **no**.

Утилита управления пулами mbuf

Утилита управления пулами mbuf контролирует размер пула, который может составлять от 32 до 16384 байт.

Эти пулы создаются путем отправки запроса на выделение памяти Администратору виртуальной памяти (VMM). Для пулов выделены закрепленные страницы памяти ядра, то есть страницы виртуальной памяти, которые никогда не выгружаются на диск. В результате объем физической памяти, который может применяться приложениями для хранения данных и подкачки, уменьшается на размер пулов mbuf.

Сетевой пул памяти распределяется равномерно между всеми процессорами. Каждый пул процессора затем разбивается на блоки размером от 32 до 16 384 байт. Каждый блок может "занимать" память у других

блоков того же процессора, однако процессор не может "занимать" память у пула другого процессора. Когда сетевой службе требуется передать данные, она может вызвать службу ядра, например **m_get()**, для получения буфера памяти. Если есть свободный буфер, закрепленный в памяти, он может быть предоставлен немедленно. Если буфер не закреплен в памяти и верхнее ограничение не достигнуто, то буфер выделяется и закрепляется. Закрепленная память остается в этом состоянии до тех пор, пока она не будет освобождена и возвращена в сетевой пул памяти. Когда число свободных буферов достигает порогового значения, часть из них открепляется и возвращается системе. Такое открепление выполняется процессом ядра **netm()**. При вызове функции **m_get()** можно указать, нужно ли ждать появления сетевого буфера памяти. Если указан флаг **M_DONTWAIT** и свободных закрепленных буферов на момент вызова нет, то счетчик сбоев увеличивается на единицу. Если указан флаг **M_WAIT**, то процесс приостанавливается до тех пор, пока какой-либо буфер не будет выделен и закреплен.

Отслеживание пулов mbuf с помощью команды netstat -m

Команда **netstat -m** позволяет обнаружить неполадки, связанные с запросами сетевой памяти (буферы mbuf или кластеры).

Команда **netstat -Zm** позволяет очистить (обнулить) статистику mbuf. Такой подход позволяет выполнять проверку с учетом только новой статистики. В команде **netstat -m** предусмотрены следующие поля:

Имя поля
Определение

По размеру
Показывает размер буфера.

занято Показывает число используемых буферов заданного размера.

вызов Показывает число вызовов или запросов на выделение для буферов всех размеров.

отказов
Показывает число запросов на выделение, которые не были выполнены из-за отсутствия доступных буферов.

отложено
Показывает число отложенных вызовов, если буферов соответствующего размера нет и в вызове был задан флаг **M_WAIT**.

свободно
Показывает число буферов всех размеров, перечисленных в списке свободной оперативной памяти и доступных для выделения.

hiwat Показывает максимальное число буферов в списке свободной оперативной памяти, определенное системой. Свободные буферы сверх этого числа постепенно возвращаются системе.

освобождено
Показывает число буферов, освобожденных после превышения ограничения **hiwat**.

Число, показанное в поле **отказов**, не должно быть большим. Как правило в нем указано небольшое значение, связанное с выделением системой большего числа буферов при увеличении размера пула буферов. После каждой перезагрузки система создает предопределенное число начальных буферов всех размеров. При необходимости число этих буферов увеличивается.

Ниже приведен пример вывода команды **netstat -m**, выполненной с двухпроцессорной системе:

```
# netstat -m

Статистика выделения памяти ядром:

******* CPU 0 *******
Размер       занято   вызовов отказов отложено свободно  hiwat освобождено
32                68       693      0         0      60    2320       0
64                55       115      0         0       9    1160       0
128               21       451      0         0      11     580       0
```

```
256                1064      5331      0         0    1384    1392      42
512                  41       136      0         0       7     145       0
1024                 10       231      0         0       6     362       0
2048               2049      4097      0         0     361     362     844
4096                  2         8      0         0     435     435     453
8192                  2         4      0         0       0      36       0
16384                 0       513      0         0      86      87     470


******* CPU 1 *******
Размер       занято   вызовов отказов отложено свободно  hiwat освобождено
32                139       710      0         0     117    2320       0
64                 53       125      0         0      11    1160       0
128                41       946      0         0      23     580       0
256                62      7703      0         0    1378    1392     120
512                37       109      0         0      11     145       0
1024               21       217      0         0       3     362       0
2048                2      2052      0         0     362     362     843
4096                7        10      0         0     434     435     449
8192                0         4      0         0       1      36       0
16384               0      5023      0         0      87      87    2667


***** Буферы размером более 16384 байт *****

Размер       занято   вызовов отказов отложено свободно  hiwat освобождено
65536               2         2      0         0       0    4096       0

Отказы на выделение потоковых mblk:
0 отказов на выделение mblk с высоким приоритетом
0 отказов на выделение mblk со средним приоритетом
0 отказов на выделение mblk с низким приоритетом
```

Настройка кэша ARP

ARP (протокол преобразования адресов) - это протокол, применяемый для преобразования 32-разрядных IP-адресов IPv4 в 48-разрядный адрес адаптера хоста, используемый протоколом передачи данных.

Преобразование адресов выполняется прозрачно для системы. Однако, система поддерживает кэш ARP, который представляет собой таблицу 32-разрядных IP-адресов и соответствующих им 48-разрядных адресов хостов. Если среда содержит большое число подключенных клиентов, то рекомендуется изменить размер кэша ARP. Это можно выполнить командами **no** и **netstat**.

Команда **no** настраивает параметры сети. Для ARP поддерживаются следующие аргументы:

- **arpqsize = 12**
- **arpt_killc = 20**
- **arptab_bsiz = 7**
- **arptab_nb = 149**

Размер таблицы ARP определяется числом сегментов, задаваемым в параметре *arptab_nb*. Каждый сегмент содержит число записей, определенное параметром **arptab_bsiz**. По умолчанию создается 149 сегментов, каждый из которых содержит по 7 записей, так что всего в таблице может храниться 1043 (149 x 7) адреса хостов. Это значение по умолчанию будет работать для систем, которые одновременно обмениваются данными не более чем с 1043 системами в сети IP. Если сервер связан более чем с 1043 системами в сети, то таблица ARP будет слишком мала, вследствие чего будет снижена производительность. Операционная система должна удалить запись из кэша и заменить ее новым адресом. Это приводит к тому, что пакеты TCP или UDP будут ждать в очереди, пока протокол ARP не поменяет информацию об адресах. Параметр *arpqsize* определяет, сколько таких ожидающих пакетов может поставить в очередь слой ARP, пока не будет получен ответ на запрос ARP. При переполнении очереди ARP отправляемые пакеты TCP или UDP будут потеряны.

Частое обновление кэша ARP может привести к снижению производительности. Это происходит по следующей причине:

1. Текущий отправляемый пакет должен ждать, пока в сети не будет найден адрес ARP.
2. Из кэша ARP должна быть удалена еще одна запись ARP. Если нужны все адреса, то в случае, когда на адрес хоста, который был удален, были отправлены пакеты, требуется еще один адрес.
3. Может произойти переполнение очереди вывода ARP, что приведет к потере пакетов.

Параметры *arpqsize*, *arptab_bsiz* и *arptab_nb* являются параметрами перезагрузки, так как при их изменении необходимо перезагрузить систему. Это связано с тем, что при изменении значений указанных параметров изменяются таблицы, создаваемые при загрузке системы или при загрузке TCP/IP.

Параметр *arpt_killc* определяет время ожидания (в минутах) перед удалением записи ARP. По умолчанию значение параметра *arpt_killc* равно 20 минутам. Записи ARP удаляются из таблицы каждые **arpt_killc** минут. Это позволяет отслеживать возможные изменения 48-разрядного адреса хоста, например, при замене сетевого адаптера. Эта процедура гарантирует, что из кэша удаляются все устаревшие записи, например, записи, которые могли бы помешать установить соединение с таким хостом из-за использования старого адреса. При увеличении этого времени сокращается время, затрачиваемое системой на поиск ARP, но при этом может увеличиться время хранения устаревших адресов хоста. *arpt_killc* - это динамический параметр, поэтому его можно изменять в процессе работы, не перезагружая систему.

Для получения статистики ARP введите команду **netstat -p arp**. Эта команда выдает следующую информацию: сколько всего запросов ARP было отправлено; сколько пакетов было удалено из таблицы при удалении записи, для того чтобы освободить место для новой записи. Если число удаленных пакетов велико, размер таблицы ARP необходимо увеличить. Ниже приведен пример команды **netstat -p arp**.

```
# netstat -p arp

arp:
     6 пакетов отправлено
     0 пакетов удалено
```

Команда **arp -a** позволяет просмотреть содержимое таблицы ARP. Вывод команды содержит информацию о том, какие адреса есть в таблице, как они хэшируются и к каким сегментам относятся.

```
 ? (10.3.6.1) at 0:6:29:dc:28:71 [ethernet] stored

сегмент:    0     содержит:    0 записей
сегмент:    1     содержит:    0 записей
сегмент:    2     содержит:    0 записей
сегмент:    3     содержит:    0 записей
сегмент:    4     содержит:    0 записей
сегмент:    5     содержит:    0 записей
сегмент:    6     содержит:    0 записей
сегмент:    7     содержит:    0 записей
сегмент:    8     содержит:    0 записей
сегмент:    9     содержит:    0 записей
сегмент:   10     содержит:    0 записей
сегмент:   11     содержит:    0 записей
сегмент:   12     содержит:    0 записей
сегмент:   13     содержит:    0 записей
сегмент:   14     содержит:    1 запись
сегмент:   15     содержит:    0 записей


...строки пропущены...

В таблице ARP содержится 1 запись.
```

Настройка преобразования имен

Процесс получения IP-адреса хоста по его имени в протоколе TCP/IP называется *преобразованием имен*.

Обратный процесс - преобразование IP-адреса в имя хоста - называется *обратным преобразованием имен*. Для преобразования имен служит *функция преобразования*. В поисках нужной информации она опрашивает DNS, NIS и, наконец, просматривает локальный файл `/etc/hosts`.

Если вы знаете, какая служба должна применяться для преобразования имен, то вы можете ускорить процесс преобразования имен, изменив порядок поиска по умолчанию. Для этого измените файл `/etc/netsvc.conf` или переменную среды *NSORDER*.

- Если применяются и файл `/etc/netsvc.conf`, и переменная среды *NSORDER*, то последняя имеет больший приоритет. Для того чтобы задать порядок поиска хостов в файле `/etc/netsvc.conf`, создайте файл, указав в нем следующую строку:

  ```
  hosts=значение,значение,значение
  ```

  где вместо *значение* может быть указано (только строчными буквами) `bind`, `local`, `nis`, `bind4`, `bind6`, `local4`, `local6`, `nis4` или `nis6` (для `/etc/hosts`). Список задается на одной строке; значения разделяются запятыми. Между запятыми и знаком равенства допускаются пробелы.

  Указанные значения и их порядок зависят от конфигурации сети. Например, если локальная сеть является одноуровневой, то необходим только файл `/etc/hosts`. Файл `/etc/netsvc.conf` будет содержать следующую строку:

  ```
  hosts=local
  ```

  Переменная среды *NSORDER* будет выглядеть так:

  ```
  NSORDER=local
  ```

- Если локальная сеть является сетью домена, преобразующей имена с помощью сервера имен и резервного файла `/etc/hosts`, то укажите обе службы. Файл `/etc/netsvc.conf` будет содержать следующую строку:

  ```
  hosts=bind,local
  ```

  Переменная среды *NSORDER* будет выглядеть так:

  ```
  NSORDER=bind,local
  ```

Согласно алгоритму, сначала будет опрошен первый источник в списке. Затем будет выбрана другая служба, если выполнено одно из следующих условий:

- Текущая служба не запущена, и, следовательно, недоступна.
- Текущей службе не удалось найти имя и она не является ответственной за данную сеть.

Анализ производительности сети

Снижение производительности не обязательно связано с работой самой системы. Оно может быть вызвано работой других устройств, подключенных к сети, находящихся во многих километрах от исходной системы.

Для того чтобы узнать, влияет ли работа с сетью на общую производительность системы, достаточно сравнить скорость выполнения сетевых операций и операций, выполняемых без участия сети. Если программа, выполняющая много операций удаленного чтения и записи, работает медленно, а все остальные программы выполняются достаточно быстро, то, скорее всего, первая программа работает медленно из-за низкой производительности сети. Чаще всего низкая производительность сети связана со следующими причинами:

- Сетевой интерфейс клиента
- Пропускная способность сети
- Топология сети
- Сетевой интерфейс сервера
- Процессор сервера
- Память сервера
- Производительность сервера
- Неэффективная настройка

Существует множество параметров сети, которые можно изменить с помощью инструментальных средств, однако на производительность влияют только некоторые из них.

Для настройки производительности применяются команды **no** (изменение параметров сети) и **nfso** (изменение параметров NFS). Кроме того, часть параметров настраивается командами **chdev** и **ifconfig**.

Команда ping

Команда **ping** позволяет проверять собственную связь и состояние удалённых хостов, находить и локализировать неполадки в программном и аппаратном обеспечении, тестировать качество сетей

Ниже перечислены опции команды **ping**, связанные с настройкой производительности:

-c Задает число пакетов. Эта опция применяется при создании протокола трассировки IP-пакетов. Задает минимальное число пакетов **ping**, которое будет получено.

-s Задает длину пакетов. Эта опция применяется для проверки работы функций фрагментации и повторной сборки пакетов.

-f Отправляет пакеты через интервал в 10 мс или немедленно после очередного ответа. Эта опция может быть указана только пользователем root.

Опция **-f** часто применяется для эмуляции максимальной нагрузки на сеть и подключенные к ней системы. Например, если вы подозреваете, что низкая производительность связана с высокой нагрузкой, вы можете искусственно загрузить сеть для подтверждения этой гипотезы. Откройте несколько окон **aixterm** и запустите в каждом из них команду **ping -f**. Нагрузка на адаптер Ethernet быстро достигнет 100 процентов. Пример приведен ниже:

```
# date; ping -c 1000 -f 192.1.6.1 ; date
Thu Feb 12 10:51:00 CST 2004
PING 192.1.6.1 (192.1.6.1): 56 байт данных
.
--- статистика ping 192.1.6.1 ---
1000 пакетов передано, 1000 пакетов получено, 0% пакетов потеряно
время оборота мин/сред/макс = 1/1/23 мс
Thu Feb 12 10:51:00 CST 2004
```

Примечание: Команду **ping** следует использовать с осторожностью, так как она сильно загружает сеть. Ее разрешено вызывать только пользователю root.

В этом примере за 1 секунду было отправлено 1000 пакетов. Учтите, что данная команда применяет протоколы IP и ICMP и, таким образом, не задействует транспортный протокол (UDP/TCP) и прикладной уровень. Полученные данные, такие как время оборота пакета, не дают полное представление о производительности.

При попытке загрузить соединение с удаленной системой учтите следующее:

- Отправка пакетов создает дополнительную нагрузку и на локальную систему.
- Во время эксперимента отслеживайте состояние сетевого интерфейса командой **netstat -i**. С помощью поля `Oerrs` можно узнать, отбрасывает ли система пакеты при отправке.
- Также рекомендуется отслеживать другие ресурсы, такие как буферы и очередь ввода-вывода. Максимальной загруженности удаленной системы добиться достаточно сложно. Первой может быть достигнут предел производительности локальной системы.
- Учтите, что полученные результаты являются лишь приблизительными. Для исключения влияния локальной системы, сети и маршрутизаторов повторите тесты на разных системах.

Команда ftp

С помощью команды **ftp** можно передать очень большой объем данных, указав **/dev/zero** в качестве ввода и **/dev/null** в качестве вывода. Это позволяет избежать использования дисков (ограничивающих пропускную способность) и кэширования всего передаваемого файла в памяти.

Введите следующие команды **ftp** (при необходимости увеличьте или уменьшите число блоков, считываемых командой **dd**):

```
> bin
> put "|dd if=/dev/zero bs=32k count=10000" /dev/null
```

Приведенная выше команда передает 10000 блоков данных по 32 КБ каждый. Для увеличения или уменьшения размера передаваемого файла измените количество блоков, считываемых командой **dd**, указанных в параметре `count`, либо размер блока, указанный в параметре `bs`. Обратите внимание, что в команде **ftp** по умолчанию применяется тип передачи `ASCII`, что замедляет обработку, поскольку необходимо просматривать каждый байт. По возможности следует применять для передачи двоичный режим (`bin`).

При работе с сетями Gigabit Ethernet, поддерживающем большие кадры, либо ATM с размером MTU не менее 9180 убедитесь, что параметрам **tcp_sendspace** и **tcp_recvspace** присвоены значения не меньше 65535. Для достижения максимальной производительности рекомендуется установить значение 131072 байт (128 КБ). Если для настройки адаптеров Gigabit Ethernet применяется **SMIT**, необходимо правильно задать системные значения ISNO по умолчанию. При запуске сетевых интерфейсов с помощью команды **ifconfig** задаются не оптимальные значения параметров ISNO.

Ниже приведен пример настройки параметров:

```
# no -o tcp_sendspace=65535
# no -o tcp_recvspace=65535
```

Соответствующие команды **ftp**:

```
ftp> bin
200 Type set to I.
ftp> put "|dd if=/dev/zero bs=32k count=10000" /dev/null
200 PORT command successful.
150 Opening data connection for /dev/null.
10000+0 records in
10000+0 records out
226 Transfer complete.
327680000 байт отправлено за 2,789 с (1,147e+05 КБ/с)
local: |dd if=/dev/zero bs=32k count=10000 remote: /dev/null
ftp> quit
221 Goodbye.
```

Описанная выше передача данных выполнялась между двумя адаптерами Gigabit Ethernet, в которых применялись MTU размером 1500 байт; скорость передачи данных была равна 114700 Кб/с, что эквивалентно 112 Мб/с или 940 Мбит/с.

Если отправитель и получатель применяют большие кадры с MTU 9000, то скорость передачи данных достигает 120700 Кб/с или 117,87 Мб/с (989 Мбит/с), как показано в приведенном ниже примере:

```
ftp> bin
200 Type set to I.
ftp> put "|dd if=/dev/zero bs=32k count=10000" /dev/null
200 PORT command successful.
150 Opening data connection for /dev/null.
10000+0 records in
10000+0 records out
226 Transfer complete.
327680000 байт передано за 2,652 секунд (1,207e+05 Кб/с)
local: |dd if=/dev/zero bs=32k count=10000 remote: /dev/null
```

Ниже приведен пример передачи данных по **ftp** между двумя интерфейсами Ethernet 10/100 Мбит/с:

```
ftp> bin
200 Type set to I.
ftp> put "|dd if=/dev/zero bs=32k count=10000" /dev/null
200 PORT command successful.
```

```
150 Opening data connection for /dev/null.
10000+0 records in
10000+0 records out
226 Transfer complete.
327680000 байт передано за 27,65 секунд (1,157e+04 Кбайт/с)
local: |dd if=/dev/zero bs=32k count=10000 remote: /dev/null
```

Пропускная способность в приведенном выше примере составляет 11570 Кб/с, что эквивалентно 11,3 Мб/с или 94,7 Мбит/с.

Команда netstat

Для проверки состояния сети используется команда **netstat**.

Обычно она применяется для устранения неполадок, а не для измерения параметров производительности. Однако с помощью команды **netstat** можно получить информацию об объеме данных, передаваемых по сети, и узнать, связана ли низкая производительность с работой в сети.

Команда **netstat** выдает следующую информацию о передаче данных через сетевые интерфейсы:

- Адреса всех управляющий блоков протокола, связанных с сокетами, и состояние всех сокетов
- Число пакетов, полученных, переданных и отброшенных подсистемой сетевого ввода-вывода
- Полную статистику для каждого интерфейса
- Маршруты и их состояние

Применение команды netstat:

Команда **netstat** показывает содержимое различных структур данных, связанных с активными сетевыми соединениями.

Команда netstat -in:

Эта функция **netstat** показывает состояние всех настроенных интерфейсов.

В приведенном ниже примере показаны статистические данные для рабочей станции с интегрированным адаптером Ethernet (`en1`), адаптером PCI-X Gigabit Ethernet (`en0`) и адаптером Fibre Channel Adapter, настроенным для TCP/IP (`fc0`):

```
# netstat -in
Имя   Mtu   Сеть          Адрес              Ipkts Ierrs    Opkts Oerrs  Coll
en1   1500  link#2      0.9.6b.3e.0.55       28800     0      506     0     0
en1   1500  10.3.104    10.3.104.116         28800     0      506     0     0
fc0   65280 link#3      0.0.c9.33.17.46         12     0       11     0     0
fc0   65280 192.6.0     192.6.0.1               12     0       11     0     0
en0   1500  link#4      0.2.55.6a.a5.dc         14     0       20     5     0
en0   1500  192.1.6     192.1.6.1               14     0       20     5     0
lo0   16896 link#1                          33339     0    33343     0     0
lo0   16896 127         127.0.0.1            33339     0    33343     0     0
```

Статистика собирается с момента запуска системы.

Имя Название интерфейса.

Mtu Максимальный блок передачи. Максимальный размер передаваемого через интерфейс пакета (в байтах).

Ipkts Полное число полученных пакетов.

Ierrs Общее число ошибок ввода. Это число учитывает пакеты неверного формата, неправильные контрольные суммы, а также случаи переполнения буфера драйвера устройства.

Opkts Общее число переданных пакетов.

Oerrs Общее число ошибок вывода. Это значение учитывает сбои при установлении соединения и случаи переполнения очереди вывода адаптера.

Coll Число обнаруженных конфликтов передачи.

Примечание: Команда **netstat -i** не показывает число конфликтов для интерфейсов Ethernet (просмотреть статистические данные для Ethernet можно с помощью команды entstat).

Ниже приведены некоторые рекомендации по настройке:

- Если в выводе команды **netstat -i** число ошибок ввода превышает 1 процент от общего числа полученных пакетов, то есть

  ```
  Ierrs > 0.01 x Ipkts
  ```

  То запустите команду **netstat -m**, чтобы проверить, достаточно ли памяти.
- Если в выводе команды **netstat -i** число ошибок вывода превышает 1 процент от общего числа отправленных пакетов, то есть

  ```
  Oerrs > 0.01 x Opkts
  ```

  То следует увеличить длину очереди передачи интерфейса (*xmt_que_size*). Текущее значение параметра *xmt_que_size* можно узнать с помощью следующей команды:

  ```
  # lsattr -El адаптер
  ```
- Если частота конфликтов превышает 10 процентов, то есть

  ```
  Coll / Opkts > 0.1
  ```

  то сеть перегружена, и необходима ее реорганизация или разделение на несколько подсетей. Для определения частоты конфликтов вызовите команду **netstat -v** или **entstat**.

Команда netstat -i -Z:

Команда **netstat** обнуляет все статистические счетчики команды **netstat -i**.

netstat -I интерфейс интервал:

Эта функция **netstat** выдает статистическую информацию о работе указанного интерфейса.

Выводимая информация схожа с выводом команды **netstat -i**, однако она будет включать в себя только сведения о заданном интерфейсе, собранные за указанный интервал времени. Например:

```
# netstat -I en0 1
    ввод     (en0)     вывод            ввод    (всего)    вывод
   пакетов ошибок пакетов ошибок конфл. пакетов ошибок пакетов ошибок конфл.
       0     0       27     0     0   799655     0   390669     0     0
       0     0        0     0     0        2     0        0     0     0
       0     0        0     0     0        1     0        0     0     0
       0     0        0     0     0       78     0      254     0     0
       0     0        0     0     0      200     0       62     0     0
       0     0        1     0     0        0     0        2     0     0
```

В предыдущем примере показан вывод команды **netstat -I** для интерфейса `ent0`. Параллельно выводятся два отчета: для указанного интерфейса и для всех интерфейсов (`Все`). Вывод этой команды аналогичен выводу команды **netstat -i**; например, `ввод, пакетов` = `Ipkts`, `ввод, ошибок` = `Ierrs` и т.п.

Команда netstat -a:

Команда **netstat -a** позволяет просмотреть сведения о состоянии всех сокетов.

Если флаг **-a** не указывается, то просмотреть сокеты, применяемые процессами сервера, нельзя. Например:

```
# netstat -a
Активные соединения Internet (включая соединения с серверами)
Proto Recv-Q Send-Q  Локальный адрес         Внешний адрес      (состояние)
```

```
tcp4       0      0  *.daytime              *.*                    LISTEN
tcp        0      0  *.ftp                  *.*                    LISTEN
tcp        0      0  *.telnet               *.*                    LISTEN
tcp4       0      0  *.time                 *.*                    LISTEN
tcp4       0      0  *.sunrpc               *.*                    LISTEN
tcp        0      0  *.exec                 *.*                    LISTEN
tcp        0      0  *.login                *.*                    LISTEN
tcp        0      0  *.shell                *.*                    LISTEN
tcp4       0      0  *.klogin               *.*                    LISTEN
tcp4       0      0  *.kshell               *.*                    LISTEN
tcp        0      0  *.netop                *.*                    LISTEN
tcp        0      0  *.netop64              *.*                    LISTEN
tcp4       0   1028  brown10.telnet         remote_client.mt.1254  ESTABLISHED
tcp4       0      0  *.wsmserve             *.*                    LISTEN
udp4       0      0  *.daytime              *.*
udp4       0      0  *.time                 *.*
udp4       0      0  *.sunrpc               *.*
udp4       0      0  *.ntalk                *.*
udp4       0      0  *.32780                *.*
Активные сокеты домена UNIX
SADR/PCB  Тип    Получ-Q Отпр-Q Узел      Соед.   Относит. След. отн. Адр.
71759200 dgram       0      0 13434d00        0        0        0 /dev/SRC
7051d580
71518a00 dgram       0      0 183c3b80        0        0        0 /dev/.SRC-unix/SRCCwfCEb
```

Просмотреть подробные сведения для каждого сокета можно с помощью команды **netstat -ao**. В приведенном ниже примере сокет **ftp** работает с адаптером Gigabit Ethernet, поддерживающим большие кадры:

```
# netstat -ao

Активные соединения Internet (включая соединения с серверами)
Proto Recv-Q Send-Q  Локальный адрес         Внешний адрес      (состояние)
[...]

tcp4       0      0  server1.ftp           client1.33122         ESTABLISHED

         so_options: (REUSEADDR|OOBINLINE)
         so_state: (ISCONNECTED|PRIV)
         timeo:0 uid:0
         so_special: (LOCKBALE|MEMCOMPRESS|DISABLE)
         so_special2: (PROC)
         sndbuf:
                 hiwat:134220 lowat:33555 mbcnt:0 mbmax:536880
         rcvbuf:
                 hiwat:134220 lowat:1 mbcnt:0 mbmax:536880
                 sb_flags: (WAIT)
         TCP:
         mss:8948  flags: (NODELAY|RFC1323|SENT_WS|RCVD_WS|SENT_TS|RCVD_TS)


tcp4       0      0  server1.telnet         sig-9-49-151-26..2387  ESTABLISHED

         so_options: (REUSEADDR|KEEPALIVE|OOBINLINE)
         so_state: (ISCONNECTED|NBIO)
         timeo:0 uid:0
         so_special: (NOUAREA|LOCKBALE|EXTPRIV|MEMCOMPRESS|DISABLE)
         so_special2: (PROC)
         sndbuf:
                 hiwat:16384 lowat:4125 mbcnt:0 mbmax:65536
                 sb_flags: (SEL|NOINTR)
         rcvbuf:
                 hiwat:66000 lowat:1 mbcnt:0 mbmax:264000
                 sb_flags: (SEL|NOINTR)
         TCP:
         mss:1375
```

```
tcp4       0    925  en6host1.login         en6host2.1023          ESTABLISHED

         so_options: (REUSEADDR|KEEPALIVE|OOBINLINE)
         so_state: (ISCONNECTED|NBIO)
         timeo:0 uid:0
         so_special: (NOUAREA|LOCKBALE|EXTPRIV|MEMCOMPRESS|DISABLE)
         so_special2: (PROC)
         sndbuf:
                 hiwat:16384 lowat:16384 mbcnt:3216 mbmax:65536
                 sb_flags: (SEL|NOINTR)
         rcvbuf:
                 hiwat:130320 lowat:1 mbcnt:0 mbmax:521280
                 sb_flags: (SEL|NOINTR)
         TCP:
         mss:1448  flags: (RFC1323|SENT_WS|RCVD_WS|SENT_TS|RCVD_TS)


tcp        0      0  *.login                *.*                    LISTEN

         so_options: (ACCEPTCONN|REUSEADDR)
         q0len:0 qlen:0 qlimit:1000      so_state: (PRIV)
         timeo:0 uid:0
         so_special: (LOCKBALE|MEMCOMPRESS|DISABLE)
         so_special2: (PROC)
         sndbuf:
                 hiwat:16384 lowat:4096 mbcnt:0 mbmax:65536
         rcvbuf:
                 hiwat:16384 lowat:1 mbcnt:0 mbmax:65536
                 sb_flags: (SEL)
         TCP:
         mss:512

tcp        0      0  *.shell                *.*                    LISTEN

         so_options: (ACCEPTCONN|REUSEADDR)
         q0len:0 qlen:0 qlimit:1000      so_state: (PRIV)
         timeo:0 uid:0
         so_special: (LOCKBALE|MEMCOMPRESS|DISABLE)
         so_special2: (PROC)
         sndbuf:
                 hiwat:16384 lowat:4096 mbcnt:0 mbmax:65536
         rcvbuf:
                 hiwat:16384 lowat:1 mbcnt:0 mbmax:65536
                 sb_flags: (SEL)
         TCP:
         mss:512


tcp4       0   6394  brown10.telnet         remote_client.mt.1254  ESTABLISHED

         so_options: (REUSEADDR|KEEPALIVE|OOBINLINE)
         so_state: (ISCONNECTED|NBIO)
         timeo:0 uid:0
         so_special: (NOUAREA|LOCKBALE|EXTPRIV|MEMCOMPRESS|DISABLE)
         so_special2: (PROC)
         sndbuf:
                 hiwat:16384 lowat:4125 mbcnt:65700 mbmax:65536
                 sb_flags: (SEL|NOINTR)
         rcvbuf:
                 hiwat:16500 lowat:1 mbcnt:0 mbmax:66000
                 sb_flags: (SEL|NOINTR)
         TCP:
         mss:1375


udp4       0      0  *.time                 *.*
```

```
         so_options: (REUSEADDR)
         so_state: (PRIV)
         timeo:0 uid:0
         so_special: (LOCKBALE|DISABLE)
         so_special2: (PROC)
         sndbuf:
                 hiwat:9216 lowat:4096 mbcnt:0 mbmax:36864
         rcvbuf:
                 hiwat:42080 lowat:1 mbcnt:0 mbmax:168320
                 sb_flags: (SEL)

[...]

Активные сокеты домена UNIX
SADR/PCB  Тип    Получ-Q Отпр-Q Узел      Соед.   Относит. След. отн. Адр.
71759200 dgram       0      0 13434d00        0        0        0 /dev/SRC
7051d580
         so_state: (PRIV)
         timeo:0 uid:0
         so_special: (LOCKBALE)
         so_special2: (PROC)
         sndbuf:
                 hiwat:8192 lowat:4096 mbcnt:0 mbmax:32768
         rcvbuf:
                 hiwat:45000 lowat:1 mbcnt:0 mbmax:180000
                 sb_flags: (SEL)

71518a00 dgram       0      0 183c3b80        0        0        0 /dev/.SRC-unix/SRCCwfCEb7051d400
         so_state: (PRIV)
         timeo:0 uid:0
         so_special: (LOCKBALE)
         so_special2: (PROC)
         sndbuf:
                 hiwat:16384 lowat:4096 mbcnt:0 mbmax:65536
         rcvbuf:
                 hiwat:8192 lowat:1 mbcnt:0 mbmax:32768
                 sb_flags: (SEL)

[...]
```

В приведенном выше примере адаптер поддерживает большие кадры, что приводит к высокому значению MSS и применению **rfc1323**.

Команда netstat -M:

Команда **netstat -M** позволяет просмотреть статистические данные пула кластеров сетевой памяти.

Ниже приведен пример вывода команды **netstat -M**:

```
# netstat -M
Статистика для пула кластеров:

  Размер кластера  Разм. пула  Вызовы    Сбои   Занято    Макс. итог
        131072           0       0       0       0             0
         65536           0       0       0       0             0
         32768           0       0       0       0             0
         16384           0       0       0       0             0
          8192           0  191292       3       0             3
          4096           0  196021       3       0             3
          2048           0  140660       4       0             2
          1024           0       2       1       0             1
           512           0       2       1       0             1
        131072           0       0       0       0             0
         65536           0       0       0       0             0
         32768           0       0       0       0             0
```

```
 16384           0       0        0       0              0
  8192           0  193948        2       0              2
  4096           0  191122        3       0              3
  2048           0  145477        4       0              2
  1024           0       0        0       0              0
   512           0       2        1       0              1
```

Команда netstat -v:

Команда **netstat -v** выдает статистическую информацию о работе всех драйверов устройств CDLI системы.

Информацию об отдельных интерфейсах можно получить с помощью команд **tokstat**, **entstat**, **fddistat** и **atmstat**.

Вывод команды содержит статистику для каждого интерфейса и общую информацию. В следующем примере показаны фрагменты вывода команды **netstat -v**, относящиеся к адаптерам Token-Ring и Ethernet; для других интерфейсов выводится аналогичная информация. Информация о различных адаптерах отличается незначительно. Наиболее важные поля вывода выделены.

```
# netstat -v
-------------------------------------------------------------
Статистика Ethernet (ent0) :
Тип устройства: Адаптер Ethernet PCI II 10/100 Мб/с (1410ff01)
Аппаратный адрес: 00:09:6b:3e:00:55
Прошедшее время: 0 дней 17 часов 38 минут 35 секунд

Статистика по передаче:                     Статистика по приему:
--------------------                        -------------------
Пакетов: 519                                Пакетов: 30161
Байт: 81415                                 Байт: 7947141
Прерываний: 2                               Прерываний: 29873
Ошибок при передаче: 0                      Ошибок при получении: 0
Отброшено пакетов: 0                        Отброшено пакетов: 0
                                            Пакетов с ошибками: 0
Максимальное число пакетов в программной очереди передачи: 3
Переполнение программной очереди передачи: 0
Текущая длина прогр. и аппар. очередей передачи: 1

Пакетов оповещения: 3                       Пакетов оповещения: 29544
Многоцелевых пакетов: 2                     Многоцелевых пакетов: 42
Несущая частота не обнаружена: 0            Ошибка CRC: 0
Выход за нижнюю границу DMA: 0              Выход за верхнюю границу DMA: 0
Ошибки потери CTS: 0                        Ошибки выравнивания: 0
Максимальное число конфликтов: 0            Ошибок отсутствия ресурсов: 0
Поздние конфликты: 0                        Конфликты при получении: 0
Задержки: 0                                 Слишком короткие пакеты: 0
Проверка SQE: 0                             Слишком длинные пакеты: 0
Ошибки ожидания: 0                          Пакеты, отброшенные адаптером: 0
Число отдельных конфликтов: 0               Начальное значение для получателя: 0
Число множественных конфликтов: 0
Текущая длина аппаратной очереди передачи: 1

Общая статистика:
-------------------
Ошибки отсутствия mbuf: 0
Число перезапусков адаптера: 0
Скорость передачи данных адаптером: 200
Флаги драйвера: Up Broadcast Running
        Simplex AlternateAddress 64BitSupport
        ChecksumOffload PrivateSegment DataRateSet

Статистика для адаптера Ethernet PCI Adapter II 10/100 Мбит/с (1410ff01):
-------------------------------------------------------------------
Состояние соединения: Включено
Выбранная скорость передачи данных: Автоматическое согласование
```

```
Текущая скорость передачи данных: 100 Мбит/с дуплекс
Размер буфера приема: 1024
Свободных буферов приема: 1024
Ошибок из-за отсутствия буферов приема: 0
Слишком маленьких буферов приема: 0
Записей передачи процедуры тайм-аута: 0
Отправленных пакетов IPsec: 0
Ошибок отправки пакетов IPsec: 0
Полученных пакетов IPsec: 0
Ошибок при получении пакетов IPsec: 0
Операций выгрузки входящих IPsec SA: 0
Отправленных больших пакетов: 0
Ошибок отправки больших пакетов: 0
Конфликтов при передаче пакетов:
 1 конфликт: 0             6 конфликтов: 0          11 конфликтов: 0
 2 конфликта: 0            7 конфликтов: 0          12 конфликтов: 0
 3 конфликта: 0            8 конфликтов: 0          13 конфликтов: 0
 4 конфликта: 0            9 конфликтов: 0          14 конфликтов: 0
 5 конфликтов: 0          10 конфликтов: 0          15 конфликтов: 0
-------------------------------------------------------------
Статистика Ethernet (ent0) :
Тип устройства: Адаптер 10/100/1000 Base-TX PCI-X (14106902)
Аппаратный адрес: 00:02:55:6a:a5:dc
Прошедшее время: 0 дней 17 часов 0 минут 26 секунд

Статистика по передаче:                    Статистика по приему:
-----------------------                    ---------------------
Пакетов: 15                                Пакетов: 14
Байт: 1037                                 Байт: 958
Прерываний: 0                              Прерываний: 13
Ошибок при передаче: 0                     Ошибок при получении: 0
Отброшено пакетов: 0                       Отброшено пакетов: 0
                                           Пакетов с ошибками: 0
Максимальное число пакетов в программной очереди передачи: 4
Переполнение программной очереди передачи: 0
Текущая длина прогр. и аппар. очередей передачи: 0

Пакетов оповещения: 1                      Пакетов оповещения: 0
Многоцелевых пакетов: 1                    Многоцелевых пакетов: 0
Несущая частота не обнаружена: 0           Ошибка CRC: 0
Выход за нижнюю границу DMA: 0             Выход за верхнюю границу DMA: 0
Ошибки потери CTS: 0                       Ошибки выравнивания: 0
Максимальное число конфликтов: 0           Ошибок отсутствия ресурсов: 0
Поздние конфликты: 0                       Конфликты при получении: 0
Задержки: 0                                Слишком короткие пакеты: 0
Проверка SQE: 0                            Слишком длинные пакеты: 0
Ошибки ожидания: 0                         Пакеты, отброшенные адаптером: 0
Число отдельных конфликтов: 0              Начальное значение для получателя: 0
Число множественных конфликтов: 0
Текущая длина аппаратной очереди передачи: 0

Общая статистика:
-------------------
Ошибки отсутствия mbuf: 0
Число перезапусков адаптера: 0
Скорость передачи данных адаптером: 2000
Флаги драйвера: Up Broadcast Running
        Simplex 64BitSupport ChecksumOffload
        PrivateSegment LargeSend DataRateSet

Статистика адаптера PCI-X 10/100/1000 Base-TX (14106902):
-------------------------------------------------------------------
Состояние соединения: Включено
Выбранная скорость передачи данных: Автоматическое согласование
Текущая скорость передачи данных: 1000 Мбит/с Дуплекс
Режим PCI: PCI-X (100-133)
Разрядность шины PCI: 64 разряда
```

```
Большие кадры: отключено
Выгрузка разбиения TCP: включено
        Передано пакетов выгрузки разбиения TCP: 0
        Ошибок пакетов выгрузки разбиения TCP: 0
Управление потоком передаваемых и получаемых данных: включено
        Передано пакетов управления потоком XON: 0
        Получено пакетов управления потоком XON: 0
        Передано пакетов управления потоком XOFF: 0
        Получено пакетов управления потоком XOFF: 0
Порог управления потоком получаемых и передаваемых данных (высокий): 32768
Порог управления потоком получаемых и передаваемых данных (низкий): 24576
Выделение памяти для полученных и переданных пакетов (TX/RX): 16/48
```

Ниже описаны поля, выделенные в выводе команды:

- **Ошибок передачи и приема**

  Число ошибок ввода-вывода для устройства. В этом поле указано число передач, прерванных из-за ошибок аппаратного обеспечения и сети.

  Такие неудачные попытки передачи замедляют работу системы.

- **Максимальное число пакетов в программной очереди передачи**

  Максимальное число пакетов, которое когда-либо находилось в программной очереди передачи.

  Если это значение равно текущему размеру очереди (*xmt_que_size*), то размер очереди следует увеличить. Такое значение говорит о том, что очередь переполнялась.

  Для проверки текущего размера очереди выполните команду **lsattr -El** *адаптер* (где адаптер - это, например, `ent0`). Поскольку очередь связана с драйвером устройства и адаптером интерфейса, необходимо указать имя адаптера, а не имя интерфейса. Размер очереди можно изменить с помощью SMIT или команды **chdev**.

- **Переполнений программной очереди передачи**

  Число переполнений программной очереди передачи. Если это значение отлично от нуля, выполните те же действия, что и случае, когда `Максимальное число пакетов в программной очереди передачи` равно *xmt_que_size*. Необходимо увеличить размер очереди передачи.

- **Пакетов оповещения**

  Число пакетов оповещения, полученных без ошибок.

  Сравните число пакетов оповещения с общим числом полученных пакетов. Это значение должно быть меньше 20 процентов от общего числа полученных пакетов. Если доля пакетов оповещения составляет больше 20 процентов, то сеть сильно загружена. В этом случае рекомендуется применять многоцелевую рассылку вместо оповещения. Многоцелевая рассылка позволяет отправить сообщение сразу группе хостов, а не всем членам группы по отдельности.

- **Переполнений DMA**

  Счетчик `переполнений DMA` увеличивается, когда операция передачи пакета с помощью DMA не была завершена. В системе были свободные буферы для размещения пакета, однако операция DMA не была выполнена. Это происходит, если шина MCA перегружена, и адаптеру не удается передать пакет с помощью DMA. В сильно загруженной системе наличие переполнений зависит от положения адаптера на шине. Обычно адаптер в самом нижнем разъеме шины, обладая большим приоритетом, занимает настолько большую часть полосы пропускания, что адаптеры в более высоких разъемах не могут получить доступ к шине. Обычно такая ситуация возникает, если в нижнем разъеме установлен адаптер ATM.

- **Макс. число конфликтов**

  Число неудачных передач, связанных со слишком большим числом конфликтов. Число обнаруженных конфликтов превысило число повторных передач, заданное для адаптера.

- **Поздних конфликтов**

  Число неудачных передач, связанных с конфликтом после начала передачи.

- **Тайм-аутов**

  Число неудачных передач, связанных с получением тайм-аута от адаптера.

- **Одиночных конфликтов**

  Число исходящих пакетов, при передаче которых был обнаружен только один конфликт.
- **Множественных конфликтов**

  Число исходящих пакетов, при передаче которых было обнаружено несколько (от 2 до 15) конфликтов.
- **Конфликтов при получении**

  Число полученных пакетов, при приеме которых произошел конфликт.
- **Ошибок из-за отсутствия mbuf**

  Число неудачных запросов на получение буферов от драйвера устройства. Обычно такие запросы отправляются при приеме пакетов. Если в системе не удастся выделить пул буферов запрошенного размера, то пакет будет отброшен. Убедитесь, что пакет был отброшен, с помощью команды **netstat -m**, а затем увеличьте значение параметра *thewall*.

  Число `ошибок из-за отсутствия mbuf` зависит от типа интерфейса и не совпадает с числом `отклоненных запросов на выделение mbuf`, которое указывается в выводе команды **netstat -m**. Сравните вывод команд **netstat -m** и **netstat -v** (отчеты об адаптерах Ethernet и Token-Ring).

Если число `ошибок` в выводе команды **netstat -v** отлично от нуля, то требуется настроить параметры сети.

Дополнительные рекомендации:

- Для того чтобы узнать, перегружена ли сеть Ethernet, вычислите следующее значение (с помощью отчета команды **netstat -v**):

  `(Макс. число конфликтов + Тайм-аутов) / Переданных пакетов`

  Если полученный результат превышает 5 процентов, реорганизуйте сеть для более равномерного распределения нагрузки.
- Кроме того, сеть перегружена, если выполнено следующее условие (значения взяты из вывода команды **netstat -v**):

  Если общее число конфликтов в выводе команды **netstat -v** (для Ethernet) превышает 10 процентов от полного числа переданных пакетов, то есть:

  `Число конфликтов / Число переданных пакетов > 0,1`

Команда netstat -p протокол:

Команда **netstat -p** протокол выводит статистику по указанному протоколу (`udp`, `tcp`, `sctp`,`ip`, `icmp`), который может быть задан либо по имени, либо по псевдониму.

Имена и псевдонимы некоторых протоколов указаны в файле `/etc/protocols`. Пустой отчет означает, что информация об указанном протоколе отсутствует. Если для указанного протокола не установлена функция сбора статистики, то выдается сообщение о том, что задан неизвестный протокол.

Ниже приведен пример вывода команды для протокола `ip`:

```
# netstat -p ip
ip:
        45775 пакетов получено
        0 с ошибками контрольной суммы заголовка
        0 с размером меньше минимального
        0 с размером данных < длины данных
        0 с длиной заголовка < размера данных
        0 с длиной данных < длины заголовка
        0 с неправильными параметрами
        0 с неправильным номером версии
        0 фрагментов получено
        0 фрагментов отброшено (повтор или нехватка памяти)
        0 фрагментов отброшено после истечения тайм-аута
        0 пакетов успешно пересобрано
        45721 пакет для данного хоста
        51 пакет неизвестных/не поддерживаемых протоколов
```

```
0 пакетов переслано
4 пакета не удалось переслать
0 перенаправлений
33877 пакетов отправлено с данного узла
0 пакетов отправлено с искусственным заголовком ip
0 исходящих пакетов отброшено из-за отсутствия буферов и пр.
0 исходящих пакетов отброшено из-за отсутствия маршрута
0 дейтаграмм вывода фрагментировано
0 фрагментов создано
0 дейтаграмм не удалось разбить на фрагменты
0 многоцелевых пакетов IP отброшено из-за отсутствия получателя
0 успешных циклов поиска путей MTU
1 повторная попытка цикла поиска путей MTU
3 оценки поиска путей MTU без ответа
3 оценки поиска путей MTU с ответом
1 снижение количества операций поиска путей MTU
8 пакетов поиска путей MTU отправлено
0 сбоев выделения путей поиска путей MTU
0 переполнений ipintrq
0 с неверным источником
0 пакетов обработано нитями
0 пакетов отброшено нитями
0 пакетов отброшено по причине переполнения буфера входящих пакетов сокета
0 пакетов обнаружения сбоев в работе шлюза отправлено
0 ошибок выделения пакетов функции обнаружения сбоев в работе шлюза
0 ошибок выделения шлюзов функции обнаружения сбоев в работе шлюза
```

Ниже описаны поля, выделенные в выводе команды:

- **Пакетов получено**

  Общее число полученных дейтаграмм IP.

- **Неправильных контрольных сумм заголовка или Фрагментов отброшено**

  Если значение в поле неправильных контрольных сумм заголовка или фрагментов отброшено из-за повтора или нехватки памяти отлично от нуля, то либо пакеты повреждаются при передаче по сети, либо размер очереди приема драйвера устройства недостаточен.

- **Фрагментов получено**

  Общее число полученных фрагментов.

- **Отброшено из-за тайм-аута**

  Ненулевое число фрагментов, отброшенных из-за тайм-аута, означает, что из-за высокой загруженности сети счетчик TTL фрагментов ip достигает нулевого значения до того, как будут получены все фрагменты дейтаграммы. Для того чтобы исправить эту ошибку, увеличьте значение параметра *ipfragttl* с помощью команды **no**. Другая возможная причина - отсутствие свободных буферов. В этом случае нужно увеличить значение *thewall*.

- **Пакетов отправлено этим хостом**

  Число дейтаграмм IP, созданных и отправленных из локальной системы. Этот счетчик не учитывает пересланные (транзитные) дейтаграммы.

- **Фрагментов создано**

  Число фрагментов, созданных в локальной системе при отправке дейтаграмм.

При просмотре статистики протокола IP обратите внимание на отношение числа полученных пакетов к числу полученных фрагментов. В сетях с небольшим значением MTU должно фрагментироваться не более 10 процентов пакетов. Большое число фрагментов означает, что протоколы более высокого уровня на удаленных хостах передают данные блоками, превышающими размер MTU интерфейса. Кроме того, пакеты могут фрагментироваться маршрутизаторами и шлюзами. То же самое справедливо для отношения числа отправленных пакетов к числу созданных фрагментов.

Фрагментация увеличивает нагрузку на процессор, поэтому важно определить ее причину. Учтите, что фрагментация неизбежна при работе некоторых приложений. Например, фрагментация происходит, если

приложение отправляет данные небольшими порциями. Если же фрагментация выполняется несмотря на то, что приложение отправляет данные большими блоками, определите причину фрагментации. Возможно, указанный размер MTU интерфейса не совпадает с размером MTU удаленных систем.

Ниже приведен пример вывода команды для протокола udp:

```
# netstat -p udp
udp:
        11623 дейтаграмм получено
        0 неполных заголовков
        0 полей неправильной длины
        0 неправильных контрольных сумм
        620 отброшено из-за отсутствия сокета
        232850 дейтаграмм оповещения и многоцелевых дейтаграмм отброшено из-за отсутствия сокета
        0 переполненных буферов сокетов
        14 дейтаграмм доставлено
        12 дейтаграмм отправлено
```

Ниже описаны поля, выделенные в выводе команды:

- **Неправильных контрольных сумм**

  Неправильные контрольные суммы появляются в результате сбоя адаптера или повреждения кабеля.

- **Отброшено из-за отсутствия сокета**

  Число полученных дейтаграмм UDP, целевой порт которых не был открыт. В этом случае возвращается сообщение Получатель ICMP недоступен - Порт недоступен. Для дейтаграмм оповещения UDP сообщения об ошибках ICMP не отправляются. Если число отброшенных дейтаграмм велико, то, скорее всего, возникают ошибки при работе с сокетами в приложении.

- **Переполнений буфера сокета**

  Переполнения буфера сокета связаны с недостаточным количеством сокетов приема и передачи UDP, демонов **nfsd** или недостаточной величиной параметров *nfs_socketsize*, *udp_recvspace* и *sb_max*.

Если в выводе команды **netstat -p udp** число переполнений буфера сокета отлично от нуля, рекомендуется увеличить число демонов **nfsd** на сервере. Сначала проверьте, не перегружен ли процессор или подсистема ввода-вывода системы, а затем вызовите команду **no -a** и убедитесь, что для параметров протоколов других уровней установлены рекомендуемые значения. Если система перегружена, необходимо уменьшить нагрузку или увеличить объем ресурсов системы.

Ниже приведен пример вывода команды для протокола tcp:

```
 # netstat -p tcp
tcp:
        576 пакетов отправлено
                512 пакетов данных (62323 байта)
                0 пакетов данных (0 байт) передано повторно
                55 пакетов уведомления (28 отложено)
                0 срочных пакетов
                0 пакетов проверки окна
                0 пакетов обновления окна
                9 управляющих пакетов
                0 операций largesend
                0 байт отправлено с помощью largesend
                0 байт в максимальном пакете largesend
        719 пакетов получено
                504 пакетов уведомления (для 62334 байт)
                19 повторных уведомлений
                0 уведомлений о неотправленных данных
                449 пакетов (4291 байт) получено в правильном порядке
                8 полных копий пакетов (8 байт)
                0 старых копий пакетов
                0 пакетов с копиями данных (0 байт копий)
                5 пакетов вне очереди (0 байт)
                0 пакетов (0 байт) данных после окна
                0 проверок окна
```

```
        2 пакетов обновления окна
        0 пакетов получено после закрытия
        0 пакетов с неправильной аппаратной контрольной суммой
        0 отброшено из-за неправильной контрольной суммы
        0 отброшено из-за неправильных полей заголовка
        0 отброшено из-за слишком малой длины
        0 отброшено получателями
        0 отброшено из-за переполнения очередей получателей
        71 правильных прогнозов для заголовков пакетов уведомления
        71 правильных прогнозов для заголовков пакетов данных
6 запросов на установление соединения
8 соединений разрешено установить
14 соединений установлено (включая разрешенные)
6 соединений закрыто (включая 0 отброшенных)
0 соединений с поддержкой ECN
0 ответов на ECN
0 начальных соединений отброшено
504 сегментов обновлено (505 попыток)
0 сегментов с уменьшенным числом разрядов окна нагрузки
0 сегментов с проверенным при нагрузке набором разрядов
0 повторных передач, связанных с вычислением MTU маршрута
0 операций поиска путей MTU прервано из-за повторной передачи
0 тайм-аутов повторной передачи
        0 соединений отброшено из-за тайм-аута rexmit
0 быстрых операций повторной передачи
        0 с окном загрузки, не превышающим 3 сегмента
0 новых операций повторной передачи
0 ошибочных операций быстрой повторной передачи предотвращено
0 постоянных тайм-аутов
        0 соединений отброшено из-за постоянных тайм-аутов
16 тайм-аутов срока действия
        16 тестов срока действия отправлено
        0 соединений отброшено из-за срока действия
0 операций расширения массивов блоков SACK
0 операций расширения массивов промежутков SACK
0 пакетов отброшено из-за сбоев выделения памяти
0 соединений использовано повторно
0 отложенных уведомлений для SYN
0 отложенных уведомлений для FIN
0 операций отправки с отключением
0 сложных соединений
0 сложных соединений закрыто
0 сложных соединений сброшено
0 тайм-аутов сложных соединений
0 постоянных тайм-аутов сложных соединений
0 тайм-аутов срока действия сложных соединений
```

Ниже описаны поля, выделенные в выводе команды:

- Пакетов отправлено
- Пакетов данных
- Пакетов данных отправлено повторно
- Пакетов получено
- Повторных пакетов
- Тайм-аутов передачи

Сравните число отправленных пакетов с числом повторно отправленных пакетов. Если число повторно отправленных пакетов превышает 10-15 процентов от общего числа пакетов, возникают тайм-ауты TCP. Скорее всего, они связаны с тем, что сеть слишком загружена, поэтому уведомление не удается передать до наступления тайм-аута. Кроме того, повторные передачи пакетов TCP могут быть связаны с высокой загруженностью системы-получателя, либо с общей низкой производительностью сети. Повторные передачи увеличивают поток данных и приводят к дальнейшему повышению нагрузки на сеть.

Кроме того, сравните число полученных пакетов с числом повторных пакетов. Если после отправки пакета возникает тайм-аут ожидания подтверждения, пакет передается повторно. При этом целевая система может получить все переданные копии пакета. Если число повторных пакетов превышает 10-15 процентов, возможно, сеть или система-получатель перегружена. Повторные передачи пакетов также увеличивают поток данных в сети.

Тайм-аут повторной передачи возникает в том случае, если узел отправил пакет, но не получил подтверждение за заданное время. При этом пакет передается еще раз. Это значение увеличивается для каждой повторной передачи. Повторные передачи увеличивают нагрузку на процессор. Если подтверждение так и не будет получено, пакет будет отброшен.

netstat -s:

Команда **netstat -s** выдает статистическую информацию о всех протоколах (в то время как команда **netstat -p** выдает статистическую информацию только об указанном протоколе).

Команда **netstat -s** выдает статистику только для следующих протоколов:
- TCP
- UDP
- SCTP
- IP
- IPv6
- IGMP
- ICMP
- ICMPv6

netstat -s -s:

Недокументированная опция **-s -s** показывает только ненулевые значения из вывода команды **netstat -s**, упрощая анализ ошибок.

netstat -s -Z:

Команда **netstat** обнуляет все статистические счетчики команды **netstat -s**.

Команда netstat -r:

Другая опция, связанная с производительностью - это вычисленный MTU маршрута (PMTU). MTU маршрута можно просмотреть с помощью команды **netstat -r**.

Если соединение между хостами проходит через несколько сетей, передаваемый пакет будет фрагментирован, если его размер превышает значение MTU хотя бы одной из этих сетей. Поскольку фрагментация пакета уменьшает производительность сети, рекомендуется передавать пакеты меньшего размера, чем минимальный MTU маршрута. Максимальный размер пакета, который не будет фрагментирован, называется *MTU маршрута*.

Ниже приведен пример вывода команды **netstat -r -f inet**, сообщающей только таблицы маршрутизации:

```
# netstat -r -f inet
Таблицы маршрутизации
Пункт назн.       Шлюз          Флаги     Ссылк.  Исп.   Инт.   PMTU Exp Groups

Дерево маршрутов для группы протоколов 2 (Internet):
default          res101141         UGc       0        0  en1     -   -
ausdns01.srv.ibm  res101141        UGHW      1        4  en1  1500   -
10.1.14.0         server1          UHSb      0        0  en1     -   -  =>
```

```
10.1.14/24        server1          U         3      112  en1     -   -
brown17          loopback          UGHS      6      110  lo0     -   -
10.1.14.255      server1           UHSb      0        0  en1     -   -
magenta          res1031041        UGHW      1       42  en1     -   -
127/8            loopback          U         6    16633  lo0     -   -
192.1.6.0        en6host1          UHSb      0        0  en0     -   -  =>
192.1.6/24       en6host1          U         0       17  en0     -   -
en6host1         loopback          UGHS      0    16600  lo0     -   -
192.1.6.255      en6host1          UHSb      0        0  en0     -   -
192.6.0.0        fc0host1          UHSb      0        0  fc0     -   -  =>
192.6.0/24       fc0host1          U         0       20  fc0     -   -
fc0host1         loopback          UGHS      0        0  lo0     -   -
192.6.0.255      fc0host1          UHSb      0        0  fc0     -   -
```

Команда netstat -D:

Опция **-D** позволяет узнать число пакетов, полученных и отправленных каждым из уровней подсистемы связи, а также число пакетов, отброшенных на каждом из уровней.

```
# netstat -D

Источник                       Ipkts               Opkts     Idrops     Odrops
-------------------------------------------------------------------------------
ent_dev1                       32556                 727          0          0
ent_dev2                           0                   1          0          0
ent_dev3                           0                   1          0          0
fcnet_dev0                        24                  22          0          0
fcnet_dev1                         0                   0          0          0
ent_dev0                          14                  15          0          0
             ---------------------------------------------------------------
Всего для устройств            32594                 766          0          0
-------------------------------------------------------------------------------
ent_dd1                        32556                 727          0          0
ent_dd2                            0                   2          0          1
ent_dd3                            0                   2          0          1
fcnet_dd0                         24                  22          0          0
fcnet_dd1                          0                   0          0          0
ent_dd0                           14                  15          0          0
             ---------------------------------------------------------------
Всего для драйверов            32594                 768          0          2
-------------------------------------------------------------------------------
fcs_dmx0                           0                 N/A          0        N/A
fcs_dmx1                           0                 N/A          0        N/A
ent_dmx1                       31421                 N/A       1149        N/A
ent_dmx2                           0                 N/A          0        N/A
ent_dmx3                           0                 N/A          0        N/A
fcnet_dmx0                         0                 N/A          0        N/A
fcnet_dmx1                         0                 N/A          0        N/A
ent_dmx0                          14                 N/A          0        N/A
             ---------------------------------------------------------------
Всего для демультиплексоров    31435                 N/A       1149        N/A
-------------------------------------------------------------------------------
IP                             46815               34058         64          8
IPv6                               0                   0          0          0
TCP                              862                 710          9          0
UDP                            12412                  13      12396          0
             ---------------------------------------------------------------
Всего для протоколов           60089               34781      12469          8
-------------------------------------------------------------------------------
en_if1                         31421                 732          0          0
fc_if0                            24                  22          0          0
en_if0                            14                  20          0          6
lo_if0                         33341               33345          4          0
             ---------------------------------------------------------------
Всего для интерфейсов          64800               34119          4          6
-------------------------------------------------------------------------------
(Примечание:  N/A соответствует отсутствию возможных значений)
```

На уровне устройств указано число пакетов, полученных и отправленных адаптером, а также число пакетов, отброшенных при приеме и передаче. Ошибки адаптера могут быть связаны с различными причинами. Для получения более подробной информации проанализируйте вывод команды **netstat -v**.

На уровне драйверов указано число пакетов, полученных драйвером устройства для каждого адаптера. Для выяснения причин ошибок вызовите команду **netstat -v**.

Для демультиплексоров указывается число пакетов, обработанных на этом уровне. Ненулевое значение в столбце `Idrops` обычно указывает на то, что некоторые пакеты были отброшены при фильтрации (например, в рассматриваемой системе не поддерживаются пакеты Netware и DecNet, поэтому они отбрасываются).

Более подробную информацию о параметрах протоколов можно получить с помощью команды **netstat -s**.

Примечание: В выводе команды значение `N/A` указывает, что соответствующий параметр не измеряется на данном уровне. В случае статистики NFS/RPC, все полученные пакеты RPC проходят через NFS, поэтому данные числа не складываются, а в поле `Всего для NFS/RPC` указано `N/A`. NFS не подсчитывает число отправленных пакетов и пакетов, отброшенных при отправке, отдельно для NFS и RPC. Поэтому в соответствующих полях указано значение `N/A`, а в поле `Всего для NFS/RPC` задано суммарное значение.

Команда netpmon

Команда **netpmon** выводит подробную информацию о работе сети в течение заданного интервала времени, собранную функцией трассировки. Поскольку команда **netpmon** вызывает функцию трассировки, ее разрешено запускать только пользователю root и членам группы system.

Команду **netpmon** нельзя запускать с другими командами сбора данных, использующими функцию трассировки, например, **tprof** и **filemon**. Как правило, команда **netpmon** выполняется в фоновом режиме, отслеживая работу параллельно выполняющихся приложений и системных команд.

Команда **netpmon** собирает следующую информацию о работе системы:

- Использование процессора
  - Процессами и обработчиками прерываний
  - Подсистемой связи
  - Что служит причиной простоя
- Операции ввода-вывода драйвера сетевого устройства
  - Отслеживает операции ввода-вывода, выполняемые с помощью драйверов сетевых адаптеров Ethernet, Token-Ring и FDDI.
  - Для операций передачи данных команда собирает информацию об использовании процессора, длине очереди и целевых хостах. Для операций приема данных команда дополнительно отслеживает время, затраченное на сборку пакетов из фрагментов.
- Обращения к сокетам Internet
  - Собирает информацию об обращении к сокетам Internet через функции **send()**, **recv()**, **sendto()**, **recvfrom()**, **sendmsg()**, **read()** и **write()** .
  - Создает для каждого процесса отчет с информацией о применении протоколов ICMP, TCP и UDP.
- Операции ввода-вывода NFS
  - На клиенте: запросы RPC и запросы NFS на чтение-запись.
  - На сервере: информация о запросах отдельных клиентов, запросах к отдельным файлам и запросах на чтение-запись.

Вычисляются следующие значения:

- Время ответа при выполнении операций приема и передачи данных на уровне драйвера устройства и объем этих данных.

- Время ответа при выполнении запросов на чтение и запись в сокет Internet и объем памяти, выделенный для обработки таких запросов.
- Время ответа при выполнении операций чтения и записи данных NFS и объем этих данных.
- Время ответа при обработке запросов на вызов удаленных процедур NFS.

Для того чтобы узнать, установлена ли в системе команда **netpmon**, вызовите следующую команду:

```
# lslpp -lI perfagent.tools
```

Трассировка запускается командой **netpmon**, приостанавливается подкомандой **trcoff**, возобновляется подкомандой **trcon**. После выключения трассировки команда **netpmon** записывает свой отчет в стандартный вывод.

Применение команды netpmon:

Если не указана опция **-d**, команда **netpmon** сразу же включает функцию трассировки.

Для того чтобы выключить трассировку, вызовите команду **trcstop**. После выключения трассировки команда **netpmon** создаст все перечисленные отчеты и завершит свою работу. В среде клиент-сервер команда **netpmon** позволяет узнать, каким образом передача данных по сети влияет на общую производительность системы. Ее можно запустить как на клиенте, так и на сервере.

Вместо выполнения трассировки команда **netpmon** может считывать данные трассировки ввода-вывода из указанного файла. В этом случае отчет команды **netpmon** будет содержать информацию об операциях обмена данными по сети, зафиксированных в файле. Автономная обработка файла трассировки может выполняться в другой системе. Кроме того, она позволяет разделить сбор данных и их обработку.

Вначале файл трассировки должен быть обработан командой **trcrpt -r**. Вывод команды должен быть перенаправлен в другой файл:

```
# gennames > gennames.out
# trcrpt -r trace.out > trace.rpt
```

Преобразованный файл трассировки можно передать команде **netpmon** для создания отчета об операциях ввода-вывода, зарегистрированных в ходе трассировки:

```
# netpmon -i trace.rpt -n gennames.out | pg
```

В данном примере команда **netpmon** считывает информацию, собранную функцией трассировки, из файла ввода `trace.rpt`. Поскольку данные трассировки уже находятся в файле, команда **netpmon** не переходит в фоновый режим. После того как будут считаны все данные из файла, команда создает отчеты о работе сети и записывает их в стандартный вывод (в данном примере - передает их на вход команде **pg**).

Если команда **trace** была запущена с флагом **-C all**, команду **trcrpt** также следует запустить с флагом **-C all** (см. “Создание отчета на основе вывода команды trace -C” на стр. 398).

Ниже приведен пример команды **netpmon**, запущенной на сервере NFS, которая вызывает команду **sleep** и создает отчет через 400 секунд. В течение работы команды в системе выполняется копирование файла в смонтированную файловую систему NFS `/nfs_mnt`.

```
# netpmon -o netpmon.out -O all; sleep 400; trcstop
```

С опцией **-O** можно задать тип отчета, который должен быть создан. Существуют следующие типы отчетов:

cpu Использование процессора

dd Операции ввода-вывода драйвера сетевого устройства

so Чтение и запись данных через сокет Internet

nfs Операции ввода-вывода NFS

all Все виды отчетов. Значение по умолчанию показано ниже.

```
# cat netpmon.out

Пят, 5 марта   15:41:52 2004
Система: AIX crusade Узел: 5 Компьютер: 000353534C00


========================================================================

Статистика использования CPU процессами:
-----------------------------
                                                    Сеть
Процессы (20 первых)         PID  Время CPU  CPU %   CPU %
------------------------------------------------------------
netpmon                    45600    0.6995   1.023   0.000
nfsd                       50090    0.5743   0.840   0.840
UNKNOWN                    56912    0.1274   0.186   0.000
trcstop                    28716    0.0048   0.007   0.000
gil                         3870    0.0027   0.004   0.004
ksh                        42186    0.0024   0.003   0.000
IBM.ServiceRMd             14966    0.0021   0.003   0.000
IBM.ERrmd                   6610    0.0020   0.003   0.000
IBM.CSMAgentRMd            15222    0.0020   0.003   0.000
IBM.AuditRMd               12276    0.0020   0.003   0.000
syncd                       4766    0.0020   0.003   0.000
sleep                      28714    0.0017   0.002   0.000
swapper                        0    0.0012   0.002   0.000
rpc.lockd                  34942    0.0007   0.001   0.000
netpmon                    28712    0.0006   0.001   0.000
trace                      54622    0.0005   0.001   0.000
reaper                      2580    0.0003   0.000   0.000
netm                        3612    0.0002   0.000   0.000
aixmibd                     4868    0.0001   0.000   0.000
xmgc                        3354    0.0001   0.000   0.000
------------------------------------------------------------
Всего (все процессы)               1.4267   2.087   0.844
Время простоя                      55.4400  81.108

========================================================================

Статистика использования CPU обработчиком прерываний первого уровня:
---------------------------------------------------
                                                    Сеть
FLIH                              Время CPU  CPU %   CPU %
------------------------------------------------------------
внешнее устройство                  0.3821   0.559   0.559
уменьшение PPC                      0.0482   0.070   0.000
страничные ошибки (данные)          0.0137   0.020   0.000
прерывания очереди                  0.0002   0.000   0.000
------------------------------------------------------------
Всего (все обработчики)             0.4441   0.650   0.559

========================================================================

Статистика использования CPU обработчиком прерываний второго уровня:
---------------------------------------------------
                                                    Сеть
SLIH                              Время CPU  CPU %   CPU %
------------------------------------------------------------
phxentdd32                          2.4740   3.619   3.619
------------------------------------------------------------
Всего (все обработчики)             2.4740   3.619   3.619

========================================================================

Статистика драйверов сетевых устройств (по устройствам):
---------------------------------------------
```

```
                      --------- Передано ---------   ------ Получено -------
Устройство            пакетов/с  байт/с  Исп. Дл.очер. пакетов/с байт/с  интерф.
------------------------------------------------------------------------------
ethernet 4             7237.33 10957295  0.0%27.303  3862.63   282624  0.2324

==============================================================================

Статистика передачи для драйверов сетевых устройств (по целевым хостам):
------------------------------------------------------------

Хост                  пакетов/с  байт/с
---------------------------------------
client_machine          7237.33 10957295

==============================================================================

Статистика обращений клиентов к серверу NFS:
-----------------------------------
                       ----- Чтение -----  ----- Запись -----    Прочих
Клиент                 вызовов/с   байт/с  вызовов/с   байт/с  вызовов/с
------------------------------------------------------------------------
client_machine      0.00         0      0.00         0    321.15
------------------------------------------------------------------------
Всего (все клиенты) 0.00         0      0.00         0    321.15

==============================================================================

Подробная статистика использования CPU обработчиком прерываний второго уровня:
------------------------------------------------------------

SLIH: phxentdd32
число:                 33256
  время CPU (мс):      avg 0.074   min 0.018   max 288.374 sdev 1.581

Общее (Все SLIH)
число:                 33256
  время CPU (мс):      avg 0.074   min 0.018   max 288.374 sdev 1.581

==============================================================================

Подробная статистика для драйверов сетевых устройств:
-----------------------------------------

Устройство: ethernet 4
получено пакетов:       33003
  прочитано  (байт):    сред 73.2   мин 60      макс 618    откл 43.8
  время записи (мс):    сред 0.000  мин 0.000   макс 0.005  откл 0.000
  время выполнения (мс):сред 0.060  мин 0.004   макс 288.36 откл 1.587
передано пакетов:           61837
  отправлено (байт):    сред 1514.0 мин 1349    макс 1514    откл 0.7
  время отправки (мс):  сред 3.773  мин 2.026   макс 293.112 откл 8.947

==============================================================================

Подробная статистика передачи для драйверов сетевых устройств (по целевым хостам):
------------------------------------------------------------

Хост: client_machine (10.4.104.159)
передано пакетов:           61837
  отправлено (байт):    сред 1514.0 мин 1349    макс 1514    откл 0.7
  время отправки (мс):  сред 3.773  мин 2.026   макс 293.112 откл 8.947

==============================================================================

Подробная статистика обращений клиентов к серверу NFS:
-----------------------------------------
```

```
Клиент: client_machine
прочие вызовы:            2744
  время обработки прочих вызовов (мс):   сред 0.192  мин 0.075   макс 0.311  откл 0.025

Общее (все клиенты)
прочие вызовы:            2744
  время обработки прочих вызовов (мс):   сред 0.192  мин 0.075   макс 0.311  откл 0.025
```

Вывод команды **netpmon** состоит из отчетов двух типов: *общих* и *подробных*. Общие отчеты содержат следующую статистическую информацию:

- Наиболее активные процессы
- Обработчики прерывания первого уровня
- Обработчики прерывания второго уровня
- Драйверы сетевых устройств
- Операции передачи данных через драйверы сетевых устройств
- Обращения к сокетам TCP
- Статистика по клиенту или серверу NFS

Общие отчеты расположены в начале вывода команды **netpmon**. Они содержат информацию, собранную за указанный интервал времени. Подробные отчеты содержат дополнительную информацию об объектах, статистика для которых приведена в общих отчетах. По умолчанию в отчетах выводится 20 максимальных значений измеряемых показателей. Строки отчета расположены в порядке уменьшения значений показателей.

Общие отчеты команды netpmon:

Отчеты команды **netpmon** содержат заголовок, в котором указывается дата, имя компьютера и интервал сбора данных в секундах.

После заголовка следует список общих и подробных отчетов различных типов.

Статистика использования процессоров:

Каждая строка отчета содержит информацию об использовании процессора указанным процессом.

Если опция подробного вывода (**-v**) не задана, то список содержит 20 процессов, на выполнение которых затрачивается больше всего процессорного времени. В нижней области отчета указано общее время процессора, затраченное на выполнение всех процессов, а также время простоя процессора. Доля времени, в течение которого процессор простаивает, вычисляется путем деления времени простоя процессора на интервал сбора данных. Сумма времени простоя и общего времени, затраченного на выполнение процессов, не совпадает с интервалом сбора данных, так как какая-та часть времени была затрачена на обработку прерываний.

Значение `Сеть (CPU %)` указывает, какая доля процессорного времени (в процентах) бала затрачена на выполнение сетевых функций.

Если указан флаг **-t**, то отчет содержит информацию об использовании процессора различными нитями. В этом случае после строки с информацией о процессе следует несколько строк с информацией о том, какое время было затрачено на выполнение отдельных нитей процесса. Для нитей указываются те же значения, что и для процессов, за исключением имени. Нитям имена не присваиваются.

В приведенном примере общего отчета об использовании процессора доля `времени простоя` (81,104) получена путем деления общего `времени простоя` (55,4400) на продолжительность `интервала измерения`, умноженную на 8 (8,54 с*8), поскольку на сервере работает восемь процессоров. Для получения информации о работе отдельных процессоров вызовите команду **sar**, **ps** или другую команду, предназначенную для

многопроцессорной системы. Аналогичным образом вычисляется общее значение CPU %. Время простоя отлично от нуля, так как выполнялись операции передачи данных по сети. Сумма итоговых значений времени CPU (55,4400 + 1,4267) не совпадает с интервалом измерения, так как в системе есть несколько процессоров, и часть времени была затрачена на обработку прерываний. В примере отчета основная часть процессорного времени была затрачена на выполнение сетевых операций: (0,844 / 2,087) = 40,44 процента.

Примечание: Если в отчете Статистика использования CPU процессами доля времени, затраченная на выполнение сетевых операций (Сеть - CPU%), поделенная на долю времени, затраченную на выполнение всех операций (Всего CPU %), больше 0,5, то основная часть процессорного времени затрачивается на выполнение сетевых функций.

Кроме того, этот отчет позволяет узнать долю времени, затраченную на выполнение отдельных процессов, не передавая вывод на обработку специальной программе.

Статистика обработчика прерываний процессора первого уровня:

В отчете указана информация об использовании процессора обработчиками прерываний первого уровня (FLIH).

В нижней области отчета указано общее процессорное время, затраченное на выполнение всех FLIH.

Процессорное время
Общее процессорное время, затраченное данным FLIH

CPU %
Доля процессорного времени, затраченная на выполнение данного обработчика прерываний.

Сеть - CPU %
Доля процессорного времени, в течение которого обрабатывались прерывания, связанные с работой сети

Статистика использования microprocessor обработчиком прерываний второго уровня:

В отчете указана информация об использовании процессора обработчиками прерываний второго уровня (SLIH). В нижней области отчета указано общее процессорное время, затраченное на выполнение всех SLIH.

Статистика драйверов сетевых устройств по устройствам:

Команда **netpmon** создаёт отчёты о работе сетевых устройств на уровне драйверов.

Каждая строка отчета содержит информацию о работе сетевого устройства.

Device Имя файла, связанного с устройством.

Передано пакетов/с
Число пакетов, передаваемых каждую секунду через данное устройство.

Передано байт/с
Объем данных в байтах, передаваемых каждую секунду через данное устройство.

Передано - Исп.
Доля времени, в течение которого устройство было занято

Передано - Дл. очер.
Среднее число запросов, находящихся в очереди передачи устройства (с учетом запроса, выполняемого в данный момент)

Получено пакетов/с
Число пакетов, получаемых каждую секунду через данное устройство.

Получено байт/с
Объем данных в байтах, получаемых каждую секунду через данное устройство

Получено - Интерф.
Доля времени, затраченная на выполнение операций на уровне интерфейса.

В данном примере параметр `Дл. очер.` равен 27,303. Значение `Получено/с` равно 10957295 (10,5 Мб/с), что достаточно близко к предельному значению для Ethernet 100 Мбит/с. Следовательно, этот отчет показывает, что сеть загружена сильно.

Статистика передачи для драйверов сетевых устройств по целевым хостам:

Команда **netpmon** создаёт отчёты о передаче данных на уровне драйвера устройства по каждому целевому хосту отдельно.

В каждой строке отчета указано, какой объем данных был передан указанному целевому хосту на уровне драйвера устройства.

Host Имя целевого хоста. Звездочка (*) указывается в тех случаях, когда имя целевого хоста неизвестно.

Pkts/s Число пакетов, передаваемых хосту каждую секунду.

Bytes/s Объем данных в байтах, передаваемых хосту каждую секунду.

Статистики вызова сокета TCP для каждого IP по процессам:

Данная статистическая информация выводится для всех протоколов Internet.

В каждой строке отчета указано, сколько раз процесс обращался к сокету того или иного протокола через функции **read()** и **write()**. В нижней области отчета указано общее число обращений к сокету указанного протокола.

Статистика обращений клиентов к серверу NFS:

В каждой строке отчета задано число операций, выполненных сервером NFS по запросу указанного клиента. В нижней области отчета указано общее число обращений к серверу.

Если команда выполняется на клиенте, то статистика обращений к серверу NFS заменяется на статистику работы клиента с различными серверами (`Статистика обращений клиента к серверам NFS (по файлам)`, `Статистика применения RPC на клиенте NFS (по серверам)`, `Статистика работы клиента NFS (по процессам)`).

Подробные отчеты netpmon:

Команда создает подробные отчеты тех типов, которые заданы в опции **-O**. Для указанных типов отчетов создаются как общие, так и подробные отчеты. Подробные отчеты содержат те же записи, что и общие отчеты, а также статистику выполнения различных транзакций.

Статистика выполнения транзакций включает число транзакций заданного типа, время ответа и, при необходимости, объем обработанных данных. Для объема обработанных данных указывается среднее, минимальное и максимальное значения, а также стандартное отклонение. Примерно две трети указанных значений находятся в пределах стандартного отклонения от среднего значения. Объем данных указывается в байтах. Время ответа указывается в миллисекундах.

Подробная статистика обработчика прерываний процессора второго уровня:

Команда **netpmon** позволяет просмотреть статистику обработчика прерываний процессора второго уровня.

Ниже описаны поля отчета:

SLIH Имя обработчика прерываний второго уровня

count Число прерываний указанного типа

время cpu (мс)
Процессорное время, затраченное на обработку прерываний данного типа

Подробная статистика драйверов сетевых устройств:

Команда **netpmon** позволяет просмотреть статистику драйвера каждого из сетевых устройств.

Ниже описаны поля отчета:

Устройство
Имя файла, связанного с устройством

получено пакетов
Число пакетов, полученных через данное устройство

получено (байт)
Общий объем полученных пакетов

время получения (мс)
Время обработки полученных пакетов

время на ур. инт. (мс)
Время обработки полученных пакетов на уровне интерфейса

передано пакетов
Число пакетов, переданных через это устройство

получено (байт)
Общий объем переданных пакетов

время передачи (мс)
Время обработки переданных пакетов

Существуют и другие типы подробных отчетов, в том числе `Подробная статистика передачи для драйвера сетевого устройства (по хостам)` и `Подробная статистика обращений к сокетам TCP (по процессам)`. Для клиента NFS создаются отчеты `Подробная статистика обращений клиента NFS к серверам (по файлам)`, `Подробная статистика вызова RPC клиентом NFS (по серверам)` и `Подробная статистика работы клиента NFS (по процессам)`. Для сервера NFS дополнительно создается отчет `Подробная статистика обращений клиентов к серверу NFS`. Поля этих отчетов аналогичны описанным ниже.

В приведенном примере значения из отчета `Подробная статистика для драйвера сетевого устройства` связаны следующим образом:

- получено байт = 33003 пакетов * 73,2 байт/пакет = 2 415 819,6 байт
- передано байт = 61837 пакетов * 1514 байт/пакет = 93 621 218 байт
- общее число байт = 2 415 819,6 + 93 621 218 = 96 037 037,6 байт
- общее число бит = 96 037 037,6 * 8 бит/байт = 768 296 300,8 бит
- скорость передачи данных в сети = 768 296 300,8 / 8,54 = 89 964 438 бит/с (примерно 90 Мбит/с) - с учетом того, что при копировании NFS выполнялась полная трассировка

Из этого отчета, как и из общего отчета для драйвера устройства, видно, что сеть загружена сильно. Средний размер полученных пакетов составлял 73,2 байта; это говорит о том, сервер NFS, для которого выполнялась трассировка, получал подтверждения для отправляемых данных. Средний размер отправляемых пакетов составил 1514 байт, что равно значению максимального блока передачи (MTU) для устройства Ethernet *интерфейс* по умолчанию. Заменив *интерфейс* на нужное имя интерфейса, например, `en0` или `tr0`, можно изменить MTU или значение длины очереди отправки и повысить производительность с помощью следующей команды:

```
# ifconfig tr0 mtu 8500
```

или

```
# chdev -l 'tok0' -a xmt_que_size='150'
```

Если сеть уже перегружена, то изменение размера MTU или очереди передачи не приведет к повышению производительности.

Примечание:

1. Если в отчете о работе драйвера устройства указано, что размер отправляемых и принимаемых пакетов невелик, то для повышения производительности рекомендуется увеличить текущий размер MTU.
2. Если в отчете о работе клиента NFS указано, что время ожидания выполнения сетевых функций велико, то низкая производительность связана с передачей данных по сети.

Ограничения команды netpmon:

Для получения статистической информации команда **netpmon** применяет функцию трассировки. Следовательно, ее применение создает дополнительную нагрузку на систему.

- Если в рабочей схеме многие приложения работают с сетью, то команда **netpmon** увеличивает нагрузку на процессор примерно на 3-5 процентов.
- В системе с сильно загруженным процессором и небольшим числом операций ввода-вывода команда **netpmon** замедляет компиляцию большой программы на 3,5 процента.

Для того чтобы избежать снижения производительности, рекомендуется выполнять автономную обработку данных трассировки, а в многопроцессорных системах - указывать в команде **trace** флаг **-C all**.

Команда traceroute

Команда **traceroute** может применяться для тестирования, администрирования и оценки значений параметров сети.

Команда **ping** позволяет убедиться, что удаленная сеть IP достижима. Однако с ее помощью нельзя обнаружить и устранить неполадки. Рассмотрим следующий пример:

- Если маршрут от системы к целевому хосту содержит много транзитных узлов (например, шлюзов и маршрутизаторов), то неполадка может возникнуть на любом из этих узлов. В этом случае нужно выяснить, на каком этапе теряется пакет.
- Если пакет не удается доставить, команда **ping** не сообщает причину ошибки.

Команда **traceroute** позволяет узнать, где в данный момент находится пакет, и почему его нельзя доставить по заданному маршруту. Если пакет будет передаваться через маршрутизаторы и каналы связи, принадлежащие другим организациям, то вам вряд ли удастся проверить состояние этих маршрутизаторов с помощью команды **telnet**. В этом случае можно воспользоваться командами **traceroute** и **ping**.

Примечание: В основном команда **traceroute** предназначена для локализации неполадок. Поскольку команда **traceroute** создает дополнительную нагрузку на сеть, не применяйте ее во время обычной работы и не включайте в сценарии.

Примеры удачного выполнения команды traceroute

Команда **traceroute** применяет протокол UDP и функцию создания отчета об ошибках, предусмотренную в ICMP. Эта команда трижды отправляет пакет UDP каждому шлюзу или маршрутизатору, входящему в маршрут. Первый пакет отправляется ближайшему шлюзу. Второй пакет отправляется следующему за ним транзитному узлу и т.д. Последний пакет отправляется целевой системе. В выводе команды указывается имя шлюза, IP-адрес шлюза и время оборота трех пакетов, отправленных шлюзу. Ниже приведен пример вывода команды:

```
# traceroute aix1
получение исходного узла для aix1
исходный узел 10.53.155.187
traceroute до aix1.austin.ibm.com (10.53.153.120) от 10.53.155.187 (10.53.155.187), не более 30 участков
MTU исходящих пакетов = 1500
 1  10.111.154.1 (10.111.154.1)  5 мс  3 мс  2 мс
 2  aix1 (10.53.153.120)  5 мс  5 мс  5 мс
```

Ниже приведен другой пример:

```
# traceroute aix1
получение исходного узла для aix1
исходный узел 10.53.155.187
traceroute до aix1.austin.ibm.com (10.53.153.120) от 10.53.155.187 (10.53.155.187), не более 30 участков
MTU исходящих пакетов = 1500
 1  10.111.154.1 (10.111.154.1)  10 мс  2 мс  3 мс
 2  aix1 (10.53.153.120)  8 мс  7 мс  5 мс
```

Команда была повторно вызвана после того, как истек срок действия записи ARP. Обратите внимание, что время оборота первого пакета, отправляемого шлюзу или целевому хосту, больше, чем у остальных пакетов. Это связано с тем, что дополнительное время затрачивается на получение информации от протокола ARP. Если маршрут проходит через сеть с коммутацией каналов (WAN), то при передаче первого пакета дополнительное время будет затрачено на установление соединения. В результате может произойти тайм-аут. Тайм-аут по умолчанию составляет 3 секунды. Его можно изменить с помощью опции **-w**.

Первые 10 мс затрачены системой (9.53.155.187) на получение адреса шлюза 9.111.154.1 с помощью протокола ARP. Следующие 8 мс затрачены шлюзом на получение адреса целевого хоста (wave). В данном случае применяется DNS, поэтому перед отправкой каждого пакета команда **traceroute** обращается к серверу DNS.

Примеры неполадок, возникающих при выполнении команды traceroute

Если маршрут к целевому хосту включает большое число транзитных узлов или проходит через сети со сложной конфигурацией, то при вызове команды **traceroute** часто возникают различного рода ошибки. Поскольку многие аспекты работы команды зависят от ее реализации, во многих случаях невозможно найти причину ошибки. Однако если все маршрутизаторы и системы, через которые передается пакет, относятся к вашей организации, то вы можете проанализировать причину ошибки.

Неполадки возникающие при передаче пакета через шлюз (маршрутизатор)

В приведенном ниже примере пакеты отправляются из системы 9.53.155.187. Маршрут к мосту проходит через две системы, играющие роль маршрутизаторов. Во второй из этих систем функция маршрутизации была временно выключена (опции **ipforwarding** было присвоено значение 0 с помощью команды **no**). Рассмотрим приведенный ниже пример:

```
# traceroute lamar
trying to get source for lamar
source should be 9.53.155.187
traceroute to lamar.austin.ibm.com (9.3.200.141) from 9.53.155.187 (9.53.155.187), 30 hops max
MTU исходящих пакетов = 1500
 1  9.111.154.1 (9.111.154.1)  12 ms  3 ms  2 ms
 2  9.111.154.1 (9.111.154.1)  3 ms !H *  6 ms !H
```

Все сообщения об ошибках ICMP, за исключением Истекло время и Порт недостижим, выдаются в следующем формате:

!H Хост недостижим

!N Сеть недостижима

!P Протокол недоступен

!S Ошибка в исходном маршруте

!F Требуется фрагментация

Неполадки, вызванные целевой системой

Если в течение 3 секунд от целевой системы не будет получен ответ, то для всех запросов возникает тайм-аут, и в выводе команды указывается звездочка (*).

```
# traceroute chuys
trying to get source for chuys
source should be 9.53.155.187
traceroute to chuys.austin.ibm.com (9.53.155.188) from 9.53.155.187 (9.53.155.187), 30 hops max
MTU исходящих пакетов = 1500
 1  * * *
 2  * * *
 3  * * *
^C#
```

Если предположительно ошибка связана с каналом связи, то увеличьте значение тайм-аута с помощью флага **-w**. Кроме того, возможно, хотя и маловероятно, что все порты были заняты. Попробуйте изменить номера портов и повторите запрос.

Число целевых узлов

Ниже приведен другой пример вывода команды:

```
# traceroute mysystem.university.edu (129.2.130.22)
traceroute to mysystem.university.edu (129.2.130.22), 30 hops max
1 helios.ee.lbl.gov (129.3.112.1) 0 ms 0 ms 0 ms
2 lilac-dmc.university.edu (129.2.216.1) 39 ms 19 ms 39 ms
3 lilac-dmc.university.edu (129.2.215.1) 19 ms 39 ms 19 ms
4 ccngw-ner-cc.university.edu (129.2.135.23) 39 ms 40 ms 19 ms
5 ccn-nerif35.university.edu (129.2.167.35) 39 ms 39 ms 39 ms
6 csgw/university.edu (129.2.132.254) 39 ms 59 ms 39 ms
7 * * *
8 * * *
9 * * *
10 * * *
11 * * *
12 * * *
13 rip.university.EDU (129.2.130.22) 59 ms! 39 ms! 39 ms!
```

Демон iptrace и команды ipreport и ipfilter

Существуют различные средства для получения информации о работе сети. Некоторые из них входят в состав операционной системы. Для работы других требуется специальное аппаратное обеспечение. Для получения подробной информации о передаче отдельных пакетов по сети можно воспользоваться демоном **iptrace** и командой **ipreport**.

Для применения демона **iptrace** в операционной системе версии 4 необходимо установить набор файлов bos.net.tcp.server. Этот набор файлов содержит демон **iptrace** и некоторые другие полезные команды, в частности, **trpt** и **tcdump**. Демон **iptrace** разрешено запускать только пользователю root.

По умолчанию демон **iptrace** включает трассировку для всех пакетов. С помощью опции **-a** можно исключить пакеты ARP. Другие опции позволяют трассировать только пакеты, полученные от

определенного хоста (**-s**), отправленные определенному хосту (**-d**) или соответствующие определенному протоколу (**-p**). Поскольку на работу демона **iptrace** затрачивается значительное процессорное время, рекомендуется максимально сократить число пакетов, для которых включается трассировка.

Поскольку программа **iptrace** является демоном, ее нужно запускать с помощью команды **startsrc**, а не напрямую из командной строки. В этом случае будет проще управлять работой команды и ее завершением. Ниже приведен пример вызова этой команды:

```
# startsrc -s iptrace -a "-i en0 /home/user/iptrace/log1"
```

Данная команда запускает демон **iptrace** для трассировки всех пакетов, передаваемых с помощью интерфейсов Gigabit Ethernet и `en0`. Данные трассировки будут сохраняться в файле `/home/user/iptrace/log1`. Для того чтобы завершить работу демона, вызовите следующую команду:

```
# stopsrc -s iptrace
```

Если демон **iptrace** был запущен без помощи команды **startsrc**, то для завершения его работы определите ИД процесса демона с помощью команды **ps** и убейте процесс с помощью команды **kill**.

Команда **ipreport** служит для форматирования файла протокола. Отформатированный протокол записывается в стандартный вывод. Различные опции позволяют распознавать и форматировать пакеты RPC (**-r**), присвоить каждому пакету номер (**-n**) и добавить к каждой строке префикс из трех символов, обозначающий протокол (**-s**). Ниже приведен пример вызова команды **ipreport** для форматирования только что созданного файла `log1`, принадлежащего пользователю root:

```
# ipreport -ns log1 >log1_formatted
```

Результатом выполнения этой команды будет последовательность отчетов о пакетах, как в приведенных ниже примерах. Первым указывается пробный пакет команды **ping**. Рекомендуется обратить внимание на следующие поля:

- Адреса отправителя (`SRC`) и получателя (`DST`), заданные в десятичном и текстовом виде
- Размер IP-пакета (`ip_len`)
- Применяемый протокол высокого уровня (`ip_p`)

```
Номер пакета 7
ETH: ====( 98 передано с помощью интерфейса en0 )==== 10:28:16.516070112
ETH:    [ 00:02:55:6a:a5:dc -> 00:02:55:af:20:2b ]  type 800  (IP)
IP:     < SRC =       192.1.6.1 >  (en6host1)
IP:     < DST =       192.1.6.2 >  (en6host2)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=84, ip_id=1789, ip_off=0
IP:     ip_ttl=255, ip_sum=28a6, ip_p = 1 (ICMP)
ICMP:   тип_icmp=8 (ECHO_REQUEST)  ид_icmp=18058  номер_icmp=3

Номер пакета 8
ETH: ====( 98 получено с помощью интерфейса en0 )==== 10:28:16.516251667
ETH:    [ 00:02:55:af:20:2b -> 00:02:55:6a:a5:dc ]  type 800  (IP)
IP:     < SRC =       192.1.6.2 >  (en6host2)
IP:     < DST =       192.1.6.1 >  (en6host1)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=84, ip_id=11325, ip_off=0
IP:     ip_ttl=255, ip_sum=366, ip_p = 1 (ICMP)
ICMP:   тип_icmp=0 (ECHO_REPLY)  ид_icmp=18058  номер_icmp=3
```

Ниже приведен фрагмент вывода **ftp**. Размер IP-пакета совпадает с размером MTU этой сети (1492 байт).

```
Номер пакета 20
ETH: ====( 1177 байт передано с помощью интерфейса en0 )==== 10:35:45.432353167
ETH:    [ 00:02:55:6a:a5:dc -> 00:02:55:af:20:2b ]  type 800  (IP)
IP:     < SRC =       192.1.6.1 >  (en6host1)
IP:     < DST =       192.1.6.2 >  (en6host2)
IP:     ip_v=4, ip_hl=20, ip_tos=8, ip_len=1163, ip_id=1983, ip_off=0
IP:     ip_ttl=60, ip_sum=e6a0, ip_p = 6 (TCP)
TCP:    <исходный_порт=32873, целевой_порт=20(данные-ftp) >
TCP:    th_seq=623eabdc, th_ack=973dcd95
TCP:    th_off=5, flags<PUSH | ACK>
```

```
TCP:    th_win=17520, th_sum=0, th_urp=0
TCP: 00000000     69707472 61636520 322e3000 00008240     |iptrace 2.0....@|
TCP: 00000010     2e4c9d00 00000065 6e000065 74000053     |.L.....en..et..S|
TCP: 00000020     59535841 49584906 01000040 2e4c9d1e     |YSXAIXI....@.L..|
TCP: 00000030     c0523400 0255af20 2b000255 6aa5dc08     |.R4..U. +..Uj...|
TCP: 00000040     00450000 5406f700 00ff0128 acc00106     |.E..T......(....|
TCP: 00000050     01c00106 0208005a 78468a00 00402e4c     |.......ZxF...@.L|
TCP: 00000060     9d0007df 2708090d 0a0b0c0d 0e0f1011     |....'...........|
TCP: 00000070 12131415 16171819 1a1b1c1d 1e1f2021 |.............. !|
TCP: 00000080     22232425 26272829 2a2b2c2d 2e2f3031     |"#$%&'()*+,-./01|
TCP: 00000090     32333435 36370000 0082402e 4c9d0000     |234567....@.L...|
            --------- Пропущена часть вывода команды -----------
TCP: 00000440     15161718 191a1b1c 1d1e1f20 21222324     |........... !"#$|
TCP: 00000450     25262728 292a2b2c 2d2e2f30 31323334     |%&'()*+,-./01234|
TCP: 00000460     353637                                  |567             |
```

Команда **ipfilter** выдает таблицу заголовков операций, полученных из файла вывода **ipreport**. Кроме того, она показывает некоторую информацию о запросах NFS и ответах на эти запросы.

Для того чтобы узнать, установлена ли команда **ipfilter** в системе, вызовите следующую команду:

```
# lslpp -lI perfagent.tools
```

Ниже приведен пример вызова этой команды:

```
# ipfilter log1_formatted
```

Распознаны следующие заголовки операций: udp, nfs, tcp, ipx, icmp. Команда **ipfilter** создает отчеты трех типов:

- Файл `ipfilter.all`, содержащий список всех операций выбранного типа. В таблице указывается номер пакета, время, адрес отправителя и получателя, размер, порядковый номер, номер подтверждения, исходный порт, целевой порт, сетевой интерфейс и тип операции.
- Файлы с отчетами для выбранных заголовков (`ipfilter.udp`, `ipfilter.nfs`, `ipfilter.tcp`, `ipfilter.ipx`, `ipfilter.icmp`). В этих файлах указывается та же информация, что и в файле `ipfilter.all`.
- Файл `nfs.rpt`, содержащий отчет о запросах NFS и ответах на эти запросы. В таблице указывается ИД транзакции, тип запроса, состояние запроса, номер пакета с запросом, время отправки запроса, размер запроса, номер пакета с ответом, время отправки ответа, размер ответа и время между отправкой запроса и получением ответа.

Статистика работы адаптера

Описанные в этом разделе команды позволяют получить примерно ту же информацию, что и команда **netstat -v**. С их помощью можно сбросить статистику адаптера (**-r**) и получить более подробный отчет, чем предоставляется командой **netstat -v** (опция **-d**).

Команда entstat

Команда **entstat** позволяет просмотреть статистическую информацию, собранную указанным драйвером адаптера Ethernet. Помимо общей информации можно просмотреть дополнительную информацию об указанном устройстве. Для этого служит опция **-d**. С ее помощью можно просмотреть полную статистическую информацию. Если флаги не заданы, то выдается только общая статистическая информация.

Команда **entstat** вызывается командой **netstat** с флагом **-v**. Команда **netstat** вызывает команду **entstat** без флагов.

```
# entstat ent0
-------------------------------------------------------------
Статистика Ethernet (ent0) :
Тип устройства: Адаптер 10/100/1000 Base-TX PCI-X (14106902)
Аппаратный адрес: 00:02:55:6a:a5:dc
Прошедшее время: 1 день 18 часов 47 минут 34 секунды
```

```
Статистика по передаче:                     Статистика по приему:
-----------------------                     ---------------------
Пакетов: 1108055                            Пакетов: 750811
Байт: 4909388501                            Байт: 57705832
Прерываний: 0                               Прерываний: 681137
Ошибок при передаче: 0                      Ошибок при получении: 0
Отброшено пакетов: 0                        Отброшено пакетов: 0
                                            Пакетов с ошибками: 0
Максимальное число пакетов в программной очереди передачи: 101
Переполнение программной очереди передачи: 0
Текущая длина прогр. и аппар. очередей передачи: 0

Пакетов оповещения: 3                       Пакетов оповещения: 3
Многоцелевых пакетов: 3                     Многоцелевых пакетов: 5
Несущая частота не обнаружена: 0            Ошибка CRC: 0
Выход за нижнюю границу DMA: 0              Выход за верхнюю границу DMA: 0
Ошибки потери CTS: 0                        Ошибки выравнивания: 0
Максимальное число конфликтов: 0            Ошибок отсутствия ресурсов: 0
Поздние конфликты: 0                        Конфликты при получении: 0
Задержки: 0                                 Слишком короткие пакеты: 0
Проверка SQE: 0                             Слишком длинные пакеты: 0
Ошибки ожидания: 0                          Пакеты, отброшенные адаптером: 0
Число отдельных конфликтов: 0               Начальное значение для получателя: 0
Число множественных конфликтов: 0
Текущая длина аппаратной очереди передачи: 0

Общая статистика:
------------------
Ошибки отсутствия mbuf: 0
Число перезапусков адаптера: 0
Скорость передачи данных адаптером: 2000
Флаги драйвера: Up Broadcast Running
        Simplex 64BitSupport ChecksumOffload
        PrivateSegment LargeSend DataRateSet
```

В приведенном выше отчете стоит обратить внимание на следующее:

Ошибки передачи
Число ошибок, возникших при передаче данных через данное устройство. Это число учитывает аппаратные ошибки и ошибки в работе сети.

Ошибки приема
Число ошибок, возникших при приеме данных через данное устройство. Это число учитывает аппаратные ошибки и ошибки в работе сети.

Отброшено пакетов
Число пакетов, принятых драйвером устройства для отправки, но не переданных устройству.

Максимальное число пакетов в программной очереди передачи
Максимальное число пакетов, которое когда-либо находилось в программной очереди передачи.

Переполнение программной очереди передачи
Число переполнений очереди передачи.

Ошибки из-за отсутствия ресурсов
Число полученных пакетов, отброшенных на аппаратном уровне из-за нехватки ресурсов. Обычно такие ошибки связаны с переполнением буфера приема адаптера. Некоторые адаптеры позволяют изменять размер буфера приема. Информацию о настройке можно найти в описании атрибутов устройства или в справке SMIT.

Число одиночных/множественных конфликтов
Число конфликтов в сети Ethernet. Эти конфликты не учитываются в соответствующем поле вывода команды **netstat -i**.

Обратите внимание, что в приведенном примере адаптер справляется с нагрузкой, так как нет `ошибок приема`. Эти ошибки иногда возникают в перегруженной сети, если по ней передаются только неполные пакеты. Неполные пакеты обычно передаются повторно, однако засчитываются как ошибка приема.

Если значение `Переполнений прогр. очереди передачи` отлично от нуля, то значение `Макс. число пакетов в прогр. очереди передачи` отражает максимальный размер очереди передачи данного адаптера (**xmt_que_size**).

Примечание: Указанные значения будут соответствовать *аппаратной очереди*, если адаптер не поддерживает программную очередь передачи. Если зафиксированы случаи переполнения очереди передачи, увеличьте ограничения на размер аппаратной или программной очереди драйвера.

Если в системе недостаточно ресурсов для приема пакетов, то значение `Отброшено пакетов:` будет отлично от нуля, а в качестве причины будет указано `Недостаточно буферов приема`, `Ошибки из-за нехватки ресурсов` или аналогичное значение.

Истекшее время указывает время, прошедшее с момента предыдущего сброса статистики. Для сброса статистики служит команда **entstat -r** *имя-адаптера*.

Для получения аналогичной информации об интерфейсах Token-Ring, FDDI и ATM служат команды **tokstat**, **fddistat** и **atmstat**.

Команда tokstat

Команда **tokstat** выводит статистическую информацию, собранную указанным драйвером устройства Token-Ring. Помимо статистики, собранной драйвером устройства, пользователь может просмотреть дополнительную информацию об указанном устройстве. Если флаги не заданы, то выводится только статистика, собранная драйвером устройства.

Эта команда вызывается командой **netstat** с флагом **-v**. Команда **netstat** вызывает команду **tokstat** без флагов.

Вывод команды **tokstat tok0** и алгоритм определения неполадки аналогичен выводу “Команда entstat” на стр. 326.

Команда fddistat

Команда **fddistat** выводит статистическую информацию, собранную указанным драйвером устройства FDDI. Помимо статистики, собранной драйвером устройства, пользователь может просмотреть дополнительную информацию об указанном устройстве. Если флаги не заданы, то выводится только статистика, собранная драйвером устройства.

Эта команда вызывается командой **netstat** с флагом **-v**. Команда **netstat** вызывает команду **fddistat** без флагов.

Вывод команды **fddistat fddi0** и алгоритм определения неполадки аналогичен выводу “Команда entstat” на стр. 326.

Команда atmstat

Команда **atmstat** выводит статистическую информацию, собранную указанным драйвером устройства ATM. Помимо статистики, собранной драйвером устройства, пользователь может просмотреть дополнительную информацию об указанном устройстве. Если флаги не заданы, то выводится только статистика, собранная драйвером устройства.

Вывод команды **atmstat atm0** и алгоритм определения неполадки аналогичен выводу “Команда entstat” на стр. 326.

Команда no

Команда **no** позволяет просматривать и изменять текущие значения параметров сети.

-a Выводит текущие значения параметров

-d Восстанавливает значения по умолчанию

-o *опция=новое-значение*

Список атрибутов команды **no** приведен в разделе “Параметры сети” на стр. 453.

Примечание: Команда **no** не проверяет, что указанное значение параметра допустимо. Вызов команды **no** с недопустимым значением параметра может нарушить работу системы.

Некоторые атрибуты сети можно изменить в любой момент. Другие атрибуты изменяются во время загрузки. Их нужно изменить до загрузки расширения ядра **netinet**.

Примечание: Изменения, внесенные с помощью команды **no**, действуют до следующей загрузки системы. После загрузки всем параметрам присваиваются значения по умолчанию.

Примечание: Для внесения изменений на постоянной основе их необходимо включить в файл `/etc/tunables/nextboot`. В командной строке выполните команду **no -r -o <***опция-no***>=<***значение***>** (например, **no -r -o arptab_bsiz=10**). В ходе последующих перезапусков параметр **arptab_bsiz=10** остается активным и применяется к файлу nextboot.

Если в системе применяется конфигурация сети типа Беркли, то поместите атрибуты в начало файла `/etc/rc.bsdnet`. При работе с системой SP внесите изменения в файл `tuning.cust`.

Производительность NFS

AIX предоставляет инструменты и способы для отслеживания и тонкой настройки NFS и для сервера, и для клиента.

Задачи, связанные с данной:

“Повышение быстродействия записи больших файлов клиента NFS” на стр. 458
Регулярная последовательная запись файлов очень большого размера в файловые системы NFS может привести к серьезному снижению скорости обмена данными с сервером NFS. Данный пример иллюстрирует диагностику такой ситуации и возможные пути устранения проблемы.

Сетевые файловые системы

NFS позволяет программам одной системы напрямую работать с файлами другой системы путем монтирования удаленного каталога.

Обычно после загрузки сервера открывается доступ к каталогам с помощью команды **exportfs**, а затем запускаются демоны (демоны **nfsd**), управляющие удаленным доступом. Аналогично, при загрузке клиента монтируются удаленные каталоги и запускаются демоны блочного ввода-вывода (**biod**).

Демоны **nfsd** и **biod** являются многонитевыми. Это значит, что в рамках одного процесса в ядре может работать несколько нитей. Кроме того, эти демоны автоматически выполняют все операции, связанные с созданием и удалением нитей в соответствии с уровнем загрузки NFS.

На следующем рисунке показана структура диалога между клиентами NFS и сервером. Если нить клиентской системы создает запрос на чтение или запись в файл, находящийся в каталоге NFS, запрос не обрабатывается обычным образом, а направляется одной из нитей **biod** системы клиента. Нить **biod**

отправляет запрос соответствующему серверу, который присваивает этот запрос одной из нитей сервера NFS (**nfsd**). Во время обработки запроса обе нити - **biod** и **nfsd** - не могут выполнять другие операции.

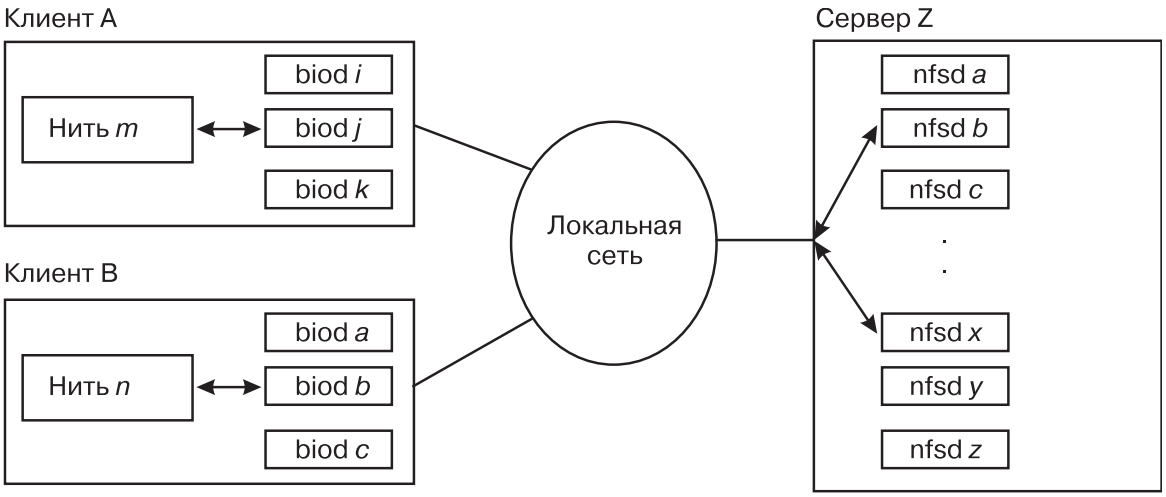


Рисунок 22. Взаимодействие клиентов с сервером NFS. На этом рисунке показано взаимодействие двух клиентов и одного сервера, расположенных в сети с топологией типа "звезда". На клиенте A выполняется нить приложения m, в которой данные передаются одной из нитей biod. Аналогичным образом на клиенте B выполняется нить n, данные которой направляются одной из нитей biod. Нити пересылают данные по сети на сервер Z, где эти данные присваиваются одной из нитей сервера NFS (nfsd).

Для передачи данных по сети в NFS применяется протокол RPC (Вызов удаленных процедур). Этот протокол основан на протоколе Внешнего представления данных (XDR), преобразующем данные в общий формат перед передачей по сети и позволяющем обмениваться информацией системам с различной архитектурой. Библиотека RPC - это библиотека процедур, позволяющая локальному (клиентскому) процессу вызывать процедуры в удаленном (серверном) процессе точно так же, как обычные процедуры в своем адресном пространстве. Поскольку клиент и сервер - это два отдельных процесса, они могут находиться в разных физических системах.

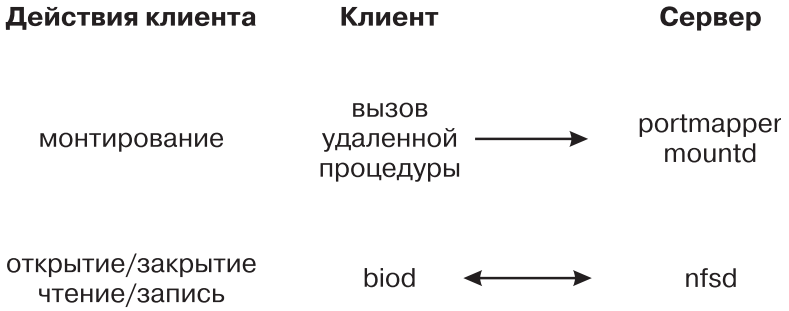


Рисунок 23. Процессы Mount и NFS. На этом рисунке показана таблица с тремя столбцами - Операции клиента, Клиент и Сервер. Вначале клиент выполняет операцию монтирования. Для этого клиент с помощью RPC вызывает демон сервера portmapper mountd. Затем клиент выполняет операции типа открыть/закрыть и чтение/запись. Нить клиента biod и нить сервера nfsd обмениваются данными друг с другом.

Демон **portmap**, **portmapper**, - это демон сетевой службы, позволяющий клиенту узнать номер порта, связанный с указанной программой. При вызове служб сервера они регистрируются демоном **portmap** в качестве доступного сервера. На основе этой информации демон **portmap** создает и обслуживает таблицу соответствия между программами и портами.

Когда клиент отправляет запрос серверу, он обращается к демону **portmap**, для того чтобы узнать номер порта службы. Демон **portmap** работает со стандартным портом, номер которого известен клиенту. Демон **portmap** сообщает клиенту номер порта службы, запрашиваемой клиентом. Все последующие запросы клиент может направлять напрямую приложению.

Демон **mountd** - это демон сервера, отвечающий на запросы клиента на монтирование экспортированной файловой системы или каталога сервера. Демон **mountd** определяет, доступна ли файловая система, считывая информацию из файла `/etc/xtab`. Процесс монтирования происходит следующим образом:

1. Клиент отправляет запрос демону **portmap**, для того чтобы узнать номер порта демона **mountd**.
2. Демон **portmap** возвращает клиенту номер порта.
3. При вызове команды **mount** клиент напрямую обращается к демону сервера **mountd** и передает ему имя монтируемого каталога.
4. Демон **mountd** на сервере проверяет с помощью файла `/etc/xtab` (создается командой **exportfs -a** на основе файла `/etc/exports`), что каталог доступен, и у клиента есть необходимые права доступа.
5. Затем демон сервера **mountd** получает описатель экспортируемого каталога (указатель на каталог файловой системы) и передает его ядру клиента.

Клиент обращается к демону **portmap** только во время обработки первого запроса на монтирование, сделанного после запуска системы. Все последующие запросы на монтирование будут направляться клиентом через полученный порт **mountd**.

Демон **biod** - это демон блочного ввода-вывода, который применяется при выполнении запросов на упреждающее чтение, отложенную запись и чтение каталогов. Нити демона **biod** повышают быстродействие NFS, заполняя или очищая кэш от имени приложений клиентов NFS. Когда пользователь клиентской системы отправляет запрос на запись или чтение файлов сервера, нити **biod** передают его запрос на сервер. Ниже перечислены операции NFS, запросы на выполнение которых напрямую передаются серверу из расширения ядра операционной системы клиента NFS без участия демона **biod**:

- **getattr()**
- **setattr()**
- **lookup()**
- **readlink()**
- **create()**
- **remove()**
- **rename()**
- **link()**
- **symlink()**
- **mkdir()**
- **rmdir()**
- **readdir()**
- **readdirplus()**
- **fsstat()**

Демон **nfsd** - это активный агент, предоставляющий доступ к службам сервера NFS. Прием любого запроса по протоколу NFS происходит с помощью нити демона **nfsd**; он работает с этим запросом до того момента, как запрос будет выполнен, а результаты выполнения будут отправлены клиенту.

Транспортный уровень в NFS

В качестве протокола передачи по умолчанию в NFS применяется TCP, но при необходимости можно выбрать и протокол UDP.

Транспортный протокол выбирается при монтировании. Протокол UDP рекомендуется применять при работе с надежными сетями, сетями с высокой скоростью передачи данных и серверами, быстро обрабатывающими запросы. При работе с глобальными сетями, сильно загруженными сетями и серверами, медленно обрабатывающими запросы, рекомендуется применять протокол TCP, обеспечивающий интегрированное управление потоком, что позволяет минимизировать число повторных передач пакетов.

Различные версии NFS

AIX поддерживает применение NFS версий 2 и 3 на одном и том же компьютере, также поддерживается NFS версии 4.

По умолчанию по-прежнему применяется NFS версии 3, если только в опции монтирования клиента AIX не указана другая версия. Аналогично транспортному протоколу, версия NFS выбирается при монтировании.

NFS версии 4:

NFS версии 4 - это новейшая версия NFS, описанная в RFC 3530.

Хотя она схожа с предыдущими версиями, главным образом с NFS версии 3, в новом протоколе предусмотрено много функциональных расширений в таких областях, как защита, расширяемость и управление данными. Поэтому NFS версии 4 - лучший вариант для крупномасштабных распределенных сред с общими файловыми ресурсами.

Некоторые из новых функций NFS версии 4:

- “Изменение реализации операций NFS”
- “Требование TCP” на стр. 333
- “Интегрированный протокол блокировок” на стр. 333
- “Встроенная поддержка монтирования” на стр. 333
- “Улучшенные механизмы защиты” на стр. 333
- “Функции интернационализации” на стр. 333
- “Модель расширяемых атрибутов” на стр. 333
- “Поддержка списков с правами доступа” на стр. 333

Однако расширенная функциональность и повышенная сложность нового протокола означают, что на его поддержку требуется больше ресурсов. Поэтому производительность NFS версии 4 может оказаться ниже, чем NFS версии 3. Разница в производительности в значительной степени зависит от того, какие из новых функций будут использоваться при работе с протоколом. Например, если в NFS версии 4 и версии 3 применяются одни и те же механизмы защиты, снижение производительности при переходе на NFS версии 4 будет незначительным. Однако если в NFS версии 3 применялась стандартная схема идентификации UNIX (AUTH_SYS), а в версии 4 будет применяться Kerberos 5 с защитой (т.е. с шифрованием всех пользовательских данных), падение производительности будет очень заметным.

Рекомендации по повышению производительности NFS версии 3, как правило, распространяются и на NFS версии 4.

Изменение реализации операций NFS:

В отличие от NFS версий 2 и 3, в версии 4 предусмотрены только две процедуры RPC: NULL и COMPOUND.

Процедура COMPOUND состоит из одной или нескольких операций NFS, которые в предыдущих версиях были реализованы как самостоятельные процедуры RPC. Это изменение позволяет сократить число RPC, выполняющих операции в логической файловой системе по сети.

Требование TCP:

NFS версии 4 может применяться только с транспортным протоколом, в котором предусмотрено управление нагрузкой для повышения производительности в средах WAN.

AIX не поддерживает применение UDP с NFS версии 4.

Интегрированный протокол блокировок:

В NFS версии 4 поддерживается возможность блокирования диапазонов байтов в файлах.

Протокол управления сетевыми блокировками (NLM) и демоны `rpc.lockd` и `rpc.statd` не используются. Для максимальной совместимости с операционными системами, отличными от UNIX, в NFS версии 4 поддерживаются резервирование открытых общих ресурсов и функции принудительной блокировки для различных серверных платформ.

Встроенная поддержка монтирования:

В NFS версии 4 поддерживается монтирование файловых систем средствами протокола.

Клиентам не требуются специализированный протокол монтирования и демон `rpc.mountd`.

Улучшенные механизмы защиты:

В NFS версии 4 поддерживается протокол RPCSEC-GSS.

Протокол защиты RPCSEC-GSS допускает применение нескольких механизмов защиты без необходимости создавать определения идентификации RPC для них. В AIX NFS поддерживает только механизм защиты Kerberos 5.

Функции интернационализации:

В NFS версии 4 символьные данные передаются в кодировке UTF-8.

Модель расширяемых атрибутов:

Модель атрибутов в NFS версии 4 лучше приспособлена для применения в средах, отличных от UNIX. В ней стало проще создавать определения новых атрибутов.

Поддержка списков с правами доступа:

В NFS версии 4 входит определение атрибута ACL.

Модель ACL схожа с моделью, применяемой в Windows NT, в том, что в ней предусмотрен набор стандартных записей, разрешающих или запрещающих доступ к ресурсу отдельным пользователям или группам.

NFS версии 3:

Настоятельно рекомендуется применять NFS версии 3 вместо NFS версии 2, так как встроенные функции протоколов позволяют повысить производительность работы.

Запись throughput:

Приложения клиентской системы могут периодически записывать данные в файл, обновляя его содержимое.

Объем данных, которые приложение может записать на сервер за указанный период времени, является мерой *производительности записи* распределенной файловой системы. Производительность записи существенно влияет на общую производительность. Все распределенные файловые системы, включая NFS, должны обеспечивать надежную запись данных в указанный файл за минимальное время.

В протоколе NFS версии 2 производительность записи можно было увеличить только путем отказа от режима синхронной записи. В версии 3 предусмотрен и другой способ. В NFS версии 3 существенное повышение производительности операций записи достигается за счет записи данных в кэш сервера (т.е. в оперативную память), а не на диск. Для принудительной записи данных сразу на диск клиент NFS должен выполнить операцию фиксации. Эта функция, которая называется *надежная асинхронная запись*, существенно уменьшает количество запросов на дисковый ввод-вывод на сервере, тем самым значительно увеличивая производительность записи.

Операции записи считаются "надежными", поскольку отслеживается состояние данных, указывающее, были ли данные записаны на диск. Таким образом, если сбой сервера происходит до операции фиксации, клиент может узнать с помощью индикатора состояния, нужно ли еще раз отправить запрос на запись.

Уменьшение числа запросов на получение атрибутов файла:

Поскольку считываемые данные иногда хранятся в кэше достаточно долго, клиентам приходится проверять, что данные в кэше не устарели. Для этого клиент NFS периодически просматривает атрибуты файла, включающие время последнего изменения файла. С помощью этого значения клиент может убедиться, что данные в кэше не устарели.

Уменьшение числа запросов на просмотр атрибутов файла повышает эффективность работы клиента, снижает нагрузку на сервер и повышает масштабируемость и производительность системы. В связи с этим в NFS версии 3 атрибуты передаются клиенту при выполнении любой операции над файлом. Это увеличивает вероятность того, что информация в кэше окажется не устаревшей, и уменьшает число запросов на получение атрибутов.

Эффективное использование сетей с высокой пропускной способностью:

Увеличение допустимого размера RPC позволяет NFS более эффективно использовать возможности высокопроизводительных сетей, например FDDI, 100baseT (100 Мбит/с) Ethernet, 1000baseT (Gigabit) Ethernet и SP Switch, и существенно повышает производительность последовательного чтения и записи.

В NFS версии 2 размер RPC не превышал 8 КБ, что ограничивало размер блока данных NFS, передаваемого по сети. В NFS версии 3 это ограничение было ослаблено. По умолчанию ввод-вывод в NFS AIX выполняется блоками по 32 КБ (максимум - 64 КБ), что позволяет NFS создавать и передавать блоки данных большего размера в одном пакете RPC.

Уменьшение числа запросов на просмотр содержимого каталога:

Для просмотра полного списка содержимого каталога, например, с помощью команды **ls -l**, необходимо получить от сервера имена и атрибуты всех файлов.

В NFS версии 2 для каждого файла отправлялся отдельный запрос на получение его имени и атрибутов. В NFS версии 3 список имен и атрибутов передается за одну операцию **READDIRPLUS**, что снижает нагрузку как на клиент, так и на сервер.

Добавлена поддержка кэширования длинных имен файлов (длиной больше 31 символа) в кэше поиска имен каталогов клиента NFS или **dnlc**. Это позволяет повысить производительность NFS при работе с длинными именами файлов, что раньше приводило к выполнению большого числа операций **LOOKUP** на сервере из-за промахов в кэше **dnlc**. В качестве примера рабочей схемы такого типа можно привести команду **ls -l**, упомянутую выше.

Отслеживание и настройка производительности NFS

Доступно несколько команд, позволяющих собирать статистику NFS и настраивать ее атрибуты.

Для достижения высокой производительности NFS требуется настройка и устранение узких мест не только самой NFS, но и операционной системы и соответствующего аппаратного обеспечения. Рабочие схемы, предусматривающие большое число операций чтения и записи, как правило требуют настройки всей системы. В этом разделе также приведена информация о рабочих схемах, при использовании которых применять NFS не рекомендуется.

Перед настройкой параметров убедитесь, что вы четко понимаете цель изменения их значений, а также возможные отрицательные побочные эффекты, которые могут быть вызваны этими изменениями.

Тонкая настройка NFS и статистика

NFS собирает статистическую информацию обо всех выполняемых операциях, а также информацию об ошибках и о производительности.

Ниже описаны команды, применяемые для обнаружения узких мест, просмотра статистической информации NFS, а также настройки параметров NFS.

Команда nfsstat:

Команда **nfsstat** выдает статистическую информацию об NFS и интерфейсе RPC ядра для клиентов и серверов.

Кроме того, с помощью этой команды можно сбросить соответствующие счетчики (**nfsstat -z**). Статистика RPC (опция **-r**) содержит наиболее важную информацию для анализа производительности. Статистика NFS позволяет оценить интенсивность работы приложений системы с NFS.

Статистика RPC:

Команда **nfsstat** предоставляет статистическую информацию о вызовах RPC.

Выводимая статистическая информация:

- Общее число полученных и отклоненных вызовов RPC
- Общее число вызовов RPC, отправленных серверу, и число вызовов, отклоненных сервером
- Число неудачных попыток получения пакета RPC
- Число пакетов со слишком короткими или неправильными заголовками
- Число повторных передач вызова
- Число ответов, не соответствующих запросу
- Число тайм-аутов вызова
- Число случаев ожидания освобождения описателя клиента
- Число обновлений идентификационной информации

Статистика NFS в выводе команды **nfsstat** делится на две части: для версии 2 и для версии 3. Статистика RPC поделена на разделы `С установлением соединения` (TCP) и `Без установления соединения` (UDP).

Вывод для конкретных разделов приведен в “Настройка производительности NFS на сервере” на стр. 343 и “Настройка NFS на клиенте” на стр. 347.

Команда nfso:

Команда **nfso** позволяет настроить атрибуты NFS.

С ее помощью вы можете просмотреть и изменить параметры NFS, связанные с текущим ядром и расширением ядра NFS. Более подробная информация о команде и ее выводе приведена в разделе Команда nfso книги *Справочник по командам, том 4*.

Примечание: Команда **nfso** не проверяет границы диапазонов. Указание в ней недопустимых значений может нарушить работу системы.

Для просмотра параметров команды **nfso** и их значений введите команду **nfso -a**:

```
# nfso -a
                    portcheck = 0
                  udpchecksum = 1
               nfs_socketsize = 60000
           nfs_tcp_socketsize = 60000
            nfs_setattr_error = 0
          nfs_gather_threshold = 4096
           nfs_repeat_messages = 0
nfs_udp_duplicate_cache_size = 5000
nfs_tcp_duplicate_cache_size = 5000
    nfs_server_base_priority = 0
         nfs_dynamic_retrans = 1
            nfs_iopace_pages = 0
         nfs_max_connections = 0
             nfs_max_threads = 3891
      nfs_use_reserved_ports = 0
    nfs_device_specific_bufs = 1
           nfs_server_clread = 1
                nfs_rfc1323 = 1
          nfs_max_write_size = 65536
           nfs_max_read_size = 65536
       nfs_allow_all_signals = 0
                  nfs_v2_pdts = 1
                  nfs_v3_pdts = 1
              nfs_v2_vm_bufs = 1000
              nfs_v3_vm_bufs = 1000
   nfs_securenfs_authtimeout = 0
   nfs_v3_server_readdirplus = 1
           lockd_debug_level = 0
           statd_debug_level = 0
           statd_max_threads = 50
             utf8_validation = 1
                  nfs_v4_pdts = 1
              nfs_v4_vm_bufs = 1000
```

Большинство атрибутов NFS можно изменить в любой момент. Однако значения некоторых атрибутов, например, *nfs_socketsize*, устанавливаются во время загрузки, поэтому для их изменения требуется перезапустить NFS. Команда **nfso -L** позволяет просмотреть более подробную информацию о каждом из этих атрибутов, включая текущее значение, значение по умолчанию и ограничения, связанные с вступлением изменений в силу:

```
# nfso -L

NAME                      CUR    DEF    BOOT   MIN    MAX    UNIT           TYPE  DEPENDENCIES
--------------------------------------------------------------------------------
portcheck                 0      0      0      0      1      On/Off           D
--------------------------------------------------------------------------------
udpchecksum               1      1      1      0      1      On/Off           D
--------------------------------------------------------------------------------
nfs_socketsize            600000 600000 600000 40000  1M     Bytes            D
--------------------------------------------------------------------------------
nfs_tcp_socketsize        600000 600000 600000 40000  1M     Bytes            D
--------------------------------------------------------------------------------
nfs_setattr_error         0      0      0      0      1      On/Off           D
--------------------------------------------------------------------------------
nfs_gather_threshold      4K     4K     4K     512    8193   Bytes            D
--------------------------------------------------------------------------------
```

```
nfs_repeat_messages       0      0      0      0      1      On/Off            D
--------------------------------------------------------------------------------
nfs_udp_duplicate_cache_size
                          5000   5000   5000   5000   100000 Req               I
--------------------------------------------------------------------------------
nfs_tcp_duplicate_cache_size
                          5000   5000   5000   5000   100000 Req               I
--------------------------------------------------------------------------------
nfs_server_base_priority  0      0      0      31     125    Pri               D
--------------------------------------------------------------------------------
nfs_dynamic_retrans       1      1      1      0      1      On/Off            D
--------------------------------------------------------------------------------
nfs_iopace_pages          0      0      0      0      65535  Pages             D
--------------------------------------------------------------------------------
nfs_max_connections       0      0      0      0      10000  Number            D
--------------------------------------------------------------------------------
nfs_max_threads           3891   3891   3891   5      3891   Threads           D
--------------------------------------------------------------------------------
nfs_use_reserved_ports    0      0      0      0      1      On/Off            D
--------------------------------------------------------------------------------
nfs_device_specific_bufs  1      1      1      0      1      On/Off            D
--------------------------------------------------------------------------------
nfs_server_clread         1      1      1      0      1      On/Off            D
--------------------------------------------------------------------------------
nfs_rfc1323               1      1      0      0      1      On/Off            D
--------------------------------------------------------------------------------
nfs_max_write_size        64K    32K    32K    512    64K    Bytes             D
--------------------------------------------------------------------------------
nfs_max_read_size         64K    32K    32K    512    64K    Bytes             D
--------------------------------------------------------------------------------
nfs_allow_all_signals     0      0      0      0      1      On/Off            D
--------------------------------------------------------------------------------
nfs_v2_pdts               1      1      1      1      8      PDTs              M
--------------------------------------------------------------------------------
nfs_v3_pdts               1      1      1      1      8      PDTs              M
--------------------------------------------------------------------------------
nfs_v2_vm_bufs            1000   1000   1000   512    5000   Bufs              I
--------------------------------------------------------------------------------
nfs_v3_vm_bufs            1000   1000   1000   512    5000   Bufs              I
--------------------------------------------------------------------------------
nfs_securenfs_authtimeout 0      0      0      0      60     Seconds           D
--------------------------------------------------------------------------------
nfs_v3_server_readdirplus 1      1      1      0      1      On/Off            D
--------------------------------------------------------------------------------
lockd_debug_level         0      0      0      0      10     Level             D
--------------------------------------------------------------------------------
statd_debug_level         0      0      0      0      10     Level             D
--------------------------------------------------------------------------------
statd_max_threads         50     50     50     1      1000   Threads           D
--------------------------------------------------------------------------------
utf8_validation           1      1      1      0      1      On/Off            D
--------------------------------------------------------------------------------
nfs_v4_pdts               1      1      1      1      8      PDTs              M
--------------------------------------------------------------------------------
nfs_v4_vm_bufs            1000   1000   1000   512    5000   Bufs              I
--------------------------------------------------------------------------------
```

Значение n/a указывает на то, что параметр не поддерживается текущей платформой или ядром

Ниже перечислены возможные типы параметров:
- S = Статический: не может быть изменен
- D = Динамический: может изменяться произвольно
- B = Загрузка: для изменения требуется выполнить команду bosboot или перезагрузить систему
- R = Перезагрузка: изменения вступают в силу после перезагрузки
- C = Соединение: изменения вступают в силу только для новых сокетов

```
    M = Монтирование: изменения вступают в силу при монтировании
    I = Дополняющий: допустимо только приращение

Ниже перечислены условные обозначения:
    K = Кило: 2^10       G = Гига: 2^30       P = Пета: 2^50
    M = Мега: 2^20       T = Тера: 2^40       E = Экса: 2^60
```

Для просмотра или изменения одного параметра введите команду **nfso -o**. Например:

```
# nfso -o portcheck
portcheck= 0
# nfso -o portcheck=1
```

Значение параметра по умолчанию можно восстановить с помощью опции **-d**. Например:

```
# nfso -d portcheck
# nfso -o portcheck
portcheck= 0
```

Руководство по настройке TCP/IP для повышения производительности NFS

NFS использует UDP или TCP для выполнения сетевого ввода-вывода.

Убедитесь, что вы применили приемы настройки, описанные в разделе “Настройка производительности TCP и UDP” на стр. 258 и “Настройка производительности пула mbuf” на стр. 292. В частности, выполните следующие действия:

- Проверьте системный протокол ошибок с помощью команды **errpt** и убедитесь в отсутствии записей о неполадках сетевого устройства и носителя.
- Установите максимальный размер очередей передачи и приема сетевого адаптера. Дополнительная информация приведена в разделе “Настройка ресурсов адаптера” на стр. 284.
- Проверьте наличие `Oerrs` с помощью команды **netstat -i**. Значительное число таких ошибок может указывать на недостаточный размер очереди передачи сетевого устройства.
- Убедитесь, что размеры буферов сокетов TCP и UDP настроены правильно. Изменить размеры буферов сокетов TCP (*tcp_sendspace* и **tcp_recvspace**), применяемых NFS, можно с помощью параметра *nfs_tcp_socketsize* команды **nfso**. Аналогичным образом параметр *nfs_udp_socketsize* управляет размерами буферов сокетов UDP (*udp_sendspace* и *udp_recvspace*), применяемых NFS. Задайте размер буферов сокетов в соответствии с рекомендациями, приведенным в разделе, посвященном настройке производительности TCP и UDP. Как и при обычной настройке протоколов TCP и UDP, значение параметра **sb_max** команды **no** должно превышать значения параметров *nfs_tcp_socketsize* и *nfs_udp_socketsize*. Как правило принятые в операционной системе AIX значения по умолчанию позволяют достичь удовлетворительных результатов, но рекомендуется выполнить дополнительную проверку. Для проверки переполнения буфера UDP выполните команду **netstat –s –p udp** и обратите внимание на число отброшенных пакетов, указанное в поле `переполнение буферов сокетов`.
- Убедитесь, что в системе настроен достаточный объем оперативной памяти. Проверьте наличие отклоненных или задержанных запросов на получение буферов mbuf с помощью команды **netstat –m**. Возможно, требуется увеличение числа буферов mbuf. Дополнительная информация об устранении неполадок, связанных с буферами mbuf, приведена в разделе “Настройка производительности пула mbuf” на стр. 292.
- Проверьте маршрутизацию. С помощью команды **traceroute** проверьте маршрут пакета.
- При работе в локальной сети рекомендуется по возможности увеличить размер максимального блока передачи (MTU). Например, в сети Gigabit Ethernet с производительностью увеличение размера MTU с 1500 байт (по умолчанию) до 9000 байт (большие пакеты) позволяет передавать запросы NFS на чтение и запись размером 8 КБ без фрагментации. Увеличение этого параметра также повышает эффективность использования памяти mbuf, снижая вероятность переполнения.
- Проверьте соответствие MTU. Запустите команду **netstat -i** и найдите значения MTU на клиенте и на сервере. Если они различны, попробуйте приравнять их и проверьте производительность. Учтите, что при передаче по "медленным" и глобальным сетям, в состав которых входят маршрутизаторы или мосты, может происходить дополнительная фрагментация пакетов. Одно из возможных решений заключается в

определении минимального размера MTU для маршрута от клиента к серверу и присвоении меньшего значения параметрам **rsize** и **wsize** при монтировании файловой системы NFS.

- В случае применения NFS версии 3 с протоколом TCP, для которого размер RPC составляет 32 КБ и более, следует указать опцию *nfs_rfc1323* в команде **nfso**. Размер окон TCP при этом может превышать 64 КБ, что позволяет сократить время ожидания подтверждений TCP. Данную опцию следует указать в обоих конечных системах соединения TCP, например, на сервере, и на клиенте NFS.
- Проверьте, не связаны ли неполадки с недостаточным интервалом между пакетами. Изредка неполадки объясняются слишком коротким интервалом между пакетами. Если между клиентом и сервером находится маршрутизатор или другое устройство, ознакомьтесь с документацией к этому устройству и выясните, нельзя ли изменить интервал между пакетами. Если это возможно, попробуйте увеличить интервал.
- Проверьте соответствие быстродействия различных сетей. Маршрутизатор может терять пакеты при передаче их из "быстрой" сети в "медленную". Например, такая ситуация наблюдается при получении маршрутизатором пакетов от сервера, находящегося в сети Gigabit Ethernet, и отправке их клиенту в сети Ethernet 100 Мбит/с. Маршрутизатор может не успевать передавать пакеты через Ethernet 100 Мбит/с с той же скоростью, с какой он получал их по Gigabit Ethernet. Вместо замены маршрутизатора можно снизить скорость отправки запросов клиентов и/или уменьшить размер считываемых или записываемых блоков данных.
- Максимальное число соединений TCP, принимаемых сервером, контролируется новой опцией *nfs_max_connections*. Значение 0 (по умолчанию) означает, что число соединений не ограничено. Обычно клиент закрывает соединение TCP после 5 минут простоя и вновь открывает его при необходимости. Сервер закрывает соединение после 6 минут простоя.
- Операционная система позволяет отключить проверку контрольной суммы UDP только для NFS. Для этого предусмотрена опция *udpchecksum* команды **nfso**. Значение по умолчанию равно 1, т.е. проверка контрольной суммы выполняется. Отключение этой опции приводит к незначительному выигрышу в производительности и к росту вероятности повреждения данных.

Отброшенные пакеты

Если клиент NFS сообщает об отброшенных пакетах, возникает сложная задача по поиску сетевого компонента на котором теряются пакеты. Пакеты могут отбрасываться клиентом, сервером или любым промежуточным узлом сети.

Пакеты, отброшенные клиентом:

Пакеты редко отбрасываются клиентом.

Так как каждая нить **biod** может одновременно обрабатывать одну операцию NFS, то перед следующим вызовом RPC она должна дождаться ответа от вызова RPC текущей операции. Такой механизм с автоматическим подтверждением значительно снижает вероятность перегрузки ресурсов системы. Как правило наиболее напряженным режимом работы является чтение, когда в систему может быть направлен большой поток данных. Хотя объем данных может быть большим, число одновременных вызовов RPC обычно невелико, причем каждая нить **biod** заранее выделяет память для ответа. Следовательно, вероятность отбрасывания пакетов клиентом очень мала.

Обычно пакеты отбрасываются сервером или промежуточными узлами сети.

Пакеты, отброшенные сервером:

Перегруженные серверы в некоторых местах могут терять пакеты.

1. Драйвер сетевого адаптера

   Если сервер NFS отвечает на очень большое число запросов, может произойти переполнение очереди вывода драйвера интерфейса. Такие ситуации можно обнаружить по статистике, выдаваемой командой **netstat -i**. Обращайте внимание на любые ненулевые значения в столбцах `Oerrs`. Каждое значение в столбце `Oerrs` указывает на один отброшенный пакет. Для устранения таких ситуаций достаточно

увеличить очередь вывода драйвера. Слишком длинные очереди передачи увеличивают задержку при обработке сообщений. Однако, поскольку NFS поддерживает уникальный XID для каждого вызова, повторный вызов может быть удовлетворен ответом на исходный запрос. Кроме того, задержки, связанные с нахождением в очереди, существенно меньше задержек, связанных с повторной передачей запроса.

2. Буферы сокетов

   Пакеты могут отбрасываться сервером и в буфере сокета UDP. Отброшенные пакеты подсчитываются на уровне UDP, соответствующая статистика выдается командой **netstat -p udp**. Проверьте статистику `переполнения буферов сокетов`.

   Как правило пакеты NFS теряются в буфере сокета только при большом объеме операций записи, проходящих через NFS. Сокет сервера NFS подключен к порту 2049, и все входящие данные буферизуются этим портом UDP. Размер буфера по умолчанию составляет 60 000 байт. Если это число разделить на размер записываемого пакета NFS версии 3 по умолчанию (32786), то окажется, что для переполнения буфера достаточно 19 одновременных запросов на запись.

   В некоторых случаях пакеты не отбрасываются из-за переполнения буфера сокета или очереди драйвера `0errs`, однако на клиенте все равно возникают тайм-ауты и передача пакетов повторяется. Здесь также возможны два варианта. Если сервер перегружен, то это может повлиять на работу демонов **nfsd** и привести к тому, что время ответа будет превышать значение тайм-аута. Другая распространенная причина, наиболее вероятная при низкой загруженности сервера, - потеря пакетов в сети.

Пакеты, отброшенные в сети:

Если сервер слабо загружен, и его буфер сокета и очередь драйвера не переполняются, но клиент часто выполняет повторные передачи, то пакеты могут отбрасываться во время передачи по сети.

В этом случае *сеть* - это структура, состоящая из физической среды передачи данных, сетевых устройств (таких как маршрутизаторы, мосты и концентраторы) и прочего аппаратного и программного обеспечения, реализующего передачу пакетов между клиентом и сервером.

Низкая производительность NFS в то время, когда сервер не перегружен и не отбрасывает пакеты, указывает на потери пакетов в сети. Подтверждение этого предположения и обнаружение причин неполадок в сети - сложная задача. Решать ее рекомендуется в зависимости от физической удаленности клиента от сервера и объема доступных ресурсов.

Иногда клиент и сервер находятся достаточно близко, чтобы между ними можно было установить прямое соединение. Если это устраняет все неполадки, то, очевидно, причина неполадок кроется в самой сети. Однако такой эксперимент возможен не всегда. Для обнаружения причины неполадки можно воспользоваться сетевым анализатором пакетов или другим инструментом.

Настройка дисковой подсистемы для NFS

Одной из наиболее распространенных причин появления узких мест, проявляющихся при интенсивном чтении и записи данных, является неправильная настройка дисковых подсистем.

Если вы планируете настройку только дисковой подсистемы сервера NFS, то учтите, что в некоторых случаях неполадка может быть связана с неправильной настройкой дисковой подсистемы клиента NFS. Примером такой ситуации может служить копирование файла приложением клиента NFS из файловой системы NFS в локальную файловую систему клиента. В этом случае дисковая подсистема клиента должна быть настроена таким образом, чтобы производительность не была ограничена скоростью записи в локальную файловую систему. См. инструкции по настройке, описанные в разделе “Производительность логических томов и дискового ввода-вывода” на стр. 178. В частности, обратите внимание на следующие особенности:

- Для простых операций чтения и записи NFS оцените производительность дисков, на которых размещаются применяемые файловые системы. Для этого выполните локальное чтение данных из файла или запись в файл. Для измерения пропускной способности дисков следует применять команду **iostat**, так как тестовые приложения как правило не записывают на диск все данные. Например, данные могут

остаться в памяти. Результаты локальных измерений представляют собой максимальную возможную производительность NFS, так как в этом случае не выполняется дополнительная обработка и отсутствуют задержки, связанные с выполнением дополнительных операций в NFS.

- Часто требуется обеспечить параллелизм доступа к данным. Одновременный доступ к одной файловой системе нескольких клиентов или нескольких процессов одного клиента может привести к перегрузке дискового устройства сервера. Определить степень загруженности дисков можно с помощью команды **iostat**. В частности, параметр *%tm_act* указывает на время активности конкретного диска в процентах, однако высокое значение может означать перегрузку адаптера, связанного с этим диском.
- Следует отметить, хотя это и не связано напрямую с настройкой дисковых подсистем, что параллельная запись в один файл может привести к конфликту блокировки inode этого файла. Большинство файловых систем применяют блокировку inode для обеспечения последовательного доступа к файлу и согласования записываемых в него данных. К сожалению, это может привести к значительному снижению скорости записи, если одновременно несколько нитей пытаются записать данные в один и тот же файл, так как запись разрешена только одной нити, обладающей блокировкой inode.
- Общая стратегия для больших серверов NFS состоит в распределении нагрузки дискового ввода-вывода по максимально возможному числу дисков и контроллеров. В системе с оптимальным распределением нагрузки дискового ввода-вывода производительность в отдельные моменты времени может быть ограничена лишь быстродействием процессоров.

Влияние неправильного применения NFS на производительность

Многие случаи неправильного применения NFS связаны с непониманием того факта, что доступ к файлам этой файловой системы осуществляется по дорогостоящему и медленному сетевому соединению.

Ниже приведено несколько примеров:

- Приложение, обслуживающее кассовый аппарат в режиме реального времени, в случайные моменты времени обновляет учетный файл склада, расположенный в файловой системе NFS.
- Среда разработки, в которой каталоги с исходным кодом в каждой системе были смонтированы в других системах, чтобы любой программист мог работать на любом компьютере в сети. В этом случае практически во всех операциях компиляции исходный код будет считываться из удаленной системы, а результат записываться в удаленную систему.
- Преобразование файлов `.o` из смонтированного каталога NFS в файл `a.out` в этом же каталоге командой **ld**.
- Запросы на запись не выровнены по границе страницы, например, если обмен ведется блоками по 10 КБ. Любая запись данных размером меньше 4 КБ приводит к выполнению *подкачки*, а в случае NFS *подкачка* выполняется по сети.

Возможны разные мнения по поводу того, насколько правильным является подобное использование возможностей NFS. Однако не вызывает сомнений, что во всех этих примерах за удобство использования пришлось заплатить избыточной загрузкой процессоров и сети, т.е. снижением общей производительности. Если во время обычной работы системы приложения обращаются к NFS, то при планировании конфигурации нужно учесть связанные с этим издержки и постараться компенсировать их некоторыми техническими приемами, например:

- Размещение всех данных или исходного кода на сервере, а не на отдельных рабочих станциях позволяет повысить степень контроля за разработкой и упрощает централизованное резервное копирование.
- Если несколько систем работают с одним набором данных, будет разумнее создать выделенный сервер, чем настроить систему, сочетающую функции клиента и сервера.

Другой пример приложения, которое не должно работать с файлами NFS, - это приложение, выполняющее сотни вызовов **lockf()** и **flock()** в секунду. В файловой системе NFS все вызовы **lockf()** и **flock()** (а также любые другие запросы на блокировку файлов) обрабатываются демоном **rpc.lockd**. Это существенно снижает производительность системы, поскольку демон блокировки не в состоянии обрабатывать такое количество запросов.

Независимо от производительности клиента и сервера, все операции через NFS будут выполняться медленнее обычных. Это происходит по нескольким причинам, но все они связаны с тем, что при чтении и записи в блокированный файл принимаются специальные меры для обеспечения синхронности. Это означает, что не кэшируются никакие данные файла, включая его атрибуты. Все операции с файлом выполняются синхронно без кэширования. Приложение, блокирующее удаленные файлы, обычно работает гораздо медленнее, чем другие приложения системы, которые обращаются к тому же серверу NFS.

Отслеживание производительности NFS на сервере

Для того чтобы проверить правильность настройки процессора, средств связи и оперативной памяти сервера NFS следует проконтролировать уровень использования процессора, средств связи и памяти с помощью команд **vmstat** и **iostat** для обычной рабочей схемы.

Команда **nfsstat** позволяет отслеживать операции NFS сервера.

Команда nfsstat -s

В разделе с информацией о сервере NFS указано число полученных (`calls`) и отклоненных в процессе идентификации (`badcalls`) вызовов NFS, а также число вызовов различных типов и доля, которую они составляют от общего числа вызовов.

Ниже приведен пример раздела Сервер вывода команды **nfsstat**, вызванной с опцией **-s**:

```
# nfsstat -s

Сервер - RPC:
С установлением соединения:
calls     badcalls  nullrecv badlen    xdrcall  dupchecks dupreqs
15835      0          0          0          0          772        0
Без установления соединения:
calls     badcalls  nullrecv badlen    xdrcall  dupchecks dupreqs
0          0          0          0          0          0          0

Сервер - NFS:
calls      badcalls   public_v2  public_v3
15835      0          0          0
Версия 2: (0 вызовов)
null       getattr    setattr    root       lookup     readlink   read
0 0%       0 0%       0 0%       0 0%       0 0%       0 0%       0 0%
wrcache    write      create     remove     rename     link       symlink
0 0%       0 0%       0 0%       0 0%       0 0%       0 0%       0 0%
mkdir      rmdir      readdir    statfs
0 0%       0 0%       0 0%       0 0%
Версия 3: (15835 вызовов)
null       getattr    setattr    lookup     access     readlink   read
7 0%       3033 19%   55 0%      1008 6%    1542 9%    20 0%      9000 56%
write      create     mkdir      symlink    mknod      remove     rmdir
175 1%     185 1%     0 0%       0 0%       0 0%       120 0%     0 0%
rename     link       readdir    readdir+   fsstat     fsinfo     pathconf
87 0%      0 0%       1 0%       150 0%     348 2%     7 0%       0 0%
commit
97 0%
```

Ниже приведено описание полей из раздела Сервер - RPC (**-s**):

вызов Общее число вызовов RPC, полученных от клиентов

badcalls
Общее число вызовов, отклоненных на уровне RPC

nullrecv
Число случаев, когда вызов RPC был недоступен, в то время как считалось, что он был получен

badlen Число усеченных и поврежденных пакетов (число вызовов RPC, длина которых была меньше минимально допустимой)

xdrcall Число вызовов RPC, заголовки которых не были закодированы в формате Внешнего представления данных (XDR)

dupchecks
Число вызовов RPC, обнаруженных в кэше повторных запросов

dupreqs
Число повторных вызовов RPC

Кроме того, в выводе команды указывается количество вызовов различных типов и их доля в процентах от общего числа вызовов.

Проверка повторных вызовов выполняется для операций, выполнение которых может давать различный результат. Классический пример - команда **rm** (удалить). Первая команда **rm** будет выполнена успешно, но, если ее ответ будет потерян, клиент может повторить вызов команды. Этот вызов будет обнаружен в кэше повторных запросов, поэтому вместо еще одной попытки удалить файл будет сразу возвращен ответ об успешном выполнении.

Информация о доле операций различных типов, таких как **getattr()**, **read()**, **write()** и **readdir()**), позволяет выбрать способ оптимизации. Например, если велика доля вызовов **getattr()**, то рекомендуется увеличить кэш атрибутов. При большом числе вызовов **write()** важна производительность диска и LVM. Обработка вызовов **read()** может быть ускорена путем увеличения объема оперативной памяти, применяемой для кэширования файлов.

Настройка производительности NFS на сервере

Настраивать параметры NFS на сервере можно с помощью команды **nfso**.

Как правило в случае правильного применения параметры настройки NFS позволяют получить следующие результаты:

- Снизить нагрузку на сеть и сервер NFS
- Устранить некоторые неполадки, связанные с сетью и использованием памяти клиента

Необходимое число нитей nfsd

На сервере NFS выполняется один многопоточный демон **nfsd**. Это значит, что в рамках процесса **nfsd** работает несколько нитей ядра. Число нитей настраивается автоматически в соответствии с нагрузкой на NFS.

Так как число нитей настраивается автоматически и число нитей **nfsd** по умолчанию (3891) является максимальным числом, то изменение этого значения требуется только в исключительных случаях. Однако вы можете ограничить максимальное количество нитей **nfsd** в системе с помощью параметра *nfs_max_threads* команды **nfso**.

Ограничения размера пакетов чтения и записи на сервере

Параметры **nfs_max_read_size** и **nfs_max_write_size** команды **nfso** позволяют управлять максимальным размером пакетов RPC, применяемых в операциях чтения и записи NFS.

В разделе “Настройка NFS на клиенте” на стр. 347 описаны ситуации, в которых может потребоваться изменение размера пакетов чтения и записи RPC. Как правило настройка этих параметров выполняется на клиенте. На сервере эти параметры команды **nfso** рекомендуется применять только в том случае, если изменение соответствующих значений на клиенте связано с какими-либо трудностями.

Настройка максимального кэширования данных файлов

В NFS не предусмотрены встроенные буферы для кэширования данных из файлов файловых систем, экспортированных в NFS.

Для управления кэшированием этих файловых страниц применяется Администратор виртуальной памяти (VMM). Если система выполняет функции выделенного сервера NFS, то для увеличения производительности можно настроить VMM таким образом, чтобы для кэширования данных использовалась вся свободная память. Если сервер экспортирует файловые системы JFS, то для этого параметру *maxperm*, отвечающему за объем памяти, выделяемой под файловые страницы JFS, нужно присвоить значение "100%". Этот параметр задается с помощью команды **vmo**. Например:

```
# vmo -o maxperm%=100
```

Если сервер экспортирует расширенные файловые системы JFS, то следует задать как параметр *maxclient* , так и параметр *maxperm* . Параметр *maxclient* задает долю оперативной памяти, выделяемую под страницы сегментов клиентов, то есть под кэш расширенной файловой системы JFS. Обратите внимание, что значение *maxclient* не может быть больше значения *maxperm* . Например:

```
# vmo -o maxclient%=100
```

В некоторых случаях чрезмерный объем данных файлов, записанных в кэш, может быть нежелательным. В разделе “Производительность файловой системы” на стр. 235 приведено описание механизма *быстрого освобождения*, позволяющего удалять из кэша данные файлов, вероятность повторного применения которых мала.

Настройка демона монтирования RPC

Демон **rpc.mountd** выполняется в нескольких нитях. По умолчанию он может использовать до 16 нитей.

Если демон **automount** используется достаточно интенсивно и в его работе наблюдаются частые тайм-ауты, рекомендуется увеличить число нитей **rpc.mountd**. Для этого выполните следующую команду:

```
# chsys -s rpc.mountd -a -h <число-нитей>
# stopsrc -s rpc.mountd
# startsrc -s rpc.mountd
```

Настройка демона блокировки RPC

Демон **rpc.lockd** выполняется в нескольких нитях. По умолчанию он может использовать до 33 нитей.

Если при работе RPC часто возникают блокировки файлов, то при достижении максимального ограничения на количество нитей демон **rpc.lockd** может стать фактором, ограничивающим производительность. При достижении максимального числа нитей все последующие запросы будут обрабатываться только после завершения предыдущих, что может привести к тайм-аутам на других стадиях процесса. Для **rpc.lockd** можно настроить до 511 нитей. Пример приведен ниже:

```
# chsys -s rpc.lockd -a <число-нитей>
# stopsrc -s rpc.lockd
# startsrc -s rpc.lockd
```

Отслеживание производительности NFS на клиенте

Для проверки правильности настройки процессора и оперативной памяти клиента NFS следует проверить использование CPU и памяти с помощью команды **vmstat** для обычной рабочей схемы.

Отслеживать операции NFS клиента можно с помощью команды **nfsstat**.

Команда nfsstat -c

В разделе с информацией о клиенте NFS указано число отправленных и отклоненных вызовов, число полученных описателей клиента (`clgets`), а также количество вызовов различных типов и их доля в процентах от общего числа вызовов.

Ниже приведен пример такого раздела в выводе команды **nfsstat**, вызванной с опцией **-c**:

```
# nfsstat -c

Клиент - RPC:
С установлением соединения:
```

```
calls     badcalls  badxids  timeouts  newcreds badverfs  timers
0          0          0          0          0          0          0
nomem     cantconn interrupts
0          0          0
Без установления соединения:
calls     badcalls retrans   badxids   timeouts newcreds  badverfs
6553       0          0          0          0          0          0
timers    nomem     cantsend
0          0          0


Клиент - NFS:
calls      badcalls   clgets     cltoomany
6541       0          0          0
Версия 2: (6541 вызовов)
null       getattr    setattr    root       lookup     readlink   read
0 0%       590 9%     414 6%     0 0%       2308 35%   0 0%       0 0%
wrcache    write      create     remove     rename     link       symlink
0 0%       2482 37%   276 4%     277 4%     147 2%     0 0%       0 0%
mkdir      rmdir      readdir    statfs
6 0%       6 0%       30 0%      5 0%
Версия 3: (0 вызовов)
null       getattr    setattr    lookup     access     readlink   read
0 0%       0 0%       0 0%       0 0%       0 0%       0 0%       0 0%
write      create     mkdir      symlink    mknod      remove     rmdir
0 0%       0 0%       0 0%       0 0%       0 0%       0 0%       0 0%
rename     link       readdir    readdir+   fsstat     fsinfo     pathconf
0 0%       0 0%       0 0%       0 0%       0 0%       0 0%       0 0%
commit
0 0%
```

Ниже приведено описание полей из раздела Клиент - RPC (**-c**):

вызов Общее число вызовов RPC, обращенных к NFS

badcalls
Общее число вызовов, отклоненных на уровне RPC

retrans
Число повторных передач вызова из-за тайм-аута при ожидании ответа от сервера. Этот показатель выдается только для вызовов RPC, передаваемых без установления соединения.

badxid Число полученных ответов от сервера, не соответствовавших ни одному из отправленных вызовов. Такой ответ означает, что операция выполнялась слишком долго.

timeouts
Число тайм-аутов ожидания ответа от сервера.

newcreds
Число обновлений идентификационной информации.

badverfs
Число вызовов, ответы на которые содержали неверный контрольный код.

timers Число случаев, когда вычисленный тайм-аут был не меньше указанного минимального тайм-аута вызова.

nomem
Число вызовов, которые не были выполнены из-за нехватки памяти.

cantconn
Число вызовов, для обработки которых не удалось установить соединение с сервером.

interrupts
Число вызовов, обработка которых была прервана по сигналу.

cantsend
Число попыток отправки, оказавшихся неудачными из-за того, что не удалось установить соединение с клиентом.

Кроме того, в выводе команды указывается количество вызовов различных типов и их доля в процентах от общего числа вызовов.

В выводе команды **nfsstat -c** указывается и число отброшенных пакетов UDP. Эта информация может применяться для анализа производительности. Пакет отбрасывается в том случае, если его не удается обработать. Кроме того, пакеты могут отбрасываться в случае слишком большого времени ответа программного или аппаратного обеспечения, либо высокой загруженности процессора сервера. Запросы из отброшенных пакетов не пропадают, а передаются повторно.

В столбце *retrans* раздела RPC указано число повторных передач запросов из-за тайм-аута ожидания ответа. Этот показатель связан с числом отброшенных пакетов UDP. Если значение *retrans* регулярно превышает пять процентов от общего числа вызовов (указанного в первом столбце), значит сервер или сеть не справляются с нагрузкой. Для выяснения степени загруженности сервера вызовите на нем команды **vmstat** и **iostat**.

Большое значение показателя *badxid* указывает на то, что запросы доходят до серверов NFS, но серверы слишком загружены и не успевают отправлять ответы до наступления тайм-аута, вследствие чего запросы передаются повторно. Значение *badxid* увеличивается при получении очередного повторного ответа на переданный запрос. *XID* запроса RPC остается неизменным на всех этапах обработки. Повторные передачи ответов повышают нагрузку на сервер, что еще больше увеличивает время ответа. Если значения *badxid* и числа тайм-аутов превышают пять процентов от общего числа запросов, то увеличьте значение параметра *timeo* в опциях монтирования NFS с помощью команды **smitty chnfsmnt**. Если значение *badxid* равно 0, а значения *retrans* и числа тайм-аутов велики, попробуйте уменьшить размер буфера NFS с помощью опций **rsize** и **wsize** команды **mount**.

Если число повторных передач близко к числу тайм-аутов, то это означает, что пакеты часто отбрасываются. Дополнительная информация приведена в разделе “Отброшенные пакеты” на стр. 339.

В некоторых случаях производительность NFS кажется низкой, в то время как вывод команды **nfsstat -c** указывает, что число тайм-аутов и повторных передач запроса невелико. Это означает, что клиент получает ответ от сервера сразу после отправки запроса. В этом случае следует выяснить, сколько экземпляров демона **biod** запущено на клиенте. Кроме того, такая ситуация может возникать при блокировке удаленных файлов. После того как клиент блокирует удаленный файл, для работы с которым применяется NFS, он переходит в синхронный режим работы. В этом режиме данные и атрибуты файла не кэшируются. Это обычное поведение клиента в таких случаях, однако оно приводит к значительному снижению производительности. Блокирующие пакеты указаны в выводе команды **ipreport** как запросы NLM.

Команда nfsstat -m

Команда **nfsstat -m** выдает для каждой точки монтирования клиента имя и адрес сервера, флаги монтирования, текущий размер блоков чтения и записи, число повторных передач. а также тайм-ауты для динамической повторной передачи.

Пример приведен ниже:

```
# nfsstat -m
/SAVE from /SAVE:aixhost.ibm.com
 Флаги:   vers=2,proto=udp,auth=unix,soft,intr,dynamic,rsize=8192,wsize=8192,retrans=5
 Поиск:   srtt=27 (67 мс), dev=17 (85 мс), cur=11 (220 мс)
 Чтение:  srtt=16 (40 мс), dev=7 (35 мс), cur=5 (100 мс)
 Запись:  srtt=42 (105 мс), dev=14 (70 мс), cur=12 (240 мс)
 Все:     srtt=27 (67 мс), dev=17 (85 мс), cur=11 (220 мс)
```

Числа в скобках, показанные в этом примере, представляют собой фактические интервалы в миллисекундах. Прочие числа - это значения времени в условных единицах, применяемые ядром операционной системы. Их

можно проигнорировать. Значения времени ответа показаны для операций поиска, чтения, записи, а также для сочетания всех этих операций (`Всего`). Прочие сокращения в этом выводе означают следующее:

srtt Усредненное время оборота пакета

dev Приближенное отклонение

cur Текущий тайм-аут

Настройка NFS на клиенте

Для настройки параметров NFS применяются команды **nfso** и **mount**.

Перед настройкой параметров убедитесь, что вы четко понимаете цель изменения их значений, а также возможные отрицательные побочные эффекты, которые могут быть вызваны этими изменениями.

Опции команды **mount** можно задать путем изменения файла настройки `/etc/filesystems` конкретной файловой системы. Изменения вступят в силу после монтирования этой файловой системы в процессе загрузки.

Как правило в случае правильного применения параметры настройки NFS позволяют получить следующие результаты:

- Снизить нагрузку на сеть и сервер NFS
- Устранить некоторые неполадки, связанные с сетью и использованием памяти клиента

Необходимое число нитей biod

На клиенте NFS выполняется один многопоточный демон **biod**. Это значит, что в рамках процесса **biod** работает несколько нитей ядра. Число нитей настраивается автоматически в соответствии с нагрузкой на NFS.

Вы можете ограничить максимальное число нитей **biod** при монтировании, указав опцию **biod**. По умолчанию число нитей **biod** составляет 4 для NFS версий 3 и 4, а для NFS версии 2 - 7.

Поскольку каждая нить **biod** может обрабатывать только один запрос на чтение или запись и поскольку время ответа NFS обычно составляет значительную долю от общего времени ответа, старайтесь, чтобы приложения не блокировались из-за нехватки нитей **biod**.

Оптимальное число демонов **nfsd** и **biod** определяется последовательно, в несколько шагов. В качестве начальной точки вы можете воспользоваться следующими рекомендациями. Ниже приведены общие рекомендации по настройке нитей **biod**:

- Увеличение числа нитей не может компенсировать недостаточную вычислительную мощность, объем памяти или пропускную способность дисковой подсистемы клиента или сервера. Перед изменением числа нитей необходимо узнать, насколько активно используются ресурсы сервера и клиента. Это можно сделать с помощью команд **iostat** и **vmstat**.
- Если процессор или дисковая подсистема компьютера уже перегружены, то увеличение числа нитей не приведет к повышению производительности.
- Нити **biod** обрабатывает только операции чтения и записи.
- Значения по умолчанию как правило позволяют получить удовлетворительные результаты. Однако, если несколько приложений одновременно обращаются к файлам точки монтирования, то число нитей **biod** для этой точки монтирования рекомендуется увеличить. Например, для каждого клиента оцените максимальное число файлов, которые будут записываться одновременно. Убедитесь, что для каждого файла доступно по крайней мере две нити **biod** для операций упреждающего чтения и отложенной записи.
- Если быстродействующие рабочие станции подключены к "медленному" серверу, то интенсивность создания запросов NFS клиентами можно ограничить. Для этого сократите количество нитей **biod** в клиентских системах, предварительно оценив загруженность и время отклика каждого клиента. Увеличение числа нитей **biod** на клиенте снижает производительность сервера, поскольку позволяет

клиенту одновременно отправлять большее число запросов, что увеличивает нагрузку на сеть и сервер. Если запросы клиента вызывают перегрузку сервера, то может потребоваться уменьшить число нитей **biod** до одной. Например:

```
# stopsrc -s biod
```

В результате выполнения этой команды на клиенте остается только один процесс ядра **biod**.

Настройка размера пакетов чтения и записи

Наиболее эффективными опциями настройки NFS являются опции **rsize** и **wsize**, определяющие максимальные размеры пакетов RPC для чтения и записи.

Ниже перечислены возможные причины изменения размера пакетов чтения и записи:

- Сервер не способен обрабатывать данные пакетами большого объема (8 КБ в NFS версии 2 и 32 КБ в NFS версий 3 и 4). Например, применение PC в качестве сервера NFS для клиента NFS. Как правило объем памяти PC недостаточен для работы с большими пакетами.
- Уменьшение пакета чтения-записи приводит к уменьшению числа фрагментов IP, создаваемых для каждого вызова. В ненадежной сети вероятность успешной передачи двух пакетов существенно выше, чем, например, семи. Аналогично, при отправке пакетов NFS через несколько сетей с различной производительностью фрагменты пакета могут не успеть достичь целевой системы до того, как возникнет тайм-аут.

Уменьшение значений **rsize** и **wsize** может повысить скорость работы NFS в перегруженных сетях за счет сокращения размера пакетов, отправляемых в каждой операции чтения или записи NFS. Побочный эффект состоит в увеличении количества пакетов и, как следствие, нагрузки на сеть и процессоры сервера и клиента.

Если работа с удаленной файловой системой NFS выполняется через сеть с высокой скоростью передачи данных, например, Gigabit Ethernet, то эффективность работы с этой файловой системой можно повысить, увеличив размер пакетов чтения и записи. Если в качестве сетевого протокола передачи данных применяется TCP, то максимальное значение опций **rsize** и **wsize** в NFS версий 3 и 4 составляет 65 536. Значение по умолчанию равно 32 768. В NFS версии 2 максимальные значения **rsize** и **wsize** составляли 8192 и применялись по умолчанию.

Настройка кэширования данных файлов NFS

VMM управляет кэшированием данных файлов NFS в страницах сегмента клиента NFS.

Если рабочая схема клиента NFS не предусматривает интенсивную работу со страницами рабочего сегмента, то рекомендуется предоставить VMM максимально возможный объем памяти для кэширования данных файлов NFS. Это можно сделать с помощью параметров *maxperm* и *maxclient*. Значение *maxclient* должно быть меньше значения *maxperm* или равно ему. Следующий пример позволяет выделить для кэширования файлов всю доступную память:

```
# vmo -o maxperm%=100
# vmo -o maxclient%=100
```

Влияние кэширования данных NFS на производительность чтения:

Производительность последовательного чтения NFS в клиентской системе можно повысить за счет применения алгоритмов упреждающего чтения VMM и кэширования.

Упреждающее чтение позволяет передать клиенту данные из файла сервера NFS до того, как эти данные были запрошены приложением клиента NFS. К тому времени, как приложение отправит запрос на данные, эти данные уже могут находиться в памяти клиентского компьютера и, следовательно, запрос будет выполнен немедленно. Кэширование VMM позволяет выполнять повторное чтение данных без задержки, так как данные остаются в памяти клиентского компьютера, вследствие чего их не нужно повторно получать с сервера NFS.

В то время как для многих приложений кэширование VMM данных NFS на клиенте предоставляет определенные преимущества, существует ряд приложений (например, базы данных), которые сами могут осуществлять управление кэшированием данных файлов. Для приложений, которые сами осуществляют управление кэшированием данных, преимуществом может являться применение прозрачного ввода-вывода (DIO) по NFS. Можно включить DIO по NFS с помощью опции **dio** команды **mount**, либо установив флаг **O_DIRECT** в системном вызове **open()**.

Далее описаны преимущества DIO:

- Предотвращается двойное кэширование данных файлов VMM и приложением.
- Возможно повышение производительности CPU при чтении из файла и записи в файл, поскольку функция DIO обходит код VMM.

Для приложений, которые сами осуществляют управление кэшированием данных и сериализацию доступа к файлам, преимуществом может являться применение параллельного ввода-вывода (CIO). В дополнение к преимуществам DIO, CIO не сериализует чтение и запись в файл, что допускает параллельное чтение или запись одного и того же файла несколькими нитями.

Примечание: Применение CIO или DIO может понизить производительность приложений, которые полностью полагаются на кэширование VMM и оптимизацию упреждающего чтения и отложенной записи VMM для повышения быстродействия системы.

При использовании клиента с небольшим объемом памяти, а также при работе с большими файлами или с медленными сетями для увеличения скорости чтения можно воспользоваться кэширующей файловой системой, которая позволяет сохранять ответы на запросы клиентов в кэше на локальном диске. Дополнительная информация приведена в разделе “Кэширующая файловая система” на стр. 352.

Кэширование данных при последовательных операциях чтения больших файлов может привести к затруднению замены страниц, поскольку память оказывается занятой данными кэша NFS. Можно повысить производительность, избегая операции замены страниц с помощью быстрого освобождения при чтении (**rbr**), опции **mount** или аргумента **nfs4cl setfsoptions** для NFS версии 4. При последовательных операциях чтения больших страниц физическая память освобождается после завершения каждой из таких последовательных операций чтения.

Если опция **rbr mount** начала освобождение памяти, которая скоро понадобится опять, можно вместо нее использовать переменную **nfs_auto_rbr_trigger** команды **nfso**. Переменная **nfs_auto_rbr_trigger**, измеряемая в мегабайтах, служит пороговым значением для включения опции быстрого освобождения при чтении. Например, если для переменной **nfs_auto_rbr_trigger** задано значение 100 МБ, то при последовательном чтении файла первые 100 МБ кэшируются, а оставшаяся часть файла освобождается из памяти.

Влияние кэширования данных NFS на производительность записи:

С помощью режима отложенной фиксации в NFS версий 3 и 4 можно повысить производительность последовательных операций записи в файлы, размер которых превышает объем памяти клиентов.

Запись файлов, размер которых превышает объем памяти клиента, приводит к интенсивной замене страниц. При этом для каждой страницы записываемых данных может выполняться операция фиксации по сети. Режим отложенной фиксации использует усовершенствованный алгоритм фиксации страниц клиента на дисках сервера и, что более важно, освобождения страниц пространства подкачки.

Режим отложенной фиксации можно включить при монтировании файловой системы - для этого нужно указать в команде **mount** опцию **combehind**. Помимо этого, вам потребуется указать особое значение переменной *numclust* в команде **mount**. Эта переменная задает количество кластеров размером по 16 КБ, обрабатываемых за одну операцию алгоритмом отложенной записи администратора виртуальной памяти (VMM). При последовательном вводе-выводе чем больше значение опции **numclust**, тем больше страниц

будет накапливаться в оперативной памяти для последующей однократной операции ввода-вывода. Рекомендуется также увеличивать значение **numclust** при работе с томами и массивами, в которых применяется чередование данных.

Настройка кэша файловых атрибутов NFS

В каждой клиентской системе NFS поддерживает кэш для атрибутов файлов и каталогов, использовавшихся недавно.

Команда **mount** позволяет задать параметры, управляющие временем хранения записей в кэше. Эти параметры описаны ниже:

actimeo
: Абсолютное время хранения атрибутов файлов и каталогов в кэше после обновления. Если это значение указано, оно переопределяет описанные ниже параметры **min* и **max*, устанавливая для них значение **actimeo**.

acregmin
: Минимальное время хранения записи о файле в кэше после обновления. Значение по умолчанию - 3 секунды.

acregmax
: Максимальное время хранения записи о файле в кэше после обновления. Значение по умолчанию - 60 секунд.

acdirmin
: Минимальное время хранения записи о каталоге в кэше после обновления. Значение по умолчанию - 30 секунд.

acdirmax
: Максимальное время хранения записи о каталоге в кэше после обновления. Значение по умолчанию - 60 секунд.

После каждого обновления файла или каталога удаление информации о нем из кэша откладывается минимум на **acregmin** или **acdirmin** секунд. В случае повторного обновления удаление информации откладывается на интервал между двумя обновлениями, но не более чем на **acregmax** или **acdirmax** секунд.

Влияние слабого и сильного монтирования NFS на производительность

При монтировании каталогов NFS можно указать опцию сильного (**-o hard**) или слабого (**-o soft**) монтирования.

Если после успешного слабого монтирования каталога в работе возникает ошибка (обычно тайм-аут), сообщение о ней немедленно передается вызывающей программе. Если ошибка возникает при доступе к сильно смонтированному каталогу, NFS повторяет операцию.

Постоянная ошибка при обращении к сильно смонтированному каталогу может существенно снизить производительность, поскольку большое число попыток (1000) в сочетании со значением тайм-аута по умолчанию (0,7 секунды) и с тем фактом, что значение тайм-аута увеличивается с каждой очередной попыткой, приводят к тому, что NFS слишком долго пытается выполнить операцию.

Технически возможно снизить число повторов или увеличить значение тайм-аута с помощью опций команды **mount**. Однако значительное изменение этих параметров может привести к выдаче избыточных сообщений об ошибках сильной связи. Вместо этого рекомендуется при монтировании каталогов указать опцию **intr**, позволяющую прервать цикл повторов с клавиатуры.

Хотя слабое монтирование каталогов позволяет обнаружить ошибку раньше, при нем велик риск повреждения данных. В общем случае, каталоги, к которым будет разрешен доступ на чтение и запись, должны быть сильно смонтированы.

Излишняя передача пакетов

С выбором между слабым и сильным монтированием тесно связан вопрос об оптимальном значении тайм-аута для заданной конфигурации сети.

Если сервер сильно загружен, соединен с клиентом через несколько мостов или шлюзов или работает по глобальной сети, значение тайм-аута по умолчанию может оказаться неподходящим. В этом случае клиент и сервер будут тратить свои ресурсы на ненужные повторы передачи. Например, если в выводе следующей команды:

```
# nfsstat -c
```

в полях `timeouts` и `badxids` указаны значения, превышающие пять процентов от общего числа запросов, то рекомендуется увеличить значение параметра *timeo* в команде **mount**.

Определите каталог для изменения и в соответствующей строке **Тайм-аут NFS** введите новое значение.

Значение по умолчанию - 0,7 секунды (**timeo=7**) но это значение изменяется расширением ядра NFS в зависимости от типа вызова. Для чтения, например, это значение удваивается, так что тайм-аут чтения по умолчанию равен 1,4 секунды.

Для управления значением *timeo* в клиентах с операционной системой версии 4 необходимо задать опцию **nfs_dynamic_retrans** команды **nfso** равной нулю. Выбор между увеличением или уменьшением *timeo* зависит от конкретного случая. Измените тайм-аут с учетом причины, по которой пакеты не успевают доставляться за указанное время.

Если пакеты приходят без потерь, но с задержками, то для устранения излишних повторных передач значение *timeo* нужно увеличить.

Если пакеты отбрасываются и никогда не достигают клиентской системы, то ожидание ответов - это пустая трата времени, поэтому значение *timeo* нужно уменьшить.

Для выбора оптимального значения запустите команду **nfsstat -cr** в системе клиента и обратите внимание на значение в столбце `badxid`. Значение `badxid` указывает число полученных клиентом ответов RPC, не соответствующих никакому активному запросу. Обычно это означает, что в результате повторной передачи запроса клиент получил два одинаковых ответа. В этом случае значение *timeo* должно быть увеличено.

Кроме того, определить причины снижения производительности можно с помощью анализатора сети. Если анализатор сети недоступен, выберите оптимальное значение *timeo* опытным путем. В некоторых случаях определить зависимость между этим значением и производительностью не удается. В этом случае рекомендуется определить причину задержки или потери пакетов и устранить неполадку.

В случае межсетевой передачи через мост попробуйте установить значение тайм-аута равное 50 (измеряется в десятых долях секунды). В случае глобальной сети укажите значение 200. Через некоторое время (но не ранее чем через день) просмотрите статистические данные NFS. Если статистика вновь покажет существенный объем излишних повторных передач, увеличьте значение *timeo* в полтора раза и повторите проверку. При этом обратите внимание на загруженность серверов, мостов и маршрутизаторов - ни один элемент сети не должен быть перегружен.

Неиспользуемая поддержка NFS ACL

Если ваши программы не используют Списки управления доступом (ACL) в смонтированной файловой системе, то вы можете снизить нагрузку на клиент и сервер, указав опцию **noacl**.

Для этого введите следующую команду:

```
options=noacl
```

Укажите эту опцию в разделе файла `/etc/filesystems`, отвечающем за создание соответствующей файловой системы.

Использование операций READDIRPLUS

В NFS версии 3 операция READDIRPLUS вместе с записями каталогов возвращает описатели и атрибуты файлов. Такой подход значительно эффективней применяемого в NFS версии 2, так как клиенту не требуется запрашивать эту информацию на сервере отдельно для каждой записи.

Однако в некоторых случаях, когда из большого числа записей каталога клиент работает лишь с небольшим подмножеством, выполнение операции READDIRPLUS NFS версии 3 может привести к снижению производительности. В таких случаях поддержку READDIRPLUS можно отключить с помощью опции **nfs_v3_server_readdirplus** команды **nfso**. Однако, использование этой возможности не рекомендуется, поскольку это противоречит стандарту NFS версии 3.

Кэширующая файловая система

Для повышения производительности удаленных файловых систем, таких как NFS, и медленных устройств, таких как дисководы CD-ROM, может применяться кэширующая файловая система (CacheFS).

В случае ее применения данные, считанные из удаленной файловой системы или с компакт-диска, сохраняются в кэше локальной системы, что обеспечивает немедленный доступ к ним при втором обращении. Алгоритм работы CacheFS заключается в отображении исходной файловой системы (например, NFS) на некоторую целевую (локальную) файловую систему, как показано на следующем рисунке:

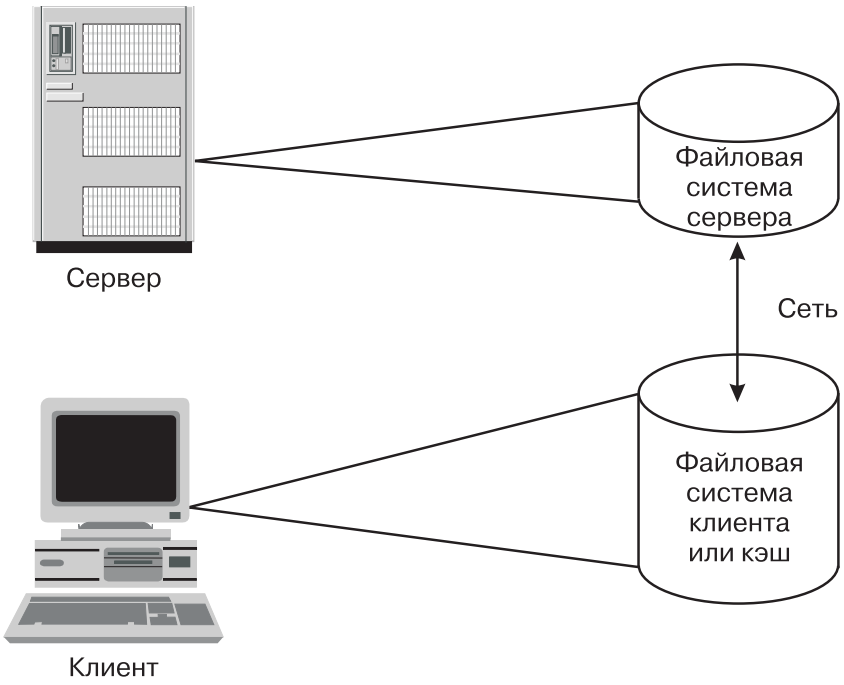


Рисунок 24. Кэширующая файловая система (CacheFS). На этом рисунке изображен клиент, подключенный к серверу по сети. На дисках сервера хранится исходная файловая система. На дисках клиента хранится кэшируемая, или целевая файловая система.

Ниже описан алгоритм работы CacheFS:

1. После создания на клиенте файловой системы CacheFS вы можете указать файловые системы, которые должны быть смонтированы в кэше.
2. При обращении к файлам исходной файловой системы они загружаются из целевой файловой системы и остаются в кэше. Кэш не заполняется заранее. Поэтому первый запрос к файлу выполняется за обычное время доступа к NFS, а последующие - за время доступа к локальной системе JFS.
3. CacheFS следит за тем, чтобы содержимое кэша не устаревало. Для этого время последнего изменения файлов в кэше регулярно сравнивается со временем их изменения в целевой файловой системе.

4. Если эти значения не совпадают, все данные и атрибуты каталога или файла удаляются из кэша и заново загружаются из исходной файловой системы.

CacheFS может применяться, например, в среде CAD. В этом случае все базовые копии чертежей будут храниться на сервере, в то время как клиенты будут работать с локальными кэшированными копиями.

CacheFS не поддерживает чтение и запись файлов размером больше 2 ГБ.

Преимущества CacheFS

Поскольку все данные после получения с сервера кэшируются, скорость выполнения запросов к файловой системе существенно увеличивается.

Небольшие объемы данных могут храниться в памяти клиента, поэтому преимущества кэширующей файловой системы проявляются в основном при работе с большими объемами данных. Дополнительное преимущество состоит в том, что данные остаются в кэше и после перезапуска системы, в то время как данные, находящиеся в оперативной памяти, при этом теряются.

Производительность NFS может быть ограничена "медленными" сетями или перегруженным сервером. В этом случае непосредственная работа клиентов с сервером будет медленной. CacheFS не ускоряет первое обращение к данным, но ускоряет обработку всех последующих запросов к тем же данным, которые выполняются без участия сервера и передачи по сети.

По мере увеличения объема данных в кэш-памяти клиентов нагрузка на сервер NFS уменьшается. Это означает, что сервер может обслуживать больше клиентов.

Уменьшение числа запросов, передаваемых по сети, снижает нагрузку на сеть, освобождая ее для других задач.

Не каждое приложение выиграет от применения CacheFS. Поскольку CacheFS повышает только производительность чтения, ее применение эффективно только для приложений, много раз считывающих одни и те же данные. Примером могут служить приложения CAD, которые часто обращаются к большим моделям при вычислениях.

Проведенные тесты показывают, что в случае применения файловой системы CacheFS скорость последовательного чтения возрастает в 2,4 - 3,4 раза по сравнению с чтением данных из памяти или с диска сервера NFS.

Влияние на производительность CacheFS

CacheFS не повышает производительность записи. Однако при монтировании файловой системы CacheFS командой **mount** вы можете указать в опции **-o** ряд параметров, управляющих записью. Они влияют на производительность последующего чтения данных.

Описание этих параметров приведено ниже:

write around

В режиме write around (применяется по умолчанию) операции записи обрабатываются также как в NFS. В этом режиме данные записываются в исходную файловую систему, а соответствующий файл удаляется из кэша. Это означает, что при последующем чтении только что записанных данных они будут получены с сервера.

non-shared

Режим non-shared применяется при исключительном доступе на запись к исходной файловой системе. В этом режиме все операции записи выполняются одновременно в локальной и в исходной файловых системах, причем соответствующий файл остается в кэше. При последующем чтении возвращаются данные из кэша без обращения к серверу.

Поскольку результаты небольших операций чтения сохраняются в памяти в любом случае, их кэширование на диске не повышает производительность. Также при кэшировании не повышается производительность неупорядоченного чтения различных блоков данных, кроме случаев повторного обращения к тем же блокам.

Первый запрос на чтение отправляется серверу, поскольку данные кэшируются только после получения запроса от пользователя. Поэтому первый запрос выполняется с такой же скоростью, как и обычные запросы NFS. Все последующие запросы к тем же данным выполняются со скоростью доступа к JFS.

Актуальность данных в кэше проверяется периодически, с некоторым интервалом. Поэтому кэшировать часто изменяющиеся данные опасно. Рекомендуется применять CacheFS для кэширования только неизменяемых или редко изменяемых данных.

Производительность записи данных в кэширующие файловые системы в NFS версий 2 и 3 различается. Проведенные тесты указывают на следующие особенности:

- Последовательная запись в новый файл из кэширующей файловой системы NFS версии 2 может выполняться на 25 процентов медленнее по сравнению с обычной версией той же файловой системы.
- Последовательная запись в новый файл из кэширующей файловой системы NFS версии 3 может выполняться в 6 раз медленнее по сравнению с обычной версией той же файловой системы.

Настройка CacheFS

Файловая система CacheFS не создается по умолчанию. Кроме того, опцию ее создания нельзя выбрать при создании файловой системы NFS. Вы должны явно указать кэширующие файловые системы.

Вы должны явно указать кэширующие файловые системы. Для этого нужно выполнить следующие действия:

1. Создайте локальную кэширующую файловую систему командой **cfsadmin**:

   ```
   # cfsadmin -c -o
   параметрыкаталог
   ```

   где *параметры* - это параметры выделения ресурсов, а *каталог* - имя каталога, в котором будет создан кэш.

2. Смонтируйте поверх нее исходную файловую систему:

   ```
   # mount -V cachefs -o backfstype=nfs,cachedir=/каталог-кэша удаленный-хост:/каталог локальная-точка-монтирования
   ```

   где *удаленный-хост:/каталог* - это имя удаленного хоста и файловой системы, в которой находятся данные, а *локальная точка монтирования* - точка монтирования в системе клиента, в которой будут доступны данные из удаленной файловой системы.

3. Файловой системой CacheFS можно также управлять с помощью SMIT (команда быстрого доступа: **smitty cachefs**).

При создании кэширующей файловой системы можно задать несколько параметров:

maxblocks
: Задает максимальное число блоков локальной файловой системы, возвращаемое в CacheFS. Значение по умолчанию - 90 процентов.

minblocks
: Задает минимальное число блоков локальной файловой системы, возвращаемое в CacheFS. Значение по умолчанию - 0 процентов.

threshblocks
: Задает число блоков, доступных клиенту в локальной файловой системе JFS, начиная с которого CacheFS возвращает число блоков, большее *minblocks*. Значение по умолчанию - 85 процентов.

maxfiles
: Максимальное число файлов, доступное CacheFS, в процентах от общего числа I-узлов локальной файловой системы. Значение по умолчанию - 90 процентов.

minfiles
Максимальное число файлов, доступное CacheFS, в процентах от общего числа I-узлов локальной файловой системы. Значение по умолчанию - 0 процентов.

maxfilesize
Наибольший размер кэшируемого файла в мегабайтах. Значение по умолчанию - 3.

Связанная с NFS информация

С NFS связано множество файлов, команд, демонов и процедур.

Дополнительные сведения приведены в *Управление сетями и средствами связи* и *Справочник по командам*.

Список файлов NFS

Описаны файлы, связанные с NFS.

Ниже приведен список файлов, в которых хранится информация о конфигурации NFS:

bootparams
Параметры загрузки бездисковых клиентов

exports
Список каталогов, которые можно экспортировать на клиенты NFS

networks
Информация о сетях в Internet

pcnfsd.conf
Параметры настройки демона **rpc.pcnfsd**

rpc База данных с информацией для программ RPC

xtab Список каталогов, экспортированных на данный момент

/etc/filesystems
Список файловых систем, монтируемых при запуске системы

Список команд NFS

Описаны команды, связанные с NFS.

Ниже приведен список команд NFS:

chnfs Запускает указанное число экземпляров демонов **biod** и **nfsd**

mknfs Настраивает систему для запуска NFS и запускает демоны NFS

nfso Настраивает сетевые параметры NFS

automount
Автоматически монтирует файловую систему NFS

chnfsexp
Изменяет атрибуты каталога, экспортированного через NFS

chnfsmnt
Изменяет атрибуты каталога, смонтированного через NFS

exportfs
Экспортирует на клиенты NFS и удаляет уже экспортированные каталоги

lsnfsexp
Выдает параметры каталогов, экспортированных через NFS

lsnfsmnt
Выдает параметры каталогов, смонтированных через NFS

mknfsexp
Экспортирует каталог через NFS

mknfsmnt
Монтирует каталог через NFS

rmnfs Останавливает демоны NFS

rmnfsexp
Удаляет каталоги, экспортированные через NFS, из списка экспортированных каталогов сервера

rmnfsmnt
Удаляет файловые системы, смонтированные через NFS, из списка смонтированных файловых систем клиента

Список демонов NFS

Описаны демоны, связанные с NFS.

Ниже приведен список демонов NFS:

lockd Обрабатывает запросы на блокировку, полученные через RPC

statd Обеспечивает восстановление после сбоя для служб блокирования NFS

Ниже приведен список сетевых демонов и утилит:

biod Отправляет серверу запросы клиента на чтение и запись

mountd
Отвечает на запросы клиентов на монтирование файловых систем

nfsd Запускает демоны, обрабатывающие запросы клиентов на работу с файловыми системами

pcnfsd Обрабатывает запросы клиентов NFS, установленных на PC

nfsstat Выдает информацию о готовности системы к приему вызовов

on Выполняет команды в удаленных системах

portmap
Преобразует номера программ RPC в номера портов Internet

rexd Принимает запросы удаленных систем на запуск программ

rpcgen Создает код на языке C, реализующий протокол RPC

rpcinfo
Выдает информацию о состоянии серверов RPC

rstatd Показывает полученные от ядра статистические данные о производительности

rup Показывает состояние удаленного хоста в локальной сети

rusers Показывает список пользователей, работающих с удаленными системами

rusersd
Отвечает на запросы команды **rusers**

rwall Отправляет сообщения всем пользователям сети

rwalld Обрабатывает запросы команды **rwall**

showmount
Показывает список всех клиентов, смонтировавших удаленные файловые системы

spray Отправляет хосту указанное число пакетов

sprayd Получает пакеты команды **spray**

Ниже приведен список демонов и утилит, обеспечивающих защиту при передаче данных по сети:

chkey Изменяет ключ шифрования пользователя

keyenvoy
Обеспечивает взаимодействие пользовательских процессов и сервера ключей

keylogin
Зашифровывает и хранит личный ключ пользователя

keyserv
Хранит общие и личные ключи

mkkeyserv
Запускает демон **keyserv** и удаляет символ комментария из соответствующих записей файла `/etc/rc.nfs`

newkey
Создает новый ключ в файле общих ключей

rmkeyserv
Останавливает демон **keyserv** и добавляет символ комментария в запись о демоне **keyserv** в файл `/etc/rc.nfs`

ypupdated
Обновляет информацию в таблицах NIS

Параметры бездисковых клиентов хранятся в следующем файле:

bootparamd
Содержит информацию, необходимую для загрузки бездисковых клиентов

Ниже приведен список функций NFS:

cbc_crypt(), des_setparity(), ecb_crypt()
Функции шифрования DES.

Производительность LPAR

В этой главе приведены рекомендации по контролю за работой и по настройке производительности логических разделов в системе AIX с архитектурой POWER4.

Дополнительная информация и разделах и их реализации приведена в *AIX 5L версии 5.3 - Установка AIX в среде с разделами* и *Консоль аппаратного обеспечения: Руководство по установке и эксплуатации*.

Производительность и логические разделы

Системы, основанные на POWER4, могут выступать как в роли больших систем с процессорами POWER4 в виде многокристальных модулей (MCM), так и в роли небольших систем с процессорами POWER4 в виде однокристальных модулей (SCM).

Производительность выполнения задач в этих системах может различаться.

Разбиение системы на разделы повышает гибкость использования аппаратного обеспечения, если приложения не масштабируются в достаточной степени по большому числу процессоров, либо если необходимо использовать разделы для различных задач. Запуск нескольких экземпляров приложения в разделах меньшего размера позволяет достичь большей производительности, чем работа одного экземпляра во всей системе. Например, если приложение представляет собой единственный процесс без поддержки или почти без поддержки нитей, то оно будет работать в двух- или четырехпроцессорных системах, но не сможет

полностью использовать ресурсы систем с большим числом процессоров. Вместо модификации приложения для использования большого числа процессоров можно запустить его в нескольких разделах меньшего размера.

Производительность в системе с логическими разделами следует оценивать с помощью анализа небольших вариаций. Гипервизор и встроенное программное обеспечение управляют памятью, процессорами и адаптерами каждого раздела. Приложения обычно не имеют информации о том, какая память, процессоры и адаптеры были присвоены разделу. Существует несколько рекомендаций по настройке производительности, связанных с расположением памяти по отношению к процессорам, совместным использованием кэшей L2 и L3, а также накладными расходами по выполнению программы гипервизора.

Замечания по операционной системе для LPAR

При работе с операционной системой с LPAR необходимо учитывать определенные факторы.

Разделы в системах с архитектурой POWER4 могут работать в следующих операционных системах:

- Операционная система AIX с 32-разрядным ядром.
- AIX с 64-разрядным ядром. В системе AIX 64-разрядное ядро оптимизировано для выполнения 64-разрядных приложений и улучшает масштабируемость, позволяя приложениям использовать больший объем физической памяти раздела.
- Linux с 64-разрядным ядром.

В каждом из разделов системы может быть установлена своя версия операционной системы. Программы, работающие в разных разделах, изолированы. Изоляция защищает как от ошибок программ, так и от преднамеренных попыток взлома барьера между LPAR. Связь между разделами обеспечивается только через сеть. Сбой программного обеспечения (как приложений, так и операционной системы) в одном разделе не повлияет на другие разделы. Использование разделами общих аппаратных ресурсов ограничено. Например, ни один раздел не может заблокировать общую шину PCI на неопределенное время.

Системные компоненты

Среда с LPAR совместно создается несколькими компонентами системы.

Связь между процессорами, встроенным программным обеспечением и операционной системой требует, чтобы каждый из этих компонентов поддерживал определенные функции. Таким образом, поддержка LPAR основана не просто на отдельных компонентах, но на их взаимодействии. Микропроцессор POWER4 поддерживает расширенный формат системного вызова - режим гипервизора - позволяющий привилегированной программе работать с определенными аппаратными устройствами. Этот режим используется для поддержки логических разделов. Он позволяет процессору работать с информацией о системах, расположенных за границами локального логического раздела. Гипервизор использует небольшой объем ресурсов процессора и памяти, поэтому сравнение системы с гипервизором с системой без него позволяет выявить лишь незначительное снижение производительности.

Система на основе архитектуры POWER4 может быть загружена в различных режимах, включая следующие:

- Система без поддержки LPAR и, следовательно, без гипервизора. Такая система называется "системой без логических разделов".
- Система с гипервизором и поддержкой логических разделов.

Логические разделы с привязкой к процессорам

В некоторых системах Платформа с процессором POWER предусмотрена возможность создания логических разделов с привязкой к процессорам. При этом процессоры и память, выделенные каждому разделу, автоматически выбираются по физическому расположению ресурсов друг рядом с другом.

Консоль аппаратного обеспечения, HMC, делит систему на одинаковые логические разделы (LPAR), содержащие по 4 или по 8 процессоров, в зависимости от выбранной администратором опции. Каждому разделу принадлежат процессоры и память одного или нескольких MCM. Это позволяет использовать систему в качестве набора идентичных кластеров, что упрощает оптимизацию при решении научных и

технических задач. При загрузке системы в этом режиме изменение объема ресурсов процессоров и памяти в разделах невозможно. Производительность компактных логических разделов выше, чем обычных.

Примечание: Поддержка сродства памяти в AIX недоступна в режиме LPAR.

Рабочая схема в разделе

В каждом разделе AIX существует свой набор средств управления рабочей схемой.

Работа средств управления рабочей схемой WLM в логическом разделе AIX не отличается от работы во всей системе. WLM действует только в рамках своего раздела. Владельцы приложений, имеющие опыт присвоения процессоров и памяти приложениям, пытаются расширить эту возможность на разделы. Однако в среде с логическими разделами распределение процессоров по логическим разделам выходит за рамки управления рабочей схемой, поэтому непосредственное выделение процессоров указанной MCM конкретной задаче невозможно. Тем не менее, WLM и команда **bindprocessor** позволяют присвоить задаче любой процессор, выделенный разделу.

Выбор между логическими разделами и управлением рабочей схемой

При выборе между логическими разделами и управлением рабочей схемой необходимо учесть конкретный набор задач, приложений и решений.

Обычно разбиение на логические разделы предпочтительнее в следующих ситуациях:

- Различным приложениям требуются для работы разные версии или несовместимые исправления операционной системы.
- Для обеспечения требований к защите необходимы различные владельцы и администраторы систем, надежное разделение защищаемых данных и приложений, связанных только через сеть.
- Приложения используют разные процедуры восстановления, например, кластеры **HA** (с высокой готовностью) и средства передачи управления при сбое.
- Необходима надежная изоляция на случай сбоя приложений и операционных систем.
- Необходима надежная изоляция нагрузки на ресурсы, чтобы производительность различных задач не зависела от доступности общих ресурсов.

Надежная изоляция нагрузки на ресурсы важна при отслеживании и настройке задач в системе с логическими разделами. Настройка производительности в системе AIX, в которой одновременно работают другие важные задачи, сильно усложняется. Эту процедуру проще выполнять в системе с четким разделением ресурсов между логическими разделами.

Влияние на производительность LPAR

Производительность системы с LPAR практически не отличается от работы той же системы в режиме SMP.

Выполнение функций гипервизора в системе с LPAR как правило добавляет не больше 5% к обычной нагрузке на ресурсы памяти и ввода-вывода. Одновременно работающие логические разделы мало влияют друг на друга, за исключением отдельных случаев. Небольшие дополнительные накладные расходы связаны с работой гипервизора по управлению виртуальной памятью. Для большинства задач эти накладные расходы минимальны, но они увеличиваются при частом перемещении страниц. Разделение на логические разделы позволяет повысить производительность слабо масштабируемых приложений в в системах SMP с большим числом процессоров путем запуска копии задачи в каждом разделе.

Моделирование систем меньшего размера

Наилучший способ моделирования системы меньшего размера заключается в уменьшении объема памяти, доступной разделу.

В системах MCM на основе архитектуры POWER4 команда **rmss** выделяет память из системы без учета расположения этой памяти по отношению к локальной MCM. Результирующая производительность может зависеть от того, какая память будет доступна и выделена разделу. Например, если вы моделируете с

помощью команды **rmss** 8-процессорный раздел с локальной памятью, то физически выделенная память может не быть локальной по отношению к MCM. Более того, 8 процессоров могут быть выделены не из одного MCM, а из списка доступных процессоров.

При удалении процессоров из раздела системы MCM существует ряд особенностей, связанных с явным учетом гипервизором связи MCM и памяти. Хотя эти особенности мало влияют на производительность, они могут усложнить ее анализ.

Микропроцессоры в разделе

Микропроцессоры можно связать с LPAR.

Присвоенные процессоры

Для того чтобы просмотреть список процессоров, выделенных LPAR, выберите объект Управляемая система (CEC) в HMC и просмотрите его свойства.

Свойства объекта содержат страницу с указанием состояния всех процессоров, присвоенных работающим разделам. AIX применяет для идентификации процессоров номера, полученные от встроенного программного обеспечения, поэтому для определения выделенных разделу процессоров необходимо сравнить их номера с кодами расположения AIX.

Ниже приведен пример проверки состояния процессоров двухпроцессорного раздела:

```
 > lsdev -C | grep proc
proc17   Доступно  00-17   Процессор
proc23   Доступно  00-23   Процессор
```

Влияние отключения процессоров

При отключении процессоров в системе MCM на основе архитектуры POWER4 эти процессоры продолжают использоваться для маршрутизации потоков выполнения и данных в системе. В результате производительность может снизиться.

Управление виртуальными процессорами в разделе

Функции планировщика ядра были расширены для динамического увеличения и уменьшения использования виртуальных процессоров в зависимости от мгновенной загрузки раздела, измеряемой физическим использованием этого раздела.

Каждую секунду планировщик ядра оценивает число виртуальных процессоров, которое следует активировать для согласования с физическим использованием раздела. Если это число указывает на значительное использование виртуальных процессоров, базовое количество требуемых виртуальных процессоров будет увеличено, для того чтобы обеспечить возможность увеличения нагрузки. Дополнительные виртуальные процессоры можно запросить с помощью команды **schedo**. Это значение затем используется для определения того, следует ли включить или выключить виртуальный процессор, поскольку планировщик изменяет число используемых виртуальных процессоров каждую секунду только на единицу. Таким образом, если вычисленное значение больше числа активированных виртуальных процессоров, будет активирован еще один виртуальный процессор. Если же это значение меньше числа активированных виртуальных процессоров, один виртуальный процессор будет выключен.

При выключении виртуальных процессоров они не удаляются динамически из раздела, как в случае DLPAR. Виртуальный процессор больше не может выполнять или получать не связанные задания, однако он может выполнять связанные задания. Число включенных логических и виртуальных процессоров, которые видны для пользователя и приложений, не изменяется. Не оказывается никакого влияния на работу промежуточного программного обеспечения и приложений, выполняемых в системе, активные и неактивные виртуальные процессоры являются внутренними для системы.

По умолчанию значение переменной *vpm_xvcpus* равно 0, означающее, что свертывание включено. Это означает, что включено управление виртуальными процессорами. С помощью команды schedo можно изменить значение переменной *vpm_xvcpus*.

Для того чтобы определить, включена или выключена функция управления виртуальными процессорами, можно применить следующую команду:

```
# schedo -o vpm_xvcpus
```

Для увеличения числа занятых виртуальных процессоров на 1 можно использовать следующую команду:

```
# schedo -o vpm_xvcpus=1
```

Каждый виртуальный процессор может загрузить только один физический процессор. Необходимое число виртуальных процессоров определяется путем вычисления суммы загрузки физического CPU и значения переменной *vpm_xvcpus* следующим образом:

```
Необходимое число виртуальных процессоров =
Загрузка физического CPU + Число дополнительных виртуальных процессоров
```

Если необходимое число виртуальных процессоров меньше текущего числа включенных виртуальных процессоров, один виртуальный процессор будет выключен. Если необходимое число виртуальных процессоров больше текущего числа включенных виртуальных процессоров, отключенный виртуальный процессор будет включен. По-прежнему разрешено выполнение нитей, подключенных к выключенному виртуальному процессору.

Примечание: Всегда следует округлять значение, полученное с помощью приведенного выше равенства.

Далее приведен пример вычисления необходимого числа виртуальных процессоров:

В течение последнего интервала разделом A использовалось два с половиной процессора. Значение переменной *vpm_xvcpus* установлено равным 1. Используя приведенное выше равенство, получаем:

```
Загрузка физического CPU = 2.5
Число дополнительных виртуальных процессоров (vpm_xvcpus) = 1

Необходимое число виртуальных процессоров = 2.5 + 1 = 3.5
```

Округляя полученное значение в большую сторону до ближайшего целого числа, получаем 4. Следовательно, в системе требуется 4 виртуальных процессора. Таким образом, если раздел A работал с 8 процессорами, 4 виртуальных процессора отключены, а 4 остались включенными. Если включена SMT, каждому виртуальному процессору соответствует два логических. То есть 8 логических процессоров выключено и 8 - включено.

В следующем примере при умеренной загрузке, если не включена функция свертывания, использование каждого виртуального процессора, выделенного разделу, минимально. В приведенном далее выводе утилиты **mpstat -s** системы с 4 виртуальными CPU показано использование каждого виртуального процессора и двух связанных с ним логических процессоров:

```
       Proc0                  Proc2                  Proc4                  Proc6
       19.15%                 18.94%                 18.87%                 19.09%
    cpu0      cpu1         cpu2     cpu3          cpu4     cpu5          cpu6     cpu7
   11.09%   8.07%          10.97%   7.98%         10.93%   7.93%         11.08%   8.00%
```

Если функция свертывания включена, система вычисляет необходимое число виртуальных процессоров с помощью приведенного выше равенства. Вычисленное значение затем применяется для уменьшения числа виртуальных процессоров до значения, необходимого для обеспечения средней загрузки системы без снижения производительности. В приведенном далее выводе утилиты **mpstat -s** системы с 4 виртуальными CPU показано использование каждого виртуального процессора и двух связанных с ним логических процессоров:

```
      Proc0                Proc2                Proc4                Proc6
      54.63%               0.01%                0.00%                0.08%
  cpu0      cpu1       cpu2      cpu3       cpu4      cpu5       cpu6      cpu7
 38.89%    15.75%     0.00%     0.00%      0.00%     0.00%      0.03%     0.05%
```

Из приведенных выше данных видны преимущества, связанные с уменьшением использования и обслуживания дополнительных процессоров, а также увеличенная привязка к процессорам при концентрации работы на одном виртуальном процессоре. Однако при большой рабочей нагрузке функция свертывания не препятствует использованию всех виртуальных CPU при необходимости.

Замечания о приложениях

Необходимо помнить о некоторых моментах, относящихся к приложениям, связанным с LPAR.

Обычно на работе приложения никак не сказывается то, что оно запущено в LPAR. Существуют некоторые отличия, которые должны учитываться пользователем, но не приложениями. За исключением этих отличий работа AIX в логическом разделе ничем не отличается от работы на отдельном сервере, как с точки зрения администратора, так и с точки зрения приложений. Чаще всего LPAR незаметны приложениям AIX и служебным утилитам AIX. При использовании приложений независимых поставщиков требуется лишь сертификация этих приложений для применяемой версии AIX.

Запуск команды uname в LPAR

С помощью команды **uname** можно получить информацию о связи системы с LPAR.

```
> uname -L
-1 NULL
```

"-1" означает, что система работает в режиме без логических разделов.

В приведенном ниже примере команда **uname** выдает номер логического раздела и его имя в HMC:

```
 > uname -L
3 Web-сервер
```

Зная, в каком именно логическом разделе LPAR выполняется приложение, вы можете легко оценить различия в производительности.

Виртуальная консоль

Логические разделы не имеет физической консоли.

Хотя им могут быть присвоены физические последовательные порты, каждым портом может пользоваться только один раздел. Для диагностики и вывода сообщений встроенное аппаратное обеспечение поддерживает виртуальный терминал, воспринимаемый системой AIX как обычное устройство. Вывод на этот терминал передается в HMC. Подсистема диагностики AIX применяет этот виртуальный терминал в качестве системной консоли. При записи большого объема данных на системную консоль производительность ограничена последовательным соединением.

Значение времени дня

Для каждого раздела поддерживаются собственные значения времени дня, поэтому каждый раздел может работать в своем часовом поясе.

Единственно возможный способ обмена информацией между разделами - связь по сети. В частности, возможность работы систем в различных часовых поясах следует учитывать при анализе протоколов трассировки и других записей о системном времени.

Серийный номер системы

Команда **uname -m** позволяет просмотреть в разделе различную системную информацию.

Эта команда показывает серийный номер всей системы. Этот номер совпадает во всех разделах.

Замечания по работе с памятью

При работе с памятью необходимо учитывать определенные факторы.

Для каждого раздела указывается максимальный, предпочтительный и минимальный объем памяти. Если при перезагрузках системы наблюдается изменение производительности, то учтите, что выделение ресурсов памяти и процессоров выполняется каждый раз из доступных ресурсов. Учтите также, что в HMC задается полный объем памяти, выделяемой разделу. Часть этой памяти используется гипервизором для поддержки преобразования таблицы страниц.

Ресурсы памяти выделяются из всей памяти системы. У приложений разделов нет возможности определить физическое расположение памяти.

Динамическое распределение ресурсов

Функции DLPAR доступны в системах System p на основе POWER4 с обновлениями микрокода от октября 2002 года или от более поздней даты. Разные логические разделы могут работать под управлением операционных систем различных уровней.

До появления поддержки DLPAR для добавления в систему ресурсов приходилось перезагружать раздел. Функция DLPAR позволяет расширить возможности системы с логическими разделами за счет динамического добавления и удаления процессоров, памяти, устройств и блоков ввода-вывода при работе с активными логическими разделами. С ее помощью можно перераспределять аппаратные ресурсы в системе, не отключая при этом логические разделы.

С помощью DLPAR можно выполнять следующие основные операции:

- Перемещение ресурса из одного логического раздела в другой
- Удаление ресурса из логического раздела
- Добавление ресурса в логический раздел

Процессоры, память и устройства ввода-вывода, не выделенные в данный момент ни одному из логических разделов, находятся в "свободном пуле". Существующие в системе разделы остаются невидимыми для других разделов и для свободного пула. Если включена поддержка DLPAR, то при удалении процессора из активного раздела система помещает его в свободный пул, после чего этот процессор можно добавить в другой активный раздел. При добавлении процессора в активный раздел этот процессор получает полный доступ ко всей памяти раздела, адресному пространству ввода-вывода и прерываниям ввода-вывода. Процессор может использоваться всеми задачами, выполняемыми в данном разделе.

Память можно удалять блоками по 256 МБ. Удаление памяти почти не влияет на работу приложения в разделе AIX, поскольку ядро AIX почти полностью работает в виртуальном режиме. Приложения, расширения ядра и большинство процессов ядра используют только виртуальную память. При удалении памяти раздел может начать подкачку. Поскольку компоненты ядра AIX допускают подкачку, это может повлиять на производительность. При удалении памяти необходимо следить, чтобы это удаление не приводило к обращению к пространству подкачки.

Можно добавлять в активный логический раздел или удалять из него устройства ввода-вывода, например, сетевые адаптеры, дисководы CD-ROM или накопители на магнитной ленте. Это позволяет сэкономить на покупке нескольких одинаковых физических устройств за счет того, что одно и то же редко используемое устройство можно подключать к разным логическим разделам. В отличие от добавления или удаления процессоров или памяти, перераспределение устройств ввода-вывода требует выполнения определенных процедур оперативной замены PCI перед добавлением устройства в активный раздел или удалением из него. Процедуры оперативной замены выполняются с помощью SMIT.

К системе подключена Консоль аппаратного обеспечения (HMC) которая позволяет выполнять динамическое изменение конфигурации. Для поддержки DLPAR необходима версия HMC R3V1.0 или выше. Список операций HMC, относящихся к DLPAR, приведен в документе *The Complete Partitioning Guide for IBM eServer pSeries Servers*.

Специальный гипервизор обеспечивает управление аппаратным обеспечением и изоляцию виртуальных машин (разделов), работающих в одной физической системе. Команды, управляющие перемещением ресурсов между разделами, можно передавать гипервизору логических разделов (LPAR) через графический пользовательский интерфейс HMC или с помощью командной строки HMC. Только гипервизор может "видеть" и подключать системные ресурсы, причем можно запустить только один экземпляр гипервизора. DLPAR никак не влияет на защиту раздела. Ресурсы, перемещаемые из одного раздела в другой, инициализируются повторно, так что не существует никаких данных, оставшихся после операции перемещения.

Замечания по производительности DLPAR

Повышение или снижение производительности DLPAR связано с несколькими факторами.

Память можно удалять или добавлять логическими блоками. При удалении памяти из раздела время, затрачиваемое на выполнение операции DLPAR, зависит от числа удаляемых блоков памяти. Например, операция DR, удаляющая 4 ГБ памяти из простаивающего раздела, занимает от 1 до 2 минут. Однако динамическое перераспределение больших страниц памяти не поддерживается. Невозможно удалить область памяти, которая содержит большую страницу.

В системах с компактными логическими разделами выделение ресурсов процессоров и памяти осуществляется на основе конфигурации многокристального модуля (MCM). В этом случае HMC не поддерживает операции динамического перераспределения процессоров и памяти. При работе с компактными логическими разделами динамическое перераспределение возможно только для ресурсов адаптеров ввода-вывода.

Для того чтобы полностью использовать возможности динамического перераспределения ресурсов, включите функцию поддержки DLPAR для приложений и промежуточного программного обеспечения. При этом приложение сможет само изменять размер в соответствии с новыми аппаратными ресурсами. В AIX предусмотрены сценарии DLPAR и API для динамического изменения размера приложений, поставляемых другими разработчиками. Инструкции по применению этих сценариев или API приведены в разделе DLPAR section of *Общие принципы программирования*.

Программы настройки DLPAR

Описаны инструменты для мониторинга и обслуживания DLPAR.

DLPAR позволяет динамически изменять число активных процессоров. Для отслеживания разницы между числом активных процессоров и максимально возможным числом процессоров в системе можно воспользоваться следующими параметрами:

_system_configuration.ncpus
Запрашивает число активных процессоров

_system_configuration.max_ncpus
Выдает максимально возможное число процессоров в данной системе

AIX поддерживает точку трассировки **38F**, предназначенную для включения сбора данных трассировки при добавлении или удалении процессоров или памяти.

В программах контроля производительности, таких как **curt**, **splat**, **filemon**, **netpmon**, **tprof** и **pprof**, обработка операции DR не предусмотрена. Эти программы рассчитаны на работу со статическими ресурсами процессоров и памяти. В некоторых случаях, например при выполнении программ **tprof** и **pprof**, динамическое перераспределение ресурсов не вызывает никаких побочных эффектов. Однако, если такая

операция выполняется во время работы других программ контроля производительности (например, **curt**), это может привести к непредвиденным или ошибочных результатам.

Существуют программы, которые поддерживают операции динамического перераспределения ресурсов. Они могут распознавать изменения конфигурации и в зависимости от этих изменений перенастраивать свой вывод. Программы, поддерживающие функцию DLPAR, перечислены ниже:

topas Интерфейс программы и формат данных вывода не изменились. При выполнении операции динамического перераспределения ресурсов программа **topas** отслеживает добавление или удаление ресурса и заносит в отчет статистическую информацию в соответствии с новым набором ресурсов.

sar, vmstat и iostat
Интерфейсы программ и формат данных вывода не изменились. При выполнении операции динамического перераспределения ресурсов пользователю отправляется сообщение, информирующее его об изменении конфигурации. Затем программы начинают сбор статистики, соответствующей новому набору ресурсов.

rmss Формат вызова команды не изменился; если во время выполнения программы **rmss** запускается динамическое перераспределение ресурсов, программа продолжает работать как обычно.

Рекомендации DLPAR по добавлению процессоров или памяти

Существуют разные способы определения потребности системы в дополнительных процессорах, памяти или разъемах ввода-вывода.

При удалении памяти из раздела операция DR будет успешно выполнена даже в том случае, если для размещения удаляемой памяти не хватает свободной физической памяти. При этом вместо физической памяти выделяется нужный объем пространства подкачки. Поэтому рекомендуется отслеживать для данного раздела статистику по операциям с пространством подкачки до и после динамического удаления памяти. Подкачкой управляет Администратор виртуальной памяти, однако, слишком интенсивный обмен страницами между памятью и диском может привести к снижению производительности.

Рекомендации, приведенные в разделах “Быстродействие памяти” на стр. 128 и “Производительность процессора” на стр. 104, помогут вам определить, когда возникает нехватка памяти или процессоров. Рекомендации по добавлению устройств ввода-вывода приведены в разделе “Быстродействие сети” на стр. 258. Кроме того, в этих разделах описано, как оценить влияние на производительность удаления одного из этих ресурсов.

Создание микроразделов

Логические разделы позволяют запускать в одной системе несколько операционных систем без их пересечения. В предыдущих версиях AIX процессоры нельзя было сделать общими для разных разделов. Можно использовать разделы с общими процессорами, или SPLPAR, известные также как Micro-Partitioning.

Факты о Micro-Partitioning

При микроразделении виртуальные процессоры отображаются на физические процессоры, а затем назначаются разделам вместо этих физических процессоров.

С помощью Hypervisor можно указать, какую долю ресурсов процессора разрешить для использования общими разделами, что определяется как выделение прав на использование процессора. Минимальная доля использования ресурсов процессора составляет 10%.

Ниже перечислены преимущества использования Micro-Partitioning:

- Оптимальное использование ресурсов
- Быстрое развертывание новых серверов
- Изоляция приложений

Micro-Partitioning доступно в системах System p5. Разные логические разделы могут работать под управлением операционных систем различных уровней, однако операции Micro-Partitioning можно выполнять только в логических разделах с AIX версии 6.1 и более поздних версий.

При работе с серверами IBM eServer p5 из консоли аппаратного обеспечения (HMC) можно выбрать следующие типы логических разделов:

Раздел с выделенным процессором

При выборе логического раздела выделенного процессора данному логическому разделу назначается процессор целиком.

Мощность выполнения операций в разделе ограничена общей мощностью процессоров, настроенных в данном разделе, и не может превысить эту мощность без добавления большего числа процессоров в раздел с помощью операции DLPAR.

Логический раздел общего процессора

При использовании раздела общего процессора физические процессоры виртуализуются, а затем назначаются разделам.

Мощность виртуальных процессоров лежит в пределах от 10 процентов до всей мощности физического процессора. Таким образом несколько логических разделов системы могут совместно использовать одни и те же процессоры, распределяя между собой мощность этих процессоров. Для одного раздела допускается использование 64 виртуальных процессоров. Более подробная информация приведена в разделе “Управление виртуальными процессорами в разделе” на стр. 360.

Реализация Micro-Partitioning

Так же как и при использовании LPAR при работе с функциями Micro-Partitioning можно определить разделы с помощь HMC.

В следующей таблице перечислены различные типы процессоров, которые можно использовать для функций Micro-Partitioning:

Тип процессора	Описание
Физический процессор	Физический процессор является реальным аппаратным ресурсом, представляющим ряд отдельных процессорных ядер (а не чипов). Каждый чип содержит два ядра. В системах на основе POWER5 максимальное число физических процессоров равно 64.
Логический процессор	Логический процессор является представлением операционной системы управляемого процессорного модуля. Максимальное число логических процессоров равно 128.
Виртуальный процессор	Виртуальный процессор является модулем, который представляет долю логического процессора, совместно используемую различными разделами. Максимальное число виртуальных процессоров равно 64.

При создании раздела необходимо определить его тип: раздел общего процессора или раздел выделенного процессора. Существование и общего, и выделенного процессоров в одном разделе не допускается. Для включения совместного использования процессоров необходимо настроить следующие параметры:

- Режим совместного использования процессора: с ограничениями или без ограничений[1]
- Мощность: вес[2]
- Число виртуальных процессоров: желаемое, минимальное и максимальное

Примечание: Режим с ограничениями означает, что мощность процессоров никогда не превысит выделенной мощности, а при режиме без ограничений эта мощность может быть превышена, если в общем пуле процессоров имеются доступные ресурсы.

Примечание: Мощность определяется числом модулей процессора, измеряемых по 0,01 от процессора. Например, чтобы выделить половину процессора, в HMC следует указать 0,50 модулей процессора.

Замечания по производительности Micro-Partitioning

Micro-Partitioning может оказать как положительное, так и отрицательное влияние на производительность.

Преимуществом Micro-Partitioning является то, что оно приводит к рациональному использованию ресурсов системы, работая только с необходимой для каждого раздела частью ресурсов процессора. Однако в связи с дополнительной нагрузкой, связанной с управлением работающими виртуальными процессорами, при выборе значений атрибутов обратите внимание на требования к вычислительной мощности.

Для достижения оптимальной производительности создавайте минимальное число логических разделов. Это уменьшит дополнительную нагрузку, связанную с управлением виртуальными процессорами.

Приложения, активно использующие ресурсы CPU, например, приложения, выполняющие большое количество вычислений, могут не подойти для работы в среде, использующей Micro-Partitioning. Если при работе приложения используется большая часть заявленной процессорной мощности, то для управления запросами такого приложения следует применить раздел выделенного процессора.

Active Memory Expansion (AME)

Технология Active Memory Expansion (AME) - это новая технология, позволяющая увеличить объем доступной системной памяти. AME сжимает данные, увеличивая объем свободной памяти.

Обзор

Технология Active Memory Expansion (AME) использует сжатие данных в памяти для увеличения объема данных, которые могут быть размещены в памяти, и таким образом расширяет эффективность использования памяти в системах на основе процессоров IBM Power Systems. Данная функция поддерживается на уровне операционной системы и ее работа прозрачна для приложений. AME настраивается для каждого логического раздела (LPAR). Поэтому данную функцию AME можно использовать для одного или нескольких LPAR в системе.

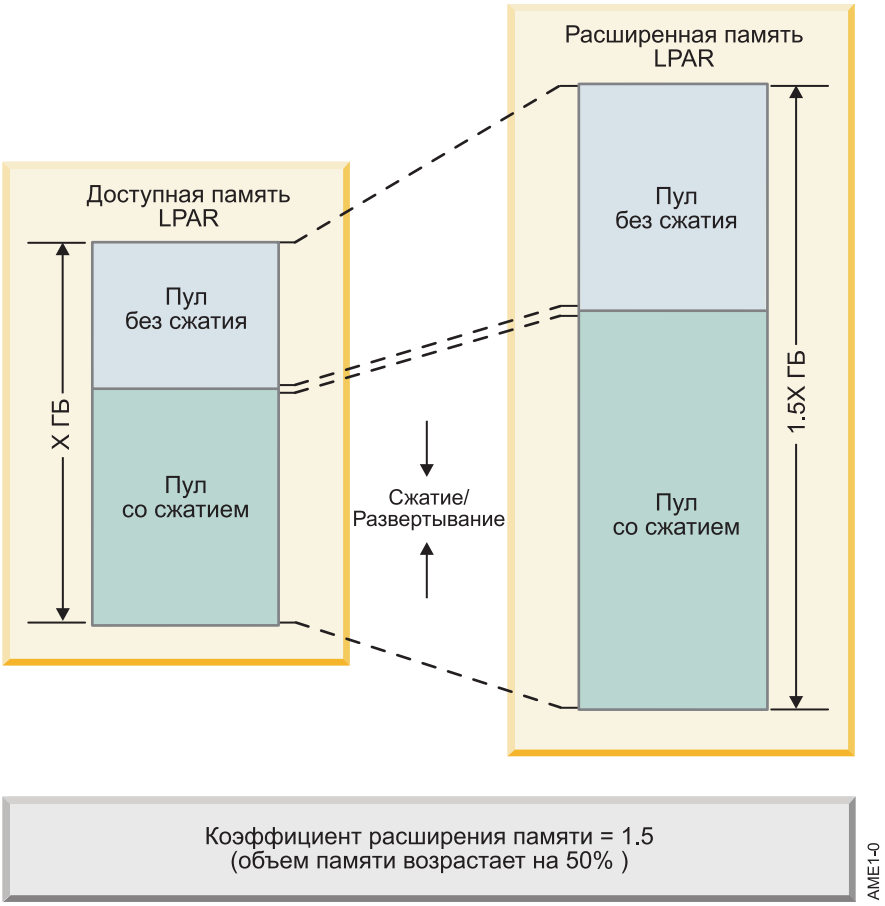
Если Active Memory Expansion включена для LPAR, то операционная система сжимает часть данных в памяти LPAR, а другая часть памяти остается неизменной. В результате память разбивается на два пула. Пулы памяти:

- Пул сжатой памяти
- Пул несжатой памяти

Операционная система динамически изменяет размер пула сжатой памяти в зависимости от загрузки системы и настроек LPAR.

Распределение данных между пулами зависит также от шаблона доступа к памяти приложения. Когда приложению требуется доступ к сжатым данным, операционная система автоматически извлекает данные и перемещает их из пула сжатой памяти в пул несжатой памяти, делая их доступными приложению. При заполнении пула распакованной памяти операционная система сжимает часть данных и перемещает их в пул несжатой памяти.

Для приложения операции сжатия и распаковки выполняются прозрачно. Поскольку AME использует сжатие памяти, при использовании данной технологии возникает дополнительная нагрузка на процессор. Значение этой нагрузки зависит от текущего состояния системы и используемого уровня расширения памяти.



Примечание: При включенном AME операционная система AIX по умолчанию использует страницы размером 4 КБ. В случае применения IBM AIX 7.2 с технологическим уровнем 1 и выше в системе POWER8 с помощью команды **vmo** с параметром `ame_mpsize_support` можно включить страницы размером 64 КБ.

Коэффициент расширения памяти и объем расширенной памяти

При использовании Active Memory Expansion для LPAR необходимо указать коэффициент расширения памяти. Данный коэффициент указывает объем доступной памяти для LPAR. На основе этого коэффициента операционная система вычисляет объем памяти, доступный приложению при использовании сжатия памяти. Данный объем памяти также называется объемом расширенной памяти. Коэффициент расширения памяти используется в качестве множителя объема исходной памяти LPAR.

```
Объем расширенной памяти LPAR = Объем исходной памяти LPAR * коэффициент расширения памяти
```

Например, если коэффициент расширения памяти равен 2,0, то объем доступной памяти удвоится. Если коэффициент расширения памяти равен 2,0, а объем памяти равен 20 ГБ, то расширенный объем памяти будет равняться 40 ГБ.

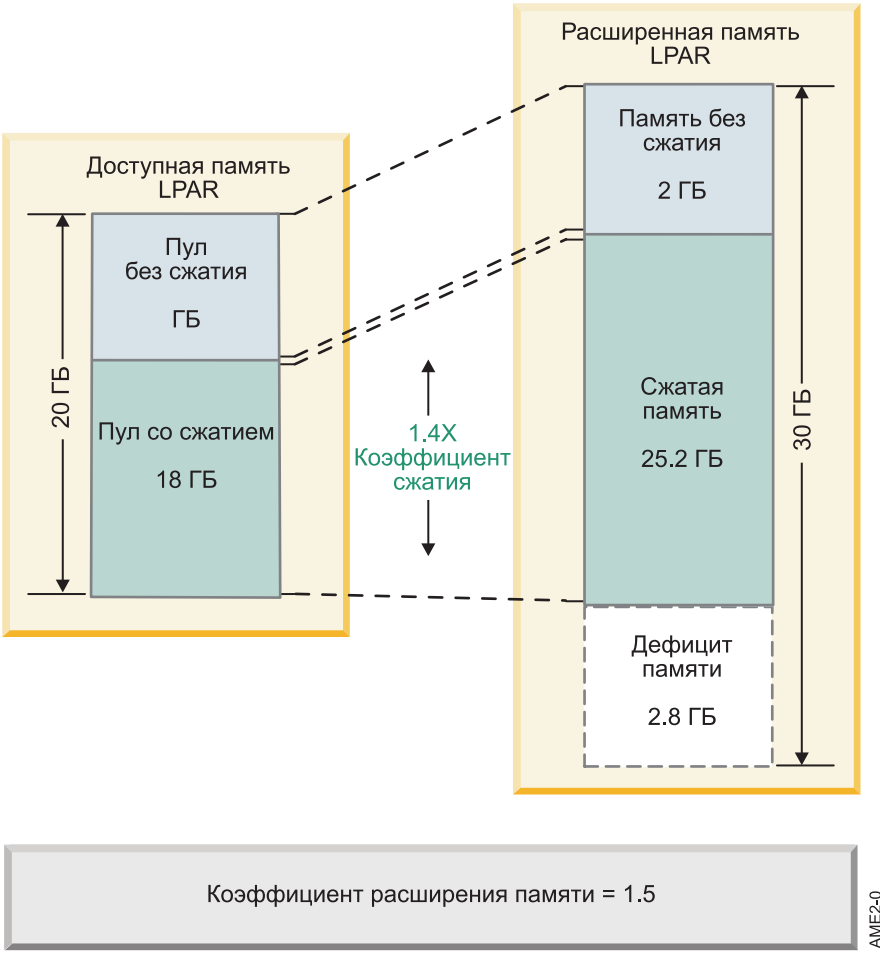
```
40 ГБ = 20 ГБ * 2,0
```

Операционная система сжимает необходимый объем данных в памяти, чтобы уместить 40 ГБ данных в 20 ГБ памяти. Значения коэффициента расширения памяти и объема расширенной памяти можно динамически изменять с помощью Консоли аппаратного обеспечения (HMC)и динамических операций LPAR. Объем расширенной памяти всегда округляется в меньшую сторону до кратного ближайшему логическому блоку памяти (LMB).

Недостаток памяти

Может возникнуть ситуация, когда заданный коэффициент расширения памяти настолько велик, что операционная система не может его применить. При этом появляется недостаток памяти, указывающий на то, что заданный коэффициент расширения памяти LPAR не может быть применен. Будем считать, что размер памяти LPAR равен 20 ГБ и задан коэффициент расширения памяти 1,5. В результате мы должны получить объем расширенной памяти, равный 30 ГБ. Наши данные сжимаются плохо (коэффициент 1,4 к 1). В данном случае невозможно достичь желаемого коэффициента расширения памяти 1.5. Операционная система ограничит объем доступной памяти в пуле сжатой памяти до 95%. Максимальный объем расширенной памяти в нашем случае будет равен 27,6 ГБ. Этот объем меньше желаемого на 2,4 ГБ. Такая ситуация называется недостатком памяти.

В результате мы можем наблюдать те же последствия, как если бы мы выделили слишком мало памяти для LPAR. При недостатке памяти операционная система не сможет выделить необходимый объем расширенной памяти и будет вынуждена использовать пространство подкачки. Т.е. если приложение использует более 27,6 ГБ памяти, то будет задействовано пространство подкачки. Операционная система сообщает о возможном недостатке памяти с помощью соответствующего показателя. Если значение данного показателя равно нулю, то коэффициент расширения памяти может быть применен и операционная система может выделить необходимый объем памяти. При ненулевом значении метрики может возникнуть недостаток памяти. Для предотвращения ситуации недостатка памяти необходимо уменьшить коэффициент расширения памяти LPAR. Обратите внимание, что при этом объем расширенной памяти также уменьшится. Для того чтобы сохранить объем расширенной памяти, необходимо уменьшить коэффициент расширения памяти и установить дополнительную память для LPAR. Объем памяти LPAR и коэффициент расширения памяти можно изменять динамически.



Замечания по планированию

Перед развертыванием системы с функцией Active Memory Expansion (AME) необходимо убедиться, что AME будет использоваться эффективно. Преимущества AME для конкретной системы зависят от параметров системы. В некоторых системах может быть достигнуто большее расширение памяти, чем в других. С помощью программы Active Memory Expansion Planning and Advisory Tool **amepat** можно спланировать и развернуть систему в среде с функцией Active Memory Expansion.

Программа Планирование AME

Программа Планирование AME, расположенная в `/usr/bin/amepat`, предназначена для выполнения следующих основных задач. А именно:

- Планирование начальной конфигурации Active Memory Expansion
- Мониторинг и точная настройка активной конфигурации AME

Программу AME Planning Tool можно запускать на LPAR даже с отключенной функцией AME. Запустите программу **amepat** в системе с отключенной функцией AME. Необходимо задать параметры программы, установив нужную продолжительность мониторинга системы. Например, можно настроить программу **amepat** так, чтобы она выполнялась в течение всей пиковой нагрузки. После завершении работы программа выведет отчет о возможных коэффициентах расширения памяти и ожидаемой загрузке процессора для каждого коэффициента. Значения коэффициентов расширения памяти являются оптимальными для данной системы и предназначены для максимального увеличения объема доступной памяти при снижении нагрузки на процессорОтчеты и рекомендации могут помочь осуществить начальное развертывание AME. В системах с включенной функцией AME программа **amepat** выполняет схожие функции. При выполнении в период пиковой нагрузки она предоставляет отчет о реальной загрузке процессора для каждого коэффициента расширения памяти. При возникновении ситуации недостатка памяти выводится соответствующая информация. Поскольку функция AME включена, программа предоставляет более точные данные о загрузке процессора. На основе полученных данных пользователю предоставляются новые рекомендации.

Пример отчета для раздела с отключенной функцией AME:

```
# amepat 5 2

Команда                         : amepat 2 5

Время запуска                   : Среда, 2 декабря 2009, 11:29:29
Общее время мониторинга         : 10 минут 58 секунд
Количество операций выборки     : 5

Конфигурация системы:
---------------------
Название раздела                : aixfvt19
Тип процессора                  : POWER5
Число процессоров               : 8
Заявленная вмест. проц.         : 4,00
Макс. вместимость проц.         : 4,00
Объем физической памяти         : 4,25 Гб
Нитей SMT                       : 2
Режим общего процессора         : отключено
Active Memory Sharing           : отключено
Active Memory Expansion         : отключено

Использование ресурсов:         среднее       минимальное максимальное
---------------------------     -----------   ----------  -----------
Загрузка проц (физ. проц.)      2,00 [ 50%]   1,00 [ 25%] 3,00 [ 75%]
Объем вирт. памяти (Мб)         1366 [ 31%]   1113 [ 26%] 2377 [ 55%]
Исп. физ. память (Мб)           1758 [ 40%]   1234 [ 28%] 3834 [ 88%]
Зафиксир. память (Мб)           673 [ 15%]     673 [ 15%]  675 [ 16%]
Размер файл. кэша (Мб)          391 [  9%]     124 [  3%] 1437 [ 33%]
Доступная память (МБ)           841 [ 65%]    1812 [ 42%] 3099 [ 71%]
```

```
После включения Active Memory Expansion
---------------------------------------
Планируемое увеличение памяти  :   4,25 Гб
Средний коэффициент сжатия     :   5,29

Коэфф.       Реальный объем    Возможный прирост    Средняя загрузка
расширения   памяти            памяти               процессора
---------    -------------     ------------------   -----------
     1,00          4,25 Гб         0,00 Кб [  0%]   0,00 [  0%]
     1,31          3,25 Гб         1,00 Гб [ 31%]   0,34 [  8%]
     1,55          2,75 Гб         1,50 Гб [ 55%]   0,39 [ 10%]
     1,89          2,25 Гб         2,00 Гб [ 89%]   0,45 [ 11%]
     2,12          2,00 Гб         2,25 Гб [112%]   0,50 [ 12%]
     2,43          1,75 Гб         2,50 Гб [143%]   0,65 [ 16%]
     2,83          1,50 Гб         2,75 Гб [183%]   0,70 [ 18%]

Рекомендации по использованию Active Memory Expansion:
------------------------------------
Для данной системы рекомендуется использовать 1,50 ГБ и настроить коэффициент
расширения памяти 2.83.  Прирост памяти составит 183%. При использовании данных
параметров дополнительная загрузка составит примерно 0,50 процессора, а во
время пиковых нагрузок - 3,50 процессора.

Примечание: рекомендации основываются на данных мониторинга за определенный период.
Если загрузка системы не изменится, то необходимо запустить программу amepat снова.

Загрузка процессора в отчете amepat является приближенным значением.
Фактическая загрузка процессоров может отличаться от указанного
значения в большую или меньшую сторону.
```

Отчет состоит из следующих шести разделов.

Сведения о команде

Приводятся сведения об аргументах команды **amepat**, времени запуска, общем времени мониторинга и количестве операций выборки. В данном отчете команда **amepat**

выполняется 10 минут с интервалом в 2 минуты между 5 операциями выборки.

Примечание: Общее время мониторинга равно 10 минут 58 секунд. Дополнительные 58 секунд используются для сбора сведений о системе, необходимых для моделирования функции Active Memory Expansion.

Конфигурация системы

Приводятся сведения о системе: имя хоста, архитектура процессора, число процессоров, фактический объем памяти, состояние SMT, режимы процессора и памяти. В приведенном отчете функция Active Memory Expansion отключена. Это означает, что программа **amepat** была запущена на разделе с отключенной функцией AME.

Примечание: Для мониторинга и точной настройки активной конфигурации AME можно запустить программу **amepat** в разделе с включенной функцией AME. В этом случае в разделе отчета Конфигурации системы выводится объем расширенной памяти и коэффициент расширения памяти.

Использование ресурсов

Приводятся сведения об использовании системных ресурсов в течение периода наблюдения. Выводятся средние, минимальные и максимальные значения, а также процент использования соответствующих системных ресурсов.

В указанном отчете в среднем используется 2,00 процессора (строка Загрузка процессора), т.е. половина от общего доступного числа (строка Макс. вместимость проц.).

Примечание: Отчет об использовании процессора включает также сведения о времени процессора, затраченном на моделирование AME. Для раздела с включенной функцией AME указанные значения включают операции сжатия и распаковки.

Также приводятся сведения об использовании памяти. Данные в процентах указаны относительно фактического объема памяти LPAR

Примечание: Высокие значения процента использования фиксированной памяти и файлового кэша говорят о том, что система не использует AME эффективно.

Статистика Active Memory Expansion

Данный раздел отображается при выполнении программы **amepat** в разделе с включенной AME.

Пример отчета:

```
Статистика AME:                 среднее       мин.        макс.
---------------                 -----------    --------   ---------
Исп. проц. для AME (физ. проц.) 0,25 [  6%]   0,01 [  0%] 0,50 [ 13%]
Сжатая память (Мб)              264 [ 13%]    264 [ 13%]  264 [ 13%]
Коэффициент сжатия              2,15          2,15        2,16
Недостаток памяти (Мб)          562 [ 55%]    562 [ 55%]   562 [ 55%]

В данном разделе приводятся сведения об использовании процессора для AME, сжатой памяти,
коэффициенте сжатия и недостатке памяти.
Данные о недостатке памяти приводятся только
при достижении объема расширенной памяти.
Если этот объем не достигнут, то сведения о недостатке памяти не выводятся.
В приведенном выше отчете средний недостаток памяти равен 562 Мб, что составляет 55%
от целевого объема расширенной памяти 2 Гб (см. раздел Конфигурация системы).
Из отчета также видно, что в среднем 264 Мб из 2 Гб расширенной памяти
находятся в сжатом виде.
```

После включения Active Memory Expansion

В данном разделе приводятся прогнозируемые данные об объеме расширенной памяти, частоте сжатия, а также таблица с числом возможных конфигураций AME. В данном отчете прогнозируемый объем расширенной памяти равен

4,25 Гб, что соответствует объему физической памяти. По умолчанию **amepat** использует физическую память раздела в качестве прогнозируемой расширенной памяти. Для изменения этих настроек воспользуйтесь опциями

-t или **-a**. Частота сжатия равна 5,29, что означает эффективное сжатие памяти. Если частота сжатия стремится к 1, то сжатие памяти маловероятно

. Конфигурации в таблице основаны на прогнозируемом объеме расширенной памяти.

В таблице также показан прогнозируемый объем физической памяти, прирост памяти, а также дополнительная нагрузка на процессор в связи с использованием AME.

Рассмотрим следующую строку в таблице:

```
1,55          2,75 Гб         1,50 Гб [ 55%]   0,39 [ 10%]
```

Здесь исходный размер фактической памяти (4,25 Гб) можно получить на основе 2,75 Гб физической памяти и коэффициента расширения 1,55. В результате использования данной конфигурации дополнительно будет задействовано

0,39 процессоров (10% от максимума).

Рекомендации по использованию Active Memory Expansion

В данном разделе приведены рекомендации по использованию Active Memory Expansion. В оптимальной конфигурации использование процессора для AME не должно превышать 15%. Также приведены сведения о дополнительной нагрузке процессора и приросте памяти. В приведенном отчете рекомендуется использовать коэффициент расширения 2,12. Целевой объект AME можно изменить с помощью опций **-c** и **-C**.

Примечание: Все приведенные рекомендации основаны на наблюдениях за системой в течение периода мониторинга. Дополнительная нагрузка на процессор является усредненным значением и может отличаться от указанного значения. Программа **amepat** не оценивает изменения в системе на различных уровнях расширения памяти. Перед развертыванием системы в рабочий режим необходимо убедиться в том, что задача повышения производительности системы выполнена. При запуске программы **amepat** в среде с включенной функцией AME будет выводится предупреждение о недостатке памяти, если такая ситуация возникла.

Примечание: Ниже приведено точное сообщения:

```
ЗАМЕЧАНИЕ: Для данного LPAR обнаружен недостаток памяти 562 Мб.
Недостаток памяти вызван слишком высоким значением коэффициента расширения памяти.
Рекомендуется изменить конфигурацию LPAR, чтобы избежать недостатка памяти.
Воспользуйтесь рекомендованными конфигурациями LPAR в приведенной выше таблице.

Для данной системы рекомендуется использовать ...
```

Дополнительные сведения об использовании программы AME Planning приведены на странице справки **amepat** (man).

Отслеживание производительности

Для мониторинга Active Memory Expansion используются несколько инструментов AIX.

В следующей таблице приведены программы и опции, используемые для мониторинга Active Memory Expansion:

Программа	Опция	Описание
amepat	**-N**	Выводит общие сведения об использовании процессора и памяти. Также выводится информация об использовании процессоров для операций сжатия и распаковки AME и сведения о недостатке памяти.
vmstat	**-c**	Выводит статистику сжатия, распаковки и сведения о недостатке памяти.
lparstat	**-c**	Выводит сведения об использовании процессоров для операций сжатия и распаковки AME и сведения о недостатке памяти.
svmon	**-O summary=ame**	Выводит сведения об использовании памяти в пуле сжатой и несжатой памяти.
topas		По умолчанию выводит статистику использования сжатой памяти (в среде с включенной функцией AME).

Команда vmstat

Для отображения статистики AME запустите команду **vmstat** с опцией **–c**.

```
# vmstat -c 2 1

Конфигурация системы: проц=2 пам=1024MB фпам=512MB выд=0.40 режим=dedicated-E

нити                память                        страница          ошибки
 r   b     avm    fre     пжп    дп    недпам  op    oc  pi  po  in    sy    cs
```

```
 0   0  309635  2163  43332   943  26267  174  386   0   0  93  351  339

процессор
us sy id wa    pc    ec
2  3 89  7  0.02   5.3
```

В приведенном выше примере содержатся следующие данные:

- Объем расширенной памяти **пам** равен 1024 Мб.
- Объем физической памяти **фпам** равен 512 Мб.
- Режим памяти **режим**: Active Memory Sharing выключено, Active Memory Expansion - включено.
- Размер пула сжатой памяти **псп** равен 43332 страницы по 4 Кб.
- Объем доступной памяти **дп** в пуле сжатой памяти равен 943 страницы по 4 Кб.
- Недостаток памяти **недпам** равен 26267 страниц по 4 Кб.
- Число операций сжатия или записи страниц в пул сжатой памяти в секунду **ос** равно 386.
- Число операций распаковки или извлечения страниц из пула сжатой памяти в секунду **ор** равно 174.

команда lparstat

Для отображения статистики AME запустите команду **lparstat** с опцией **–c**.

```
# lparstat -c 2 5

Конфигурация системы: тип=общий режим=без_ограничений  Режим_памяти=Ded-E smt=включено
проц=2 пам=1024 Мб фпам=512MB разм=14 огр=0.40

%польз %сис   %ожид  %прос физ %огр    зан    прил  vcsw phint  %xcpu    dxm
----- ----- ------ ------ ----- ----- ------   --- ----- ----- ------ ------
 45,6  51,3    0,2    2,8  0,95 236,5   62,6 11,82  7024     2    5,8    165
 46,1  50,9    0,1    2,8  0,98 243,8   64,5 11,80  7088     7    6,0    162
 46,8  50,9    0,3    2,1  0,96 241,1   69,6 11,30  5413     6   19,4    163
 49,1  50,7    0,0    0,3  0,99 247,3   60,8 10,82   636     4    8,6    152
 49,3  50,5    0,0    0,3  1,00 248,9   56,7 11,47   659     1    0,3    153
```

В приведенном выше примере содержатся следующие данные

- Режим памяти **режим**: Active Memory Sharing выключено, AME Memory Expansion - включено.
- Объем расширенной памяти**пам** равен 1024 Мб.
- Объем физической памяти **фпам** равен 512 Мб.
- Процент использования процессора в операциях Active Memory Expansion **%xcpu.**
- Недостаток памяти **dxm** в Мб.

Команда topas

Основная панель **topas** в логическом разделе с включенной технологией Active Memory Expansion

отображает данные о сжатии в разделе **AME.**

```
Монитор Topas для хоста: proc7                СОБЫТИЯ/ОЧЕРЕДИ    ФАЙЛЫ/TTY
Пон 14 Дек 2009   16:30:50 Интервал: 2         Cswitch    1240  Readch  43,2 Мб
                                                Свыз     110,8 Кб  Writech 102,5 Кб
Проц  польз%  ядр%  ожид%  прост%  физ   огр    Чтений    12594  Rawin     0
ВСЕ   49,1   50,7    0,0    0,3   1,00  249,7   Записи      515  Ttyout   388
                                                Forks       218  Igets      0
Сеть     KBPS   I-Pack  O-Pack   KB-In  KB-Out  Execs       218  Namei    5898
Всего     1.2      7,5     1,0     0,9     0,3  Runqueue    1,0  Dirblk      0
                                                Waitqueue   0,0
Диск    Занято%     KBPS     TPS KB-Read KB-Writ                   ПАМЯТЬ
Всего    0,0      0,0      0,0     0,0     0,0  PAGING           Физ,Мб   1024
                                                Сбой   53184  % Comp     85
Файл. система     KBPS     TPS KB-Read KB-Writ  Steals        0  % Noncomp   0
```

```
Всего       75,4 Кб   21,1 КБ  75,3 КБ  95,4    PgspIn        0  % Client    0
                                                  PgspOut       0
WLM-Class (Active)     CPU%    Mem%  Blk-I/O%   PageIn        0  ПРОСТРАНСТВО ПОДКАЧКИ
Система                   0      61         0   PageOut       0  Размер,Мб    512
По умолчанию              0       4         0   Sios          0  % использ      1
                                                                  % свободно     99
Имя             PID CPU% PgSp Class            AME
inetd        364682  3,5  0,5 wpar1            TMEM,Мб     512  Активн WPAR  1
xmtopas      622740  0,4  0,7 wpar1            CMEM,Мб     114  Всего WPAR   1
topas        413712  0,1  1,5 System           EF[T/A] 2.0/1.5  Нажмите: "h" для справки
random       204934  0,1  0,1 System           CI:5.5  CO:0.0          "q" для выхода
```

В приведенном выше примере содержатся следующие данные.

- Объем физической памяти **TMEM, Мб** равен 512 Мб.
- Размер пула сжатой памяти **CMEM,Мб** равен Мб.
- **EF[T/A]** – целевой коэффициент расширения равен 2,0, фактический коэффициент равен 1,5.
- Частота сжатия **co** и распаковки **ci** в секунду равна 0,0 и 5,5 страниц соответственно.

svmon, команда

Программа **svmon** выводит подробные сведения об использовании AME.

```
# svmon -G -O summary=ame,pgsz=on,unit=MB
Ед. изм: Мб
--------------------------------------------------------------------------------
               разм.     занято      своб.    фикс.   вирт.     дост.    mmode
память      1024,00      607,54     144,11   306,29  559,75    136,61   Ded-E
 несжатая         -      387,55          -
 сжатая           -      219,98          -
пр. подк.   512,00        5,08

               раб.       пост.       клнт.        проч.
фикс.        213,34           0           0        28,9
занято       534,12           0        9.42
 несжатая    314,13
 сжатая      219,98

разм.стр.   разм.пула   занято   пр.пдкчк.      закр.      вирт.        несж.
s    4 Кб         -      543,54        5,02      242,27     560,59      323,55
L   16 Мб         4           0           0       64.0          0           0
--------------------------------------------------------------------------------
Объем физической памяти: 512,00

              CurSz  %Cur       TgtSz        %Tgt       MaxSz   %Max   CRatio
несжатая     405,93  79,28      168,38       32,89      -          -    -
сжатая       106,07  20,72      343,62       67,11      159,59   31,17  2,51

                txf         cxf         dxf         dxm
AME            2,00        1,46        0,54      274,21
```

В приведенном выше примере содержатся следующие данные:

- Режим памяти **режим**: Active Memory Sharing выключено, AME Memory Expansion - включено.
- Всего памяти: 607,54 Мб (**memory_inuse**). Несжатые страницы **ucomprsd_inuse** занимают 387,55 Мб. Сжатые страницы **comprsd_inuse** занимают оставшиеся 219,98 Мб
- Всего рабочих страниц: 534,12 Мб (**inuse_work**). Несжатые страницы **ucomprsd_work** занимают 314,13 Мб. Сжатые страницы **comprsd_work** занимают 219,98 Мб.
- Всего используется 543,54 Мб страниц (**4KB_inuse**) в пуле страниц по 4 Кб. Несжатые страницы**4KB_ucomprsd** занимают 323,55 Мб.
- Объем расширенной памяти **memory_size** равен 1024 Мб.
- Объем физической памяти **True Memory** равен 512 Мб.

- Текущий размер пула несжатой памяти **ucomprsd_CurSz** равен 405,93 Мб (79,28% от общего объема памяти).
- Текущий размер пула сжатой памяти **comprsd_CurSz** равен 106,07 Мб (20,72% от общего объема памяти).
- Целевой размер пула сжатой памяти **comprsd_TgtSz**, необходимый для достижения коэффициента расширения памяти **txf** 2,00, равен 343,62 Мб(67.11% от общего объема памяти).
- Текущий размер пула несжатой памяти **ucomprsd_TgtSz** равен 168,38 Мб (32,89% от общего объема памяти).
- Максимальный размер пула сжатой памяти **comprsd_MaxSz** равен 159,59 Мб (31,17% от общего объема памяти).
- Текущая частота сжатия **CRatio** равна 2,51, а текущий коэффициент расширения **cxf** равен 1,46
- Недостаток памяти **dxm** составляет 274,21 Мб, а коэффициент расширения недостатка **dxf** равен 0,54.

Опция **–O summary=longame** выводит краткие сведения о сжатой памяти в следующем виде:

```
# svmon -G -O summary=longame,unit=MB
Ед. изм: Мб
                                Active Memory Expansion
---------------------------------------------------------------
 Размер   Исп    Своб    DXMSz  UCMInuse  CMInuse   TMSz    TMFr
1024,00  607,91  142,82  274,96   388,56    219,35   512,00  17,4

CPSz  CPFr  txf  cxf   CR
106,07 18,7 2,00 1,46   2,50
```

В приведенном выше примере содержатся следующие данные:

- Общий объем расширенной памяти **Размер** равен 1024,00 Мб. Используется **Исп** 607,91 Мб. Свободно **Своб** 142,82 Мб. Недостаток памяти **DXMSz** равен 274,96 Мб.
- Общий объем используемой памяти **Исп** равен 607,91 Мб. Объем несжатых страниц **UCMInuse** равен 388,56 Мб. Объем сжатых страниц **CMInuse** равен 219,35 Мб.
- Общий объем физической памяти **TMSz** равен 512,00 Мб. Доступно только 17,4 Мб **TMFr**.
- Общий размер пула сжатой памяти **CPSz** равен 106,07 Мб. Доступно только 18,7 Мб **CPFr**.
- При этом целевой коэффициент расширения **txf** равен 2,00, а текущий **cxf** равен 1,46.
- коэффициент сжатия (КС) равен 2,50.

Информация, связанная с данной:

Команда vmo

Настройка приложений

Прежде чем прилагать большие усилия для повышения производительности программы, попробуйте определить, насколько можно повысить производительность программы, и применительно к каким фрагментам программы оптимизация и настройка дадут наибольший эффект. В этой главе описаны способы выполнения этих задач.

В общем случае, процесс оптимизации состоит из нескольких этапов:

- В некоторых случаях требуется изменить исходный код, например, путем переупорядочивания операторов и выражений. Этот прием называется *настройкой вручную*.
- Для программ на языках FORTRAN и C предусмотрены оптимизирующие препроцессоры, настраивающие и преобразующие исходный код до его компиляции. Результатом работы препроцессоров FORTRAN и C является оптимизированный исходный код.
- Компиляторы FORTRAN и C++ преобразуют исходный код в код на промежуточном языке.
- Генератор кода преобразует промежуточный код в код на машинном языке. В зависимости от выбранных опций компиляции, генератор кода может оптимизировать окончательный исполняемый код для повышения его производительности. С помощью препроцессора или вручную вы можете задать уровень оптимизации, выполняемой на этом этапе.

На повышение производительности влияют два фактора:

- Число операций по оптимизации, выполненных по отношению к отдельным фрагментам программы
- Частота использования этих фрагментов во время выполнения программы

Повышение производительности отдельной функции может значительно ускорить работу программы, если эта функция выполняет основной объем работы, и, напротив, почти не повлиять на общую производительность программы, если эта функция вызывается редко и выполняется сравнительно недолго. Помните об этом при выборе способов настройки производительности и объектов для настройки.

Более подробно эти способы рассмотрены в книге *Optimization and Tuning Guide for XL Fortran, XL C and XL C++*. За дополнительной информацией обратитесь к разделу “Эффективное проектирование программы и ее реализация” на стр. 89.

Приемы оптимизации компиляции

Описаны методы оптимизации с помощью компилятора.

Средства и приемы оптимизации исходного кода можно разбить на три основные категории:

- Приемы программирования, основанные на использовании оптимизирующих компиляторов и особенностей системной архитектуры.
- Библиотека BLAS (библиотека элементарных линейных алгебраических преобразований). Если программа выполняет большой объем вычислений, то применение функций из этой библиотеки может дать существенный выигрыш в производительности. Расширением BLAS является ESSL, библиотека функций для научно-инженерных расчетов. Помимо BLAS, библиотека ESSL содержит и другие высокоэффективные математические процедуры для физики, химии и инженерного дела. Для компьютеров SMP предоставляется пакет Параллельный ESSL (PESSL).
- Опции компилятора и применение препроцессоров типа KAP и VAST других фирм.

Помимо вышеперечисленных средств, можно воспользоваться программой изменения структуры объектного кода **fdpr**. Работа программы **fdpr** описана в разделе “Изменение структуры исполняемых программ с помощью программы fdpr” на стр. 121.

Компиляция с оптимизацией

При создании эффективной программы прежде всего необходимо воспользоваться встроенными средствами оптимизации компилятора.

Оптимизирующая компиляция позволяет повысить быстродействие программы и сократить объем дальнейших действий по настройке производительности.

Рекомендации

Ниже приведены некоторые рекомендации:

- При компиляции программы на FORTRAN, C или C++ укажите опцию **-O2** или **-O3 -qstrict**. В случае высокопроизводительных программ FORTRAN (HPF) не указывайте опцию **-qstrict**.
- Если основной объем операций в программе приходится на обработку циклов или массивов, укажите опцию **-qhot**. Всегда указывайте опцию **-qhot** для программ HPF.
- Если время компиляции несущественно, укажите опцию **-qipa** в конце цикла разработки.

Опция **-qipa** предназначена для применения процедур оптимизации, называемых *межпроцедурным анализом*. У опции **-qipa** есть несколько подопций, описанных в руководстве по компилятору. Ее можно применять двумя способами:

- Первый способ: укажите опцию **-qipa** как на этапе компиляции, так и на этапе компоновки. Во время компиляции информация межпроцедурного анализа сохраняется в файле `.o`. Во время компоновки опция **-qipa** означает, что должна быть выполнена повторная компиляция всего приложения.

- Второй способ: откомпилируйте программу для профилирования с опцией **-p**/**-pg** (с опцией **-qipa** или без нее), а затем запустите ее со стандартным набором данных. При последующей компиляции укажите отчет программы профилирования с опцией **-qipa**, чтобы компилятор мог выделить наиболее часто используемые фрагменты программы и оптимизировать их прежде всего.

Флаг **-O4** эквивалентен флагам **-O3 -qipa** с автоматическим созданием архитектуры и оптимальной настройкой для платформы. Флаг **-O5** эквивалентен флагу **-O4** с опцией **-qipa** = level = 2.

Оптимизация с помощью компилятора дает следующие преимущества:

Оптимизация ветвления
Код программы реорганизуется с целью минимизировать число операций перехода и объединить различные блоки кода.

Переупорядочение кода
Если переменные, используемые в цикле, не изменяются в его теле, то такие переменные могут быть вычислены заранее вне цикла.

Исключение повторных вычислений
В некоторых стандартных выражениях одно и то же значение вычисляется несколько раз. В этом случае повторное вычисление можно исключить.

Применение констант
На основе констант, применяемых в выражении, генерируются новые константы. При этом выполняются некоторые неявные преобразования целых и вещественных типов.

Исключение лишнего кода
Исключается код, который никогда не используется или не влияет на дальнейшие вычисления.

Исключение лишних операций сохранения
Исключаются операции по сохранению не используемых в дальнейшем значений. Например, если между двумя операциями сохранения нет операций по извлечению значения, то первая операция сохранения не нужна, поэтому она удаляется.

Глобальное распределение по регистрам
Переменные и выражения распределяются по доступным аппаратным регистрам с помощью алгоритма "раскрашивания графа".

Встраивание
Вызовы функций заменяются на их код

Планирование команд
Машинные команды переупорядочиваются с целью минимизировать время выполнения

Межпроцедурный анализ
Обрабатываются перекрестные ссылки и исключаются операции загрузки, сохранения и вычисления, которые нельзя было исключить ранее.

Модификация инвариантного кода IF (без переключения)
Из циклов удаляется инвариантный код перехода с целью расширить возможности других способов оптимизации.

Учет результатов профилирования
Результаты выполнения модельных программ используются для оптимизации условных переходов и часто исполняемых фрагментов кода.

Изменение порядка операций
Порядок вычисления элементов массивов изменяется с целью предоставить дополнительные возможности для исключения повторных вычислений.

Перемещение операций сохранения
Команды сохранения выносятся за пределы циклов.

Повышение эффективности команд
Менее эффективные команды заменяются на более эффективные. Например, при работе с индексами элементов массива команда умножения заменяется на команду добавления.

Обработка числовых значений
Включает создание новых констант, исключение одинаковых выражений, свертывание нескольких инструкций в одну и т.п.

Условия для компиляции без оптимизации

Не указывайте опцию **-O** для программ, которые вы собираетесь отлаживать с помощью символьного отладчика, независимо от того, применяется ли опция **-g**. Однако для программ HPF рекомендуется указывать опции **-O3 -qhot** даже во время отладки, поскольку оптимизация таких программ крайне важна.

Оптимизатор реорганизует инструкции на ассемблере, после чего становится довольно сложно сопоставить инструкции отдельным строкам исходного кода. Если при компиляции была указана опция **-g**, то из-за реорганизации при запуске символьного отладчика может создаться впечатление, что операторы исходного кода выполняются в неправильном порядке.

Если программа выдает неверные результаты при компиляции с любой из опций **-O**, то причиной может быть наличие псевдонимов переменных в вызовах процедур.

Компиляция для работы на определенных аппаратных платформах

Приведены замечания по компиляции программ для определенных аппаратных платформ.

В системах устанавливаются процессоры различных типов. С помощью опций **-qarch** и **-qtune** вы можете оптимизировать программы с учетом специальных команд и особенностей этих процессоров.

Рекомендации

Ниже приведены некоторые рекомендации по компиляции для конкретных аппаратных платформ:

- Если программа будет работать только в одной системе, либо в группе систем с процессорами одного типа, то укажите тип процессора с помощью опции **-qarch**.
- Если программа будет выполняться в системах с различными типами процессоров, то с помощью параметров **-qarch** и **-qtune** вы можете указать один тип процессора как наиболее приоритетный. Для динамической настройки этих параметров пользователи XL FORTRAN и XL HPF могут воспользоваться командами **xxlf** и **xxlhpf**.
- Если нельзя сказать, что программа предназначена преимущественно для какого-то одного типа процессоров, и она будет использоваться в системах с разными типами процессоров, то параметры **-qarch** и **-qtune** использовать не нужно.

Компиляция для оптимизации операций с плавающей точкой

Некоторые опции оптимизации операций над вещественными числами, применяемые по умолчанию, можно изменить с целью повысить производительность программ, выполняющих большое число таких операций.

Некоторые из этих опций могут повлиять на соответствие стандартам операций над вещественными числами. Применение таких опций в принципе может повлиять на результаты, однако в большинстве случае оно повышает точность вычислений.

Рекомендации

Ниже приведены некоторые рекомендации:

- В случае программ с одинарной точностью операций, предназначенных для платформ семейство POWER и POWER2, вы можете повысить производительность, не снижая точность, с помощью следующих опций:
  ```
  -qfloat=fltint:rsqrt:hssngl
  ```

Если программа с одинарной точностью не очень активно работает с памятью (например, программа не запрашивает больше данных, чем умещается в кэше), то вы можете повысить точность вычислений, сохранив или повысив производительность, с помощью следующих опций:

```
-qfloat=fltint:rsqrt -qautodbl=dblpad4
```

Для программ, не содержащих переменных одинарной точности, следует указывать только опцию **-qfloat=rsqrt:fltint**. Учтите, что опция **-O3** без опции **-qstrict** автоматически задает режим **-qfloat=rsqrt:fltint**.

- Программы с одинарной точностью обычно более эффективны, чем программы с двойной точностью, поэтому замена типа REAL (по умолчанию) на тип REAL(8) может снизить производительность. Поддерживаются следующие подопции **-qfloat**:

Выбор размера кэша

Если программу предполагается применять только в одной системе или конфигурации, то вы можете предоставить компилятору дополнительную информацию о конфигурации памяти компьютера, указав опцию FORTRAN **-qcache**.

Вместе с опцией **-qcache** нужно указать опцию **-qhot**. Опция **-qhot** определяет подходящие варианты оптимизации управления памятью на основе информации **-qcache**.

Существуют кэши трех типов: для данных, для команд и комбинированный. Все системы подразделяются на две категории: системы с кэшем данных и кэшем команд и системы с одним комбинированным кэшем. Подопция TYPE позволяет указать, к какому типу кэша относится опция **-qcache**.

С помощью опции **-qcache** можно также задать размер и уровень ассоциативности кэша второго уровня и таблицы преобразования адресов (TLB) - таблицы, которая служит для поиска недавно использовавшихся страниц памяти. Опцию **-qcache** не нужно указывать для TLB, за исключением случаев, когда программа использует область данных размером более 512 КБ.

В некоторых случаях уменьшение значения атрибута SIZE приводит к повышению производительности. Это зависит от загруженности системы во время выполнения программы.

Встраивание кода функций

Встраивание означает замену вызова функции на ее код. Это дает возможность не выполнять дополнительные операции по вызову функций и оптимизировать код внутри встраиваемых функций.

В случае программ на FORTRAN и C вы можете указать опцию **-Q** (вместе с **-O2** или **-O3**) для встраивания функций путем замены вызова функции на ее код.

В некоторых случаях встраивание повышает производительность, в некоторых - наоборот. Снижение производительности возможно в случае, если размер кода становится слишком большим, что повышает нагрузку на кэш и частоту подкачки страниц, либо если в системе недостаточно регистров для хранения всех локальных переменных встроенной функции.

При использовании опции **-Q** обязательно сравните производительность версии программы, откомпилированной с опциями **-O3** и **-Q**, и версии, откомпилированной только с опцией **-O3**. Производительность программ, откомпилированных с опцией **-Q**, может быть значительно выше, значительно ниже или почти такой же.

Компилятор решает вопрос о встраивании функций в зависимости от их размера. Возможно, вам удастся повысить производительность приложения, задав другие критерии встраивания. Для функций, вызов которых в обычной ситуации маловероятен (примерами могут служить функции отладки и исправления ошибок), рекомендуется выборочно отключить встраивание с помощью опции **-Q-имена**. Для часто используемых функций, наоборот, рекомендуется выбрать обязательное встраивание, указав опцию **-Q+имена**.

Использование динамического и статического подключения

В операционной системе предусмотрены средства создания и применения динамически подключаемых общих библиотек. При динамическом подключении внешние идентификаторы, встречающиеся в пользовательском коде и определенные в общей библиотеке, преобразуются загрузчиком на этапе загрузки. Во время компиляции программы, применяющей общие библиотеки, они динамически связываются с ней по умолчанию.

Смысл использования общих библиотек заключается в хранении одной копии стандартных библиотек, доступной всем приложениям. Это позволяет значительно сократить размер исполняемых программ и тем самым сэкономить место на диске.

Вы можете сократить размер программ за счет применения динамической компоновки, однако, как правило, это приводит к снижению производительности. Код общих библиотек хранится на диске не вместе с исполняемым кодом, а в отдельном файле библиотеки. Общий код загружается в память один раз и используется всеми процессами, в которых есть ссылка на него. Таким образом, применение динамически подключаемых библиотек позволяет снизить объем виртуальной памяти, применяемой программой, при условии, что несколько параллельных приложений (или экземпляров одного приложения) обращаются к общей библиотеке. Кроме того, это позволяет сократить объем дисковой памяти, необходимой программе, при условии, что несколько различных приложений данной системы обращаются к общей библиотеке. Ниже перечислены другие преимущества применения общих библиотек:

- Возможное снижение времени загрузки, поскольку код общих библиотек может быть уже загружен в память.
- Возможное ускорение выполнения, поскольку вероятность того, что операционная система выгрузит код общих библиотек, значительно ниже в случае, когда этот код используется несколькими приложениями или несколькими экземплярами одного приложения, а не одним приложением. Следовательно, в этом случае подкачка страниц выполняется реже.
- Функции подключаются к приложению не статически, а динамически, при его загрузке. Это позволяет программам автоматически использовать изменения, внесенные в библиотеки, без перекомпиляции и перекомпоновки.

Однако динамическая компоновка обладает и некоторыми недостатками:

- Для обращения к коду общих библиотек в исполняемой программе должна быть предусмотрена специальная "процедура интеграции". На каждое обращение к функции из общей библиотеки затрачивается примерно восемь машинных циклов. Программы, обращающиеся к общим библиотекам, обычно работают медленнее по сравнению с теми, в которых применяется статическая компоновка библиотек.
- Менее явным эффектом является возможный разброс ссылок. Программа может использовать только несколько функций из общей библиотеки, удаленных друг от друга в пространстве виртуальных адресов библиотеки. Таким образом, для обращения к функции нужно просмотреть намного больше страниц, чем в случае, когда все функции напрямую связаны с исполняемой программой. В частности, если вы являетесь единственным пользователем этих функций, то при их загрузке в оперативную память возникнет большое число страничных ошибок. Кроме того, из-за большого числа просматриваемых страниц может возникнуть промах при обращении к таблице TLB.
- Когда программа обращается к ограниченному числу функций библиотеки, каждая страница библиотеки, содержащая нужную функцию, загружается в память отдельно. Если функции невелики, то при статической компоновке несколько функций из различных областей библиотеки могут быть размещены на одной странице, что повышает производительность по сравнению с динамической компоновкой.
- Программы с динамической компоновкой зависят от наличия совместимой библиотеки. В случае изменения библиотеки (например, в новом выпуске компилятора) может потребоваться модифицировать приложения с целью обеспечить совместимость с новой версией библиотеки. В случае удаления библиотеки использующие ее программы перестают работать.

В программах со статической компоновкой весь код содержится в одном исполняемом модуле. Ссылки на библиотеки обрабатываются более эффективно, поскольку библиотечные функции статически подключены к

программе. Статическая компоновка увеличивает размер файла, содержащего программу, а также, возможно, размер кода в памяти, если в системе запущены другие приложения или другие экземпляры того же приложения.

По умолчанию команда **cc** использует поддержку общих библиотек. Если вы хотите переопределить значение по умолчанию для создания статически подключаемых объектных файлов, при компиляции программы укажите опцию **-bnso**:

```
cc xxx.c -o xxx.noshr -O -bnso -bI:/lib/syscalls.exp
```

Эта опция означает, что компоновщик должен поместить библиотечные функции, на которые ссылается программа, в объектный файл. Файл `/lib/syscalIs.exp` содержит имена системных функций, которые следует импортировать в программу из системы. Этот файл обязателен при статической компоновке. При динамической компоновке указывать этот файл необязательно, поскольку перечисленные в нем функции автоматически импортируются с помощью модуля `libc.a`. Дополнительные сведения об этих опциях приведены в разделе “Повышение эффективности работы команды ld” на стр. 409 и описание работы команды **ld**.

Повышение производительности с помощью статически связанных библиотек:

Один из способов определить, зависит ли эффективность работы программы от наличия общих библиотек, заключается в повторной компиляции исполняемой программы с опцией отключения общих библиотек.

Если производительность заметно увеличилась, вы можете отказаться от применения общих библиотек в пользу быстродействия. Не забудьте, что измерения нужно проводить в рабочей среде. В слабо загруженной системе программа без использования общих библиотек обычно работает быстрее. Однако когда в системе работает много пользователей, запуск такой программы может привести к переполнению памяти и снижению общей производительности системы.

Предварительная загрузка библиотек

Общие библиотеки, которые должны быть предварительно загружены процессом, указываются в переменных среды*LDR_PRELOAD* и *LDR_PRELOAD64*. Переменная среды *LDR_PRELOAD* предназначена для 32-разрядных процессоров, а *LDR_PRELOAD64* - для 64-разрядных.

Поиск имён функций начинается с библиотек, перечисленных в переменных среды, и, если он не дал результатов, выполняется обычный поиск. Замещение символов из предварительно загруженных библиотек работает как для компоновки AIX по умолчанию, так и для динамической компоновки. Поведение отложенной обработки имён не меняется.

Более подробная информация об этих переменных среды приведена в разделе “Прочие настраиваемые параметры” на стр. 429.

Выбор порядка компоновки больших программ для снижения интенсивности подкачки

На этапе компоновки во время компиляции программы компоновщик переупорядочивает блоки программы, пытаясь повысить компактность ссылок.

Например, если функция вызывает другую функцию, компоновщик может разместить их рядом в загрузочном модуле, так чтобы обе функции уместились в одной странице виртуальной памяти. Это снижает интенсивность подкачки. После того как при первом обращении к первой функции содержащая ее страница будет загружена в память, вторая функция будет готова к использованию без дополнительных операций подкачки.

Если размер программы очень велик и подкачка выполняется очень интенсивно, вы можете задать определенный порядок компоновки. Для этого расположите управляющие разделы в нужном порядке и с помощью опции **-bnoobjreorder** запретите компоновщику изменять этот порядок. Управляющий раздел, или

CSECT, - это минимальный блок кода или данных в объектном модуле XCOFF, который может быть перемещен. Более подробная информация приведена в книге *Справочник по файлам* .

Однако определение жесткого порядка компоновки обладает и рядом недостатков. Любое изменения порядка должно сопровождаться тщательным тестированием с целью убедиться, что заданный порядок компоновки обеспечивает более высокую производительность по сравнению с тем, который выбирает компоновщик. Перед выбором собственного порядка компоновки обратите внимание на следующее:

- Вы должны определить порядок компоновки для всех управляющих разделов (CSECT) в программе. Разделы CSECT должны быть расположены в том порядке, в котором они будут скомпонованы. Если программа велика, вам придется выполнить большой объем работы, и вероятность ошибиться будет весьма высока.
- Предполагаемый выигрыш в производительности может обернуться проигрышем, поскольку при изменении размера кода разделы CSECT, ранее расположенные вместе на странице, могут быть разнесены по разным страницам.
- Изменение порядка компоновки может повысить частоту конфликтов строк в кэше команд. При наличии кэша команд или комбинированного кэша данных и команд с двукратной ассоциативностью любая строка кода может храниться в одной из двух строк кэша. Если три или более коротких независимых функции относятся к одному классу подобия кэша, то конфликт в кэше команд может привести к снижению производительности. Изменение порядка компоновки может вызвать появление новых, ранее не существовавших конфликтов. Возможен и обратный эффект - снижение числа конфликтов по сравнению со случаем, когда опция **-bnoobjreorder** не задана.

Если вы изменяете порядок компоновки программы, обязательно протестируйте ее в системе, в которой общий объем памяти и показатели ее использования другими программами близки к аналогичным параметрам предполагаемой рабочей среды. Порядок компоновки, оправдывающий себя в системе с небольшим числом задач, может привести к конфликтам подкачки в загруженной системе.

Вызов библиотек BLAS и ESSL

Библиотека BLAS (библиотека элементарных линейных алгебраических преобразований) обеспечивает эффективное выполнение линейных алгебраических преобразований над матрицами и векторами. Ее расширение, библиотека ESSL (библиотека функций для научно-инженерных расчетов), содержит дополнительные функции. Она специально оптимизирована для архитектуры семейства C процессором POWER family, POWER2 и PowerPC.

Применение BLAS и ESSL позволяет значительно ускорить выполнение многих арифметических операций и дает выигрыш в производительности даже по сравнению с настройкой вручную или автоматической оптимизацией этих операций. Функции из обеих библиотек можно вызывать в программах на FORTRAN, C и C++.

Библиотека BLAS содержит функции, специально оптимизированные для применяемой архитектуры. Она поставляется вместе с операционной системой (`/lib/libblas.a`).

Пользователям настоятельно рекомендуется применять эту библиотеку при выполнении операций над матрицами и векторами, поскольку уровень оптимизации функций из этой библиотеки намного выше того уровня, которого можно добиться путем настройки вручную.

Функции BLAS предназначены для использования в программах на FORTRAN, но могут вызываться и в программах на C. При работе с матрицами необходимо учесть различия между этими языками. Например, в языке FORTRAN матрицы хранятся по столбцам, а в языке C - по строкам.

Для включения в программу библиотеки BLAS, хранящейся в файле `/lib/libblas.a`, укажите опцию **-lblas** в команде компиляции (**xlf -O prog.f -lblas**). Если библиотека BLAS применяется в программе на C, дополнительно укажите опцию **-lxlf**, так как эта библиотека написана на языке FORTRAN (**cc -O prog.c -lblas -lxlf**).

Библиотека ESSL содержит более широкий набор математических функций, применяемых в физических, химических и инженерных расчетах.

Ниже перечислены некоторые преимущества библиотек BLAS и ESSL:

- Если применяются функции BLAS и ESSL, то не нужно кодировать соответствующие им операции.
- Функции BLAS и ESSL не зависят от платформы. Имена функций и формат вызова стандартны.
- Функции BLAS выполняются эффективно на любой платформе. Внутренняя реализация функций обычно привязана к конкретной платформе и учитывает ее особенности.

В примере программы следующие девять строк кода на языке FORTRAN:

```
do l=1,control
do j=1,control
        xmult=0.d0
        do k=1,control
                xmult=xmult+a(i,k)*a(k,j)
        end do
        b(i,j)=xmult
end do
end do
```

были заменены на следующую строку FORTRAN, вызывающую функцию библиотеки BLAS:

```
call dgemm (`n','n',control,control,control,1,d0,a, control,a,1control,1.d0,b,control)
```

Это привело к следующему повышению производительности:

Размерность массива	Затраченное время - MULT	Затраченное время - BLAS	Коэффициент
101 x 101	0,1200	0,0500	2,40
201 x 201	0,8900	0,3700	2,41
301 x 301	16,4400	1,2300	13,37
401 x 401	65,3500	2,8700	22,77
501 x 501	170,4700	5,4100	31,51

Этот пример демонстрирует эффективность применения функции уровня 3 BLAS вместо обычных операций умножения матриц. Обратите внимание, что эффективность повышается с увеличением размерности массива.

Учет результатов профилирования

PDF - это опция компилятора, позволяющая выполнить дополнительную оптимизацию на уровне функций: распределение регистров, планирование команд, реорганизацию блоков и т.п.

Для применения PDF выполните следующие действия:

1. Откомпилируйте исходные файлы программы с опцией **-qpdf1** (а также функцию **main()**). На этапе компоновки укажите опцию **-lpdf**. Если вы применяете другие опции компиляции, то все они должны быть указаны на шаге 3.
2. Выполните программу со стандартным набором данных. По окончании работы программа запишет информацию профилирования в файл .__BLOCKS в каталоге, указанном в переменной среды *PDFDIR*, либо в текущем рабочем каталоге, если эта переменная не задана. Вы можете выполнить программу несколько раз с различными наборами данных. В этом случае вы получите совокупную информацию профилирования, предоставляющую более точные сведения о частоте отдельных переходов и частоте выполнения различных блоков кода. Важно, чтобы данные, которые вы вводите, соответствовали тем, которые вы будете применять при дальнейшей работе с программой.
3. Еще раз откомпилируйте программу с теми же опциями, что и на шаге 1, но вместо **-qpdf1** укажите **-qpdf2**. Помните, что **-L** и **-l** - это опции компоновщика, и вы можете изменить их на этом этапе; в частности, опустите опцию **-lpdf**. Во время второй компиляции программа дополнительно

оптимизируется на основе совокупной информации профилирования. В результате программа не будет содержать "узких мест" и будет работать с максимальной скоростью.

Для работы с каталогом, указанным в переменной *PDFDIR*, предусмотрены две команды:

resetpdf *путь*
Удаляет всю информацию профилирования (но не файлы данных) из указанного каталога. Если каталог не задан, то по умолчанию применяется каталог, указанный в переменной *PDFDIR*, а если переменная *PDFDIR* не задана, то текущий каталог. При внесении изменений в приложение и повторной компиляции некоторых файлов информация профилирования для этих файлов автоматически сбрасывается. После внесения существенных изменений, которые могут повлиять на показатели выполнения фрагментов программы, не компилировавшихся заново, вызовите команду **resetpdf** для сброса информации профилирования для всего приложения.

cleanpdf *путь*
Удаляет всю информацию профилирования из указанного каталога, каталога, заданного в переменной *PDFDIR*, или текущего каталога. Удаление информации профилирования сокращает нагрузку во время выполнения, если вы измените программу и затем еще раз выполните процедуру PDF. Запустите программу после ее компиляции с опцией **-qpdf2**.

Команда fdpr

Команда **fdpr** позволяет реорганизовать код в откомпилированной исполняемой программе с целью повысить производительность операций ветвления и перехода, отделить редко используемый код от активных фрагментов программы и выполнить другие глобальные операции по оптимизации.

Она наиболее эффективна для больших программ со значительным числом условных переходов, а также программ с высокой степенью структуризации и большим числом произвольно расположенных процедур. Работа с командой **fdpr** описана в разделе “Изменение структуры исполняемых программ с помощью программы fdpr” на стр. 121.

Оптимизирующие препроцессоры для FORTRAN и C

Применение препроцессоров дает выигрыш в производительности в среднем от 8 до 18 процентов.

Препроцессоры KAP и VAST для компилятора FORTRAN реструктурируют исходный код на языке FORTRAN с целью более эффективного использования ресурсов процессоров семейства семейство POWER, POWER2 и PowerPC и иерархической структуры памяти. Существует также версия препроцессора KAP для программ на C. Препроцессоры выполняют оптимизацию управления памятью, арифметические преобразования, встраивание, межпроцедурный анализ и другие операции по повышению производительности приложений на FORTRAN и C.

Препроцессоры KAP и VAST пытаются преобразовать исходные алгоритмы в алгоритмы, позволяющие в полной мере воспользоваться оптимизирующими возможностями компилятора. Кроме того, препроцессоры создают распечатки с указанием выполняемых преобразований, а также фрагментов кода, препятствующих преобразованиям. Препроцессоры анализируют исходный код и выполняют преобразования, повышающие производительность программы.

Любое преобразование, выполняемое препроцессорами, можно выполнить и вручную. Однако применение препроцессора обладает следующими преимуществами:

- В большинстве случаев программы, обработанные препроцессором, работают так же или даже более эффективно, чем их эквиваленты, настроенные вручную, причем это достигается без существенных затрат времени программиста. Для применения препроцессоров не требуется хорошо разбираться в системной архитектуре и приемах настройки, обсуждаемых в этой книге.
- В некоторых случаях удается эффективно оптимизировать код просто за счет правильного выбора опций препроцессора в командной строке и добавления небольшого числа директив в исходный код программы. Если применение препроцессора не дает значительного эффекта, изучите его распечатки и определите, какие фрагменты кода препятствуют оптимизации.

- Некоторые преобразования, выполняемые препроцессорами, приводят к значительному увеличению объема исходного кода. Хотя такое расширение повышает эффективность программы, выполнение его вручную сопряжено с высокой вероятностью ошибок и опечаток, ухудшением читаемости исходного кода и усложнением дальнейшей работы с программой.
- Препроцессоры могут оптимизировать код с учетом особенностей конкретной архитектуры, даже отличной от платформ семейства семейство POWER, POWER2 и PowerPC. На основе одной версии исходного кода вы можете создать несколько различных версий для моделей компьютеров семейство POWER, POWER2 и PowerPC, а также для систем с другими характеристиками кэша и процессора.
- Препроцессоры часто позволяют улучшить настроенный вручную код. Хотя вручную вы можете добиться такой же эффективности настройки, что и с помощью препроцессора, попытка выполнить вручную некоторые сложные преобразования почти наверняка приведет к ошибкам.

Приемы оптимизации кода

Снижение производительности вследствие неэффективного использования памяти намного более ощутимо, чем вследствие неэффективного использования кэша, поскольку разница в быстродействии памяти и диска намного больше, чем разница в быстродействии кэша и памяти.

Ниже перечислены приемы оптимизации кода:

- Для минимизации рабочего набора кода располагайте часто выполняемые фрагменты коды рядом друг с другом и отделяйте редко используемые фрагменты кода. Иными словами, не встраивайте в программу большие функции исправления ошибок и загружайте часто вызываемые модули сразу после функций, которые их вызывают.
- Для минимизации рабочего набора данных попытайтесь собрать все часто обрабатываемые данные вместе и удалить ненужные ссылки на страницы виртуальной памяти. Для этого вместо функции **calloc()** воспользуйтесь функцией **malloc()**, инициализирующей структуры данных непосредственно перед их использованием и освобождающей занимаемое ими пространство сразу после того, как они становятся ненужными.
- Для минимизации объема закрепленной памяти поместите резидентный код в отдельно загружаемые модули. Применяйте резидентный код только в случае необходимости. Некоторые системные структуры (например, пулы mbuf) не выгружаются из памяти (т.е. являются резидентными); не увеличивайте их размер без особой необходимости.
- Возможна оптимизация и во время выполнения. Например, вы можете применять функцию **plock()** для закрепления кода в памяти и функцию **setpri()** для указания приоритетов.

Файлы, размещенные в памяти

Одним из приемов оптимизации является применение отображенных файлов.

С помощью системных вызовов **shmat()** и **mmap()** приложения могут обращаться к файлам по их адресам, вместо того чтобы многократно вызывать системные функции чтения и записи. Поскольку обработка таких вызовов всегда отнимает много ресурсов, лучше уменьшить их количество. Применение **shmat()** и **mmap()** вместо обычных функций **read()** и **write()** повышает производительность во много раз (до 50). При вызове функции **shmat()**, так же, как и при выполнении операции чтения или записи, открывается файл и возвращается его дескриптор (*fd*). Затем **shmat()** возвращает адрес отображенного файла. Выбор элементов файла по последующим адресам, а не с помощью операций чтения, позволяет работать с файлом как с матрицей.

Вызов **mmap()** позволяет отобразить память в набор сегментов. Пользователь может работать более чем с 10 областями памяти. Функции **mmap()** обеспечивают защиту областей памяти на уровне страниц. Для каждой страницы может быть установлено разрешение на чтение, разрешение на запись или запрет на доступ. Вызов **mmap()** позволяет отобразить только одну страницу файла.

Вызов **shmat()** позволяет отобразить несколько сегментов, если отображаемый файл занимает больше одного сегмента.

В следующем примере программа считывает данные из файла с помощью операторов read:

```
fd = open("myfile", O_RDONLY);
for (i=0;i<cols;i++) {
        for (j=0;j<rows;j++) {
                read(fd,&n,sizeof(char));
                *p++ = n;
        }
}
```

Ту же самую задачу можно выполнить с помощью функции **shmat()**:

```
fd = open("myfile", O_RDONLY);
nptr = (signed char *) shmat(fd,0,SHM_MAP | SHM_RDONLY);
for (i=0;i<cols;i++) {
        for (j=0;j<rows;j++) {
                *p++ = *nptr++;
        }
}
```

Применение явно отображенных файлов обладает одним недостатком, проявляющимся при выполнении операций записи. Системная функция отложенной записи, периодически записывающая группы обновленных страниц на диск, неприменима, когда в приложении используется функция **shmat()** или **mmap()**. В этом случае обновленные страницы накапливаются в памяти. Они записываются на диск случайным образом, когда Администратору виртуальной памяти (VMM) требуется свободное пространство. Эта ситуация часто приводит к выполнению большого числа коротких операций записи на диск, что снижает производительность процессора и дисковой памяти.

Отслеживание производительности программ на Java

Приведены советы по обнаружению "узких мест" в приложениях на Java™, влияющих на их производительность.

Java - это объектно-ориентированный язык программирования, разработанный корпорацией Oracle. Он был создан на основе языка C++. При его разработке основными целями были небольшой размер программ, их простота и независимость от платформы как на уровне исходного кода, так и на двоичном уровне. Таким образом, программы на Java, включающие аплеты и приложения, могут работать на любом компьютере, на котором установлена виртуальная машина Java (JVM).

Преимущества Java

У языка Java есть много преимуществ перед другими языками программирования, что позволяет решать с его помощью практически любые задачи.

Ниже перечислены основные преимущества Java:

- Язык Java прост для изучения.

  При разработке Java было уделено большое внимание простоте языка, поэтому программы на Java, по сравнению с программами на других языках, проще писать, компилировать, отлаживать и изучать.
- Java - это объектно-ориентированный язык.

  Это позволяет создавать модульные программы, исходный код которых может использоваться многократно.
- Язык Java не зависит от платформы.

  Одним из основных преимуществ языка Java является возможность переноса программ из одной системы в другую. Поскольку программы на Java не зависят от платформы как на уровне исходного кода, так и на двоичном уровне, их можно запускать в различных системах, что особенно важно для программ, предназначенных для World Wide Web.

Широкие возможности Java, простота применения, независимость от платформы и встроенные функции защиты делают этот язык программирования одним из лучших для создания приложений для Internet.

Оптимизация программ на Java

Производительность программ на Java в AIX моно улучшить разными способами.

- Для того чтобы при выполнении большого числа операций со строками избежать создания ненужных объектов, после которых нужно будет освобождать память, вместо конкатенации строк можно использовать функцию **StringBuffer**.
- Постарайтесь максимально сократить число операций вывода данных на консоль Java, так как для этого требуется выполнить некоторые действия над строками, отформатировать текст и записать данные.
- По возможности создавайте переменные простых типов, так как для работы с ними требуется меньше ресурсов системы.
- Заносите часто используемые объекты в кэш; это позволит уменьшить нагрузку на функцию сбора мусора и избежать многократного создания этих объектов.
- Объединяйте в группы внутренние операции для уменьшения числа вызовов интерфейса JNI Java.
- Для снижения числа процессов в JVM и операционной системе не применяйте синхронизированные методы, когда задачу можно решить другими способами.
- Вызывайте функцию сбора мусора только при необходимости. Запускать эту функцию рекомендуется только во время простоя или низкой загруженности системы.
- Используйте тип **int** вместо **long** всегда, когда это возможно, потому что 32-разрядные операции выполняются быстрее, чем 64-разрядные.
- Объявляйте методы с ключевым словом final всегда, когда это возможно. В JVM такие методы обрабатываются быстрее.
- Для снижения числа операций по инициализации переменных создавайте константы с ключевым словом **static final**.
- Постарайтесь сократить число ссылок "cast" и "instanceof", поскольку соответствующие ссылки в Java преобразуются во время выполнения.
- Старайтесь использовать массивы вместо векторов.
- Добавляйте и удаляйте элементы из конца вектора.
- Избегайте операций выделения памяти под объекты внутри циклов.
- Пользуйтесь буферизованным вводом-выводом и выберите оптимальный размер буфера.
- Пользуйтесь пулами соединений и предварительной подготовкой операторов при работе с базами данных.
- Пользуйтесь пулами соединений с базой данных и повторно используйте соединения вместо того, чтобы всякий раз открывать и закрывать соединения.
- Применяйте максимальное возможное время жизни нити и минимизируйте количество операций по созданию и уничтожению нитей.
- Минимизируйте конкуренцию за общие ресурсы.
- Минимизируйте количество операций создания объектов, используемых в течение непродолжительного времени.
- Старайтесь как можно реже вызывать удаленные методы.
- Пользуйтесь функциями callback для того, чтобы избежать блокирования обращений к удаленным методам.
- Не создавайте объекты, которые будут применяться только для обращения к методам.
- По возможности не пользуйтесь синхронизированными методами внутри циклов.
- Храните строковые и символьные данные в базе данных в формате Unicode.
- Выберите оптимальный порядок библиотек в переменной CLASSPATH.

Средства мониторинга Java

Сведения о производительности программ на Java можно собирать с помощью различных программ.

vmstat Эта команда предоставляет сведения о ресурсах системы. Приводится статистика по нитям ядра в

очереди выполнения и очереди ожидания, по загрузке памяти и пространства подкачки, по дисковым операциям, прерываниям, системным вызовам, контекстным переключателям и нагрузке на процессор.

iostat Эта команда предоставляет подробные сведения о дисковых операциях.

topas Эта команда выдает сведения о процессоре, сети, дисковых операциях, WLM и процессах.

tprof С помощью этой команды можно определить, какие методы или процедуры в максимальной степени влияют на скорость выполнения программы.

ps -mo *нить*
С помощью этой команды можно определить, к какому процессору привязаны нить или процесс.

Профилировщики Java [-Xrunhprof, Xrunjpa64]
Определяет, какие процедуры и методы используются наиболее часто.

java -verbose:gc
С помощью этой опции можно оценить влияние процессов сбора мусора на скорость выполнения программы. Выдаются значения общей продолжительности сбора мусора, средней продолжительности сбора мусора, среднего объема освобожденной памяти и среднего объема собранного мусора.

Настройка Java для AIX

Описаны рекомендуемые параметры AIX для среды Java.

AIXTHREAD_SCOPE=S
По умолчанию этой переменной присвоено значение **P**, указывающее на то, что областью действия является система (M:N). Значение **S** обозначает системную область действия (1:1). Значение этой переменной по умолчанию для приложений Java: **S**.

AIXTHREAD_MUTEX_DEBUG=OFF
Определяет список активных взаимных блокировок, применяемый отладчиком.

AIXTHREAD_COND_DEBUG=OFF
Определяет список переменных, используемых отладчиком.

AIXTHREAD_RWLOCK_DEBUG=OFF
Определяет список активных взаимных блокировок, условных переменных и блокировок чтения-записи, которые могут использоваться отладчиком. После инициализации блокировка добавляется в список (если ее еще нет в списке). Список представляет собой обычный связанный список, поэтому когда он достигает больших размеров, на его просмотр и поиск блокировок может уходить значительное время. Ситуация осложняется тем, что на время каждой операции поиска список блокируется и становится недоступен другим процессам. При этом все остальные вызовы функции **pthread_mutex_init()** должны ожидать окончания поиска. Для достижения максимальной производительности данной опции следует присвоить значение `OFF`. По умолчанию ей присвоено значение `ON`.

SPINLOOPTIME=500
Задает максимальное количество попыток обращения к занятой блокировке для процесса. По умолчанию данной опции присвоено значение 40. Если результаты выполнения команды **tprof** свидетельствуют о частом обращении к процедуре **check_lock**, а блокировки обычно устанавливаются на короткие промежутки времени, то рекомендуется увеличить это значение до 500 или больше.

Кроме того, для среды Java рекомендуется дополнительно задать следующие параметры:

ulimit -d *unlimited*

ulimit -m *unlimited*

ulimit -n *unlimited*

ulimit -s *unlimited*

Для настройки производительности программ на Java в операционной системе предусмотрено несколько параметров и переменных среды. На производительность программ на Java влияют и такие компоненты системы, как процессор, память, сеть, подсистема ввода-вывода и т. д. Настроить эти компоненты можно с помощью соответствующих системных средств. Информация о том, каким образом нужно настроить эти компоненты, приведена в соответствующих главах этой книги.

Максимальная производительность и масштабируемость приложений на Java достигается при работе с последними версиями операционной системы, Java и компилятора Just-In-Time (JIT).

Сбор мусора и производительность Java

Самый распространенный источник снижения производительности в среде Java - это механизм сбора мусора.

Чем больше размер кучи Java, тем чаще она будет частично вытесняться в виртуальную память. Это приводит к активизации процессов подкачки, которые влияют на производительность Java.

Кроме того, на заполнение большой кучи требуется несколько секунд. Таким образом, даже если процедура сбора мусора будет запускаться реже, увеличится время, в течение которого работа приложений будет приостановлена для сбора мусора.

Для настройки кучи виртуальной машины Java (JVM) используйте команду **java** с опцией **-ms** или **-mx**. Оптимальные значения параметров лучше всего подбирать с учетом статистики сбора мусора.

Анализ производительности с помощью функции трассировки

Функция трассировки, предусмотренная в операционной системе, является одним из основных средств контроля за работой системы.

Функция трассировки записывает сообщения о событиях, происходящих в системе, указывая время их возникновения, и предоставляет подробную информацию о работе системы. Сведения о событиях представлены в хронологическом порядке и связаны с друг с другом. Трассировка - это полезное средство для наблюдения за работой системы и выполнением приложений. В отличие от других инструментов, предоставляющих информацию только об использовании CPU и времени ожидания ввода-вывода, функция трассировки позволяет понять, какие события происходят в системе, по чьей инициативе, когда, каким образом эти события влияют на работу системы и почему.

В операционной системе предусмотрены средства для наблюдения за работой системных программ. Пользователи могут включить в список отслеживаемых программ и собственные приложения, добавив дополнительные события и задав правила форматирования.

При разработке и реализации этой функции особое внимание уделялось эффективности сбора данных трассировки. Ее влияние на производительность системы было сведено до минимума. По этой причине функция трассировки исключительно полезна в качестве средства анализа производительности и обнаружения проблем.

Подробное описание функции трассировки

Функция трассировки является более гибким средством, чем стандартные системные мониторы, которые предназначены для просмотра статистики, собираемой системой.

Эта функция не позволяет заранее выбрать тип собираемой информации. Вместо этого она собирает информацию обо всех событиях и позволяет пользователю указать, какую информацию он хочет просмотреть. При работе с обычными мониторами обработка данных (преобразование системных событий

в статистику) в большой степени связана с набором инструментов, доступных в системе. Например, во многих системах можно получить информацию о минимальном, максимальном и среднем времени выполнения задачи.

Функция трассировки не сокращает объем данных до необходимого, а создает поток событий трассировки (обычно кратко называемый *события*). При этом не требуется заранее указывать, какие статистические данные требуются собрать; обработка данных практически не зависит от набора инструментов системы. Пользователь может выбрать из полученного потока событий минимальное, максимальное и среднее время выполнения задачи. Кроме того, можно получить следующую информацию:

- Получить среднее время выполнения задачи А в рамках процесса Б
- Получить среднее время выполнения задачи А, если выполнены условия XYZ
- Вычислить среднее квадратичное отклонение для времени выполнения задачи А
- Исходя из полученной информации выбрать другую задачу, для которой следует запустить функцию трассировки.

Такая гибкость полезна при диагностике производительности или неполадок в работе системы.

Функция трассировки не только предоставляет подробную информацию о работе системы, но и позволяет отслеживать события, возникающие в ходе работы отдельных приложений, наряду с системными событиями. В файл трассировки заносится полная информация о работе приложения и системы в хронологической последовательности с указанием точного времени возникновения событий.

Реализация функции трассировки

Точкой трассировки называется отслеживаемое событие. Этому событию присваивается уникальный номер, называемый *идентификатором точки трассировки*. Информация об этих точках трассировки собирается командой **trace**.

Команда **trace** служит для сбора статистических данных о процессах пользователей и подсистемах ядра. Двоичная информация заносится в два альтернативных буфера в памяти. После этого процесс **trace** записывает информацию на диск в файл протокола трассировки. Размер этого файла быстро растет. Программа **trace** выполняется в виде процесса, информацию о котором можно получить с помощью команды **ps**. Команда **trace** выполняется в виде демона, аналогичного демону учета.

На следующем рисунке изображена схема работы функции трассировки.

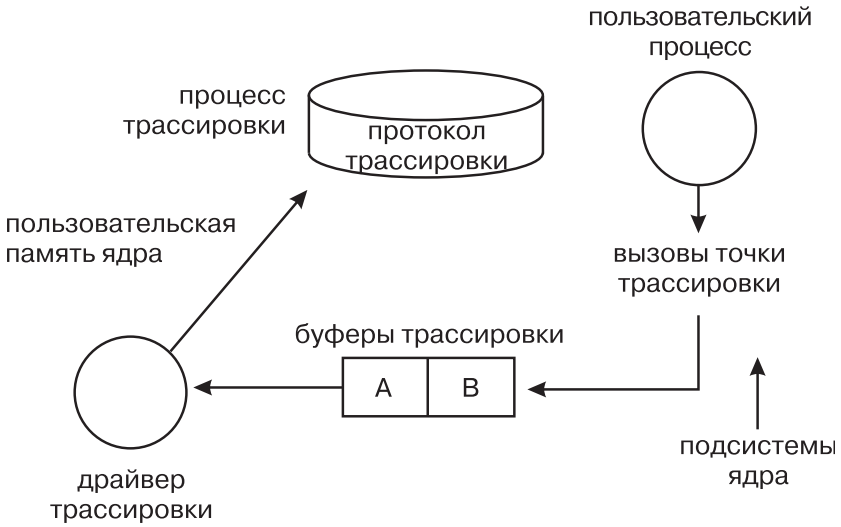


Рисунок 25. Работа функции трассировки. На этом рисунке изображен процесс трассировки. Пользовательский процесс (подсистемы ядра) отправляет запросы на запись информации о точках трассировки в буферы А и Б. Информация из этих буферов передается драйверу трассировки, а затем записывается в файл протокола трассировки пользовательского ядра.

На работу любой функции сбора данных затрачиваются ресурсы системы. В общем случае дополнительная нагрузка, связанная с запуском этой функции, должна быть как можно меньше, чтобы она не влияла на работу других программ. При выполнении программы **trace** загруженность процессора увеличивается не более чем на 2 процента. Нагрузка на процессор возрастает, когда буфер переполняется и требуется записать данные в файл протокола, так как для этого требуется выполнить операции ввода-вывода. Как правило, при этом нагрузка составляет не более 5 процентов. Если объем памяти системы невелик, то запуск функции трассировки может привести к снижению производительности системы, поскольку программа **trace** занимает часть памяти под буферы. Учтите, что файлы отчетов и протокола трассировки могут стать очень большими.

Ограничение объема данных трассировки

Функция трассировки собирает много различных данных. Если трассировка выполняется в течение длительного периода времени, память переполняется.

Существует два способа эффективного применения функции трассировки:

- Для сбора данных о работе системы можно периодически включать и выключать функцию трассировки. Информация трассировки может собираться в течение нескольких секунд или минут и сохраняться для последующей обработки. Например, функцию трассировки можно включать во время выполнения приложением основных транзакций, либо во время выполнения важного этапа длительной задачи.
- Функцию трассировки можно настроить так, что информация о потоке событий будет направляться в стандартный вывод. За счет этого поток событий можно передать на вход другому процессу, работающему в режиме реального времени, который будет будет выполнять обработку полученной информации, удаляя ненужные данные. В результате трассировка может выполняться длительное время. Существуют специальные инструменты, направляющие поток данных на вспомогательное устройство большого объема или динамически сокращающие данные. Они применяются такими средствами сбора данных о производительности, как **tprof**, **pprof**, **netpmon** и **filemon**.

Запуск и управление трассировкой

Функция трассировки может работать в трех режимах:

Режим субкоманд

Трассировка запускается с помощью команды оболочки (**trace**). Для ее управления применяются субкоманды. Рабочая схема, отслеживаемая функцией трассировки, должна выполняться в других процессах, так как исходный процесс оболочки используется.

Режим команд

Трассировка запускается с помощью команды оболочки (**trace -a**) с флагом, устанавливающим асинхронный режим работы. В исходном процессе оболочки могут вызываться обычные команды, в том числе и те команды, которые отслеживаются функцией трассировки.

Режим управления из приложений

Трассировка запускается с помощью функции **trcstart()**. Для ее управления в приложении применяются такие функции, как **trcon()** и **trcoff()**.

Форматирование вывода трассировки

С помощью команды **trcrpt** можно вызвать функцию создания глобальных отчетов трассировки.

Эта команда незначительно сокращает объем данных, но преобразует поток двоичных данных в текстовую форму. Полученные данные можно просмотреть с помощью программы чтения, либо дополнительно сократить их объем с помощью специально созданных средств.

Функция создания отчета выводит описания событий в соответствии с правилами, заданными в файле формата данных трассировки. По умолчанию формат данных трассировки хранится в файле `/etc/trcfmt`, который содержит разделы для всех идентификаторов событий. В каждом разделе задаются правила форматирования отчета о событии с указанным идентификатором. Благодаря такому подходу пользователи могут использовать в своих программах собственные идентификаторы событий, добавив для них соответствующие разделы в файл с описанием формата.

Просмотр данных событий трассировки

Обычно в отформатированном выводе трассировки информация о событии занимает одну строку.

Однако в некоторых случаях создается подробное описание события, которое размещается в дополнительных строках. В зависимости от параметров форматирования, длина строк может превышать 80 символов. Отчеты функции трассировки рекомендуется просматривать на устройствах вывода, поддерживающих ширину текста в 132 колонки.

Пример работы с трассировщиком

Обычно в ходе трассировки выполняются операции сбора, форматирования и фильтрации данных и чтения файла трассировки.

Примечание: Этот пример будет более наглядным, если файл ввода еще не занесен в кэш, расположенный в оперативной памяти системы. Выберите в качестве исходного любой файл, размер которого составляет около 50 КБ, и который не использовался в последнее время.

Получение примера файла трассировки

Данные трассировки накапливаются быстро. Максимально ограничьте время сбора данных. Это можно сделать, введя сразу несколько команд.

Например:

```
# trace -a -k "20e,20f" -o trc_raw ; cp ../bin/track /tmp/junk ; trcstop
```

В этом примере отслеживается выполнение команды **cp**. В команде **trace** указано две дополнительные опции. Если задана опция **-k "20e,20f"**, то сбор данных о событиях функций **lockl()** и **unlockl()** не выполняется. В однопроцессорных системах, в отличие от систем SMP, такие функции вызываются очень часто, и при этом в отчет функции трассировки заносится большой объем ненужной информации. Если задана опция **-o trc_raw**, то файл с выводом функции трассировки будет помещен в локальный каталог.

Форматирование вывода трассировки

Команда **trcrpt** позволяет отформатировать вывод трассировки.

```
# trcrpt -O "exec=on,pid=on" trc_raw > cp.rpt
```

Эта команда заносит в отчет полное имя файла и связанный с ним идентификатор процесса.

В полученном файле отчета было указано, что во время выполнения трассировки часто выделялись и освобождались страницы памяти VMM. Ниже приведен пример фрагмента отчета:

```
1B1 ksh            8526          0.003109888       0.162816
         VMM page delete:      V.S=0000.150E ppage=1F7F
             working_storage delete_in_progress process_private computational

1B0 ksh            8526          0.003141376       0.031488
         VMM page assign:      V.S=0000.2F33 ppage=1F7F
             working_storage delete_in_progress process_private computational
```

Настолько подробная информация о работе VMM в данный момент не нужна, поэтому вывод трассировки был заново отформатирован следующим образом:

```
# trcrpt -k "1b0,1b1" -O "exec=on,pid=on" trc_raw > cp.rpt2
```

Параметр **-k "1b0,1b1"** исключает из отформатированного вывода ненужную информацию о событиях VMM. Если вы укажете этот параметр, то вам не придется выполнять повторную трассировку рабочей схемы для удаления этой информации. Для временного исключения из вывода информации о событиях **lockl()** и **unlockl()**, которая может понадобиться в дальнейшем, вместо указанной команды **trace** можно вызвать команду **trcrpt** с опцией **-k**. Если нужно получить информацию о небольшом числе событий, то можно задать список этих событий с опцией **-d "***ИД-точки-трассировки-1*,*ИД-точки-трассировки-2***"**. Поскольку ИД точек трассировки указывается в самом левом столбце отчета, можно быстро составить список точек трассировки, информацию о которых нужно включить в отчет или исключить из него. Полный список идентификаторов точек трассировки приведен в файле `/usr/include/sys/trchkid.h`.

Описание отчета трассировки

В заголовке отчета трассировки указано, когда и в какой системе была выполнена трассировка, а также формат команды трассировки.

Пример заголовка:

```
четверг 28 октября 1999 г. 13.34.05
Система: AIX texmex
Узел: 4
Машина: 000691854C00
IP-адрес: 09359BBB 9.53.155.187
Буферизация: куча ядра

trace -a -k 20e,20f -o trc_raw
```

Тело отчета выводится мелким шрифтом. Оно выглядит примерно следующим образом:

```
ИД  ИМЯ ПРОЦЕССА  PID     ВРЕМЯ(СЕК)    ДЕЛЬТА(МСЕК) ПРИЛ.  ВЫЗОВ    ЯДРО  ПРЕРЫВАНИЕ
101 ksh           8526    0.005833472    0.107008          kfork LR = D0040AF8
101 ksh           7214    0.012820224    0.031744          execve LR = 10015390
134 cp            7214    0.014451456    0.030464          exec:  cmd=cp ../bin/track /tmp/junk pid=7214 tid=24713
```

Файл `cp.rpt2` содержит информацию о следующих событиях:

- Вызов функций **fork()** и **exec()** в рамках процесса **cp** и возникновение страничных ошибок.
- Открытие файла ввода для чтения и создание файла `/tmp/junk`
- Вызов функций **read()** и **write()** для копирования данных.
- Процесс **cp** блокируется до завершения операции ввода-вывода и запускается процесс **wait**.
- Преобразование запросов к логическим томам в запросы к физическим томам.
- Файлы не помещаются в стандартные буферы ядра, а явно копируются в оперативную память. В итоге запросы на чтение приводят к возникновению страничных ошибок, которые обрабатываются Администратором виртуальной памяти.

- Администратор виртуальной памяти обнаруживает, что данные последовательно выбираются из памяти, и считывает в кэш страницы файлов с упреждением.
- Размер буфера упреждающего чтения увеличивается по мере последовательного чтения данных.
- По возможности драйвер диска объединяет несколько запросов к файлам в один запрос на ввод-вывод к этому диску.

На первый взгляд, вывод трассировки выглядит громоздким. Это хороший учебный пример. Если вы сможете найти в нем описанные события, значит вы уже готовы работать с функцией трассировки и применять ее для диагностики и повышения производительности системы.

Фильтрация отчета трассировки

Обычно пользователю не нужны все данные трассировки. Вы можете выбрать несколько событий, информация о которых вас интересует.

Например, вам может потребоваться узнать, сколько раз произошло указанное событие. Для того чтобы узнать, сколько операций открытия было выполнено командой копирования в описанном выше примере, определите ИД события, соответствующий системному вызову **open()**. Для этого введите следующую команду:

```
# trcrpt -j | grep -i open
```

В выводе команды будет указано, что событию OPEN SYSTEM CALL присвоен идентификатор 15b. Теперь нужно обработать вывод трассировки, вызвав следующую команду:

```
# trcrpt -d 15b -O "exec=on" trc_raw
```

Отчет с информацией о вызовах функции **open()** будет записан в стандартный вывод. Для того чтобы просмотреть информацию только о тех вызовах **open()**, которые были выполнены процессом **cp**, введите команду создания отчета со следующими параметрами:

```
# trcrpt -d 15b -p cp -O "exec=on" trc_raw
```

Запуск и управление трассировкой из командной строки

Для настройки функции трассировки и запуска сбора данных служит команда **trace**, подробное описание которой приведено в книге *Справочник по командам, том 5*.

После настройки трассировки с помощью команды **trace** вы можете включить, выключить или завершить сбор данных трассировки. Управлять трассировкой можно с помощью подкоманд, команд и функций. Интерфейсы функций рассмотрены в “Запуск и управление трассировкой из программы” на стр. 396.

Управление трассировкой в режиме подкоманд

Если в функции **trace** не задана опция **-a**, она будет выполняться в режиме подкоманд.

При работе утилиты trace в режиме подкоманд вместо обычного приглашения оболочки будет показано приглашение "->". В этом режиме можно вводить следующие подкоманды:

trcon Запускает или возобновляет сбор данных о событиях

trcoff Приостанавливает сбор данных о событиях

q или quit
Прекращает сбор данных о событиях и завершает работу процедуры **trace**

!команда
Запускает указанную команду оболочки

? Показывает список доступных команд

Например:

```
# trace -f -m "Трассировка событий во время выполнения mycmd"
->!mycmd
-> q
#
```

Управление трассировкой вызовом команд

Описаны инструменты для управления функциями **trace**.

Если процедура **trace** настроена для работы в асинхронном режиме (**trace -a**), то ее работой можно управлять с помощью следующих команд:

trcon Запускает или возобновляет сбор данных о событиях

trcoff Приостанавливает сбор данных о событиях

trcstop Прекращает сбор данных о событиях и завершает работу процедуры **trace**

Например:

```
# trace -a -n -L 2000000 -T 1000000 -d -o trace.out
# trcon
# cp /a20kfile /b
# trcstop
```

С помощью опции **-d** (отложить трассировку до ввода подкоманды **trcon**) можно сократить объем данных трассировки, относящихся к самой команде **trace**. Если опция **-d** не указана, то трассировка запускается немедленно. В этом случае в протокол будет занесена информация об инициализации буферов памяти команды **trace**. Как правило, информация о событиях, относящихся к команде **trace**, не требуется.

По умолчанию размер буфера ядра (опция **-T**) может быть равен половине размера буфера протокола (опция **-L**). Если указан флаг **-f**, то размеры буферов могут совпадать.

Флаг **-n** применяется для трассировки системных вызовов расширений ядра.

Запуск и управление трассировкой из программы

Функцию трассировки можно запустить из программы с помощью функции **trcstart()**, которая расположена в библиотеке `librts.a`.

Ниже описан формат вызова функции **trcstart()**:

```
int trcstart(char *args)
```

где *args* - это список параметров команды **trace**. По умолчанию запускается трассировка системных событий (канал 0). Для запуска трассировки общего назначения укажите в строке параметров *args* опцию **-g**. После успешного завершения функция **trcstart()** возвратит ИД канала. В случае трассировки общего назначения этот ИД канала может применяться для выбора необходимых данных и их записи в частный канал общего назначения.

При компиляции программы, использующей эту функцию, может потребоваться явно указать библиотеку `librts.a` (в качестве опции компиляции укажите **-l rts**).

Управление трассировкой вызовом функций trace

Для управления работой функции **trace** применяются функции из библиотеки `librts.a`.

В случае успешного завершения функции возвращают нулевое значение. Ниже приведен список функций:

int trcon()
Начинает или возобновляет сбор данных трассировки.

int trcoff()
Приостанавливает сбор данных трассировки.

int trcstop()
Прекращает сбор данных трассировки и завершает работу процедуры **trace**.

Применение команды trcrpt для форматирования отчета

Функция создания отчета трассировки считывает файл протокола трассировки, форматирует записи трассировки и создает отчет.

Команда **trcrpt** выводит описания событий в соответствии с правилами, заданными в файле формата данных трассировки (`/etc/trcfmt`). В разделах файла формата приведены правила форматирования данных для событий и точек трассировки. Если в программе применяются пользовательские точки трассировки, то в файл формата можно добавить разделы для соответствующих событий и задать в них правила вывода информации об этих событиях (дополнительная информация приведена в разделе “Добавление новых событий для трассировки” на стр. 398).

Функция **trcrpt** не поддерживает создание итоговых отчетов, однако вы можете создать простые итоговые отчеты путем обработки вывода команды **trcrpt** командой **awk**.

Подробное описание формата команды **trcrpt** приведено в книге *Справочник по командам, том 5*.

Создание отчета в той же системе

Команда **trcrpt** создает отчеты на основе информации о событиях трассировки, хранящейся в файле протокола трассировки.

С помощью этой команды вы можете указать, какие события нужно включить в отчет, а какие - исключить из него. Кроме того, вы можете задать представление вывода.

Для запуска команды **trcrpt** можно вызвать следующую команду SMIT:

```
# smitty trcrpt
```

Для создания отчета трассировки в файле `новый_файл` введите:

```
# trcrpt -o новый_файл
```

Создание отчета в другой системе

Часто команду **trcrpt** лучше запускать не в той системе, в которой были собраны данные трассировки.

Это может быть обусловлено различными причинами, например:

- У вас нет доступа к системе, в которой были собраны данные **trcrpt**. Например, трассировка могла быть выполнена по вашей просьбе администратором или другим пользователем удаленной системы.
- Система слишком сильно загружена для того, чтобы в ней можно было запустить команду **trcrpt**.
- В системе недостаточно места на диске для размещения очень большого файла **trcrpt**.

Команду **trcrpt** можно вызвать для обработки файла трассировки, созданного командой **trace** в другой системе. Для этого необходимо получить вывод команды **trcnm**, созданный в той системе, в которой выполнялась трассировка. Вызовите команду **trcnm**, перенаправив вывод в файл:

```
# trcnm > trace.nm
```

Если вы хотите обработать файл трассировки с помощью другой функции анализа производительности, например, **tprof**, **pprof**, **netpmon** или **filemon**, введите команду **gennames** *файл*.

Имя созданного файла нужно указать с флагом **-n** при вызове команды **trcrpt**:

```
# trcrpt -n trace.nm -o новый_файл
```

Если флаг **-n** не указан, то команда **trcrpt** создает таблицу имен для системы, в которой **trcrpt** была вызвана.

Кроме того, из системы, в которой выполнялась трассировка, рекомендуется скопировать файл /etc/trcfmt и указать его в качестве параметра команды **trcrpt**, так как в удаленной системе могут применяться другие правила форматирования вывода трассировки. Файл формата можно указать с флагом **-t** команды **trcrpt** (по умолчанию применяется файл /etc/trcfmt той системы, в которой вызвана команда **trcrpt**). Например:

```
# trcrpt -n trace.nm -t файл_формата -o новый_файл
```

Создание отчета на основе вывода команды trace -C

Когда трассировка выполняется с флагом **-C**, может быть создано несколько файлов вывода.

Например, если в качестве файла вывода трассировки был задан файл trace.out и трассировка была запущена в четырехпроцессорной системе с параметром **-C all** , то будут созданы файлы trace.out, trace.out-1, trace.out-2, trace.out-3 и trace.out-4. При вызове команды **trcrpt** укажите параметр **trcrpt -C all** и файл trace.out. В этом случае будут обработаны все файлы вывода:

```
# trcrpt -C all -r trace.out > trace.tr
```

Полученный файл trace.tr можно обработать другими командами (он будет содержать данные трассировки для всех процессоров). Флаг **-C** применяется в системах с большим числом процессоров (например, больше 12) для сбора данных обо всех процессорах. Если указан флаг **-C all**, то размер буфера трассировки ограничивает размер буфера каждого процессора, а не общий размер буфера.

Добавление новых событий для трассировки

В операционной системе есть ряд стандартных событий. Для получения информации об этих событиях пользователю достаточно запустить функцию трассировки. При отладке приложения у разработчика может возникнуть необходимость добавить собственные события трассировки. Это помогает лучше понять взаимодействие приложения с системой.

Для добавления события трассировки необходимо определить формат записей трассировки, создаваемых пользовательской программой. При этом нужно учесть общие правила создания записей трассировки. Затем в соответствующих точках программы нужно поместить макрокоманды, задающие точки трассировки. После этого можно выполнять трассировку одним из стандартных способов (с помощью команд, подкоманд или вызовов функций). Если вы планируете применять команду **trcrpt** для форматирования вывода трассировки, добавьте в файл формата данных трассировки разделы с описанием формата записей для пользовательских точек трассировки.

Возможные форматы записей о событиях трассировки

Запись о событии содержит слово с информацией о точке трассировки. Дополнительно она может содержать несколько слов данных и отметку системного времени.

Как видно на рисунке, каждому варианту записи о событии соответствует четырехразрядный тип. Функция регистрации событий указывает в записи ее тип, поэтому функция создания отчета всегда может просмотреть запись о событии, даже если правила форматирования для этого события не заданы, либо заданы неправильно.

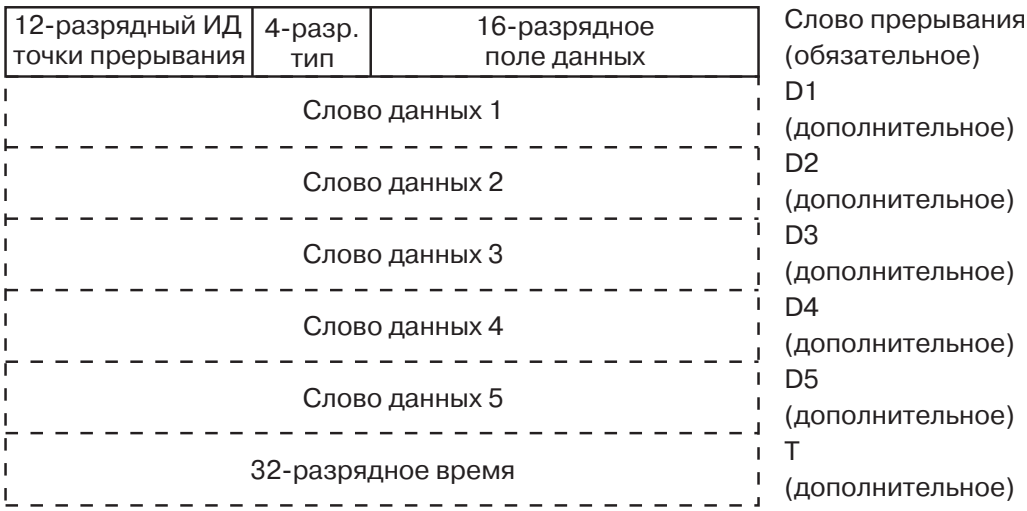


Рисунок 26. Формат записи о событии трассировки. На рисунке изображена таблица, содержащая 7 строк. В первой строке содержатся 12-разрядный ИД точки трассировки, 4-разрядный тип и 16-разрядное поле данных. Следующие 6 строк содержат слова данных с 1 по 5, а последняя строка - 32-разрядное значение системного времени. Первая строка представляет слово с информацией о точке трассировки (обязательная часть записи). Следующие пять строк представляют слова D1 - D5 (необязательная часть записи). Последняя строка содержит заголовок T (обязательная часть записи).

Размер записи о событии должен быть минимальным. Для многих системных событий запись содержит только слово с информацией о точке трассировки и системное время. Длинный формат позволяет пользователям сохранять данные различной длины. В этом формате 16-разрядное поле данных в слове с информацией о точке трассировки заменяется на поле, содержащее длину записи о событии.

Каналы трассировки

Функция трассировки позволяет одновременно работать с восьмью каналами трассировки с номерами от 0 до 7.

Канал 0 всегда используется для трассировки системных событий, однако его можно использовать и для событий приложений. С помощью остальных семи каналов, которые называются каналами общего назначения, можно выполнять трассировку программ.

При запуске трассировки по умолчанию применяется канал 0. Для запуска трассировки с записью информации в канал общего назначения введите команду **trace -n** *номер_канала*. При применении каналов общего назначения действуют некоторые ограничения:

- При работе с каналом общего назначения затрачивается больше процессорного времени, чем при работе с каналом 0, поскольку предварительно требуется выбрать канал. Кроме того, в каналы общего назначения заносятся записи переменной длины, что отнимает больше времени, чем вывод обычных записей.
- Порядок возникновения событий, зарегистрированных в канале 0 и каналах общего назначения, можно восстановить только по отметкам системного времени. В некоторых случаях реальную последовательность событий восстановить нельзя.

Макрокоманды для записи событий трассировки

Макрокоманды, позволяющие регистрировать события различных типов, определены в файле `/usr/include/sys/trcmacros.h`.

Идентификаторы событий определены в файле `/usr/include/sys/trchkid.h`. Эти файлы необходимо включить в программу, которая заносит информацию о событиях трассировки.

Для записи информации о событии в канал 0 с указанием системного времени предназначены следующие макрокоманды:

```
TRCHKL0T(hw)
TRCHKL1T(hw,D1)
TRCHKL2T(hw,D1,D2)
TRCHKL3T(hw,D1,D2,D3)
TRCHKL4T(hw,D1,D2,D3,D4)
TRCHKL5T(hw,D1,D2,D3,D4,D5)
```

В предыдущих версиях AIX используйте следующие макрокоманды для записи событий на канале 0 без системного времени:

```
TRCHKL0(hw)
TRCHKL1(hw,D1)
TRCHKL2(hw,D1,D2)
TRCHKL3(hw,D1,D2,D3)
TRCHKL4(hw,D1,D2,D3,D4)
TRCHKL5(hw,D1,D2,D3,D4,D5)
```

Все события трассировки содержат системное время, независимо от используемых макрокоманд.

Тип записи о событии трассировки зависит от выбранной макрокоманды. Значение, указанное в соответствующих четырех разрядах параметра `hw`, игнорируется.

Для записи информации о событиях в каналы общего назначения (1-7) предназначены определенные макрокоманды. Эти команды таковы:

```
TRCGEN(ch,hw,D1,len,buf)
TRCGENT(ch,hw,D1,len,buf)
```

Эти макрокоманды записывают в канал, заданный в параметре `ch`, слово с информацией о событии (`hw`), слово данных (`D1`) и `len` байт из пользовательского сегмента данных, выбранных начиная с позиции `buf`.

Использование ИД события

Идентификатор события в записи трассировки указывает, к какому классу относится данное событие. ИД события может служить критерием для записи или игнорирования события в ходе трассировки, а также для включения или исключения соответствующей записи из отчета, создаваемого командой **trcrpt**.

В более ранних версиях AIX 6.1, а также в 32-разрядных приложениях под AIX 6.1 и более поздними версиями, ИД события представляет собой 12-разрядное двоичное число (три шестнадцатиричные цифры), поэтому всего есть 4096 вариантов идентификаторов. Зарезервированные ИД событий, для которых программный код поставляется вместе с системой, фиксируются для предотвращения дублирования. Событиям, которые пользователи определяют в собственных средах и приложениях, могут присваиваться идентификаторы в диапазоне от 0x010 до 0x0FF (в шестнадцатеричном формате). При этом под пользовательской средой понимается любой набор систем, в котором можно гарантировать уникальность события с идентификатором из указанного диапазона.

В 64-разрядных приложениях и процедурах ядра под AIX 6.1 и более поздней версией можно использовать 16-разрядные ИД событий (четыре шестнадцатеричных цифры) для возможных 65536 идентификаторов. Идентификаторы событий меньшие, чем 0x1000, должны иметь наименьшую значимую цифру - 0 (в формате "0x0hh0"). Событиям, которые пользователи определяют в собственных средах и приложениях, могут присваиваться идентификаторы в диапазоне от 0x0100 до 0x0FF0 (в шестнадцатеричном формате).

Примечание: Необходимо убедиться в том, что программа, в которой применяются события с идентификаторами из этого диапазона, будет использоваться только в исходной пользовательской среде. Если программа, в которой применяются временные ИД событий, будет выполняться в другой среде, могут возникнуть конфликты с другими программами из этой среды, которые также используют эти ИД.

Количество доступных ИД событий невелико, поэтому не следует выделять их без необходимости. Для увеличения количества идентификаторов можно воспользоваться 16-разрядным полем данных. В этом случае каждому формальному ИД события можно сопоставить 65536 различных событий. Единственная

причина, по которой стоит применять уникальные идентификаторы, заключается в том, что сбор и фильтрация данных утилитой трассировки выполняется именно на уровне ИД.

Запись о пользовательском событии можно отформатировать с помощью команды **trcrpt**, если указанный файл формата данных трассировки содержит соответствующий раздел. Файл формата данных трассировки представляет собой обычный текстовый файл (дополнительная информация приведена в разделе “Синтаксис разделов в файле формата трассировки” на стр. 402).

Примеры кода и форматирования событий

Приведен пример оценки времени выполнения цикла программы с помощью событий трассировки.

```
#include <sys/trcctl.h>
#include <sys/trcmacros.h>
#include <sys/trchkid.h>
char *ctl_file = "/dev/systrctl";
int ctlfd;
int i;
main()
{
  printf("настройка сбора данных трассировки \n");
  if (trcstart("-ad")){
    perror("trcstart");
    exit(1);
  }

  printf("открытие устройства трассировки \n");
  if((ctlfd = open(ctl_file,0))<0){
    perror(ctl_file);
    exit(1);
  }

  printf("начало трассировки \n");
  if(ioctl(ctlfd,TRCON,0)){
    perror("TRCON");
    exit(1);
  }

  for(i=1;i<11;i++){
    TRCHKL1T(HKWD_USER1,i);

    /* Здесь следует контролируемый фрагмент программы.  */ Интервал */
    /* между появлениями THKWD_USER1 в файле трассировки */
    /* равен времени выполнения одной итерации /       */
  }

  printf("завершение трассировки \n");
  if(ioctl(ctlfd,TRCSTOP,0)){
    perror("TRCOFF");
    exit(1);
  }

  printf("завершение работы демона трассировки \n");
  if (trcstop(0)){
    perror("trcstop");
    exit(1);
  }

  exit(0);
}
```

При компиляции примера программы необходимо подключить библиотеку `librts.a`, как показано ниже:

```
# xlc -O3 sample.c -o sample -l rts
```

HKWD_USER1 - это событие с ИД x010 (список идентификаторов событий приведен в файле `/usr/include/sys/trchkid.h`). Функция создания отчетов отформатирует запись о событии HKWD_USER1

только в том случае, если файл формата данных трассировки содержит соответствующие правила форматирования. Ниже приведен пример раздела, который можно добавить в файл для события HKWD_USER1:

```
# Раздел описания правил форматирования HKWD_USER1
# Данные правила предназначены для события, применяемого в примере программы
010 1.0 L=APPL "USER EVENT - HKWD_USER1" O2.0      \n \
               "Количество итераций =" U4  \n \
               "Время, прошедшее с начала последнего цикла = " \
                endtimer(0x010,0x010) starttimer(0x010,0x010)
```

Не добавляйте этот пример раздела в основной файл формата `/etc/trcfmt`. Вместо этого создайте его копию и сохраните в своем пользовательском каталоге (например, скопируйте его в файл **mytrcfmt**). При запуске примера программы информация о событиях заносится в файл протокола по умолчанию, поскольку для функции **trcstart()** не был задан другой файл протокола. Для того чтобы получить из файла протокола информацию только о пользовательских событиях, задайте фильтр. Для этого введите команду **trcrpt** со следующими параметрами:

```
# trcrpt -d 010 -t mytrcfmt -O "exec=on" > sample.rpt
```

Информация о пользовательских событиях будет записана в файл `sample.rpt`.

Синтаксис разделов в файле формата трассировки

Файл формата данных трассировки содержит правила представления и просмотра данных о событиях с различными идентификаторами. Наличие такого файла позволяет задавать правила форматирования для новых событий, не изменяя функцию создания отчетов.

Правила для новых событий просто добавляются в файл формата. Синтаксис правил позволяет представлять данные различным образом.

Раздел файла формата может содержать любое число правил, необходимых для описания события. Для того чтобы продолжить раздел на следующей строке, в конце текущей строки указывают символ продолжения '\'. Описание полей приведено в в *Справочник по файлам* .

Информация о прочих макрокомандах и правилах форматирования, а также об определении дополнительных макрокоманд приведена в файле `/etc/trcfmt`.

Создание отчетов о проблемах производительности

Если вы считаете, что производительность операционной системы недостаточна по какой-то причине, то вы можете воспользоваться специальными средствами и процедурами для создания отчета и сбора данных для анализа проблемы. Эти средства значительно ускоряют и упрощают получение необходимой информации.

Измерение базового уровня

Снижение производительности часто наблюдается после изменения программной или аппаратной конфигурации. Если данные о производительности системы до внесения изменений отсутствуют, то неполадку трудно описать количественно.

Производительность может измениться при внесении следующих изменений:

- Аппаратная конфигурация - Добавление, удаление и изменение конфигурации, например, способа подключения дисков
- Операционная система - Установка или добавление набора файлов, PTF, изменение параметров
- Приложения - Установка новых версий и исправлений
- Приложения - Настройка и изменение расположения данных
- Настройка приложений
- Настройка опций операционной системы, баз данных и приложений

- Другие изменения

Лучший способ оценки влияния внесенного изменения на производительность заключается в измерении производительности до и после изменения. Другой способ - выполнение измерений через фиксированный интервал времени (например, раз в месяц). В случае снижения производительности для сравнения может быть использован результат предыдущего измерения. Локализация проблем производительности упрощается при наличии результатов большого числа измерений.

Для увеличения ценности собираемых данных выполняйте измерения в те рабочие часы и дни недели, когда система максимально загружена. Например, пики нагрузки могут наблюдаться в следующие моменты:

- В середине дня для интерактивных пользователей
- Поздней ночью для пакетных заданий
- При создании отчетов в конце месяца
- Про обработке больших объемов данных

Выполните отдельные измерения для всех перечисленных событий, поскольку снижение производительности может наблюдаться только в течение одного из указанных периодов.

Примечание: Любое измерение влияет на работу и производительность измеряемой системы. Для измерений рекомендуется пользоваться утилитами сбора статистики администратора производительности AIX (perfpmr). Эти утилиты можно загрузить из сети Internet со следующего сервера: ftp://ftp.software.ibm.com/aix/tools/perftools/perfpmr. Инструкции по установке и использованию утилит для вашей версии AIX приведены в файле README.

Проблемы производительности

Сотрудники службы технической поддержки должны знать, что описывает переданным им отчет: неполадку в работе или проблему производительности.

Если приложение, устройство или сеть работает неправильно, это называется *неполадкой в работе*. Например, утечка памяти - это неполадка в работе.

Некоторые неполадки в работе приводят к проблемам производительности, например, если необходимого результата можно достичь, но за большее время. В этих случаях вместо настройки производительности следует определить причину неполадки и устранить ее. Другой пример - уменьшение скорости передачи данных из-за отключения некоторых сетей или серверов.

Описание проблемы производительности

Сотрудники службы технической поддержки часто получают отчеты, в которых говорится о проблемах производительности и приводится некоторый анализ данных. Этой информации недостаточно для точного определения причин снижения производительности. Отчет может содержать информацию о 100-процентной загруженности процессора или большой длине очереди выполнения, но эти факты ничего не говорят о причине возникновения проблемы производительности.

Например, пользователи подключаются к системе по сети через несколько маршрутизаторов. Пользователи жалуются на низкую скорость работы системы. Данные отчета показывают, что процессор системы перегружен. Однако истинной причиной низкой производительности может быть задержка в выводе символов на терминалы из-за потерь пакетов в сети (связанных с перегрузкой сетей и маршрутизаторов). Таким образом, источник проблемы не обязательно связан с нагрузкой на процессор системы. С другой стороны, слишком долгое выполнение пакетного задания имеет прямое отношение к загруженности процессора и подсистемы ввода-вывода.

Всегда собирайте как можно больше данных перед их анализом и созданием отчета. Для этого ответьте на следующие вопросы о проблеме производительности:

- Можно ли воспроизвести проблему, запустив определенную команду или повторив некоторую последовательность действий? (например: **ls** `/slow/fs` или **ping** *xxxxx*). Если это невозможно, опишите наиболее простой пример возникновения проблемы.
- Постоянно ли снижается производительность? Можно ли сказать, что работа системы замедляется, а затем снова возвращается к нормальному уровню? Происходит ли это в определенное время дня или при выполнении в системе определенных действий?
- Замедляется ли работа всех приложений системы или только некоторых из них?
- Какое именно действие выполняется медленно? Например, это может вывод символа, выполнение транзакции или обновление экрана.
- Когда впервые была замечена проблема? Изменилась ли производительность с момента установки системы? Изменился ли режим работы с системой перед возникновением проблемы (были ли добавлены новые пользователи, загружены дополнительные данные)?
- Если вы работаете в среде клиент-сервер, можно ли воспроизвести проблему на сервере (без участия сети)?
- Если проблема связана с сетью, как настроена сеть (какова ее пропускная способность - 10 Мбит/с или 9600 бод)? Есть ли между клиентом и сервером какие-либо маршрутизаторы?
- Какие приложения других фирм работают в системе? Затрагивает ли эти приложения проблема производительности?
- Каким образом проблема производительности влияет на работу пользователей?

Создание отчета о проблеме производительности

Отчеты о проблемах производительности операционной системы следует направлять в службу технической поддержки фирмы IBM. Для этого можно воспользоваться обычным способом сообщения об ошибках программного обеспечения. Если вы не знаете, какой способ сообщения об ошибках применяется в вашей организации, свяжитесь с представителем фирмы IBM.

Утилиты сбора статистики диспетчера производительности AIX (perfpmr) предоставляют широкие возможности по обнаружению проблем, связанных с производительностью системы AIX. Эти утилиты можно загрузить из Internet со следующего сервера: ftp://ftp.software.ibm.com/aix/tools/perftools/perfpmr. Инструкции по установке и использованию утилит для вашей версии AIX приведены в файле README. Помимо этого, в файле README указаны способы передачи статистики, собранной с помощью этих утилит, в фирму IBM для дальнейшего анализа.

При создании отчета о проблеме производительности недостаточно просто собрать и проанализировать данные. Без описания характера проблемы производительности анализ данных может оказаться пустой тратой времени, поскольку не будет иметь к проблеме никакого отношения.

Перед отправкой отчета о неполадке подготовьте всю информацию, которая может понадобиться сотрудникам службы технической поддержки для анализа и устранения неполадки.

Вы можете ускорить решение проблемы следующими способами:

1. Подготовьте четкое описание одного конкретного случая возникновения проблемы, основываясь только на реальных симптомах и фактах, а не на собственных домыслах и заключениях. При получении отчета со словами "система медленно работает" значительная часть времени уйдет на определение значения слово "медленно", то есть относительно чего система работает медленно и какой должна быть ее производительность.
2. Предоставьте информацию обо всех изменениях, произошедших в системе перед возникновением проблемы. Пропуск какого-либо важного изменения может значительно увеличить время анализа проблемы. Если будут доступны все факты, те из них, которые не относятся к описываемой проблеме, будут быстро выявлены и удалены.
3. Собирайте информацию в правильной системе. На очень больших сайтах легко ошибиться и начать сбор данных в системе, не имеющей отношения к проблеме. Это серьезно затруднит анализ проблемы.

При создании отчета включите в него следующую информацию:

- Описание проблемы, которое позволит найти в базе данных проблем отчеты об аналогичных проблемах и способах их решения.
- Что привело вас к заключению, что причина проблемы связана с ошибкой в операционной системе?
- В какой аппаратной и программной конфигурации возникает проблема?
  - Связана ли проблема с отдельной системой, или она наблюдается в нескольких системах?
  - Какова аппаратная конфигурация системы: модель, объем памяти, число и размер дисков?
  - Какие локальные сети и другие каналы передачи данных подключены к системе?
  - Зависит ли конфигурацию системы от параметров других операционных систем?
- Каковы характеристики программы или рабочей схемы, которая работает медленно?
  - Следует ли из вывода команд **time**, **iostat** и **vmstat**, что производительность соответствующей программы зависит от ресурсов процессора или подсистемы ввода-вывода?
  - Какова роль системы: рабочая станция, сервер, многопользовательская система, или нечто среднее?
- Какие показатели производительности вас не устраивают?
  - Какой из параметров для вас наиболее важен - время ответа при работе с терминалом, пропускная способность или время ответа в реальном времени?
  - Были ли какие-либо результаты получены в других системах? Если да, какова конфигурация этих систем?

При отправке первого отчета о неполадке ему присваивается номер PMR, который следует указывать при передаче дополнительных данных и получении любой информации.

Включите в отчет о проблеме производительности следующие данные **perfpmr**:

- Способ воспроизведения проблемы
  - Если возможно, программу или сценарий оболочки, демонстрирующий наличие проблемы
  - Обязательно - описание условий, при которых обнаружена проблема производительности.
- Приложение, при выполнении которого возникает проблема
  - Полная версия и выпуск каждого продукта, в котором наблюдается проблема производительности, а также всех программ, от которых зависят эти продукты.
  - Если исходный код пользовательского приложения не может быть передан службе технической поддержки, следует сообщить по крайней мере опции компилятора, с которыми был создан исполняемый файл этого приложения.

Отслеживание и настройка производительности, команды и функции

В системе доступны команды и функции для отслеживания и настройки производительности.

Средства настройки производительности делятся на два типа: это средства для получения информации о событиях и средства для выполнения определенных действий при возникновении тех или иных событий. Некоторые средства выполняют обе функции. Более подробная информация о формате и назначении команд приведена в книге *Справочник по командам*.

Команды настройки производительности находятся в наборах файлов `perfagent.tools`, `bos.acct`, `bos.sysmgt.trace` , `bos.adt.samples`, `bos.perf.tools` и `bos.perf.tune`, поставляемых с Базовой операционной системой.

Для того чтобы определить, установлены ли в системе средства настройки производительности, запустите одну из следующих команд:

```
# lslpp -lI perfagent.tools bos.sysmgt.trace bos.acct bos.perf.tools bos.perf.tune
```

Команды сбора статистики и анализа производительности

Команды сбора информации и анализа производительности позволяют получить информацию о производительности одного или нескольких компонентов системы, либо сведения об одном или нескольких параметрах, влияющих на производительность.

К ним относятся следующие команды:

Команда
Описание

alstat Выдает отчет о числе исключительных ситуаций, связанных с ошибками выравнивания

atmstat
Показывает статистику адаптера ATM

curt Показывает использование процессора каждой нитью ядра.

emstat Выдает отчет о числе команд эмуляции

entstat Показывает статистику драйвера и устройства Ethernet

fddistat
Показывает статистику драйвера и устройства FDDI

filemon
С помощью средств трассировки собирает информацию об операциях ввода-вывода для физических томов, логических томов и отдельных файлов, а также для Администратора виртуальной памяти

fileplace
Показывает информацию о физическом или логическом расположении блоков файла в физическом или логическом томе

gprof Выдает отчет о порядке выполнения процедур программы и о процессорном времени, затраченном на выполнение каждой процедуры

ifconfig
Изменяет или показывает значения параметров сетевого интерфейса TCP/IP

ioo Задает параметры настройки ввода-вывода (совместно с **vmo**, заменяет **vmtune**.

iostat Показывает информацию об использовании следующих ресурсов:

- Терминалы
- процессор
- Диски
- Адаптеры

ipfilter Выдает таблицу заголовков операций, полученных из файла вывода **ipreport**

ipreport
Создает отчет о трассировке пакетов на основе указанного файла трассировки пакетов

iptrace Выполняет трассировку пакетов на уровне интерфейса протоколов Internet

locktrace
Включает трассировку блокировок

lsattr Показывает атрибуты системы, влияющие на производительность, например:

- Скорость процессора
- Объем кэша
- Объем физической памяти
- Максимальное число страниц в кэше буфера блочного ввода-вывода
- Максимальный объем памяти, который может занимать mbuf, в КБ

- Минимальное и максимальное число ожидающих операций ввода-вывода

lsdev Выдает информацию о системных устройствах и их свойствах

lslv Выдает информацию о логическом томе

lsps Показывает параметры пространств подкачки

lspv Выдает информацию о физическом томе, входящем в группу томов

lsvg Выдает информацию о группах томов

mtrace Показывает путь многоцелевой рассылки от отправителя до получателя

netpmon
С помощью средств трассировки создает отчет о работе сети, содержащий следующую информацию:

- Загрузка CPU
- Скорость передачи данных
- Время ответа

netstat Показывает различную информацию о конфигурации и статистическую информацию о работе средств связи, в частности:

- Текущее состояние пула mbuf
- Таблицы маршрутизации
- Суммарная статистика работы сети

nfso Показывает (или изменяет) значения параметров NFS

nfsstat Выводит статистику работы серверов и клиентов Сетевой файловой системы (NFS) и службы Вызова удаленных процедур (RPC).

no Показывает (или изменяет) значения сетевых параметров, в том числе:

- Размер буферов сокетов приема и передачи по умолчанию
- Максимальный объем памяти, который может быть выделен для пулов кластеров и mbuf

pdt_config
Запускает средства диагностики производительности, завершает их работу или изменяет параметры

pdt_report
Создает отчет PDT на основе текущих данных хронологии

pprof Выдает отчет об использовании CPU всеми нитями ядра за определенный период времени

prof Показывает содержимое профайла для объектного файла

ps Выводит статистическую информацию и информацию о состоянии процессов в системе, например:

- Идентификатор процесса
- Число операций ввода-вывода
- Использование CPU

sar Показывает статистику работы операционной системы, например:

- Число обращений к каталогам
- Число системных вызовов чтения и записи
- Число операций порождения и запуска процессов
- Число операций подкачки

schedo Задает параметры настройки планировщика процессора (заменяет команду **schedtune**.

smitty Показывает (или изменяет) значения параметров управления системой

splat Инструмент анализа конфликтов блокировок

svmon Сообщает о состоянии памяти на уровне системы, процессов и сегментов

tcpdump
Выводит заголовки пакетов

time, timex
Показывает общее и процессорное время, затраченное на выполнение команды.

topas Выдает статистику по выбранной локальной системе

tokstat Показывает статистику устройства и драйвера Token-Ring

tprof С помощью средств трассировки создает отчет об использовании CPU службами ядра, библиотечными процедурами, модулями прикладных программ и отдельными инструкциями прикладной программы

trace, trcrpt
Создает файл, в котором записывается последовательность операций, выполняемых системой

traceroute
Показывает маршрут IP-пакетов до хоста сети

vmo Задает параметры настройки VMM (совместно с **ioo**, заменяет **vmtune**.

vmstat Показывает информацию VMM, в том числе:

- Число работающих и ожидающих процессов
- Размер списка свободных страниц физической памяти
- Число страничных ошибок
- Использование CPU

Команды для настройки производительности

AIX имеет набор команд, позволяющих изменять параметры системы, влияющие на ее производительность.

Команда
Функция

bindprocessor
Связывает или удаляет связывание нитей ядра с процессором

caccelstat
Отображает статистику, связанную со всеми или отдельными когерентными ускорителями системы.

chdev Изменяет параметры устройства

chlv Изменяет параметры логического тома

chps Изменяет атрибуты пространства подкачки

fdpr Утилита настройки производительности для уменьшения времени выполнения и оптимизации использования физической памяти пользовательскими приложениями

ifconfig
Изменяет или показывает значения параметров сетевого интерфейса TCP/IP

ioo Задает параметры настройки ввода-вывода (совместно с **vmo**, заменяет **vmtune**.

migratepv
Перемещает выделенные физические разделы с одного физического тома на другие физические тома

mkps Расширяет пространство подкачки

nfso Задает параметры Сетевой файловой системы (NFS)

nice Запускает команду с пониженным или повышенным приоритетом

no Задает сетевые атрибуты

renice Изменяет приоритет работающего процесса

reorgvg
Реорганизует расположение физических разделов группы томов

rmss Эмулирует системы с различными объемами памяти для тестирования производительности приложений

schedo Задает параметры настройки планировщика процессора (заменяет команду **schedtune**).

smitty Показывает (или изменяет) значения параметров управления системой

tuncheck
Проверяет файл со значениями параметров настройки.

tundefault
Устанавливает для всех параметров тонкой настройки значения по умолчанию.

tunrestore
Восстанавливает для всех параметров настройки значения из файла настройки.

tunsave
Сохраняет значения всех параметров настройки в файле настройки.

vmo Задает параметры настройки VMM (совместно с **ioo**, заменяет **vmtune**.

Информация, связанная с данной:

Команда caccelstat

Функции оценки и настройки производительности

Для оценки и настройки производительности в AIX имеются следующие подпрограммы.

bindprocessor()
Связывает нити ядра с процессором

getpri()
Позволяет узнать приоритет выполняемого процесса

getpriority()
Позволяет узнать значение nice выполняемого процесса

getrusage()
Выдает информацию об использовании ресурсов системы

nice() Увеличивает значение nice текущего процесса

psdanger()
Выдает информацию об использовании пространства подкачки

setpri()
Устанавливает фиксированный приоритет выполняемого процесса

setpriority()
Устанавливает значение nice выполняемого процесса

Повышение эффективности работы команды ld

Редактор связей, вызываемый на последнем этапе компиляции или напрямую с помощью команды **ld**, содержит функции, которые отсутствуют в обычном компоновщике UNIX.

Эти функции позволяют ускорить компоновку программ. В данном разделе описаны некоторые способы применения редактора связей.

Примеры

Ниже приведен пример использования команды **ld**:

1. Для предварительной компоновки библиотеки вызовите следующую команду для архивного файла:
   ```
   # ld -r libfoo.a -o libfooa.o
   ```
2. После этого для компиляции и компоновки программы `something.f`, написанной на языке FORTRAN, введите:
   ```
   # xlf something.f libfooa.o
   ```
   Обратите внимание, что библиотека, для которой выполнялась предварительная компоновка, задается как обычный файл ввода, а не как библиотека (**-lfoo**).
3. Для перекомпиляции и перекомпоновки модуля после исправления ошибки введите:
   ```
   # xlf something.f a.out
   ```
4. Однако если после исправления программа стала вызывать другую процедуру из библиотеки, компоновка завершится неудачно. Приведенный ниже сценарий оболочки Korn проверяет код возврата и автоматически вызывает нужную команду:
   ```
   # !/usr/bin/ksh
   # Shell script for source file replacement bind
   #
   xlf something.f a.out
   rc=$?
   if [ "$rc" != 0 ]
   then
   echo "Добавлена новая функция libfooa.o"
   xlf something.o libfooa.o
   fi
   ```

Повторная компоновка исполняемых программ

В документации по редактору связей указано, что он может получать на вход исполняемый файл (загрузочный модуль).

Использование такой возможности может заметно повысить производительность при выполнении задач, связанных с разработкой программного обеспечения, а также уменьшить время ответа отдельных команд **ld**.

В большинстве систем UNIX на вход команде **ld** передается набор объектных файлов в виде отдельных файлов `.o` или в виде архивных библиотек файлов `.o`. Команда `ld` обрабатывает ссылки между этими файлами и создает исполняемый файл, который по умолчанию называется `a.out`. Файл `a.out` является в чистом виде исполняемым файлом. Если в одном из модулей, составляющих `a.out`, будет обнаружена ошибка, то придется исправлять и заново компилировать этот модуль, а затем повторять весь процесс **ld** для полного набора объектных файлов `.o`.

В данной операционной системе редактор связей может обрабатывать как файлы `.o`, так и файлы `a.out`, поскольку он записывает в исполняемый файл информацию Словаря внешних имен (ESD) и Словаря перемещений (RLD). Это означает, что вместо полной перекомпиляции проекта достаточно заменить один файл `.o` и повторно скомпоновать исполняемую программу. Так как объем памяти и ресурсов процессора, затрачиваемых на компоновку, напрямую зависят от числа участвующих в компоновке файлов и количества внешних ссылок, компоновка исполняемой программы с одним новым модулем выполняется значительно быстрее, чем повторная компиляция всей программы.

Предварительная компоновка библиотек

В некоторых случаях важна возможность внутренней компоновки библиотеки процедур до ее применения в программах.

Системные библиотеки функций, такие как `libc.a`, поставляются в виде вывода редактора связей, а не в виде архива файлов `.o`. Это позволяет сэкономить процессорное время при связывании приложений с

системными библиотеками, так как редактор связей должен обработать только ссылки на процедуры библиотеки из приложения. Связи между функциями внутри библиотеки уже обработаны.

Многие библиотеки независимых поставщиков по-прежнему поставляются в виде архивов файлов `.o`. При компоновке приложений, использующих такие библиотеки, необходимо каждый раз обрабатывать имена для всей библиотеки. Это приводит к тому, что процедура компоновки выполняется очень долго, особенно при работе с большими библиотеками в медленных системах.

Предварительная компоновка библиотек дает значительный выигрыш в производительности, особенно в системах с минимальным набором ресурсов.

Работа с таймером процессора

Результаты традиционного измерения очень маленьких интервалов времени часто оказываются неточными из-за того, что какое-то время тратиться на выполнение промежуточных фоновых операций операционной системы и функций определения системного времени. Для получения более точного результата можно определить начальное и конечное время интервала с помощью непосредственного обращения к таймеру процессора, а затем вычесть время, затрачиваемое на обработку прерывания.

В архитектурах POWER и POWER2 таймер процессора реализован в виде пары специальных регистров. Архитектура С процессором POWER включает в себя 64-разрядный регистр *TimeBase*. К этим регистрам могут обращаться только программы, написанные на ассемблере.

Примечание: Таймер процессора отсчитывает абсолютное время. Если в промежуток времени между обращениями к таймеру произойдет прерывание, вычисленная продолжительность интервала будет содержать время обработки прерывания, а также, возможно, время работы других процессов, которые могли быть выполнены до того, как управление было возвращено анализируемой программе. Все результаты измерений, полученные с помощью таймера процессора, являются неточными. Их можно использовать только после тщательной проверки.

Для обращения к регистрам TimeBase предусмотрены три библиотечные функции, не зависящие от архитектуры. Ниже приведено описание этих функций:

read_real_time()
: Эта функция получает точное время из соответствующего источника и сохраняет его в виде двух 32-разрядных значений.

read_wall_time()
: Эта функция получает значение *TimeBase* из соответствующего источника и сохраняет его в виде двух 32-разрядных значений.

time_base_to_time()
: Эта функция преобразует содержимое регистра TimeBase в секунды и наносекунды.

Для определения времени и преобразования значения в обычный формат предусмотрено две разные функции, что позволяет минимизировать объем ресурсов, необходимых для получения значения времени.

Ниже приведен пример применения функций **read_real_time()** и **time_base_to_time()** для определения времени выполнения некоторой программы:

```
#include <stdio.h>
#include <sys/time.h>

int main(void) {
   timebasestruct_t start, finish;
   int val = 3;
   int w1, w2;
   double time;

   /* считывание времени перед началом операции */
```

```
   read_real_time(&start, TIMEBASE_SZ);

   /* начало контролируемого фрагмента кода */
   printf("This is a sample line %d \n", val);
   /* конец контролируемого фрагмента кода */

   /* считывание времени после завершения операции
   read_real_time(&finish, TIMEBASE_SZ);

   /* безусловный вызов процедур преобразования, гарантирующий, что */
   /* оба значения представлены в  */
   /* секундах и наносекундах */
   /* независимо от аппаратной платформы.                                    */
   time_base_to_time(&start, TIMEBASE_SZ);
   time_base_to_time(&finish, TIMEBASE_SZ);

   /* Вычисление разности конечного и начального времени */
   w1 = finish.tb_high - start.tb_high; /* возможно, ноль */
   w2 = finish.tb_low - start.tb_low;

   /* Если при измерении произошел перенос из младших разрядов в */
   /* старшие, результаты могут оказаться неверными                  */
   if (w2 < 0) {
      w1--;
      w2 += 1000000000;
   }

   /* преобразование чистого времени исполнения в число микросекунд в */
   /* формате с плавающей точкой */
   time = ((double) w2)/1000.0;
   if (w1 > 0)
      time += ((double) w1)*1000000.0;

   printf("Время исполнения %9.3f мкс \n", time);
   exit(0);
}
```

Для того чтобы максимально упростить процедуру вызова функции таймера и возврата, попробуйте выполнить статическое связывание теста (дополнительная информация приведена в разделе “Использование динамического и статического подключения” на стр. 381).

Если эти тесты дают близкие к истинным показатели производительности, то измерения кода будут повторяемыми. Зная это время, можно рассчитать среднее время операции, однако это среднее время может включать затраты на обработку прерываний и другие посторонние операции. При измерении времени выполнения для каждого отдельного запуска можно проанализировать результаты и выбрать наиболее разумные, однако при этом в каждом измерении будут присутствовать отклонения, связанные с доступом к таймеру. Иногда бывает полезно объединить оба подхода и сравнить результаты. В любом случае при выборе метода измерения следует учитывать, с какой целью они проводятся.

Обращение к таймеру POWER

В процессорах семейство POWER и POWER2 есть два специализированных регистра (верхний и нижний), в которых содержится сигнал таймера с высокой точностью.

Примечание: Приведенная ниже информация относится только к архитектурам семейство POWER и POWER2 (и процессору 601). Приведенная программа будет работать и в системе PowerPC, но некоторые из команд будут эмулированы. Цель обращения к таймеру процессора заключается в получении точной информации о времени с минимальным влиянием на скорость выполнения программы, а эмуляция снижает точность получаемых результатов.

В верхнем регистре содержится время в секундах, а в нижнем регистре - доли секунды, выраженные в наносекундах. Точность времени в нижнем регистре определяется частотой обновления регистра и зависит от модели процессора.

Ассемблерные процедуры для работы с регистрами таймеров С процессором POWER

Модуль на ассемблере `timer.s` содержит функции (`rtc_upper` и `rtc_lower`) для обращения к верхнему и нижнему регистрам таймера.

```
          .globl  .rtc_upper
.rtc_upper: mfspr   3,4         # копирование RTCU в регистр возврата
            br

          .globl  .rtc_lower
.rtc_lower: mfspr   3,5         # копирование RTCL в регистр возврата
            br
```

Функция С для вывода времени в секундах

Модуль (`second.c`) содержит функцию на языке С, вызывающую функции `timer.s` для получения значений из верхнего и нижнего регистров.

Эта функция возвращает значение времени в секундах в виде действительного числа с двойной точностью.

```
double second()
{
  int ts, tl, tu;

  ts = rtc_upper();     /* секунды */
  tl = rtc_lower();     /* наносекунды */
  tu = rtc_upper();     /* Проверка переноса разряда из */
  if (ts != tu)         /* нижнего регистра в верхний      */
    tl = rtc_lower();   /* Возврат в исходное состояние. */
  return ( tu + (double)tl/1000000000 );
}
```

Функцию **second()** можно вызывать из программ на языке С или FORTRAN.

Примечание: Точность результата, возвращаемого программой `second.c`, может зависеть от того, сколько времени прошло с момента последней перезагрузки системы. Чем оно больше, тем большее количество значащих разрядов может быть потрачено на представление целой части времени в секундах. Способ, описанный в первой части данного приложения, позволяет обойти эту проблему путем вычисления разности двух значений времени до преобразования в формат с плавающей точкой.

Работа с регистрами таймера в системах PowerPC

В архитектуре на PowerPC предусмотрен 64-разрядный регистр *TimeBase*, логически разделенный на два 32-разрядных поля - TBU (верхнее) и TBL (нижнее).

Частота обновления регистра *TimeBase* зависит от аппаратной и программной конфигурации и может меняться со временем. Преобразование значений из регистра TimeBase в секунды требует более сложной процедуры, чем предусмотрено в архитектуре С процессором POWER. В архитектуре PowerPC для этого предусмотрены интерфейсы **read_real_time()**, **read_wall_time()** и **time_base_to_time()**.

Пример функции second()

В программах можно вызывать функцию **second()**.

```
#include <stdio.h>
double second();
main()
{
      double t1,t2;
```

```
      t1 = second();
      my_favorite_function();
      t2 = second();

     printf("my_favorite_function time: %7.9f\n",t2 - t1);
      exit();
}
```

Ниже приведен пример программы на языке FORTRAN (`main.f`), также использующей подпрограмму **second()**:

```
      double precision t1
      double precision t2

      t1 = second()
      my_favorite_subroutine()
      t2 = second()
      write(6,11) (t2 - t1)
11    format(f20.12)
      end
```

Для компиляции программы `main.c` или `main.f` вызовите следующую команду:

```
xlc -O3 -c second.c timer.s
xlf -O3 -o mainF main.f second.o timer.o
xlc -O3 -o mainC main.c second.o timer.o
```

Определение быстродействия процессора

В этом разделе описано, как можно определить быстродействие процессора.

Для того чтобы определить тактовую частоту процессора в герцах (Гц) выполните следующую команду:

```
lsattr -E -l  proc0 | grep "Processor Speed"
```

В системах предыдущих выпусков для этого нужно вызвать команду **uname**. Команда **uname -m** выдает значение в следующем формате:

```
xxyyyyyymmss
```

где:

xx 00

yyyyyy Идентификатор процессора

mm Модель (по этому значению можно определить быстродействие процессора)

ss 00 (тип модели)

Для того чтобы определить быстродействие процессора, найдите значение `mm`, полученное в выводе команды **uname -m**, в приведенной ниже таблице.

Модель	Тип компьютера	Быстродействие процессора	Архитектура
02	7015-930	25	Power
10	7013-530	25	Power
10	7016-730	25	Power
11	7013-540	30	Power
14	7013-540	30	Power
18	7013-53H	33	Power
1C	7013-550	41.6	Power
20	7015-930	25	Power
2E	7015-950	41	Power
30	7013-520	20	Power
31	7012-320	20	Power
34	7013-52H	25	Power
35	7012-32H	25	Power

37	7012-340	33	Power
38	7012-350	41	Power
41	7011-220	33	RSC
43	7008-M20	33	Power
43	7008-M2A	33	Power
46	7011-250	66	PowerPC
47	7011-230	45	RSC
48	7009-C10	80	PowerPC
4C		См. примечание 1.	
57	7012-390	67	Power2
57	7030-3BT	67	Power2
57	9076-SP2 Thin	67	Power2
58	7012-380	59	Power2
58	7030-3AT	59	Power2
59	7012-39H	67	Power2
59	9076-SP2 Thin w/L2	67	Power2
5C	7013-560	50	Power
63	7015-970	50	Power
63	7015-97B	50	Power
64	7015-980	62.5	Power
64	7015-98B	62.5	Power
66	7013-580	62.5	Power
67	7013-570	50	Power
67	7015-R10	50	Power
70	7013-590	66	Power2
70	9076-SP2 Wide	66	Power2
71	7013-58H	55	Power2
72	7013-59H	66	Power2
72	7015-R20	66	Power2
72	9076-SP2 Wide	66	Power2
75	7012-370	62	Power
75	7012-375	62	Power
75	9076-SP1 Thin	62	Power
76	7012-360	50	Power
76	7012-365	50	Power
77	7012-350	41	Power
77	7012-355	41	Power
77	7013-55L	41.6	Power
79	7013-591	77	Power2
79	9076-SP2 Wide	77	Power2
80	7015-990	71.5	Power2
81	7015-R24	71.5	Power2
89	7013-595	135	P2SC
89	9076-SP2 Wide	135	P2SC
94	7012-397	160	P2SC
94	9076-SP2 Thin	160	P2SC
A0	7013-J30	75	PowerPC
A1	7013-J40	112	PowerPC
A3	7015-R30	См. примечание 2.	PowerPC
A4	7015-R40	См. примечание 2.	PowerPC
A4	7015-R50	См. примечание 2.	PowerPC
A4	9076-SP2 High	См. примечание 2.	PowerPC
A6	7012-G30	См. примечание 2.	PowerPC
A7	7012-G40	См. примечание 2.	PowerPC
C0	7024-E20	См. примечание 3.	PowerPC
C0	7024-E30	См. примечание 3.	PowerPC
C4	7025-F30	См. примечание 3.	PowerPC
F0	7007-N40	50	ThinkPad

Примечание:

1. Если в выводе команды **uname -m** указана модель 4C, то единственный способ определить быстродействие процессора - перезагрузить систему, запустить Службы управления системой и просмотреть параметры конфигурации системы. Однако в некоторых случаях быстродействие процессора можно определить и по выводу команды **uname -M**, как показано в приведенной ниже таблице:

```
uname -M        Тип компьютера  Быстродействие процессора Архитектура
IBM,7017-S70         7017-S70          125                RS64
IBM,7017-S7A         7017-S7A          262                RD64-II
IBM,7017-S80         7017-S80          450                RS-III
IBM,7025-F40         7025-F40          166/233            PowerPC
IBM,7025-F50         7025-F50       См. примечание 4.     PowerPC
IBM,7026-H10         7026-H10          166/233            PowerPC
IBM,7026-H50         7026-H50       См. примечание 4.     PowerPC
IBM,7026-H70         7026-H70          340                RS64-II
IBM,7042/7043 (ED)   7043-140          166/200/233/332    PowerPC
IBM,7042/7043 (ED)   7043-150          375                PowerPC
IBM,7042/7043 (ED)   7043-240          166/233            PowerPC
IBM,7043-260         7043-260          200                Power3
IBM,7248             7248-100          100                PowerPersonal
IBM,7248             7248-120          120                PowerPersonal
IBM,7248             7248-132          132                PowerPersonal
IBM,9076-270      9076-SP Silver Node См. примечание 4.   PowerPC
```

2. В системах MCA SMP типа J-Series, R-Series и G-Series быстродействие процессора можно определить по номеру FRU карты процессора, который можно узнать с помощью следующей команды:

```
# lscfg -vl cpucard0 | grep FRU
Номер FRU      Тип процессора    Быстродействие процессора
     E1D           PowerPC 601       75
     C1D           PowerPC 601       75
     C4D           PowerPC 604       112
     E4D           PowerPC 604       112
     X4D           PowerPC 604e      200
```

3. В системах типа E-series и F-30 для определения быстродействия процессора вызовите следующую команду:

```
# lscfg -vp | pg
```

Найдите следующий раздел:

```
procF0                      Карта CPU

Код устройства.............093H5280
Уровень EC.................00E76527
Серийный номер.............17700008
Номер FRU .................093H2431
Сообщение с описанием.......Карта CPU
Параметр устройст.(PL)......
Параметр устройст.(ZA)......PS=166,PB=066,PCI=033,NP=001,CL=02,PBH
                                      Z=64467000,PM=2.5,L2=1024
Параметр устройст.(RM)......10031997 140951 VIC97276
Уровень и ИД ROS............03071997 135048
```

В строке `Параметр устройст. (ZA)` значение `PS=` задает быстродействие процессора в МГц.

4. В системах F-50, H-50 и SP Silver Node для определения быстродействия процессора вызовите следующую команду:

```
# lscfg -vp | more
```

Найдите следующий раздел:

```
Orca M5 CPU:
Код устройства.............08L1010
Уровень EC.................E78405
Серийный номер.............L209034579
Номер FRU..................93H8945
Изготовитель...............IBM980
Версия.....................RS6K
Сообщение с описанием.......OrcaM5 CPU DD1.3
Параметр устройст.(ZC)......PS=0013c9eb00,PB=0009e4f580,SB=0004f27
                                      ac0,NP=02,PF=461,PV=05,KV=01,CL=1
```

В строке `Параметр устройст.(ZC)`, значение `PS=` задает быстродействие процессора в шестнадцатеричном формате. Для преобразования этого значения в обычный формат воспользуйтесь следующими соотношениями:

```
0009E4F580 = 166 МГц
0013C9EB00 = 332 МГц
```

Значение `PF=` задает конфигурацию процессора.

```
251 = 1 процессор с быстродействием 166 МГц
261 = 2 процессора с быстродействием 166 МГц
451 = 1 процессор с быстродействием 332 МГц
461 = 2 процессора с быстродействием 332 МГц
```

Поддержка национальных языков (NLS): Зависимость производительности от локали

Поддержка национальных языков (NLS) дает возможность пользователям, говорящим на разных языках, работать с операционной системой. Поскольку для достижения оптимальной производительности системы необходимо соблюдать некоторые правила применения NLS, в этом приложении приведено краткое описание NLS.

NLS позволяет изменить язык и национальные стандарты, применяемые в операционной системе. *Локалью* называется группа параметров языка и национальных стандартов, объединенных под общим именем, например, en_US (английский язык, США). Каждой локали соответствует набор сообщений, таблиц сортировки и другой информации, определяющей требования локали. Администратор выбирает локаль при установке операционной системы. Пользователи могут изменить локаль отдельной оболочки с помощью переменных *LANG* и *LC_ALL*.

Одной из локалей, не соответствующих описанной выше структуре, является локаль C (или POSIX). Локаль C - это локаль системы по умолчанию, если другая локаль не указана явно. В этой локали запускаются порождаемые процессы. При этом процесс будет работать примерно так же, как и в исходной версии операционной системы UNIX, поддерживающей только один язык. Каталоги сообщений в C отсутствуют. Программы, которые пытаются считать сообщение из каталога, получают сообщение по умолчанию, установленное в программе при компиляции. Некоторые команды, например, **sort**, изменяют алгоритм своей работы и выполняют обработку с учетом набора символов.

По своему влиянию на производительность системы NLS можно поделить на три группы. Как правило, быстрее всего команды выполняются в локали C. За ней следуют локали с набором однобайтовых символов (с латинским алфавитом), такие как en_US. Медленнее всего команды выполняются при использовании локалей с набором многобайтовых символов.

Советы программистам

При разработке программы с поддержкой национальных языков следует принимать во внимание несколько моментов.

Исторически сложилось так, что в языке C длина символа (тип character) составляет 1 байт, т.е. слова "символ" (character) и "байт" (byte) - синонимы. Таким образом, размер массива, объявленного как `char foo[10]`, составляет 10 байт. Однако не все языки мира позволяют применять однобайтовый набор символов. Например, в японском и китайском языках для идентификации символа применяется два или более байт. Поэтому мы будем различать понятия "байт" (означающее 8 бит данных) и "символ" (представляющее собой объем информации, необходимый для идентификации одного графического символа).

Локаль характеризуется двумя символами - числом байт, необходимым для описания одного символа, и числом позиций экрана, необходимым для вывода одного символа. Эти значения могут быть получены с помощью макроопределений **MB_CUR_MAX** и **MAX_DISP_WIDTH**. Если оба значения равны единице, то в

текущей локали понятия "байт" и "символ" эквивалентны. Если хотя бы одно из значений больше 1, то программы, выполняющие посимвольную обработку или следящие за выводом на экран, должны применять функции поддержки национальных языков.

Так как многобайтовые кодировки используют переменное число байт для представления символа, они не могут обрабатываться с помощью массивов типа char. Для повышения эффективности работы с символами многобайтовых кодировок был определен тип данных фиксированного размера **wchar_t**. Размер **wchar_t** достаточен для размещения преобразованного кода любого символа из поддерживаемых кодировок. Программисты могут объявлять массивы типа **wchar_t** и работать с ними примерно так же, как и с массивами типа **char**, используя функции работы с многобайтовыми символами, аналогичные стандартным функциям `libc.a`.

К сожалению, преобразование символов из многобайтового представления, в котором текст вводится, хранится на диске и выводится на экран, в формат **wchar_t** требует значительных ресурсов системы. Этот тип следует применять только тогда, когда эффективность работы с массивами **wchar_t** будет явно компенсировать затраты на преобразование из многобайтового представления в формат **wchar_t**.

Приемы, позволяющие упростить программу

Многоязычное приложение может работать слишком медленно, если программист не знаком с некоторыми особенностями применения наборов многобайтовых символов, позволяющими эффективно выполнять программы в многобайтовой среде с ограниченным использованием языковых функций.

Например:

- Во всех кодовых наборах, поддерживаемых фирмой IBM, символы с кодами от 0x00 до 0x3F не изменяются и соответствуют стандартной кодировке ASCII. Это означает, что их битовые последовательности никогда не входят в состав многобайтового символа. Так как пустой символ входит в этот набор, функции **strlen()**, **strcpy()** и **strcat()** могут работать со строками как многобайтовых, так и однобайтовых символов. Программист должен помнить, что **strlen()** возвращает число байт в строке, а не число символов.
- Аналогично, стандартная функция **strchr(foostr, '/')** будет правильно работать во всех локалях, так как символ "/" (косая черта) входит в набор уникальных кодовых знаков. Значительная часть стандартных разделителей входят в диапазон 0x00 - 0x3F, поэтому большинство задач синтаксического анализа можно выполнять без использования языковых функций или преобразования в формат **wchar_t**.
- Операции сравнения строк подразделяются на два класса - проверка равенства и проверка неравенства. Для сравнения строк применяется стандартная функция **strcmp()**. Условие

  ```
  if (strcmp(foostr,"a rose") == 0)
  ```

  означает, что необходим в точности текст `"a rose"`. Если строка `foostr` содержит текст `"a rosE"`, то такое вхождение найдено не будет.
- Проверка неравенства применяется для упорядочения строк в соответствии с правилами текущей локали. В этом случае следует применять

  ```
  if (strcoll(foostr,barstr) > 0)
  ```

  что приводит к снижению производительности из-за чтения информации об упорядочении для каждого символа.
- При запуске программы всегда устанавливается локаль C. Если в программе будет применяться поддержка языковых функций, включая работу с каталогами сообщений, то перед вызовом языковых функций необходимо выполнить команду

  ```
  setlocale(LC_ALL, "");
  ```

  Эта команда устанавливает локаль родительского процесса.

Изменение локали

Изменить локаль можно с помощью команды **export**.

Последовательность команд

```
LANG=C
export LANG
```

устанавливает локаль `C` в качестве локали по умолчанию (другими словами, локаль `C` будет применяться во всех случаях, когда явным образом не задана языковая переменная, например, *LC_COLLATE*).

Последовательность команд

```
LC_ALL=C
export LC_ALL
```

устанавливает локаль `C` для всех языковых переменных, независимо от их текущих значений.

Текущее значение языковых переменных можно определить с помощью команды `locale`.

Настраиваемые параметры

На производительность операционной системы влияет множество параметров.

В каждом из разделов параметры перечислены в алфавитном порядке.

Переменные среды

Описаны два типа переменных среды, влияющие на работу нитей и прочие параметры.

Настраиваемые параметры поддержки нити

Существует множество параметров поддержки нити, которые можно настроить.

1. **ACT_TIMEOUT**

Элемент	**Описание**
Назначение:	Задает тайм-аут активации в секундах.
Значения:	По умолчанию: DEF_ACTOUT. Диапазон: Натуральное число
Просмотр:	**echo $ACT_TIMEOUT** Значение присваивается системой, поэтому начальное значение по умолчанию нельзя просмотреть командой **echo**.
Изменение:	**ACT_TIMEOUT=n export ACT_TIMEOUT** В данной оболочке изменения вступают в силу немедленно. Изменение действительно до выхода из оболочки. Для того чтобы сделать изменение постоянным, добавьте команду **ACT_TIMEOUT=***n* в файл `/etc/environment`.
Рекомендации:	Нет
Настройка:	Нет

Дополнительная информация: “Переменные среды нити” на стр. 72.

2. **AIXTHREAD_COND_DEBUG**

Элемент	Описание
Назначение:	Определяет список переменных, используемых отладчиком.
Значения:	По умолчанию: OFF. Диапазон: ON, OFF.
Просмотр:	**echo $AIXTHREAD_COND_DEBUG** Значение присваивается системой, поэтому начальное значение по умолчанию нельзя просмотреть командой **echo**.
Изменение:	`AIXTHREAD_COND_DEBUG={ON|OFF}export AIXTHREAD_COND_DEBUG` В данной оболочке изменения вступают в силу немедленно. Изменение действительно до выхода из оболочки. Для того чтобы сделать изменение постоянным, добавьте команду **AIXTHREAD_COND_DEBUG=*{ON|OFF}*** в файл `/etc/environment`.
Рекомендации:	Указание в этой переменной значения **ON** упрощает отладку многонитевых приложений, но уменьшает производительность.
Настройка:	Если в программе используется много активных переменных условия, то их интенсивное создание и уничтожение может потребовать слишком больших затрат ресурсов. Для отключения списка переменных условия присвойте этой переменной среды значение **OFF**.

За дополнительной информацией обратитесь к разделу “Опции отладки нити” на стр. 77.

3. **AIXTHREAD_DISCLAIM_GUARDPAGES**

Элемент	Описание
Назначение:	Управляет использованием вспомогательных страниц при создании стека.
Значения:	По умолчанию: OFF. Диапазон: ON, OFF.
Просмотр:	**echo $AIXTHREAD_DISCLAIM_GUARDPAGES** Значение присваивается системой, поэтому начальное значение по умолчанию нельзя просмотреть командой **echo**.
Изменение:	`AIXTHREAD_DISCLAIM_GUARDPAGES={ON|OFF};export AIXTHREAD_DISCLAIM_GUARDPAGES` В данной оболочке изменения вступают в силу немедленно. Изменение действительно до выхода из оболочки. Для того чтобы сделать изменение постоянным, добавьте команду **AIXTHREAD_GUARDPAGES=*n*** в файл `/etc/environment`.
Рекомендации:	NA
Настройка:	Если в стеках pthread используются вспомогательные страницы, установка значения **AIXTHREAD_DISCLAIM_GUARDPAGES = ON** приводит к тому, что при последующем создании pthread вспомогательные страницы отключатся. Этот параметр позволяет уменьшить использовании памяти в многопоточных приложениях.

За дополнительной информацией обратитесь к разделу “Переменные среды нити” на стр. 72.

4. **AIXTHREAD_ENRUSG**

Элемент	Описание
Назначение:	Разрешает или запрещает использование набора ресурсов pthread.
Значения:	По умолчанию: OFF. Диапазон: ON, OFF.
Просмотр:	**echo $AIXTHREAD_ENRUSG** Этот режим включается системой, поэтому начальное значение по умолчанию нельзя просмотреть командой **echo**.
Изменение:	`AIXTHREAD_ENRUSG={ON|OFF}export AIXTHREAD_ENRUSG` В данной оболочке изменения вступают в силу немедленно. Изменение действительно до выхода из оболочки. Для того чтобы сделать изменение постоянным, добавьте команду **AIXTHREAD_ENRUSG=*{ON|OFF}*** в файл `/etc/environment`.
Рекомендации:	Если этому параметру присвоено значение **ON**, то всем нитям pthread процесса будет разрешено применять набор ресурсов, однако при этом возрастет нагрузка на систему.
Настройка:	Нет

За дополнительной информацией обратитесь к разделу “Переменные среды нити” на стр. 72.

5. **AIXTHREAD_GUARDPAGES**

Элемент	Описание
Назначение:	Задает число вспомогательных страниц, добавляемых в конец стека pthread.
Значения:	Значение по умолчанию: 1 (где 1 - десятичное число для количества страниц, которое может принимать значения 4K, 64K и т. д.) Диапазон: диапазон n.
Просмотр:	**echo $AIXTHREAD_GUARDPAGES** Этот режим включается системой, поэтому начальное значение по умолчанию нельзя просмотреть с помощью команды **echo**.
Изменение:	`AIXTHREAD_GUARDPAGES=`*n*`export AIXTHREAD_GUARDPAGES` В данной оболочке изменения вступают в силу немедленно. Изменение действительно до выхода из оболочки. Для того чтобы сделать изменение постоянным, добавьте команду **AIXTHREAD_GUARDPAGES=*n*** в файл `/etc/environment`.
Рекомендации:	Нет
Настройка:	Нет

За дополнительной информацией обратитесь к разделу “Переменные среды нити” на стр. 72.

6. **AIXTHREAD_MINKTHREADS**

Элемент	Описание
Назначение:	Задает минимальное число используемых нитей ядра.
Значения:	По умолчанию: 8. Диапазон: Положительное целое число.
Просмотр:	**echo $AIXTHREAD_MINKTHREADS** Этот режим включается системой, поэтому начальное значение по умолчанию нельзя просмотреть с помощью команды **echo**.
Изменение:	**AIXTHREAD_MINKTHREADS=*n*export AIXTHREAD_MINKTHREADS** В данной оболочке изменения вступают в силу немедленно. Изменение действительно до выхода из оболочки. Для того чтобы сделать изменение постоянным, добавьте команду **AIXTHREAD_MINKTHREADS =*n*** в файл `/etc/environment`.
Рекомендации:	Нет
Настройка:	Планировщик библиотеки не возвращает нити ядра, если их число меньше указанного. Нить ядра может быть возвращена практически в любой момент. Обычно нить ядра возвращается после завершения нити pthread.

Дополнительная информация:“Переменные локальной области действия” на стр. 77.

7. **AIXTHREAD_MNRATIO**

Элемент	Описание
Назначение:	Определяет коэффициент масштабирования библиотеки. Этот коэффициент применяется при создании нитей pthread и завершении их работы.
Значения:	По умолчанию: 8:1 Диапазон: Отношение двух натуральных чисел (*p:k*), где *k* - число нитей ядра, которые должны применяться для выполнения переменной *p*.
Просмотр:	**echo $AIXTHREAD_MNRATIO** Этот режим включается системой, поэтому начальное значение по умолчанию нельзя просмотреть с помощью команды **echo**.
Изменение:	**AIXTHREAD_MNRATIO=*p:k*export AIXTHREAD_MNRATIO** В данной оболочке изменения вступают в силу немедленно. Изменение действительно до выхода из оболочки. Для того чтобы сделать изменение постоянным, добавьте команду **AIXTHREAD_MNRATIO=*p:k*** в файл `/etc/environment`.
Рекомендации:	Нет
Настройка:	Рекомендуется для приложений с очень большим количеством нитей. Рекомендуется всегда пытаться установить коэффициент 1:1, который во многих случаях обеспечивает максимальную производительность.

Дополнительная информация: “Переменные локальной области действия” на стр. 77

8. **AIXTHREAD_MUTEX_DEBUG**

Элемент	Описание
Назначение:	Определяет список активных взаимных блокировок, применяемый отладчиком.
Значения:	По умолчанию: OFF. Допустимые значения: ON, OFF.
Просмотр:	**echo $AIXTHREAD_MUTEX_DEBUG** Этот режим включается системой, поэтому начальное значение по умолчанию нельзя просмотреть с помощью команды **echo**.
Изменение:	**AIXTHREAD_MUTEX_DEBUG=***{ON|OFF}***export AIXTHREAD_MUTEX_DEBUG** Изменение вступает в силу немедленно и действует до вашего выхода из оболочки. Для того чтобы сделать изменение постоянным, добавьте команду **AIXTHREAD_MUTEX_DEBUG=***{ON|OFF}* в файл `/etc/environment`.
Рекомендации:	Указание в этой переменной значения `ON` упрощает отладку многонитевых приложений, но уменьшает производительность.
Настройка:	Если в программе используется много активных взаимных блокировок, то их интенсивное создание и уничтожение может потребовать слишком больших затрат ресурсов. Если в переменной указано значение `OFF`, список не создается.

Дополнительная информация: “Опции отладки нити” на стр. 77

9. **AIXTHREAD_MUTEX_FAST**

Элемент	Описание
Назначение:	Включает применение оптимизированного механизма взаимной блокировки.
Значения:	По умолчанию: OFF. Допустимые значения: ON, OFF.
Просмотр:	**echo $AIXTHREAD_MUTEX_FAST** Этот режим включается системой, поэтому начальное значение по умолчанию нельзя просмотреть с помощью команды **echo**.
Изменение:	**AIXTHREAD_MUTEX_FAST=***{ON|OFF}***export AIXTHREAD_MUTEX_FAST** Изменение вступает в силу немедленно и действует до вашего выхода из оболочки. Для того чтобы сделать изменение постоянным, добавьте команду **AIXTHREAD_MUTEX_FAST=***{ON|OFF}* в файл `/etc/environment`.
Рекомендации:	Если для переменной задано значение `ON` приложения с нитями будут использовать оптимизированный механизм взаимной блокировки, что приведет к повышению производительности.
Настройка:	Если производительность программы заметно снижается из-за взаимных операций, то при назначении для данной переменной значения `ON` библиотека pthread будет использовать оптимизированный механизм взаимной блокировки, работающий только в частных взаимных блокировках процесса. Эти частные взаимные блокировки процесса должны быть инициализированы с помощью процедуры pthread_mutex_init и должны быть уничтожены с помощью процедуры pthread_mutex_destroy.

Дополнительная информация: “Опции отладки нити” на стр. 77

10. **AIXTHREAD_READ_GUARDPAGES**

Элемент	Описание
Назначение:	Устанавливает права на чтение вспомогательных страниц, добавляемых в конец стека pthread.
Значения:	По умолчанию: OFF. Диапазон: ON, OFF.
Просмотр:	**echo $AIXTHREAD_READ_GUARDPAGES** Этот режим включается системой, поэтому начальное значение по умолчанию нельзя просмотреть с помощью команды **echo**.
Изменение:	`AIXTHREAD_READ_GUARDPAGES={ON|OFF}export AIXTHREAD_GUARDPAGES` В данной оболочке изменения вступают в силу немедленно. Изменение действительно до выхода из оболочки. Для того чтобы сделать изменение постоянным, добавьте команду **AIXTHREAD_READ_GUARDPAGES={ON|OFF}** в файл `/etc/environment`.
Рекомендации:	Нет
Настройка:	Нет

За дополнительной информацией обратитесь к разделу “Переменные среды нити” на стр. 72.

11. **AIXTHREAD_RWLOCK_DEBUG**

Элемент	Описание
Назначение:	Определяет список блокировок чтения-записи, применяемый отладчиком.
Значения:	По умолчанию: OFF. Диапазон: ON, OFF.
Просмотр:	**echo $AIXTHREAD_RWLOCK_DEBUG** Этот режим включается системой, поэтому начальное значение по умолчанию нельзя просмотреть с помощью команды **echo**.
Изменение:	**AIXTHREAD_RWLOCK_DEBUG=***{ON|OFF}***export AIXTHREAD_RWLOCK_DEBUG** В данной оболочке изменения вступают в силу немедленно. Изменение действительно до выхода из оболочки. Для того чтобы сделать изменение постоянным, добавьте команду **AIXTHREAD_RWLOCK_DEBUG=***{ON|OFF}* в файл `/etc/environment`.
Рекомендации:	Задав для этого параметра значение `ON` вы сделаете процесс отладки легче, но это может привести к увеличению нагрузки на систему.
Настройка:	Если в программе используется много активных блокировок чтения-записи, то их интенсивное создание и уничтожение может потребовать слишком больших затрат. Для отключения списка переменных условия присвойте этой переменной среды значение `OFF`.

Дополнительная информация: “Опции отладки нити” на стр. 77

12. AIXTHREAD_SUSPENDIBLE

Элемент	Описание
Назначение:	Предотвращает тупики в приложениях, использующих следующие процедуры с процедурами **pthread_suspend_np** или **pthread_suspend_others_np**: • pthread_getrusage_np • pthread_cancel • pthread_detach • pthread_join • pthread_getunique_np • pthread_join_np • pthread_setschedparam • pthread_getschedparam • pthread_kill
Значения:	По умолчанию: OFF. Диапазон: ON, OFF.
Просмотр:	**echo $AIXTHREAD_SUSPENDIBLE** Этот режим включается системой, поэтому начальное значение по умолчанию нельзя просмотреть с помощью команды **echo**.
Изменение:	**AIXTHREAD_SUSPENDIBLE=***{ON|OFF}***export AIXTHREAD_SUSPENDIBLE** В данной оболочке изменения вступают в силу немедленно. Изменение действительно до выхода из оболочки. Для внесения изменения на постоянной основе добавьте команду **AIXTHREAD_SUSPENDIBLE=***{ON|OFF}* в файл `/etc/environment`.
Рекомендации:	При использовании этой переменной слегка снижается производительность.
Настройка:	Включат эту переменную следует только если упомянутые выше функции используются с процедурой **pthread_suspend_np routine** или **pthread_suspend_others_np**.

Дополнительная информация: “Опции отладки нити” на стр. 77

13. AIXTHREAD_SCOPE

Элемент	Описание
Назначение:	Определяет область действия. Значение P обозначает локальную (внутри процесса) область действия (M:N). Значение S - системную область действия (1:1).
Значения:	По умолчанию : P. Допустимые значения: P или S.
Просмотр:	**echo $AIXTHREAD_SCOPE** Этот режим включается системой, поэтому начальное значение по умолчанию нельзя просмотреть с помощью команды **echo**.

Элемент	Описание
Изменение:	**AIXTHREAD_SCOPE=***{P|S}***export AIXTHREAD_SCOPE** В данной оболочке изменения вступают в силу немедленно. Изменение действительно до выхода из оболочки. Для внесения изменения на постоянной основе добавьте команду **AIXTHREAD_SCOPE=***{P|S}* в файл `/etc/environment`.
Рекомендации:	Если запускается меньше нитей, чем ожидалось, попробуйте установить системную область действия.
Настройка:	Тестирование показало, что некоторые приложения работают лучше, если установлена системная область действия (S). Данная переменная среды влияет только на нити, при создании которых был установлен атрибут по умолчанию. Атрибут по умолчанию применяется в том случае, если в параметре *attr* команды **pthread_create** указано пустое значение.

Дополнительная информация: “Переменные среды нити” на стр. 72

14. AIXTHREAD_SLPRATIO

Элемент	Описание
Назначение:	Задает число нитей ядра, которые следует зарезервировать для ожидающих нитей.
Значения:	По умолчанию: 1:12. Диапазон: Отношение двух натуральных чисел (*k:p*), где k - число нитей, которые следует зарезервировать для p ожидающих нитей pthread.
Просмотр:	**echo $AIXTHREAD_SLPRATIO** Этот режим включается системой, поэтому начальное значение по умолчанию нельзя просмотреть с помощью команды **echo**.
Изменение:	**AIXTHREAD_SLPRATIO=***k:p***export AIXTHREAD_SLPRATIO** В данной оболочке изменения вступают в силу немедленно. Изменение действительно до выхода из оболочки. Для того чтобы сделать изменение постоянным, добавьте команду **AIXTHREAD_SLPRATIO=***k:p* в файл `/etc/environment`.
Рекомендации:	Нет
Настройка:	В общем случае, для поддержки ожидающих нитей pthread требуется меньше нитей ядра, так как обычно ожидающие нити активируются по одной. Это позволяет сэкономить ресурсы ядра.

Дополнительная информация: “Переменные локальной области действия” на стр. 77

15. AIXTHREAD_STK=n

Элемент	Описание
Назначение:	Количество байт, выделяемое каждой нити pthread (значение должно быть указано в десятичной системе счисления). Это значение может быть переопределено процедурой **pthread_attr_setstacksize**.
Значения:	По умолчанию: 98 304 байт для 32-разрядных приложений, 196 608 байт для 64-разрядных приложений. Диапазон: Десятичные целые значения от 0 до 268 435 455, округляемые до ближайшей страницы (в настоящее время размер страницы составляет 4 096).
Просмотр:	**echo $AIXTHREAD_STK** Этот режим включается системой, поэтому начальное значение по умолчанию нельзя просмотреть с помощью команды echo.
Изменение:	**AIXTHREAD_STK=sizeexport AIXTHREAD_STK** В данной оболочке изменения вступают в силу немедленно. Изменение действительно до выхода из оболочки. Для того чтобы сделать изменение постоянным, добавьте строку **AIXTHREAD_STK=size** в файл `/etc/environment`.
Рекомендации:	Если анализ программы показывает переполнение стека, можно увеличить размер стека по умолчанию.
Настройка:	Уменьшение размера стека по умолчанию может потребоваться для достижения ограничения на 32 000 нитей для 32-bit-разрядных приложений.

16. AIXTHREAD_AFFINITY

Элемент	Описание
Назначение:	Определяет расположение структур, стеков и локальных хранилищ нитей pthread в системах с включенной опцией enhanced affinity.
Значения:	Значение по умолчанию: existing. Диапазон: existing, always, attempt.
Просмотр:	**echo $AIXTHREAD_AFFINITY** Этот режим включается системой, поэтому начальное значение по умолчанию нельзя просмотреть с помощью команды **echo**.
Изменение:	**AIXTHREAD_AFFINITY** ={default|strict|first-touch} **export** AIXTHREAD_AFFINITY
Рекомендации:	Установка значения **strict** позволит улучшить производительность потоков, но время запуска при этом будет увеличено. При установке значения **default** будет использоваться предыдущая оптимальная реализация. При установке значения **first-touch** будет произведен поиск оптимальной настройки.
Настройка:	Если потоки будут выполняются достаточно долго, то можно улучшить производительность, указав значение **strict** . При большом количестве быстро выполняющихся потоков необходимо указать значение **default** или **first touch**.

Дополнительная информация: “Переменные среды нити” на стр. 72

17. MALLOCBUCKETS

Элемент	Описание
Назначение:	Разрешает программе выделения памяти, применяемой по умолчанию, динамически выделять области памяти с заданными параметрами. Эта опция позволяет повысить скорость работы приложений, делающих большое число запросов на выделение небольших областей памяти.
Значения:	**MALLOCTYPE=**buckets **MALLOCBUCKETS=**[[**number_of_buckets**:*n* | **bucket_sizing_factor**:*n* | **blocks_per_bucket**:*n* | **bucket_statistics**:[**stdout|stderr|файл**]],...] В следующей таблице приведены значения по умолчанию для **MALLOCBUCKETS**. **Опции MALLOCBUCKETS** Значение по умолчанию **number_of_buckets**[1] 16 **bucket_sizing_factor (32-разрядный)**[2] 32 **bucket_sizing_factor (64-разрядный)**[3] 64 **blocks_per_bucket** 1024[4] **Примечание:** 1. Допустимы значения от 1 до 128. 2. В 32-разрядной реализации значение атрибута bucket_sizing_factor должно быть кратно 8. 3. В 64-разрядной реализации значение атрибута bucket_sizing_factor должно быть равно 16. 4. По умолчанию опция **bucket_statistics** выключена.
Просмотр:	**echo $MALLOCBUCKETS**; **echo $MALLOCTYPE**
Изменение:	Воспользуйтесь командой экспорта переменных среды, применяемой в текущей оболочке.
Рекомендации:	Если запросы malloc обрабатываются медленно, и приложение делает большое число запросов на выделение небольших областей памяти, то данная опция позволяет повысить скорость работы приложения.

Элемент	Описание
Настройка:	Для того чтобы включить режим динамического выделения областей памяти с заданными параметрами, переменной среды *MALLOCTYPE* нужно присвоить значение "buckets". Переменная среды *MALLOCBUCKETS* позволяет переопределить параметры областей памяти, выделяемых динамически, однако в большинстве случаев можно применять значения по умолчанию. Атрибут **number_of_buckets:n** позволяет задать число областей памяти, доступных в куче (*n* - число областей). Значение *n* относится ко всем кучам. Атрибут **bucket_sizing_factor:n** позволяет задать коэффициент для изменения размера области памяти, где *n* - число байт. Атрибут **blocks_per_bucket:n** задает число блоков, выделяемых в качестве области памяти, где *n* - число блоков. Это значение относится ко всем динамически выделяемым областям памяти. Кроме того, значение *n* указывает, на сколько блоков нужно расширить область памяти, когда будет исчерпан запас блоков, выделенных первоначально. Если опция **bucket_statistics** включена, то после завершения процесса, применяющего подсистему malloc, эта подсистема будет создавать статистику выделения областей памяти. В частности, эта статистика будет содержать параметры областей памяти и число запросов на выделение памяти, обработанных для каждой области. Если в системе создано несколько куч, то указанное число запросов на выделение памяти будет представлять собой суммарное значение по всем кучам. Статистическая информация о выделении областей памяти будет записана в поток вывода, указанный в атрибуте **bucket_statistics**. **stdout** Стандартный вывод **stderr** Стандартный поток сообщений об ошибках **файл** Указанный пользователем файл Если было указано имя файла, то статистика будет добавлена к текущему содержимому этого файла. Не записывайте статистику в стандартный вывод в том случае, если вывод процесса с помощью конвейера перенаправлен на вход другому процессу.

Дополнительная информация: Сегменты Malloc

18. MALLOCMULTIHEAP

Элемент	Описание
Назначение:	Задает число куч, создаваемых внутри сегмента памяти процесса.
Значения:	Значение по умолчанию: 32. Диапазон: От 1 до 32.
Просмотр:	**echo $MALLOCMULTIHEAP** Этот режим включается системой, поэтому начальное значение по умолчанию нельзя просмотреть с помощью команды **echo**.
Изменение:	**MALLOCMULTIHEAP=**[[**heaps**:*n* | **considersize**],...] **export MALLOCMULTIHEAP** В данной оболочке изменения вступают в силу немедленно. Изменение действительно до выхода из оболочки. Для того чтобы сделать изменение постоянным, добавьте команду **MALLOCMULTIHEAP=**[[**heaps**:*n* | **considersize**],...] в файл `/etc/environment`.
Рекомендации:	Посмотрите, нет ли конфликтов в подсистеме динамического выделения памяти (сегмент F) и достаточно ли запущено нитей.

Элемент	Описание
Настройка:	Сокращение числа динамически выделяемых областей памяти может уменьшить объем памяти, используемый процессом. Некоторые пользовательские процессы с несколькими нитями, активно использующие подсистему malloc, будут работать быстрее, если перед их запуском экспортировать переменную среды **MALLOCMULTIHEAP**=1. В частности, это рекомендуется делать для программ C++ с несколькими нитями, так как они применяют подсистему динамического выделения памяти при вызове конструкторов и деструкторов. Повышение производительности будет наиболее заметно при запуске пользовательских приложений с несколькими нитями в системе SMP, в особенности если для нитей будет установлена глобальная область действия (соотношение M:N равно 1:1). Однако в некоторых случаях повышения производительности можно добиться и в другой среде, в частности, в однопроцессорной системе. Если задано значение **considersize**, то применяется альтернативный алгоритм динамического выделения памяти, согласно которому система выбирает любую кучу, размер которой достаточен для обработки данного запроса. При этом за счет сокращения числа вызовов **sbrk()** можно уменьшить размер рабочего набора процесса. Однако при этом немного возрастет время выполнения процесса.

Дополнительная информация: “Переменные среды нити” на стр. 72

19. NUM_RUNQ

Элемент	Описание
Назначение:	Изменяет количество очередей выполнения по умолчанию.
Значения:	По умолчанию: Число активных процессоров, определенное на момент выполнения. Диапазон: Натуральное число
Просмотр:	**echo $NUM_RUNQ** Этот режим включается системой, поэтому начальное значение по умолчанию нельзя просмотреть с помощью команды **echo**.
Изменение:	**NUM_RUNQ=n export NUM_RUNQ** В данной оболочке изменения вступают в силу немедленно. Изменение действительно до выхода из оболочки. Для того чтобы сделать изменение постоянным, добавьте команду **NUM_RUNQ** =*n* в файл `/etc/environment`.
Рекомендации:	Нет
Настройка:	Нет

Дополнительная информация: “Переменные среды нити” на стр. 72

20. NUM_SPAREVP

Элемент	Описание
Назначение:	Задает количество структур vp, выделяемых malloc за время `pth_init`.
Значения:	По умолчанию: NUM_SPARE_VP. Диапазон: Натуральное число
Просмотр:	**echo $NUM_SPAREVP** Этот режим включается системой, поэтому начальное значение по умолчанию нельзя просмотреть с помощью команды **echo**.
Изменение:	**NUM_SPAREVP=n export NUM_SPAREVP** В данной оболочке изменения вступают в силу немедленно. Изменение действительно до выхода из оболочки. Для того чтобы сделать изменение постоянным, добавьте команду **NUM_SPAREVP** =*n* в файл `/etc/environment`.
Рекомендации:	Нет
Настройка:	Нет

Дополнительная информация: “Переменные среды нити” на стр. 72

21. SPINLOOPTIME

Элемент	Описание
Назначение:	Задает число попыток захвата блокировки перед передачей управления другому процессору (только для нитей libpthread).
Значения:	По умолчанию: 1 в однопроцессорных системах и 40 в многопроцессорных системах. Диапазон: Натуральное число
Просмотр:	**echo $SPINLOOPTIME** Этот режим включается системой, поэтому начальное значение по умолчанию нельзя просмотреть с помощью команды **echo**.
Изменение:	**SPINLOOPTIME=*n*export SPINLOOPTIME** В данной оболочке изменения вступают в силу немедленно. Изменение действительно до выхода из оболочки. Для того чтобы сделать изменение постоянным, добавьте команду **SPINLOOPTIME**=*n* в файл `/etc/environment`.
Рекомендации:	Значение **SPINLOOPTIME** может быть недостаточным, если нити будут часто находиться в состоянии ожидания (т.е. если время простоя будет значительным).
Настройка:	Если в многопроцессорной системе существует конфликт взаимных блокировок нитей pthread, то вместо значения по умолчанию (40) укажите большее значение.

Дополнительная информация: “Переменные среды нити” на стр. 72

22. **STEP_TIME**

Элемент	Описание
Назначение:	Задает количество попыток создания VP за время тайм-аута активации.
Значения:	По умолчанию: DEF_STEPTIME. Диапазон: Натуральное число
Просмотр:	**echo $STEP_TIME** Этот режим включается системой, поэтому начальное значение по умолчанию нельзя просмотреть с помощью команды **echo**.
Изменение:	**STEP_TIME=n export STEP_TIME** В данной оболочке изменения вступают в силу немедленно. Изменение действительно до выхода из оболочки. Для того чтобы сделать изменение постоянным, добавьте команду **STEP_TIME** =*n* в файл `/etc/environment`.
Рекомендации:	Нет
Настройка:	Нет

Дополнительная информация: “Переменные среды нити” на стр. 72

23. **VP_STEALMAX**

Элемент	Описание
Назначение:	Задает максимальное количество используемых VP или отключает использование off.
Значения:	По умолчанию: None. Диапазон: Натуральное число
Просмотр:	**echo $VP_STEALMAX** Этот режим включается системой, поэтому начальное значение по умолчанию нельзя просмотреть с помощью команды **echo**.
Изменение:	**VP_STEALMAX=n export VP_STEALMAX** В данной оболочке изменения вступают в силу немедленно. Изменение действительно до выхода из оболочки. Для того чтобы сделать изменение постоянным, добавьте команду **VP_STEALMAX** =*n* в файл `/etc/environment`.
Рекомендации:	Нет
Настройка:	Нет

Дополнительная информация: “Переменные среды нити” на стр. 72

24. **YIELDLOOPTIME**

Элемент	Описание
Назначение:	Указывает, сколько раз требуется уступить процессор другой нити перед повторной попыткой блокировки (только для нитей libpthread). Процессор передается другой нити ядра только в том случае, если есть другая готовая к выполнению нить ядра с достаточно высоким приоритетом.
Значения:	По умолчанию: 0. Диапазон: Положительное целое число.
Просмотр:	**echo $YIELDLOOPTIME** Этот режим включается системой, поэтому начальное значение по умолчанию нельзя просмотреть с помощью команды **echo**.
Изменение:	**YIELDLOOPTIME=*n*export YIELDLOOPTIME** В данной оболочке изменения вступают в силу немедленно. Изменение действительно до выхода из оболочки. Для того чтобы сделать изменение постоянным, добавьте команду **YIELDLOOPTIME=*n*** в файл `/etc/environment`.
Рекомендации:	Значение **YIELDLOOPTIME** может быть недостаточным, если нити будут часто находиться в состоянии ожидания (т.е. если время простоя будет значительным).
Настройка:	Если вы не хотите, чтобы нити переходили в состояние ожидания во время ожидания блокировки, измените значение по умолчанию (0) на положительное число.

Дополнительная информация: “Переменные среды нити” на стр. 72

Прочие настраиваемые параметры

AIX позволяет настраивать несколько дополнительных параметров.

1. **AIX_TZCACHE**

Элемент	Описание
Назначение:	Применяется для хранения фиксированной копии переменной *TZ* во время выполнения процесса.
Значения:	По умолчанию: Нет Допустимые значения: ON (включает параметр)
Просмотр:	**$AIX_TZCACHE**
Изменение:	**export AIX_TZCACHE=ON** Изменения применяются для всех процессов, которые затем будут запускаться из этой оболочки. Указывает, что приложение должно всегда использовать начальное значение переменной *TZ*. Этот процесс повышает производительность, если приложение часто запрашивает часовой пояс. Например, если приложение часто проверяет локальное время. Однако, после запуска приложения любые изменения переменной *TZ* игнорируются.
Рекомендации:	Этот параметр не рекомендуется использовать в универсальной конфигурации системы в файле `/etc/environment`. Укажите этот параметр для приложений, которые не изменяют переменную *TZ*, но часто запрашивают часовой пояс.
Настройка:	Нет

2. **EXTSHM**

Элемент	Описание
Назначение:	Включает опцию расширенной общей памяти.
Значения:	По умолчанию: Нет Допустимые значения: ON, 1SEG, MSEG
Просмотр:	**echo $EXTSHM**
Изменение:	**export EXTSHM** В данной оболочке изменения вступают в силу немедленно. Изменение действительно до выхода из оболочки. Для того чтобы сделать это изменение постоянным, добавьте команду **EXTSHM=ON** , **EXTSHM=1SEG** или **EXTSHM=MSEG** в файл `/etc/environment`.
Рекомендации:	Нет

Элемент	Описание
Настройка:	Установка значения *ON*, *1SEG* или *MSEG* позволяет уменьшить размер выделяемых для процесса сегментов общей памяти до одного байта, округленного до размера целой страницы. Таким образом снимается ограничение в 11 пользовательских сегментов общей памяти. Для 32-разрядных процессов максимальный общий размер всех сегментов памяти составляет 2,75 ГБ. Значение переменной **EXTSHM** *ON* эквивалентно значению *1SEG*. С любым из этих значений любой фрагмент общей памяти размером меньше 256 МБ создается в системе как сегмент mmap, и тем самым влияет на производительность аналогично mmap. Общая память с размером 256 МБ и больше представляется в виде рабочего сегмента. Если **EXTSHM** задана равной *MSEG*, то вся общая память создается в системе как сегмент mmap, что позволяет улучшить работу с памятью.

Дополнительная информация: “Расширенная общая память” на стр. 161

3. **LDR_CNTRL**

Элемент	Описание
Назначение:	Разрешает настройку загрузчика ядра.
Значения:	По умолчанию: Нет Возможные значения: PREREAD_SHLIB, LOADPUBLIC, IGNOREUNLOAD, USERREGS, MAXDATA, MAXDATA32, MAXDATA64, DSA, PRIVSEG_LOADS, DATA_START_STAGGER, LARGE_PAGE_TEXT, LARGE_PAGE_DATA, HUGE_EXEC, NAMEDSHLIB, SHARED_SYMTAB и SED
Просмотр:	**echo $LDR_CNTRL**
Изменение:	**LDR_CNTRL=**{*PREREAD_SHLIB* **|** *LOADPUBLIC*| ...}**export LDR_CNTRL** В данной оболочке изменения вступают в силу сразу после выполнения команды. Изменение действительно до выхода из оболочки. Для того чтобы сделать изменение постоянным, добавьте следующую команду в файл `/etc/environment`: **LDR_CNTRL=**{*PREREAD_SHLIB* **|** *LOADPUBLIC*| ...}
Рекомендации:	Нет

Элемент	Описание
Настройка:	С помощью переменной среды **LDR_CNTRL** можно задать многие аспекты работы системного загрузчика ядра. В переменной **LDR_CNTRL** можно задать многие атрибуты. Атрибуты следует отделять друг от друга символом '@'. Например: **LDR_CNTRL=**PREREAD_SHLIB@LOADPUBLIC. Если задан атрибут PREREAD_SHLIB, то при обращении к библиотеке считывается все ее содержимое. Если в VMM настроен режим упреждающего чтения, библиотеку можно считать с диска и записать в кэш программы, обращающейся к ее страницам. Хотя в этом случае будет расходоваться больше памяти, применение такого подхода позволяет повысить скорость работы программ, обращающихся к большому числу страниц общих библиотек при условии, что применяется произвольный способ доступа к данным (например, Catia). Если задана опция LOADPUBLIC, то загрузчик ядра загружает все модули, запрошенные приложением, в глобальный сегмент памяти общей библиотеки. Если модуль нельзя загрузить в глобальный сегмент памяти общей библиотеки, предоставив доступ к этому модулю всем процессам, то сегмент загружается только для исходного приложения. Опция IGNOREUNLOAD позволяет запретить выгрузку библиотек. Благодаря этому можно предотвратить фрагментацию памяти и избавиться от лишней нагрузки, возникающей от постоянной загрузки и выгрузки библиотек. Если опция IGNOREUNLOAD не указана, может возникнуть ситуация, когда в память загружено два экземпляра модуля: один при запуске приложения, а второй по динамическому запросу. Если задана опция USERREGS, то система сохраняет содержимое пользовательских регистров общего назначения. Таким образом, эти регистры могут использоваться последующими системными вызовами, сделанными в рамках этого приложения. Эта опция может применяться в том случае, если приложение выполняет сбор мусора. Опция MAXDATA позволяет ограничить размер кучи и при необходимости переопределить значение, указанное в исполняемом файле. Значение MAXDATA задает начальное ограничение ресурсов процесса. В 32-разрядных программах, если значение MAXDATA больше нуля, то включается модель большого адресного пространства. См. Поддержка больших программ. Для выключения модели большого адресного пространства переменную MAXDATA необходимо обнулить: **LDR_CNTRL=MAXDATA=0**. В 64-разрядных программах MAXDATA задает гарантированный максимальный размер кучи данных программы. Часть адресного пространства, зарезервированная для кучи, недоступна процедурам **shmat()** и **mmap()** даже в том случае, если адрес указан явным образом. Можно указать произвольное значение, однако область данных не может выходить за пределы 0x06FFFFFFFFFFFFF вне зависимости от значения maxdata. Два дополнительных параметра существуют для оптимизации отдельно для 32-разрядных и 64-разрядных процессов. Дополнительные параметры максимального размера переопределяют опцию MAXDATA в двух соответствующих объектных моделях. Опция MAXDATA32 выполняет роль MAXDATA в 32-разрядных процессах, а в 64-разрядных игнорируется. Аналогично, опция MAXDATA64 влияет только на 64-разрядные процессы. Опция PRIVSEG_LOADS указывает системному загрузчику, что динамически загружаемые личные модули процесса нужно помещать в личный процесс сегмента. Это может повысить коэффициент готовности памяти при работе объемных приложений, выполняющих динамическую загрузку личных модулей и сталкивающихся с нехваткой памяти в куче процесса. Если в личном сегменте процесса недостаточно места, опция PRIVSEG_LOADS не играет роли. Опция PRIVSEG_LOADS действует только для 32-разрядных приложений с ненулевым значением MAXDATA. Если будет указана опция DATA_START_STAGGER=Y, сегмент данных процесса будет начинаться со смещения, зависящего от MCM и определенного опцией **data_stagger_interval** команды **vmo**. У процесса с номером n, выполняемого в определенном MCM, раздел данных начинается по адресу (n * **data_stagger_interval** * PAGESIZE) % 16 МБ. Опция DATA_START_STAGGER=Y допустима только для 64-разрядных процессов в 64-разрядном ядре. Опция LARGE_PAGE_TEXT=Y указывает, что загрузчик должен пытаться разместить текстовый сегмент процесса в больших страницах памяти. Опция LARGE_PAGE_TEXT=Y допустима только для 64-разрядных процессов в 64-разрядном ядре.

Элемент	Описание

	Если указана опция LARGE_PAGE_DATA=M, то большие страницы выделяются для сегмента данных только в пределах значения **brk**, тогда как без этой опции большие страницы выделяются для всего сегмента. Изменение значения **brk** может не сработать, если для нового значения **brk** будет недостаточно больших страниц памяти.
	
	Опция RESOLVEALL указывает, что загрузчик должен принудительно преобразовать все неопределенные символы, импортированные во время загрузки программы или динамических модулей. Преобразование символов выполняется в стандартной для AIX последовательности. Если вы укажете опцию **LDR_CNTRL**=RESOLVEALL и при этом импортированные символы будет невозможно преобразовать, то программа или динамический модуль не будут загружены.
	
	Опция HUGE_EXEC переменной среды позволяет пользователю управлять процесс расположением адресного пространства процессов сегментов, доступных только для чтения, для некоторых 32-разрядных исполняемых файлов. Дополнительная информация приведена в разделе Большие исполняемые файлы.
	
	Задание опции `NAMEDSHLIB=имя,[атрибут-1],[атрибут-2]...[атрибут-N]` запускает процесс доступа к области общей библиотеки или создания такой библиотеки с указанным именем. Для создания именованной области общей библиотеки можно использовать следующие способы:
	• Без атрибутов
	• С атрибутом **doubletext32**, который создает именованную область общей библиотеки, имеющую два сегмента, выделенных для текста общей библиотеки.
	
	Если для процесса требуется несуществующая именованная область общей библиотеки, такая область с указанным именем будет создана автоматически. Если указано недопустимое имя, параметр `NAMEDSHLIB=имя,[атрибут-1],[атрибут-2]...[атрибут-N]` проигнорируется. Допустимые имена имеют ненулевую длину и содержат только символы букв, цифр, подчеркивания и точки.
	
	Если указана опция `SHARED_SYMTAB=Y`, то для экспорта символов в 64-разрядных программах система создаст общую таблицу символов. Если параллельно выполняются несколько программ, общая таблица символов позволит уменьшить объем используемой системной памяти.
	
	Если указана опция `SHARED_SYMTAB=N`, общая таблица символов не создается. Эта опция переопределяет флаг `AOUT_SHR_SYMTAB` в необязательном заголовке XCOFF.
	
	Опция SED запрещает процессу использовать стек для выполнения кода и блокирует все опции SED, указанные в исполняемом файле. Опция может иметь одно из следующих значений:
	`SED=system`
	`SED=request`
	`SED=exempt`

4. LDR_PRELOAD LDR_PRELOAD64

Элемент	Описание
Назначение:	Включает предварительную загрузку общих библиотек. Опция LDR_PRELOAD предназначена для 32-разрядных процессоров, а LDR_PRELOAD64 - для 64-разрядных. Поиск имён функций начинается с библиотек, перечисленных в данных переменных, и, если он не дал результатов, выполняется обычный поиск. Замещение символов из предварительно загруженных библиотек работает как для компоновки AIX по умолчанию, так и для динамической компоновки. Поведение отложенной обработки имён не меняется.
Значения:	По умолчанию: Нет Допустимые значения: имена библиотек **Примечание:** несколько библиотек разделяются двоеточием (:). Библиотеки, входящие в состав архивов, указываются в скобках.
Просмотр:	**echo $LDR_PRELOAD** **echo $LDR_PRELOAD64**
Изменение:	**$LDR_PRELOAD="libx.so:liby.a(shr.o)"** Поиск имён функций выполняется сначала в `libx.so`, потом в объекте `shr.o` архива `liby.a`, а затем в библиотеках, от которых зависит процесс. Поиск динамически (с помощью dlopen() или load()) загружаемых модулей также будет выполняться сначала среди предварительно загруженных библиотек, указанных в переменной среды.
Рекомендации:	Нет

5. NODISCLAIM

Элемент	Описание
Назначение:	Задает способ обработки вызовов **free()**. Если параметр **PSALLOC** равен *early*, то все вызовы **free()** обрабатываются с помощью системных вызовов **disclaim()**. Если параметр **NODISCLAIM** равен *true*, этого не происходит.
Значения:	По умолчанию: Нет Возможное значение: True
Просмотр:	**echo $NODISCLAIM**
Изменение:	**NODISCLAIM=true export NODISCLAIM** В данной оболочке изменения вступают в силу немедленно. Изменение действительно до выхода из оболочки. Для того чтобы сделать изменение постоянным, добавьте команду **NODISCLAIM=true** в файл `/etc/environment`.
Рекомендации:	Если число системных вызовов **disclaim()** велико, то рекомендуется включить эту опцию.
Настройка:	Если для переменной **PSALLOC** установлено значение *early*, то настройка данной переменной приведет к исключению вызовов **disclaim()** из функции **free()**.

Дополнительная информация: “Статическое выделение пространства подкачки” на стр. 157

6. **NSORDER**

Элемент	Описание
Назначение:	Переопределяет порядок поиска при преобразовании имени набора.
Значения:	По умолчанию: bind, nis, local Возможные значения: bind, local, nis, bind4, bind6, local4, local6, nis4 и nis6
Просмотр:	**echo $NSORDER** Этот режим включается системой, поэтому начальное значение по умолчанию нельзя просмотреть с помощью команды **echo**.
Изменение:	**NSORDER=***значение***,***значение***,** ... **export NSORDER** В данной оболочке изменения вступают в силу немедленно. Изменение действительно до выхода из оболочки. Для того чтобы сделать это изменение постоянным, добавьте команду **NSORDER=***значение* в файл `/etc/environment`.
Рекомендации:	Нет
Настройка:	Переменная **NSORDER** имеет больший приоритет, чем файл `/etc/netsvc.conf` .

Дополнительная информация: “Настройка преобразования имен” на стр. 296

7. **PSALLOC**

Элемент	Описание
Назначение:	Устанавливает значение переменной среды **PSALLOC**, задающей стратегию выделения пространства подкачки.
Значения:	По умолчанию: Нет Возможное значение: early
Просмотр:	**echo $PSALLOC**
Изменение:	**PSALLOC=early export PSALLOC** В данной оболочке изменения вступают в силу немедленно. Изменение действительно до выхода из оболочки.
Рекомендации:	Нет
Настройка:	Малый объем пространства подкачки может привести к уничтожению процесса. Избежать этого позволяет стратегия предварительного выделения пространства подкачки для процесса. Однако при этом может возрасти объем неиспользуемого пространства подкачки. Вместе с данной переменной среды рекомендуется задать переменную среды **NODISCLAIM**.

Дополнительная информация: “Выделение и освобождение блоков пространства подкачки” на стр. 53 и “Статическое выделение пространства подкачки” на стр. 157

8. **RT_GRQ**

Элемент	Описание
Назначение:	Указывает, что нить должна быть помещена в глобальную очередь выполнения, а не в очередь выполнения конкретного CPU.
Значения:	По умолчанию: Нет Диапазон: ON, OFF
Просмотр:	**echo $RT_GRQ**
Изменение:	**RT_GRQ=***{OFF/ON***export RT_GRQ** Изменение вступает в силу немедленно и действует до следующей загрузки. Для внесения изменения на постоянной основе добавьте команду **RT_GRQ=***{ON|OFF}* в файл `/etc/environment`.
Рекомендации:	Нет
Настройка:	Эту опцию можно включить в многопроцессорной системе. Если значение параметра равно *ON*, то нить будет помещена в глобальную очередь выполнения. В этом случае планировщик будет выбирать из глобальной очереди выполнения нить с минимальным приоритетом. В итоге нити с алгоритмом планирования SCHED_OTHER, для управления которыми применяются прерывания, будут быстрее передаваться на выполнение.

Дополнительная информация: “Очередь выполнения планировщика” на стр. 43

9. **RT_MPC**

Элемент	Описание
Назначение:	Когда ядро работает в режиме реального времени (см. описание команды **bosdebug**), то другому процессору можно отправить прерывание MPC, для того чтобы немедленно запустить нить с более высоким приоритетом.
Значения:	По умолчанию: Нет Диапазон: ON
Просмотр:	**echo $RT_MPC**
Изменение:	**RT_MPC=ON export RT_MPC** Изменение вступает в силу немедленно и действует до следующей загрузки. Для того чтобы сделать изменение постоянным, добавьте строку **RT_MPC=ON** в файл `/etc/environment`.
Рекомендации:	Нет

10. **TZ**

Элемент	Описание
Назначение:	Задает часовой пояс.
Значения:	По умолчанию: часовой пояс (из базы данных Олсона) Возможные значения: часовой пояс по Олсону или POSIX
Просмотр:	**echo $TZ**
Изменение:	**TZ = значение-TZ** В оболочке изменения вступают в силу немедленно. Изменения действуют до выхода из оболочки. Для того чтобы сделать изменение постоянным, добавьте команду **TZ=значение** в файле `/etc/environment`.
Рекомендации:	Нет
Настройка:	Формат POSIX следует использовать в приложениях, в которых более критично быстродействие, а не точность значения часового пояса

11. **VMM_CNTRL**

Элемент	Описание
Назначение:	Разрешает настройку администратора виртуальной памяти.
Значения:	По умолчанию: Нет Возможные значения: vmm_fork_policy, ESID_ALLOCATOR, SHM_1TB_SHARED, SHM_1TB_UNSHARED
Просмотр:	**echo $ VMM_CNTRL**
Изменение:	**VMM_CNTRL={vmm_fork_policy=... | ESID_ALLOCATOR=... | ...}export VMM_CNTRL** В данной оболочке изменения вступают в силу немедленно. Изменение действительно до выхода из оболочки. Для того чтобы сделать изменение постоянным, добавьте переменную среды `VMM_CNTRL=` в файл `/etc/environment`.
Рекомендации:	Нет

Элемент	Описание
Настройка:	С помощью переменной среды VMM_CNTRL можно управлять администратором виртуальной памяти. С помощью переменной среды VMM_CNTRL можно указать несколько опций; в качестве разделителя применяется символ '@'. Пример указания нескольких опций: `VMM_CNTRL=vmm_fork_policy=COW@SHM_1TB_SHARED=5` При указании опции vmm_fork_policy=COW каждый раз при порождении процесса vmm использует стратегию дерева порождения с копированием по команде. Это поведение по умолчанию. Для того чтобы запретить vmm применение стратегии записи по команде, укажите опцию vmm_fork_policy=COR. Если указан параметр vmm_fork_policy, то глобальный параметр vmm_fork_policy игнорируется. Если опция ESID_ALLOCATOR указана, то она управляет распределителем из присвоений shmat и mmap без направления. Дополнительная информация приведена в разделе “Использование псевдонимов для сегмента 1 ТБ” на стр. 162. Параметры SHM_1TB_SHARED и SHM_1TB_UNSHARED управляют использованием областями общей памяти объемом 1 ТБ. Дополнительная информация приведена в разделе “Использование псевдонимов для сегмента 1 ТБ” на стр. 162.

12. AIX_STDBUFSZ

Элемент	Описание
Назначение:	Настраивает размер буфера ввода-вывода для вызовов чтения и записи, создаваемых командами **cp**,**mv**,**cat**, **cpio**. Также применимо для потоковой буферизации.
Значения:	По умолчанию: Нет. Возможные значения: целое число, указывающее размер буфера в байтах, килобайтах или мегабайтах.
Просмотр:	**echo** $ **AIX_STDBUFSZ**
Изменение:	**AIX_STDBUFSZ=1024; export AIX_STDBUFSZ** (Для настройки размера буфера 1024) В данной оболочке изменения вступают в силу немедленно. Изменение действительно до выхода из оболочки. Для того чтобы сделать изменение размера буфера постоянным, добавьте переменную среды AIX_STDBUFSZ в файл /etc/environment.
Рекомендации:	Нет
Настройка:	Укажите значение следующими способами. • Укажите целое значение в формате export AIX_STDBUFSZ=1024 • Укажите шестнадцатеричное значение в формате export AIX_STDBUFSZ=0x400 • Ограничения: минимальное значение - 64 байт, максимальное значение - 127 МБ. • Недопустимые целые числа за пределами этих ограничений заменяются на ближайшее допустимое значение. • Если значение не кратно двум, то оно округляется до ближайшего четного числа в меньшую сторону. • Если значение параметра AIX_STDBUFSZ недопустимо, то оно игнорируется.

13. AIX_LDSYM

Элемент	Описание
Назначение:	Информация об исходной строке в файле Lightweight_core не отображается по умолчанию, если размер страницы текста равен 64 КБ. Если размер страницы текста равен 64 КБ, то информацию об исходной строке в файле Lightweight_core можно получить с помощью переменной среды AIX_LDSYM=ON.
Значения:	По умолчанию: Нет. Допустимые значения: ON.
Просмотр:	**echo** $ **AIX_LDSYM**
Изменение:	**export AIX_LDSYM=ON** В данной оболочке изменения вступают в силу немедленно. Изменение действительно до выхода из оболочки. Для того чтобы сделать изменение системы постоянным, добавьте переменную среды AIX_LDSYM=ON в файл /etc/environment.
Рекомендации:	Нет
Настройка:	Этот параметр можно использовать для приложений с размером страницы текста 64 КБ с целью получения информации об исходной строке в файле Lighweight_core.

Большие 32-разрядные исполняемые файлы

Для большинства 32-разрядных исполняемые файлы, имеющих секции загрузчика и текста в пределах первых 256 МБ файла, AIX резервирует сегмент 0x1 адресной области процесса для информации исполняемого файла, доступной только для чтения.

Однако, для 32-разрядных больших исполняемых файлов, в которых суммарный размер разделов текста и загрузчика больше, чем размер сегмента, требуются последовательные сегменты, доступные только для чтения.

Расположение адресного пространства процессов больших исполняемых сегментов, доступных только для чтения.:

Опция **HUGE_EXEC** переменной среды **LDR_CNTRL** позволяет пользователю управлять процесс расположением адресного пространства процессов сегментов, доступных только для чтения.

Данная опция используется следующим образом:

```
LDR_CNTRL=[...@]HUGE_EXEC={<segno>|0}[,<attribute>][@...]
```

, где *segno* - это номер запрошенного начального сегмента в 32-разрядном адресном пространстве или 0.

Если будет указано значение segno, отличное от 0, загрузчик системы попытается вставить большие исполняемые сегменты, доступные только для чтения, в адресное пространство процесса в месте, соответствующем номеру запрошенного начального сегмента.

Если будет указано значение segno , равное 0, загрузчик системы попытается вставить большие исполняемые сегменты, доступные только для чтения, в адресное пространство процесса в месте, соответствующем номеру запрошенного начального сегмента.

Если указано значение segno, равное 0 (или, при отсутствии опции **HUGE_EXEC** в **LDR_CNTRL**), загрузчик системы выбирает номер начального сегмента на основании модели адресного пространства. Алгоритм, используемый для этого выбора, подобен флагу **MAP_VARIABLE** функции **mmap**:

- Если процесс не требует ни динамического распределения сегментов (DSA), ни больших страниц, система задает сегменты, следующие за кучей процесса.
- В противном случае система задает последующие сегменты сразу же под нижним сегментом общей библиотеки (если она присутствует).

Номер начального сегмента не должен конфликтовать с сегментами, зарезервированными моделью адресного пространства запрошенного процесса. Чтобы определить, есть ли такой конфликт, необходимо учесть сегменты кучи процесса и области общей библиотеки. В случаях, когда сегменты кучи процесса размещаются динамически (DSA или большие страницы), зарезервированным считается только начальный сегмент кучи. Если есть конфликт номера выбранного начального сегмента и зарезервированных сегментов, произойдет ошибка исполнения **ENOMEM**.

Доступность сегмента 0x1 для текста общей библиотеки:

Большой исполняемый файл состоит из нескольких последовательных сегментов, защищенных от записи, и не может располагаться в сегменте 0x1. Т.к. большой исполняемый файл не может использовать сегмент 0x1, он может использоваться для других целей.

Дополнительный атрибут для опции контроля загрузчика **HUGE_EXEC** позволяет разместить сегмент текста общей библиотеки в сегменте 0x1, а не в 0xD:

```
HUGE_EXEC={<segno>|0},shtext_in_one
```

Т.к. пред-перемещенные данные области общей библиотеки используется только при размещении сегмента общего текста в 0xD, процессы, требующие эту опцию, не имеют преимуществ пред-перемещенных данных

библиотеки. Следовательно, все данные общей библиотеки располагаются в куче процесса. Это позволяет освободить все сегменты 0x3–0xF для разделенного использования кучей процесса (mmap/shmat) и исполняемым файлом.

Примечание: Атрибут **shtext_in_one**, используемый вместе с **maxdata** и параметрами DSA, которые обычно не дают процессу использовать область общей библиотеки (например, `maxdata>0xA0000000/dsa` или `maxdata=0/dsa`), позволяет процессу воспользоваться преимуществами быстродействия, предоставляемыми текстом общей библиотеки.

Если область общая библиотека процесса - это именованная область, созданная с атрибутом **doubletext32**, то сегмента пред-перемещенных данных нет, и необходимо использовать оба сегмента текста общей библиотеки. В этом случае первый сегмент (обычно расположенный в сегменте 0xD) переносится в сегмент 0x1, а второй сегмент текста общей библиотеки остается в сегменте 0xF. Это максимизирует число последовательных сегментов (0x3–0xE), которое может быть разделено для использования кучей процесса (mmap/shmat) и исполняемым файлом.

Небольшие исполняемые файлы со значением **maxdata** больше, чем 0xA0000000 и включенным DSA не могут использовать зоны общей библиотеки. Большие исполняемые файлы, которые используют область именованной общей библиотеки, созданную с атрибутом **doubletext32**, и имеют атрибут **shtext_in_one**, могут иметь значение **maxdata** вплоть до 0xC0000000 при использовании области.

Примеры больших исполняемых файлов:

Пример сценариев использования больших исполняемых файлов.

Пример модели адресного пространства большой программы

Если предпочитаемая модель адресного пространства имеет следующий вид:

- `MAXDATA=0x50000000` (non-DSA)
- в сегментах 0xB, 0xC and 0xE необходимы области shmat/mmap
- Текст и пред-перемещенные данные доступа области библиотеки

Формат адресного пространства:

```
0x0: системный сегмент
0x1:
0x2: частные зависимости времени исполнения / стек
0x3: куча процесса
0x4: куча процесса
0x5: куча процесса
0x6: куча процесса
0x7: куча процесса
0x8:
0x9:
0xA:
0xB: shmat/mmap (размещено после exec)
0xC: shmat/mmap (размещено после exec)
0xD: текст общей библиотеки
0xE: shmat/mmap (размещено после exec)
0xF: пред-перемещенные данные общей библиотеки
```

Из данного примера можно видеть, что сегменты 0x8–0xA доступны для исполняемого файла.

Приняв, что размер исполняемый файл больше, чем 256 MB и меньше, чем 512 MB, наилучшие установки **HUGE_EXEC** в данной ситуации приведены ниже:

1. `HUGE_EXEC=0`
2. `HUGE_EXEC=0x8`
3. `HUGE_EXEC=0x9`

Опции 1 и 2 поместят исполняемый файл в сегменты 0x8–0x9, а опция 3 - в сегмент 0x9–0xA.

Пример модели адресного пространства очень большой программы

Если предпочитаемая модель адресного пространства имеет следующий вид:

- MAXDATA=0x50000000 DSA
- в сегментах 0xB, 0xC and 0xE необходимы области shmat/mmap
- Текст и пред-перемещенные данные доступа области библиотеки

Формат адресного пространства:

```
0x0: системный сегмент
0x1:
0x2: частные зависимости времени исполнения / стек
0x3: куча процесса
0x4:      |
0x5:      |
0x6:      v
0x7: _____________ (лимит данных)
0x8:
0x9:
0xA:
0xB: shmat/mmap (размещено после exec)
0xC: shmat/mmap (размещено после exec)
0xD: текст общей библиотеки
0xE: shmat/mmap (размещено после exec)
0xF: пред-перемещенные данные общей библиотеки
```

Из данного примера можно видеть, что сегменты 0x4–0xA доступны для исполняемого файла.

Приняв, что размер исполняемый файл больше, чем 256 MB и меньше, чем 512 MB, наилучшие установки **HUGE_EXEC** в данной ситуации приведены ниже:

1. HUGE_EXEC=0x8
2. HUGE_EXEC=0x9

Опции 1 поместит исполняемый файл в сегменты 0x8–0x9, а опция 2 - в сегмент 0x9–0xA.

Примечание: Параметр HUGE_EXEC=0 не будет подходящим для данного клиента, т.к. система выберет сегменты 0xB–0xC для исполняемого файла (из-за динамического распределения сегментов (DSA). Это сделает эти сегменты недоступными для shmat/mmap после exec. Задание для **HUGE_EXEC** любого из значений сегментов 0x4, 0x5, 0x6 или 0x7 при разрешенной запрошенной вставке вызовет ограничение роста кучи процесса сегментом под запрошенным стартовым сегментом.

Пример модели адресного пространства очень большой программы без доступа к общей библиотеки

Если предпочитаемая модель адресного пространства имеет следующий вид:

- MAXDATA=0xB0000000 DSA
- Нет областей shmat/mmap
- Нет доступа к общей библиотеке

Формат адресного пространства:

```
0x0: системный сегмент
0x1:
0x2: частные зависимости времени исполнения / стек
0x3: куча процесса
0x4:      |
0x5:      |
0x6:      |
0x7:      |
```

```
0x8:      |
0x9:      |
0xA:      |
0xB:      |
0xC:      v
0xD: _____________ (лимит данных)
0xE:
0xF:
```

Из данного примера можно видеть, что сегменты 0x4–0xF доступны для исполняемого файла.

Приняв, что размер исполняемый файл больше, чем 256 MB и меньше, чем 512 MB, наилучшие установки **HUGE_EXEC** в данной ситуации приведены ниже:

1. `HUGE_EXEC=0`
2. `HUGE_EXEC=0xE`

Обе опции поместит исполняемый файл в сегменты 0xE–0xF.

Примечание: Задание для HUGE_EXEC любого из значений сегментов 0x4-0xD при разрешенной запрошенной вставке вызовет ограничение роста кучи процесса сегментом под запрошенным стартовым сегментом.

Пример модели адресного пространства процесса по умолчанию

Если предпочитаемая модель адресного пространства имеет следующий вид:

- `MAXDATA=0` (не DSA)
- Нет областей shmat/mmap
- Текст и пред-перемещенные данные доступа области библиотеки

Формат адресного пространства:

```
0x0: системный сегмент
0x1:
0x2: частные зависимости времени исполнения / куча процесса / стек
0x3:
0x4:
0x5:
0x6:
0x7:
0x8:
0x9:
0xA:
0xB:
0xC:
0xD: текст общей библиотеки
0xE:
0xF: пред-перемещенные данные общей библиотеки
```

Из данного примера можно видеть, что сегменты 0x3–0xC доступны для исполняемого файла.

Приняв, что размер исполняемый файл больше, чем 256 MB и меньше, чем 512 MB, наилучшие установки **HUGE_EXEC** в данной ситуации приведены ниже:

```
 1. HUGE_EXEC=0
 2. HUGE_EXEC=0x3
 ...
10. HUGE_EXEC=0xB
```

Опции 1 и 2 имеют одинаковые результаты – вставка исполняемого файла в сегмент 0x3–0x4.

Пример сегмента текста shtext_in_one с одной областью общей библиотеки

Если предпочитаемая модель адресного пространства имеет следующий вид:

- `MAXDATA=0x70000000` (non-DSA)
- в сегментах 0xC, 0xD, 0xE и 0xF необходимы области shmat/mmap
- Доступ к общей библиотеке

Формат адресного пространства:

```
0x0: системный сегмент
0x1: текст общей библиотеки
0x2: частные зависимости времени исполнения / стек
0x3: куча процесса
0x4: куча процесса
0x5: куча процесса
0x6: куча процесса
0x7: куча процесса
0x8: куча процесса
0x9: куча процесса
0xA:
0xB:
0xC: shmat/mmap (размещено после exec)
0xD: shmat/mmap (размещено после exec)
0xE: shmat/mmap (размещено после exec)
0xF: shmat/mmap (размещено после exec)
```

Из данного примера можно видеть, что сегменты 0xA–0xB доступны для исполняемого файла.

Приняв, что размер исполняемый файл больше, чем 256 MB и меньше, чем 512 MB, наилучшие установки **HUGE_EXEC** в данной ситуации приведены ниже:

1. `HUGE_EXEC=0,shtext_in_one`
2. `HUGE_EXEC=0xA,shtext_in_one`

Обе опции поместит исполняемый файл в сегменты 0xA–0xB, а общую библиотеку в сегмент 0x1.

Примечание: Задание для HUGE_EXEC любого из значений сегментов 0xB–0xE при разрешенной запрошенной вставке вызовет недоступность некоторых сегментов 0xC–0xF для shmat/mmap после исполняемого файла.

Пример сегментов текста shtext_in_one с двумя областями общей библиотеки

Если предпочитаемая модель адресного пространства имеет следующий вид:

- `MAXDATA=0x70000000 DSA`
- в сегментах 0xA и 0xB необходимы области shmat/mmap
- Доступность текста области общей библиотеки (созданная с помощью атрибута **doubletext32**)

Формат адресного пространства:

```
0x0: системный сегмент
0x1: текст общей библиотеки (первичный)
0x2: частные зависимости времени исполнения / стек
0x3: куча процесса
0x4:      |
0x5:      |
0x6:      |
0x7:      |
0x8:      v
0x9: ___________ (лимит данных)
0xA: shmat/mmap (размещено после exec)
0xB: shmat/mmap (размещено после exec)
```

```
0xC:
0xD:
0xE:
0xF: текст общей библиотеки (вторичный)
```

Из данного примера можно видеть, что сегменты 0xC–0xE доступны для исполняемого файла.

Приняв, что размер исполняемый файл больше, чем 512 MB и меньше, чем 768 MB, наилучшие установки **HUGE_EXEC** в данной ситуации приведены ниже:

1. `HUGE_EXEC=0,shtext_in_one`
2. `HUGE_EXEC=0xC,shtext_in_one`

Обе опции поместит исполняемый файл в сегменты 0xC–0xE, а общую библиотеку в сегменты 0x1 и 0xF.

Примечание: Задание для HUGE_EXEC любого из значений сегментов 0x4–0x7 при разрешенной запрошенной вставке вызовет ограничение роста кучи процесса сегментом под запрошенным стартовым сегментом.

Переменная среды ASO

Настройками активного системного оптимизатора (ASO) можно управлять с помощью команды **asoo**. Настройки ASO для отдельных процессов можно изменять с помощью переменных ASO.

ASO_ENABLED

Элемент	Описание
Назначение:	Явно включает или исключает процесс из оптимизации ASO.
Значения:	По умолчанию: ASO оптимизирует процесс, если он удовлетворяет критериям оптимизации ASO. Допустимые значения: ALWAYS и NEVER ALWAYS - ASO присваивает приоритет процессу. NEVER - ASO не оптимизирует процесс.
Просмотр:	**echo $ASO_ENABLED** Значение присваивается системой, поэтому начальное значение по умолчанию нельзя просмотреть с помощью команды **echo**.
Изменение:	**ASO_ENABLED=[ALWAYS|NEVER] export ASO_ENABLED** Изменение имеет силу только для процессов, которые были запущены после изменения. Изменение действует до выхода из оболочки. Для того чтобы сделать изменение постоянным, добавьте строку **ASO_ENABLED=[ALWAYS|NEVER]** в файл `/etc/environment`.
Рекомендации:	нет
Настройка:	нет

Настраиваемые параметры ядра

Параметры, которые можно настроить для ядра AIX, подразделяются на шесть групп: планировщик и управление загрузкой памяти, VMM, синхронный ввод-вывод, асинхронный ввод-вывод, диск и дисковые адаптеры, взаимодействие между процессами.

Изменения

В AIX добавлен новый гибкий и централизованный способ задания большинства параметров настройки ядра AIX.

Теперь для сохранения изменений не требуется редактировать файлы **rc**. Все настраиваемые параметры, применяемые при загрузке системы, теперь помещаются в файл `/etc/tunables/nextboot`. При перезагрузке системы эти параметры автоматически применяются.

Файл настройки `/etc/tunables/lastboot` создается автоматически после загрузки и содержит список используемых значений параметров. Это позволяет в любой момент вернуться к этим значениям. В файл протокола `/etc/tunables/lastboot.log` заносятся сообщения об изменениях, внесенных в процессе перезагрузки, а также о возникших при этом ошибках.

Существуют следующие команды для изменения настроечных файлов:

Команда	Назначение
tunsave	Сохраняет значение в файле настройки
tunchange	Обновляет значения в файле stanza
tunrestore	Применяет значения параметров, указанные в файле
tuncheck	Проверяет файлы, созданные вручную
tundefault	Восстанавливает значения по умолчанию для настраиваемых параметров

Все перечисленные команды действуют как на текущие значения, так и на значения, запланированные для следующей загрузки. Более подробная информация приведена в документации по этим командам.

Дополнительная информация о настройке параметров ядра приведена в соответствующем разделе книги *Performance Tools Guide and Reference*.

Замена команд vmtune и schedtune

Команды **vmtune** и **schedtune** заменены новыми командами: **vmo**, **ioo** и **schedo**. Команды **vmo** и **ioo** заменяют команду **vmtune**, а команда **schedo** заменяет команду **schedtune**. Новые команды поддерживают все действия, которые можно было выполнить с помощью прежних команд.

Команда **ioo** управляет всеми параметрами настройки, связанными с операциями ввода-вывода, а команда **vmo** управляет остальными параметрами Администратора виртуальной памяти (VMM), для изменения которых ранее применялась команда **vmtune**. Все три команды входят в состав нового набора файлов `bos.perf.tune`, включающего также команды **tunsave**, **tunrestore**, **tuncheck** и **tundefault**. Набор файлов `bos.adt.samples` для совместимости все еще включает команды **vmtune** и **schedtune**, представляющие собой сценарии оболочки, вызывающие команды **vmo**, **ioo** и **schedo**. Указанные сценарии поддерживают изменение только тех параметров, которые можно изменять интерактивно. Параметры, требующие выполнения **bosboot** и последующей перезагрузки системы, сценарием **vmtune** больше не поддерживаются. Для изменения таких параметров следует пользоваться командой **vmo -r**. Ниже перечислены соответствующие опции и параметры команды **vmtune**:

Предыдущая опция vmtune	Формат	Новая команда
-C 0|1	разметка страниц	**vmo -r -o pagecoloring=0|1**
-g *n1* -L *n2*	размер большой страницы число резервируемых больших страниц	**vmo -r -o lgpg_size=n1 -o lgpg_regions=n2**
-v *n*	число кадров в пуле памяти	**vmo -r -o framesets=n**
-i *n*	интервал идентификаторов специальных сегментов данных	**vmo -r -o spec_dataseg_int=n**
-V *n*	число резервируемых идентификаторов специальных сегментов данных	**vmo -r -o num_spec_dataseg=n**
-y 0|1	сродство памяти p690	**vmo -r -o memory_affinity=0|1**

В комплект поставки AIX не входят сценарии обеспечения совместимости **vmtune** и **schedtune**. Для применения существующих значений параметров к новым командам обратитесь к следующим таблицам:

Опция schedtune	Аналог schedo	Функция
-a число	-o affinity_lim=число	Задает число контекстных переключателей, по достижении которого обработка нити стратегией SCHED_FIF02 прекращается.
-b число	-o idle_migration_barrier=число	Устанавливает пороговое значение простоя при переносе данных.
-c число	-o %usDelta=число	Задает смещение системного времени.
-d число	-o sched_D=число	Устанавливает коэффициент для ограничения загрузки CPU.
-e число	-o v_exempt_seconds=число	Задает временной интервал, по истечении которого можно возобновить выполнение приостановленного процесса.
-f число	-o pacefork=число	Задает задержку в тактах перед повторным выполнением вызова **fork** после сбоя.
-F число	-o fixed_pri_global=число	Позволяет задавать фиксированные приоритеты нитей в глобальной очереди выполнения.
-h число	-o v_repage_hi=число	Позволяет задать системный критерий для начальной и конечной границ приостановки процессов.
-m число	-o v_min_process=число	Позволяет задавать минимальный уровень мультипрограммирования.
-p число	-o v_repage_proc=число	Позволяет изменять критерии приостановки для отдельных процессов.
-r число	-o sched_R=число	Устанавливает скорость накопления нагрузки CPU.
-s число	-o maxspin=число	Задает количество прокруток для блокировки перед выключением.
-t число	-o timeslice=число	Задает квантов времени по 10 мс.
-w число	-o v_sec_wait=число	Время ожидания в секундах после окончания перегрузки перед возобновлением обработки процессов.

Опция vmtune	Аналог vmo	Аналог ioo	Функция
-b число		-o numfsbuf=число	Задает число структур bufstruct файловой системы.
-B число		-o hd_pbuf_cnt=число	Этот параметр заменен на *pv_min_pbuf*.
-c число		-o numclust=число	Задает число кластеров размером 16 КБ, обрабатываемых операцией отложенной записи.
-C 0|1	-r -o pagecoloring=0|1		Позволяет включать и выключать разметку страниц для определенных аппаратных платформ.
-d 0|1	-o deffps=0|1		Позволяет включать и выключать отложенное выделение пространства подкачки.
-e 0|1		-o jfs_clread_enabled=0|1	Управляет применением в JFS кластеризованного чтения файлов.
-E 0|1		-o jfs_use_read_lock=0|1	Управляет применением в JFS общей блокировки при чтении файлов.
-f число	-o minfree=число		Задает число кадров в списке свободных записей.
-F число	-o maxfree=число		Задает число кадров в списке свободных записей, по достижении которого необходимо завершить перераспределение кадров.
-g число	-o lgpg_size число		Задает размер в байтах больших страниц, поддерживаемых на аппаратном уровне
-H число		-o pgahd_scale_thresh=число	Задает число свободных страниц в пуле памяти, на которые система распространяет упреждающее чтение.
-i число	-r -o spec_dataseg_int=число		Задает временной интервал для резервирования идентификаторов специальных сегментов данных.

Опция vmtune	Аналог vmo	Аналог ioo	Функция
-j число		-o j2_nPagesPerWriteBehindCluster= число	Задает число страниц для кластера отложенной записи.
-J число		-o j2_maxRandomWrite=число	Задает пороговое значение для случайных операций записи.
-k число	-o npskill=число		Задает число страниц пространства подкачки, по достижении которого будут выполняться операции удаления процессов.
-l число	-o lrubucket=число		Задает размер наиболее давно использовавшегося сегмента заменителя страницы.
-L число	-o lgpg_regions=число		Задает число резервируемых больших страниц.
-M число	-o maxpin=число		Задает максимальную долю физической памяти, которая может быть закреплена (в процентах).
-n число	-o nokilluid=число		Задает диапазон идентификаторов процессов **uid**, которые не следует убивать при малом объеме пространства подкачки.
-N число		-o pd_npages=число	Задает число страниц, которые следует одновременно удалять из оперативной памяти при удалении файла.
-p число	-o minperm%=число		Устанавливает пороговое значение, по достижении которого страницы данных файла не будут обрабатываться повторно.
-P число	-o maxperm%=число		Устанавливает пороговое значение, по достижении которого алгоритм перераспределения страниц будет обрабатывать только страницы данных файла.
-q число		-o j2_minPageReadAhead=число	Задает минимальное число страниц смещения для упреждающего чтения.
-Q число		-o j2_maxPageReadAhead=число	Задает максимальное число страниц смещения для упреждающего чтения.
-r число		-o minpgahead=число	Задает начальное число страниц для последовательного упреждающего чтения.
-R число		-o maxpgahead=число	Задает максимальное число страниц для упреждающего чтения.
-s 0|1		-o sync_release_ilock=0|1	Позволяет включить или выключить код, минимизирующий время блокировки inode при выполнении команды **sync**.
-S 0|1	-o v_pinshm=0|1		Позволяет включить и выключить флаг **SHM_PIN** системного вызова **shmget**.
-t число	-o maxclient%=число		Устанавливает пороговое значение, по достижении которого алгоритм перераспределения страниц будет обрабатывать только страницы данных клиентского файла.
-T число	-o pta_balance_threshold= число		Задает пороговое значение для выделения нового сегмента PTA.
-u число	-o lvm_bufcnt=число		Задает число буферов LVM для прямых операций ввода-вывода из физической памяти.
-v число	-r -o framesets=число		Задает число наборов фреймов для пула памяти.
-V число	-r -o num_spec_dataseg= число		Задает число идентификаторов специальных сегментов данных, которые необходимо зарезервировать

Опция vmtune	Аналог vmo	Аналог ioo	Функция
-w число	-o npswarn=число		Задает число свободных страниц пространства подкачки, по достижении которого процессам будет передаваться сигнал SIGDANGER.
-W число		-o maxrandwrt=число	Задает максимальный объем накопленных операций случайной записи в память, по достижении которого выполняется синхронизация страниц на дисковом накопителе с помощью алгоритма отложенной записи.
-y 0|1	-r -o memory_affinity=0|1		Параметр не существует. Привязка памяти включена, если это поддерживается аппаратным обеспечением.
-z число		-o j2_nRandomCluster=число	Задает пороговое расстояние для случайных операций записи.
-Z число		-o j2_nBufferPerPagerDevice= число	Задает число буферов для устройства пространства подкачки.

Расширения команд no и nfso

Команды **no** и **nfso** были модернизированы и теперь поддерживают сохранение изменений настраиваемых параметров в файле `/etc/tunables/nextboot`. Кроме того, обе команды поддерживают новый флаг **-h** , позволяющий просмотреть справку по любому параметру.

Справка включает в себя следующую информацию:

- Назначение параметра
- Возможные значения, такие как значение по умолчанию, допустимый диапазон и тип
- Информация о диагностике и настройке, позволяющая определить необходимость изменения значения параметра

Все такие команды настройки, как **ioo**, **nfso**, **no**, **vmo**, **raso** и **schedo**, поддерживают общий синтаксис. Более подробная информация и полный список поддерживаемых параметров настройки приведены в описании команд.

Режим совместимости AIX

При переходе в режим совместимости применяются только команды **no** и **nfso**, поскольку команды **vmtune** и **schedtune** более не поддерживаются. Режим совместимости можно применять при переходе к новой среде параметров настройки; тем не менее, не рекомендуется использовать его в выпусках AIX.

В режиме совместимости остается возможность внесения постоянных изменений путем добавления команд настройки в сценарии загрузки. Единственное существенное отличие заключается в файлах `/etc/tunables/lastboot` и `/etc/tunables/lastboot.log`, создаваемых при загрузке. Файл `lastboot.log` содержит только предупреждение о том, что AIX запускается в режиме совместимости, и содержимое файла `nextboot` не применяется.

Кроме параметров типа *Bosboot* (см. раздел “Замена команд vmtune и schedtune” на стр. 442) не будет действовать ни одна из новых опций следующей загрузки и сохранения изменений (флаги **-r** и **-p** соответственно), поскольку содержимое изменяемого файла не применяется при перезагрузке. В результате команды настройки не управляют значениями параметров после перезагрузки, как они это делают в обычном режиме. Параметры типа *Bosboot* при обновлении версии сохраняются в файле `/etc/tunables/nextboot` и их можно изменить с помощью опции **-r** независимо от того, применяется режим совместимости или нет. Файл `/etc/tunables/nextboot` удалять не следует.

Режимом совместимости управляет новый атрибут **sys0** с именем **pre520tune**, для которого при обновлении версии автоматически указывается значение `enable`. В этом режиме вызовы команд настройки в сценариях загрузки переопределяются содержимым файла `nextboot`. Текущее значение атрибута **pre520tune** можно просмотреть следующей командой:

```
# lsattr -E -l sys0
```

или изменить с помощью следующей команды:

```
# chdev -l sys0 -a pre520tune=disable
```

Если режим совместимости не применяется, то следующие параметры команды **no** типа *Reboot* можно изменить только в том случае, если в процессе перезагрузки указан флаг **-r**:

- arptab_bsiz
- arptab_nb
- extendednetstats
- ifsize
- inet_stack_size
- ipqmaxlen
- nstrpush
- pseintrstack

Для того чтобы переключиться из режима совместимости в обычный режим с сохранением текущих параметров загрузки, выполните следующую команду, предварительно изменив значение атрибута **pre520tune**:

```
# tunrestore -r -f lastboot
```

В результате содержимое файла `lastboot` будет скопировано в файл `nextboot`. Дополнительная информация о новом режиме настройки приведена в разделе, посвященном настройке ядра *Performance Tools Guide and Reference*.

Процедуры восстановления системы AIX

Если система после перезагрузки работает нестабильно, а в параметре **pre520tune** указано значение `enable`, необходимо удалить из сценариев загрузки команды настройки, приводящие к ошибкам.

Параметры, выбранные при перезагрузке, приведены в файле `/etc/tunables/lastboot` и не отмечены комментарием # DEFAULT VALUE. Дополнительная информация о файлах настройки приведена в разделе tunables File Format книги *Справочник по файлам* .

Для того чтобы восстановить значения по умолчанию для всех параметров настройки, выполните следующие действия:

1. Удалите файл `/etc/tunables/nextboot`.
2. Присвойте атрибуту **pre520tune** значение `disable`.
3. Выполните команду **bosboot**.
4. Перезагрузите систему.

Параметры управления нагрузкой на планировщик и память

В AIX доступно несколько настраиваемых параметров, относящихся к управлению нагрузкой на планировщик и память.

Большая часть параметров нагрузки на планировщик и память полностью описаны в справке по команде **schedo**. Остальные связанные параметры перечислены ниже:

1. Тонкая настройка параметра **maxuproc**:

Элемент	Описание
Назначение:	Задает максимальное число процессов, которые могут быть запущены от имени одного ИД пользователя.
Значения:	По умолчанию: 40; диапазон: от 1 до 131072
Просмотр:	**lsattr -E -l sys0 -a maxuproc**
Изменение:	**chdev -l sys0 -a maxuproc=***новое-значение*Изменение вступает в силу немедленно и сохраняется после перезагрузки. Если значение уменьшено, то оно вступит в силу после перезагрузки системы.
Рекомендации:	Это значение ограничивает число процессов, которые может запустить пользователь.
Настройка:	Данный параметр позволяет ограничить число процессов, запускаемых пользователями.

2. Тонкая настройка параметра **ncargs**:

Элемент	Описание
Назначение:	Задает максимальный размер списка ARG/ENV (в блоках по 4 КБ). Это значение применяется при выполнении функций **exec()**.
Значения:	По умолчанию: 256; диапазон: от 256 до 1024
Просмотр:	**lsattr -E -l sys0 -a ncargs**
Изменение:	**chdev -l sys0 -a ncargs=***новое-значение*Изменение вступает в силу немедленно и сохраняется после перезагрузки.
Рекомендации:	Пользователи не могут запускать дополнительные процессы, так как системному вызову **exec()** передается слишком длинный список аргументов. Чрезмерно малое значение по умолчанию может привести к тому, что некоторые программы будут завершаться с ошибкой `слишком длинный список аргументов` - в этом случае можно попробовать увеличить значение **ncargs** с помощью команды **chdev** и запустить программу еще раз.
Настройка:	С помощью этой переменной можно предотвратить сбой функции **exec()**, вызванный тем, что был передан слишком длинный список аргументов. Обратите внимание, что увеличение значения **ncargs** повышает требования к объему памяти системы.

Переменные параметры Администратора виртуальной памяти (VMM)

Команда **vmo** управляет переменными параметрами Администратора виртуальной памяти.

Дополнительная информация приведена в разделе Команда **vmo**.

Синхронный ввод-вывод, настраиваемые параметры

Для синхронного ввода-вывода имеется несколько настраиваемых параметров.

Большая часть параметров синхронного ввода-вывода полностью описаны в справке по команде **ioo**. Остальные связанные параметры перечислены ниже:

1. **maxbuf**

Элемент	Описание
Назначение:	Число страниц размером 4 КБ в кэше буфера блочного ввода-вывода.
Значения:	По умолчанию: 20; диапазон: от 20 до 1000
Просмотр:	**lsattr -E -l sys0 -a maxbuf**
Изменение:	**chdev -l sys0 -a maxbuf=***новое-значение*Изменение вступает в силу немедленно и действует постоянно. Если задан флаг **-T**, то изменение вступит в силу немедленно и будет действовать до следующей перезагрузки системы. Если указан флаг **-P**, то изменение вступит в силу после следующей загрузки и будет действовать постоянно.
Рекомендации:	Если в выводе команды **sar —b** для параметра `breads` или `bwrites` указано низкое значение `%rcache` или `%wcache`, то рекомендуется настроить этот параметр.
Настройка:	Обычно значение этого параметра почти не влияет на производительность систем, в которых кэш буфера блочного ввода-вывода не применяется при выполнении обычного ввода-вывода.

Обратитесь к разделу Настройка асинхронного ввода-вывода диска

2. **maxpout**

Элемент	Описание
Назначение:	Задает максимальное число ожидающих операций ввода-вывода для файла.
Значения:	По умолчанию: 8193; диапазон: от 0 до n (n+1 должно быть кратным 4)
Просмотр:	**lsattr -E -l sys0 -a maxpout**
Изменение:	**chdev -l sys0 -a maxpout=***новое-значение*Изменение вступает в силу немедленно и действует постоянно. Если задан флаг **-T**, то изменение вступит в силу немедленно и будет действовать до следующей перезагрузки системы. Если указан флаг **-P**, то изменение вступит в силу после следующей загрузки и будет действовать постоянно.
Рекомендации:	Если при выполнении программы с большим числом операций записи на диск, в которых применяется последовательный способ доступа, время интерактивного ответа слишком велико, то необходимо сделать ограничение ввода-вывода более активным. Если скорость последовательного ввода-вывода сильно упадет, необходимо уменьшить или выключить ограничение ввода-вывода.
Настройка:	Если время интерактивного ответа недопустимо велико, необходимо уменьшить значения **maxpout** и **minpout**. Если скорость последовательного ввода-вывода сильно упадет, увеличьте одно или оба значения или задайте для них 0 для выключения ограничения ввода-вывода.

3. minpout

Элемент	Описание
Назначение:	Задает пороговое значение, при достижении которого программы, превысившие ограничение **maxpout**, могут возобновить запись в файл.
Значения:	По умолчанию: 4096; Диапазон: от 0 до n (n должно кратным на 4 по крайней мере на четыре меньше, чем **maxpout**)
Просмотр:	**lsattr -E -l sys0 -a minpout**
Изменение:	**chdev -l sys0 -a minpout=***новое-значение*Изменение вступает в силу немедленно и действует постоянно. Если задан флаг **-T**, то изменение вступит в силу немедленно и будет действовать до следующей перезагрузки системы. Если указан флаг **-P**, то изменение вступит в силу после следующей загрузки и будет действовать постоянно.
Рекомендации:	Если при выполнении программы с большим числом операций записи на диск, в которых применяется последовательный способ доступа, время интерактивного ответа слишком велико, то необходимо сделать ограничение ввода-вывода более активным. Если скорость последовательного ввода-вывода сильно упадет, необходимо уменьшить или выключить ограничение ввода-вывода.
Настройка:	Если время интерактивного ответа недопустимо велико, необходимо уменьшить значения **maxpout** и **minpout**. Если скорость последовательного ввода-вывода сильно упадет, увеличьте одно или оба значения или задайте для них 0 для выключения ограничения ввода-вывода.

4. mount -o nointegrity

Элемент	Описание
Назначение:	Опция mount (nointegrity) позволяет повысить производительность локальной файловой системы для некоторых приложений, выполняющих большое число операций записи. Благодаря ее применению практически исключается запись в протокол JFS. Учтите, что повышение производительности достигается за счет нарушения целостности метаданных. В связи с этим данную опцию следует применять с большой осторожностью, поскольку файловую систему, которая монтировалась с этой опцией, нельзя восстановить после сбоя системы. Тем не менее, для работы некоторых приложений не требуется сохранять целостность данных после сбоя системы, поэтому для них можно задать опцию nointegrity. В частности, применение опции nointegrity оправданно при работе с временными файлами компилятора и при выполнении установки без перехода или установки mksysb.

5. Размер пространства подкачки

Элемент	Описание
Назначение:	Объем дискового пространства, применяемый для хранения страниц рабочей памяти.
Значения:	По умолчанию: зависит от конфигурации; диапазон: от 32 МБ до n МБ для hd6, от 16 МБ до n МБ для других устройств
Просмотр:	**lsps -a mkps** или **chps** или **smitty pgsp**
Изменение:	Изменение вступит в силу немедленно и будет действовать постоянно. Однако новое пространство подкачки не всегда начинает использоваться немедленно.
Рекомендации:	Вызовите команду **lsps -a**. Если некоторые процессы были убиты из-за отсутствия пространства подкачки, получите более подробную информацию с помощью функции **psdanger()**.

Элемент	Описание
Настройка:	Если пространства подкачки не хватает для нормальной работы системы, создайте новое пространство подкачки на отдельном физическом томе или увеличьте существующее пространство подкачки.

6. **Интервал syncd**

Элемент	Описание
Назначение:	Частота, с которой демон **syncd** вызывает функцию **sync()**.
Значения:	По умолчанию: 60; диапазон: любое натуральное значение
Просмотр:	**grep syncd** `/sbin/rc.boot` **vi** `/sbin/rc.boot` или
Изменение:	Изменение вступит в силу со следующей загрузки и будет действовать постоянно. Другой способ изменения этого значения заключается в вызове команды **kill** для демона **syncd** и вызова команды `/usr/sbin/syncd интервал`.
Рекомендации:	Операции ввода-вывода в файл блокируются во время работы демона **syncd**.
Настройка:	Значение по умолчанию не влияет на производительность. Вносить изменения не рекомендуется. Если для сохранения целостности данных вы значительно уменьшите интервал **syncd** (например, для HACMP), то производительность может понизиться.

Изменение значений переменных асинхронного ввода-вывода

Все переменные асинхронного ввода-вывода имеют текущее значение, значение по умолчанию, минимальное и максимальное значения, которые можно просматривать с помощью команды **ioo**.

Только текущее значение можно изменять с помощью команды **ioo**. Остальные три значения изменять нельзя. Они служат для сообщения пользователю границ переменной. Можно изменить текущее значение переменной и присвоить ей постоянное значение, сохраняемое при перезапусках системы. Все переменные подчиняются правилам команды **ioo**, которая содержится в наборе наборе Performance Tools.

Переменные без ограничений приведены в следующей таблице:

Элемент	Описание
`minservers`	Минимальное число процессов ядра для процессора, выделенного для асинхронного ввода-вывода. Каждый процесс ядра использует память, поэтому значение переменной `minservers`, умноженное на число процессоров не должно быть большим, если число асинхронных вводов-выводов маленькое. Значение переменной `minservers` по умолчанию - 3.
`maxservers`	Максимальное число процессов ядра для процессора, выделенного для асинхронного ввода-вывода. Значение этой переменной, умноженное на число процессоров, обозначает ограничение на запросы ввода-вывода медленного пути, выполняемые одновременно, а также ограничение для числа параллельных вводов-выводов. Значение переменной `maxservers` по умолчанию - 30.
`maxreqs`	Задает максимальное число запросов на асинхронный ввод-вывод, которые одновременно могут быть приняты системой. Это число учитывает как обрабатываемые запросы, так и запросы, ожидающие обработки. Максимальное число запросов на асинхронный ввод-вывод должно быть не меньше значения `AIO_MAX`, указанного в файле `/usr/include/sys/limits.h`. В системах с большим числом запросов на асинхронный ввод-вывод максимальное число запросов может превосходить значение `AIO_MAX`. Значение переменной `maxreqs` по умолчанию - 16384.
`server_inactivity`	Задает значение тайм-аута выхода в секундах при простое сервера без обслуживания запросов асинхронного ввода-вывода. Если при выходе серверов их количество становиться меньше числа процессоров `minservers *`, сервер снова переходит в спящий режим ожидания запроса асинхронного ввода-вывода. Этот механизм увеличивает общее быстродействие системы, уменьшая число спящих процессов, не обрабатывающих запросы асинхронного ввода-вывода. Значение переменной `server` по умолчанию - 300.

Настраиваемые параметры диска и адаптера

В ядре операционной системы AIX предусмотрены различные параметры для диска и адаптера диска.

1. **Ограничение на число ожидающих запросов к адаптеру**

Элемент	Description
Назначение:	Максимальное число запросов, ожидающих освобождения шины SCSI. (Применяется только для адаптера SCSI-2 Fast/Wide.)
Значения:	По умолчанию: 40; диапазон: от 40 до 128
Просмотр:	**lsattr -E -l scsi***n* **-a num_cmd_elems**
Изменение:	**chdev -l scsi***n* **-a num_cmd_elems=***новое-значение* Изменение вступит в силу немедленно и будет действовать постоянно. Если задан флаг **-T**, то изменение вступит в силу немедленно и будет действовать до следующей перезагрузки системы. Если указан флаг **-P**, то изменение вступит в силу после следующей загрузки и будет действовать постоянно.
Рекомендации:	Приложения, выполняющие большое число операций записи в логические тома с прямым доступом и чередованием данных, работают медленно.
Настройка:	Значение должно быть равно числу физических дисков (включая диски, входящие в массивы дисков) шины SCSI, умноженному на длину очереди отдельного диска.

2. **Длина очереди дискового накопителя**

Элемент	Description
Назначение:	Максимальное число запросов в очереди дискового накопителя.
Значения:	По умолчанию: диски IBM - 3; другие диски - 0; диапазон: задается производителем диска
Просмотр:	**lsattr -E -l hdisk***n*
Изменение:	**chdev -l hdisk***n* **-a q_type=simple -a queue_depth=***новое-значение* Изменение вступит в силу немедленно и будет действовать постоянно. Если задан флаг **-T**, то изменение вступит в силу немедленно и будет действовать до следующей перезагрузки системы. Если указан флаг **-P**, то изменение вступит в силу после следующей загрузки и будет действовать постоянно.
Рекомендации:	Нет
Настройка:	Если диск другой фирмы поддерживает очередь запросов, измените значение этого параметра, чтобы эта функция применялась операционной системой.

Дополнительная информация: Настройка пределов очередей адаптера SCSI и дисковых устройств

3. **Ограничение на число ожидающих запросов к адаптеру Fibre Channel**

Элемент	Описание
Назначение:	Максимальное число ожидающих запросов в адаптере Fibre Channel.
Значения:	По умолчанию: 200; диапазон: от 200 до 4096
Просмотр:	**lsattr -E -l fcs***n* **-a num_cmd_elems**
Изменение:	**chdev -l fcs***n* **-a num_cmd_elems=***новое-значение*. Если этот атрибут требуется изменить немедленно, то адаптер `fcs`*n* должен находиться в состоянии `defined`. В противном случае для изменения атрибута применяется флаг **`-P`**. Флаг **`-P`** откладывает изменение до следующей операции загрузки и применяет его как постоянное изменение. **Примечание:** Значение по умолчанию и диапазон зависят от конкретного устройства Fibre Channel. Для отдельных адаптеров Fibre Channel и Fibre Channel over Ethernet (FC/FCoE) максимальное значение параметра **num_cmd_elems** может быть меньше максимального значения из Object Data Manager (ODM). Если значение параметра **num_cmd_elems**, указанное в команде **chdev**, превышает значение, поддерживаемое адаптерами FC/FCoE, то в протоколы адаптеров заносится сообщение об ошибке.
Настройка:	Для того чтобы обеспечить оптимальную производительность, укажите в параметре **num_cmd_elems** максимальное значение из поддерживаемого диапазона.

Настраиваемые параметры для межпроцессного взаимодействия

В AIX есть множество параметров для межпроцессного взаимодействия

1. Тонкая настройка параметра **msgmax**:

Элемент	Описание
Назначение:	Задает максимальный размер сообщения.
Значения:	Динамически изменяется, но не превосходит 4 МБ
Просмотр:	Нет
Изменение:	Нет
Рекомендации:	Нет
Настройка:	Изменять значение этого параметра не требуется, так как оно динамически изменяется ядром.

2. Тонкая настройка параметра **msgmnb**:

Элемент	Описание
Назначение:	Задает максимальный размер очереди в байтах
Значения:	Динамически изменяется, но не превосходит 4 МБ
Просмотр:	Нет
Изменение:	Нет
Рекомендации:	Нет
Настройка:	Изменять значение этого параметра не требуется, так как оно динамически изменяется ядром.

3. Тонкая настройка параметра **msgmni**:

Элемент	Описание
Назначение:	Задает максимальное число ИД очередей сообщений.
Значения:	Динамически изменяется, но не превосходит 131072
Просмотр:	Нет
Изменение:	Нет
Рекомендации:	Нет
Настройка:	Изменять значение этого параметра не требуется, так как оно динамически изменяется ядром.

4. Тонкая настройка параметра **msgmnm**:

Элемент	Описание
Назначение:	Задает максимальное число сообщений в очереди
Значения:	Динамически изменяется, но не превосходит 524288
Просмотр:	Нет
Изменение:	Нет
Рекомендации:	Нет
Настройка:	Изменять значение этого параметра не требуется, так как оно динамически изменяется ядром.

5. Тонкая настройка параметра **semaem**:

Элемент	Описание
Назначение:	Задает максимальное значение для корректировки при выходе.
Значения:	Динамически изменяется, но не превосходит значения 16384
Просмотр:	Нет
Изменение:	Нет
Рекомендации:	Нет
Настройка:	Изменять значение этого параметра не требуется, так как оно динамически изменяется ядром.

6. Тонкая настройка параметра **semmni**:

Элемент	Описание
Назначение:	Задает максимальное число ИД семафоров.
Значения:	Динамически изменяется, но не превосходит 131072
Просмотр:	Нет
Изменение:	Нет
Рекомендации:	Нет
Настройка:	Изменять значение этого параметра не требуется, так как оно динамически изменяется ядром.

7. Тонкая настройка параметра **semmsl**:

Элемент	Описание
Назначение:	Задает максимальное число семафоров для ИД.
Значения:	Динамически изменяется, но не превосходит значения 65535
Просмотр:	Нет
Изменение:	Нет
Рекомендации:	Нет
Настройка:	Изменять значение этого параметра не требуется, так как оно динамически изменяется ядром.

8. Тонкая настройка параметра **semopm**:

Элемент	Описание
Назначение:	Задает максимальное число операций для одного вызова **semop()**.
Значения:	Динамически изменяется, но не превосходит 1024
Просмотр:	Нет
Изменение:	Нет
Рекомендации:	Нет
Настройка:	Изменять значение этого параметра не требуется, так как оно динамически изменяется ядром.

9. Тонкая настройка параметра **semume**:

Элемент	Описание
Назначение:	Задает максимальное число записей об отмене для процесса.
Значения:	Динамически изменяется, но не превосходит 1024
Просмотр:	Нет
Изменение:	Нет
Рекомендации:	Нет
Настройка:	Изменять значение этого параметра не требуется, так как оно динамически изменяется ядром.

10. Тонкая настройка параметра **semvmx**;

Элемент	Описание
Назначение:	Задает максимальное значение семафора.
Значения:	Динамически изменяется, но не превосходит 32767
Просмотр:	Нет
Изменение:	Нет
Рекомендации:	Нет
Настройка:	Изменять значение этого параметра не требуется, так как оно динамически изменяется ядром.

11. Тонкая настройка параметра **shmmax**;

Элемент	Описание
Назначение:	Задает максимальный размер общего сегмента памяти.
Значения:	Динамически изменяется, но не превосходит 256 МБ для 32-разрядных процессов или 0x80000000u для 64-разрядных процессов
Просмотр:	Нет
Изменение:	Нет
Рекомендации:	Нет
Настройка:	Изменять значение этого параметра не требуется, так как оно динамически изменяется ядром.

12. Тонкая настройка параметра **shmmin**:

Элемент	Описание
Назначение:	Задает минимальный размер общего сегмента памяти.
Значения:	Динамически изменяется, но всегда больше либо равно 1
Просмотр:	Нет
Изменение:	Нет
Рекомендации:	Нет
Настройка:	Изменять значение этого параметра не требуется, так как оно динамически изменяется ядром.

13. Тонкая настройка параметра **shmmni**:

Элемент	Описание
Назначение:	Задает максимальное число ИД общих сегментов памяти.
Значения:	Динамически изменяется, но не превосходит значения 1048576
Просмотр:	Нет
Изменение:	Нет
Рекомендации:	Нет
Настройка:	Изменять значение этого параметра не требуется, так как оно динамически изменяется ядром.

Параметры, касающиеся работы с сетью

В число таких параметров входят опции NFS и собственно параметры сети.

Параметры сети

В AIX доступно несколько настраиваемых параметров, относящихся к работе с сетью.

Большая часть настраиваемых параметров сети полностью описана в справке по команде **no** в *Справочник по командам, том 4*. Параметры сети, на которые нужно обратить особое внимание при работе в среде SP, описаны в публикации *RS/6000 SP System Performance Tuning*. Остальные связанные параметры перечислены ниже:

1. **maxmbuf**

Элемент	Описание
Назначение:	Максимальный объем оперативной памяти, который может быть выделен для структур MBUF (в КБ).
Значения:	По умолчанию: 0; диапазон: от x до y
Просмотр:	**lsattr -E -l sys0 -a maxmbuf**
Изменение:	**chdev -l sys0 -a maxmbuf=***новое-значение* Изменение вступит в силу немедленно и будет действовать постоянно. Если задан флаг **-T**, то изменение вступит в силу немедленно и будет действовать до следующей перезагрузки системы. Если указан флаг **-P**, то изменение вступит в силу после следующей загрузки и будет действовать постоянно.
Рекомендации:	Нет
Настройка:	Если значение *maxmbuf* больше 0, то значение *maxmbuf* применяется независимо от значения *thewall*. Размер пулов mbuf ограничен сверху большим из значений *maxmbuf* и *thewall*.

Элемент	Описание
Дополнительная информация:	“Отслеживание пулов mbuf с помощью команды netstat -m” на стр. 294

2. MTU

Элемент	Описание
Назначение:	Ограничивает размер пакетов, передаваемых по сети.
Значения:	По умолчанию: зависит от конфигурации
Просмотр:	**lsattr -E -l** *интерфейс*
Изменение:	**chdev -l** *интерфейс* **-a mtu=***новое-значение* Команда **chdev** не позволяет изменить параметры интерфейса, когда он используется. Изменение действует постоянно. Эта команда изменяет размер MTU в работающей системе, однако это изменение действует только до следующей загрузки.
Рекомендации:	Статистика фрагментации пакетов.
Настройка:	Увеличьте размер MTU для сетевых интерфейсов. Для того чтобы разрешить передачу больших фрагментов через адаптер Gigabit Ethernet, установите атрибут устройства **jumbo_frames=yes** (недостаточно присвоить параметру MTU значение 9000).
Дополнительная информация:	“Настройка производительности TCP и UDP” на стр. 258

3. rfc1323

Элемент	Описание
Назначение:	Разрешает применение расширенных функций TCP, описанных в документе RFC 1323 (TCP Extensions for High Performance). Значение 1 указывает, что значения *tcp_sendspace* и *tcp_recvspace* могут превышать ограничение в 64 КБ.
Значения:	По умолчанию: 0; диапазон: 0 или 1
Просмотр:	**lsattr -El** *интерфейс* или **ifconfig** *интерфейс*
Изменение:	**ifconfig** *интерфейс* **rfc1323** *новое-значение* или **chdev -l** *интерфейс* **-a rfc1323=***новое-значение* Команда **ifconfig** временно изменяет значение атрибута. Она может применяться при тестировании. Команда **chdev** изменяет запись ODM, поэтому новое значение вступает в силу после следующей загрузки системы.
Рекомендации:	Нет
Настройка:	Значение 0 устанавливает глобальный запрет на применение расширенных функций, описанных в RFC. Значение 1 указывает, что применение расширенных функций, описанных в RFC, будет согласовываться для всех соединений TCP. Приложение SOCKETS может переопределить это значение для отдельного соединения TCP с помощью функции **setsockopt()**. Этот атрибут применяется во время выполнения. Перед присвоением опциям *tcp_sendspace* и *tcp_recvspace* значений, превышающих 64 КБ, установите этот атрибут.
Дополнительная информация:	“Настройка рабочей схемы TCP” на стр. 272

4. tcp_mssdflt

Элемент	Описание
Назначение:	Максимальный размер сегмента для соединений с удаленными сетями.
Значения:	По умолчанию: 512 байт
Просмотр:	**lsattr -El** *интерфейс* или **ifconfig** *интерфейс*
Изменение:	**ifconfig** *интерфейс* **tcp_mssdflt***новое-значение* или **chdev -l** *интерфейс* **-a tcp_mssdflt=***новое-значение* Команда **ifconfig** временно изменяет значение атрибута. Она может применяться при тестировании. Команда **chdev** изменяет запись ODM, поэтому новое значение вступает в силу после следующей загрузки системы.
Рекомендации:	Нет
Настройка:	*tcp_mssdflt* применяется, если не включена функция определения MTU маршрута, либо если этой функции не удалось определить MTU маршрута. Если это значение будет равно (MTU - 52), то во всех возможных случаях будут отправляться полные пакеты. Этот атрибут применяется во время выполнения.
Дополнительная информация:	“Настройка максимального размера сегмента TCP” на стр. 288

5. tcp_nodelay

Элемент	Описание
Назначение:	Указывает, что при передаче данных по этому интерфейсу через сокеты TCP должен применяться алгоритм Nagle. Протокол TCP применяет алгоритм Nagle по умолчанию.
Значения:	По умолчанию: 0; диапазон: 0 или 1
Просмотр:	**lsattr -El** *интерфейс* или **ifconfig** *интерфейс*
Изменение:	**ifconfig** *интерфейс* **tcp_nodelay** *новое-значение*OR **chdev -l** *интерфейс* **-a tcp_nodelay=***новое-значение* Команда **ifconfig** временно изменяет значение атрибута. Она может применяться при тестировании. Команда **chdev** изменяет запись ODM, поэтому новое значение вступает в силу после следующей загрузки системы.
Рекомендации:	Нет
Настройка:	Этот атрибут относится к параметрам сетевого интерфейса (ISNO).
Дополнительная информация:	“Сетевые опции, связанные с интерфейсом” на стр. 269

6. tcp_recvspace

Элемент	Описание
Назначение:	Задает размер по умолчанию для буфера сокета, в который записываются принимаемые данные. Это значение влияет на размер окна TCP.
Значения:	По умолчанию: 16384 байт
Просмотр:	**lsattr -El** *интерфейс* или **ifconfig** *интерфейс*
Изменение:	**ifconfig** *интерфейс* **tcp_recvspace** *новое-значение* или **chdev -l** *интерфейс* **-a tcp_recvspace=***новое-значение* Команда **ifconfig** временно изменяет значение атрибута. Она может применяться при тестировании. Команда **chdev** изменяет запись ODM, поэтому новое значение вступает в силу после следующей загрузки системы.
Рекомендации:	Нет
Настройка:	Скорость передачи данных в сетях Ethernet и Token-Ring повысится, если размер буфера сокета будет увеличен до 16 КБ (16 384). Значение по умолчанию равно 16 384. Оптимальный размер буфера в сетях с низкой пропускной способностью, например, SLIP, и высокой пропускной способностью, например, Последовательный оптоволоконный канал связи, будет разным. Оптимальный размер буфера зависит от пропускной способности среды передачи данных и среднего времени оборота пакета. Размер буфера сокета, задаваемый атрибутом *tcp_recvspace*, не должен быть больше значения атрибута **sb_max**. Этот атрибут изменяется динамически. Однако для того чтобы его изменение вступило в силу для демонов, запущенных демоном **inetd**, вызовите команды: • **stopsrc-s inetd** • **startsrc -s inetd**
Дополнительная информация:	“Настройка рабочей схемы TCP” на стр. 272

7. tcp_sendspace

Элемент	Описание
Назначение:	Задает размер по умолчанию для буфера сокета, в который записываются отправляемые данные.
Значения:	По умолчанию: 16384 байт
Просмотр:	**lsattr -El** *интерфейс* или **ifconfig** *интерфейс*
Изменение:	**ifconfig** *интерфейс* **tcp_sendspace** *новое-значение*OR **chdev -l** *интерфейс* **-a tcp_sendspace=***новое-значение* Команда **ifconfig** временно изменяет значение атрибута. Она может применяться при тестировании. Команда **chdev** изменяет запись ODM, поэтому новое значение вступает в силу после следующей загрузки системы.
Рекомендации:	Нет

Элемент	Описание
Настройка:	Это значение влияет на размер окна TCP. Скорость передачи данных в сетях Ethernet и Token-Ring повысится, если размер буфера сокета будет увеличен до 16 КБ (16 384). Значение по умолчанию равно 16 384. Оптимальный размер буфера в сетях с низкой пропускной способностью, например, SLIP, и высокой пропускной способностью, например, Последовательный оптоволоконный канал связи, будет разным. Оптимальный размер буфера равен произведению пропускной способности и среднего времени оборота пакета: `optimum_window=bandwidth * average_round_trip_time` Размер буфера сокета, задаваемый атрибутом **tcp_sendspace**, не должен быть больше значения атрибута **sb_max**. **tcp_sendspace** изменяется динамически. Однако для того чтобы его изменение вступило в силу для демонов, запущенных супердемоном **inetd**, требуется выполнение следующих команд: • **stopsrc-s inetd** • **startsrc -s inetd**
Дополнительная информация:	“Настройка рабочей схемы TCP” на стр. 272

8. **use_sndbufpool**

Элемент	Описание
Назначение:	Указывает, должны ли для сокетов применяться пулы буферов передачи.
Значения:	По умолчанию: 1
Просмотр:	**netstat -m**
Изменение:	Для включения этой опции присвойте ей значения 1, для отключения - значение 0.
Рекомендации:	Нет
Настройка:	Это булевская опция времени загрузки.

9. **xmt_que_size**

Элемент	Описание
Назначение:	Задает максимальное число буферов отправки, которые могут быть помещены в очередь для обработки интерфейсом.
Значения:	По умолчанию: зависит от конфигурации
Просмотр:	**lsattr -E -l** *интерфейс*
Изменение:	**ifconfig** *интерфейс* **detach** chdev **-l** *интерфейс* **-a***que_size_name***=***новое-значение* **ifconfig** *хост интерфейса* **up**. Значение атрибута нельзя изменить во время передачи данных по интерфейсу. Изменение действует постоянно.
Рекомендации:	**netstat -i** (`0err` > 0)
Настройка:	Увеличьте значение.
Дополнительная информация:	“Команда netstat” на стр. 300

Настраиваемые параметры NFS

В AIX доступно несколько настраиваемых параметров, относящихся к NFS.

Большая часть параметров NFS полностью описаны в справке по команде **nfso**. Остальные связанные параметры перечислены ниже:

1. **Счетчик biod**

Элемент	Описание
Назначение:	Число процессов biod, доступных на клиенте для обработки запросов NFS.
Значения:	По умолчанию: 6; диапазон: от 1 до любого положительного целого числа
Просмотр:	**ps -efa | grep biod**
Изменение:	**chnfs -b***новое-значение* Обычно изменение вступает в силу немедленно и действует постоянно.. Если указан флаг **-N**, то изменение вступит в силу немедленно и будет действовать только в течение одного сеанса. Если указан флаг **-I**, то изменение вступит в силу после следующей загрузки.
Рекомендации:	С помощью команды **netstat -s** оцените частоту переполнения буферов сокетов UDP.
Настройка:	Увеличивайте значение до тех пор, пока не станет снижаться количество переполнений буферов сокетов.

Элемент	Описание
Дополнительная информация:	“Необходимое число нитей biod” на стр. 347

2. combehind

Элемент	Описание
Назначение:	Включает отложенную фиксацию для клиента NFS при записи очень больших файлов через точки монтирования NFS версии 3.
Значения:	По умолчанию: 0; диапазон: 0 или 1
Просмотр:	**mount**
Изменение:	**mount -o combehind**
Рекомендации:	Низкая пропускная способность при записи очень больших файлов (главным образом файлов, превышающих объем памяти клиента NFS) через точки монтирования NFS версии 3.
Настройка:	Включите эту опцию монтирования в системе клиента NFS, если запись больших файлов на сервер NFS происходит часто. Отрицательным эффектом включения этой опции является фактическое отключение кэширования данных NFS администратором VMM в системе клиента. Поэтому применение этой опции в средах, в которых важна производительность чтения из NFS, не рекомендуется.

3. nfsd Count

Элемент	Описание
Назначение:	Задает максимальное число нитей сервера NFS, применяемых для выполнения поступающих запросов NFS.
Значения:	По умолчанию: 3891; диапазон: от 1 до 3891
Просмотр:	**ps -efa | grep nfsd**
Изменение:	**chnfs -n***новое-значение* Обычно изменение вступает в силу немедленно и действует постоянно.. Если указан флаг **-N**, то изменение вступит в силу немедленно и будет действовать только в течение одного сеанса. Если указан флаг **-I**, то изменение вступит в силу после следующей загрузки.
Рекомендации:	См. **nfs_max_threads**
Настройка:	См. **nfs_max_threads**
Дополнительная информация:	“Необходимое число нитей biod” на стр. 347

4. numclust

Элемент	Описание
Назначение:	Применяется вместе с опцией **combehind** для улучшения производительности записи больших файлов в NFS версии 3.
Значения:	По умолчанию: 128; диапазон: от 8 до 1024
Просмотр:	**mount**
Изменение:	`mount -o numclust=`*`новое-значение`*
Рекомендации:	Низкая пропускная способность при записи очень больших файлов (главным образом файлов, превышающих объем памяти клиента NFS) через точки монтирования NFS версии 3.
Настройка:	Включите эту опцию монтирования в системе клиента NFS, если запись больших файлов на сервер NFS происходит часто. Это значение главным образом задает минимальное число страниц, для которых VMM будет пытаться вызвать операцию фиксации из клиента NFS. Слишком маленькое значение может привести к низкой пропускной способности из-за большого числа фиксаций (каждая из которых вызывает синхронную запись на сервере). Слишком большое значение может привести к снижению производительности из-за переполнения памяти клиента NFS измененными страницами, что приведет к запуску демона LRU для их удаления. При запуске **lrud** операции записи фактически становятся синхронными, поскольку каждая из них сопровождается фиксацией. Опции **numclust** и **combehind** следует настраивать, избегая описанной ситуации.

Настраиваемые параметры потоков

Полный список параметров потоков приведен в справке по команде **no** с опцией **-L**.

Тестовые примеры

В каждом примере описаны тип системы и характер проблемы. Далее описаны средства обнаружения и устранения этой проблемы. Эти примеры могут оказаться очень полезны, особенно если ваша среда схожа хотя бы с одним из них.

При настройке производительности следует учитывать множество факторов, зависящих от конкретных систем и приложений, однако можно привести ряд общих рекомендаций, которые подойдут к большинству систем AIX.

Повышение быстродействия записи больших файлов клиента NFS

Регулярная последовательная запись файлов очень большого размера в файловые системы NFS может привести к серьезному снижению скорости обмена данными с сервером NFS. Данный пример иллюстрирует диагностику такой ситуации и возможные пути устранения проблемы.

Замечания

Описанная ниже процедура была протестирована в отдельных версиях AIX. Результаты, которые вы можете получить, в значительной степени зависят от конкретных версии и уровня AIX.

Пусть в системе работает приложение, осуществляющее последовательную запись очень больших файлов (файлов, размер которых превышает объем оперативной памяти) в файловую систему NFS. Файловая система смонтирована по протоколу NFS V3. Сервер и клиент NFS связаны сетью Ethernet с пропускной способностью 100 Мбит/с. При последовательной записи маленьких файлов в среднем скорость передачи составляет около 10 Мб/с. Однако при записи больших файлов скорость падает до 1 Мб/с.

Снижение скорости передачи происходит из-за того, что на запись большого файла уходит вся оперативная память клиента. Это вызвано тем, что система AIX на клиенте регулярно освобождает страницы оперативной памяти для загрузки следующего набора страниц файла с помощью процедуры **kproc** LRUD.

Данную проблему можно диагностировать несколькими способами:

- Во время записи большого файла регулярно выполняйте команду **nfsstat** (например, раз в 10 секунд):
  ```
  nfsstat
  ```
  Просмотрите вывод команды **nfsstat**. Если количество операций фиксации V3 примерно линейно возрастает с количеством операций записи V3, то скорее всего в вашей системе наблюдается данная проблема.
- С помощью команды **topas** (набор файлов `bos.perf.tools`) определите скорость передачи данных на сервер NFS:
  ```
  topas -i 1
  ```

Если хотя бы одна из этих операций свидетельствует о наличии проблем, рекомендуется перемонтировать файловую систему NFS на клиенте с опцией **combehind** команды **mount**. Выполните следующие действия:

1. Когда в файловой системе не выполняются операции, размонтируйте ее:
   ```
   unmount /mnt
   ```
   (предполагается, что локальная точка монтирования - /mnt)
2. Вновь смонтируйте файловую систему с опцией **combehind** команды **mount**:
   ```
   mount -o combehind сервер:/удаленная-точка-монтирования /mnt
   ```

Понятия, связанные с данным:

“Производительность NFS” на стр. 329
AIX предоставляет инструменты и способы для отслеживания и тонкой настройки NFS и для сервера, и для клиента.

Информация, связанная с данной:

команда mount

Команда nfsstat

команда topas

Оптимизация процедур защиты с помощью индексации паролей

В данном примере проиллюстрировано выявление больших издержек на процедуры идентификации пользователей и устранение этой проблемы за счет индексации файла паролей.

Замечания

Описанная ниже процедура была протестирована в отдельных версиях AIX. Результаты, которые вы можете получить, в значительной степени зависят от конкретных версии и уровня AIX.

Данный пример применялся в среде с двухпроцессорной системой, выполнявшей функции двустороннего почтового сервера. Пользователи получали почту локальными почтовыми клиентами, которые напрямую подключались к удаленному серверу POP3. Для отправки почты применялся демон **sendmail**. На почтовом сервере в силу его природы всегда выполняется много процедур идентификации пользователей. После перехода с однопроцессорной системы на двухпроцессорную команда **uptime** показала около 200 процессов (в однопроцессорной системе значение было меньше единицы).

Для того чтобы определить причину снижения производительности в такой ситуации, выполните следующие действия:

1. Определите, какие процессы потребляют больше всего процессорного времени, и как распределена нагрузка между ядром и пользовательскими процессами, с помощью следующей команды (потребуется набор файлов `bos.perf.tools`):

   ```
   topas -i 1
   ```

   В данном примере команда **topas** показала, что около 90% процессорного времени использовалось пользовательскими процессами, и больше всего времени расходовали процессы **sendmail** и **pop3d**. (Если бы большую часть процессорного времени потребляло ядро, следующим шагом было бы разумно начать трассировку ядра.)
2. Определите, чем потребляется больше процессорного времени: кодом приложения или общими библиотеками. Для этого нужно собрать данные за 1 минуту с помощью следующей команды:

   ```
   tprof -ske -x "sleep 60"
   ```

   Команда **tprof** определяет список процедур, вызванных из общих библиотек, и сортирует их по количеству потребленного процессорного времени. В данном примере команда **tprof** показала, что большая часть пользовательского процессорного времени была потрачена системной библиотекой `libc.a`, отвечающей за процедуры защиты и идентификации. (Если бы команда **tprof** показала, что большая часть времени была потрачена кодом приложения, то следующим шагом было бы разумно провести отладку приложения.)
3. Для того чтобы каждой процедуре защиты не требовалось считывать файл `/etc/passwd`, проиндексируйте его с помощью следующей команды:

   ```
   mkpasswd -f
   ```

В данном примере за счет индексирования файла с идентификационной информацией нагрузку на систему удалось сократить со значения 200 до значения 0.6.

Дополнительная информация приведена в разделах:

- Описание команд **topas**, **tprof** и **uptime** в *Справочник по командам, том 5*.
- Описание демонов **pop3d** и **sendmail** в *Справочник по командам, том 4*.

Общая память BSR

Регистр синхронизации границ (BSR) - это аппаратное обеспечение, которое эффективно обеспечивает совместный доступ к небольшим или сжатым областям памяти. Области памяти обновляются одновременно в несколько потоков.

Память BSR обеспечивает сохраненным блокам памяти более быструю передачу в системе, чем обычная кэш-память. Память BSR использует семантику управления кэш-памятью, которая выделена для манипулирования небольшими блоками памяти от нескольких процессоров в параллельных вычислениях. Семантика управления кэш-памятью изначально не предназначалась для управления разделяемой памятью, но этот механизм полезен для эффективной реализации схемы синхронизации границ, которая используется в высокопроизводительных параллельных потоках.

Для памяти BSR требуется поддержка процессора, и ресурсы логического раздела (LPAR) должны быть настроены на использование функции BSR.

Для выделения общей памяти BSR необходимо выполнить следующее:

1. Выделить общую память V для областей общей памяти системы BSR с помощью функции `shmctl()`.
2. Включить резервное копирование выделенных областей общей системной памяти V в память BSR, для чего вызвать функцию `shmctl()` и выполнить команду **SHM_BSR**.

   Примечание: В общей памяти используется функция `shmctl()` с командой **SHM_BSR**. Этот шаг выполняется непосредственно после создания резервной копии с помощью функции `shmget()` и до того, как первый процесс будет добавлен в общую память. С помощью команды **SHM_BSR** функция `shmctl()` пытается использовать память BSR для заданной области общей памяти.
3. При недостатке памяти BSR и в том случае, если функции BSR не поддерживаются аппаратной платформой, выводится сообщение об ошибке. Функция `shmget()` завершается с `errno`, установленным в `ENOMEM`

   Примечание: Пользователь, не имеющий прав доступа root, должен иметь разрешение `CAP_BYPASS_RAC_VMM` для выделения памяти BSR. В противном случае функция `shmctl()` с командой **SHM_BSR** завершится с `errno`, установленным в `EPERM`.

При использовании общей памяти BSR, в области общей памяти разрешаются только 1-байтовые и 2-байтовые команды хранилища. Команды хранилища, длина которых превышает 2 байта, не работают правильно с общей памятью BSR. Команды загрузки любого размера разрешаются для разделяемой памяти BSR.

Команда **VMINFO** функции `vmgetinfo()` используется для сбора сведений об имеющейся поддержке BSR. При указании команды **VMINFO** в функции `vmgetinfo()` возвращается структура `vminfo`. В поле `bsr_mem_total` указан общий объем памяти, настроенный LPAR. В поле `bsr_mem_free` указан общий объем BSR, который в настоящее время доступен для выделения.

Области общей памяти BSR не изменяются динамически с помощью опции `SHM_SIZE` функции `shmctl()`. При попытке изменения размера области общей памяти BSR путем указания параметра `SHM_SIZE` в функции `shmctl()` функция `shmctl()` завершается с `errno`, установленным в `EINVAL`. Общая память BSR не поддерживается с переменной среды `EXTSHM`. В том случае если переменная среды `EXTSHM` задана, при вызове`shmctl()` с флагом `SHM_BSR` функция `shmctl()` завершается с `EINVAL`.

Пример

Следующий пример показывает, что приложение может запрашивать объем доступной памяти BSR системы и затем выделять и присоединять область общей памяти BSR. В примере показано, как приложение будет отсоединять и удалять область общей памяти BSR.

```
#include <errno.h>
#include <stdio.h>
#include <sys/shm.h>
#include <sys/vminfo.h>

/* shm_rgn_size - размер области общей памяти область для выделения.
 * В этом примере выбрано - 4 КБ (PAGESIZE).  4 КБ - наименьший поддерживаемый
 * размер области общей памяти.  Ожидается, что 4 КБ должно быть достаточно для
 * большинства пользователей.
 */
const size_t shm_rgn_size = PAGESIZE;

int main(int argc, char *argv[])
{
 struct vminfo my_info = { 0 };
 int           id;
 void          *ptr;

 /* Определяется объем доступной памяти BSR */
 if (vmgetinfo(&my_info, VMINFO, sizeof(my_info)) != 0)
 {
  perror("неожиданный сбой vmgetinfo()");
  return 1;
 }

 /* Проверка, что имеется достаточно памяти BSR */
 if (my_info.bsr_mem_free < shm_rgn_size)
 {
  fprintf(stderr, "недостаточно памяти BSR\n");
  return 2;
 }

 /* Выделяется новая область общей памяти */
 id = shmget(IPC_PRIVATE, shm_rgn_size, IPC_CREAT|IPC_EXCL);

 if (id == -1)
 {
  perror("не удалось выполнить shmget()");
  return 3;
 }

 /* Запрос памяти BSR для области общей памяти */
 if (shmctl(id, SHM_BSR, NULL))
 {
  perror("не удалось выполнить shmctl(SHM_BSR)");
  shmctl(id, IPC_RMID, 0);
  return 4;
 }

 /* Присоединение области общей памяти */
 ptr = shmat(id, NULL, 0);
 if ((int)ptr == -1)
 {
  perror("не удалось выполнить shmat()");
  shmctl(id, IPC_RMID, 0);
  return 5;
 }

 /* BSR память может быть доступна, начиная с адреса - ptr */

 /* Отсоединение области общей памяти */
 if (shmdt(ptr))
 {
  perror("не удалось выполнить shmdt()");
  shmctl(id, IPC_RMID, 0);
  return 6;
 }
```

```
 /* Удаление области общей памяти */
 if (shmctl(id, IPC_RMID, 0))
 {
  perror("не удалось выполнить shmctl(IPC_RMID)");
  return 7;
 }

 return 0;
}
```

Стратегия порождения процессов VMM

Можно изменить способ управления памятью, выделенной процессу, при порождении процесса.

Администратор виртуальной памяти (VMM) не копирует все адресное пространство процесса при его разветвлении. Страницы копируются по запросу при изменении родительским или дочерним процессом. Ссылки на страницы, которые еще не были изменены, связывающая с общей памятью родительского и дочернего процессов. Если впоследствии страница будет изменена, то она копируется во время редактирования.

Если память считывается и затем немедленно записывается, то проще создать копию страницы при первом обращении, а не первой записи. Такой подход можно использовать для всех системы путем изменения параметра **vmm_fork_policy** с помощью команды **vmo**. Глобальный настраиваемый параметр можно переопределить с помощью одного процесса путем экспорта переменной среды `VMM_CNTRL` и указания ключевого слова **vmm_fork_policy**.

Примечания

Данная информация была разработана для продуктов и услуг, предлагаемых на территории США.

Компания IBM может не предоставлять в других странах продукты и услуги, обсуждаемые в данном документе. Информацию о продуктах и услугах, распространяемых в вашей стране, вы можете получить в местном представительстве IBM. Ссылки на продукты, программы или услуги IBM не означают, что можно использовать только указанные продукты, программы или услуги IBM. Вместо них можно использовать любые другие функционально эквивалентные продукты, программы или услуги, не нарушающие прав IBM на интеллектуальную собственность. Однако ответственность за проверку действия любых продуктов, программ и услуг других компаний лежит на пользователе.

Компания IBM может обладать заявками на патенты или патентами на предметы обсуждения в данном документе. Обладание данным документом не предоставляет лицензии на эти патенты. Запросы на получение лицензии можно отправлять в письменном виде по адресу:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

За получением лицензий, имеющих отношение к двухбайтовому набору символов (DBCS), обращайтесь в местное отделение компании IBM по интеллектуальной собственности или направьте запрос в письменной форме по следующему адресу:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

КОМПАНИЯ IBM ПРЕДОСТАВЛЯЕТ НАСТОЯЩУЮ ПУБЛИКАЦИЮ НА УСЛОВИЯХ "КАК ЕСТЬ", БЕЗ КАКИХ-ЛИБО ЯВНЫХ ИЛИ ПОДРАЗУМЕВАЕМЫХ ГАРАНТИЙ, ВКЛЮЧАЯ, НО НЕ ОГРАНИЧИВАЯСЬ ЭТИМ, НЕЯВНЫЕ ГАРАНТИИ СОБЛЮДЕНИЯ ПРАВ, КОММЕРЧЕСКОЙ ЦЕННОСТИ И ПРИГОДНОСТИ ДЛЯ КАКОЙ-ЛИБО ЦЕЛИ. В некоторых юрисдикциях освобождение от явных и подразумеваемых гарантий запрещено в некоторых сделках, поэтому это заявление может к вам не относиться.

Эта информация может содержать технические неточности или типографические ошибки. В информацию периодически вносятся изменения, которые будут учтены во всех последующих изданиях этой книги. IBM может вносить обновления или изменения в этот документ без предварительного уведомления.

Любые ссылки на веб-сайты других компаний приведены в данной публикации исключительно для удобства пользователей и не должны рассматриваться как рекомендация этих веб-сайтов. Материалы, размещенные на этих веб-сайтах, не являются частью информации по данному продукту IBM, и ответственность за применение этих материалов лежит на пользователе.

IBM может использовать и распространять предоставленную вами информацию любым способом без каких-либо обязательств перед вами.

Лицам, обладающим лицензией на данную программу и желающим получить информацию о ней с целью: (i) настройки обмена данными между независимо разработанными программами и другими программами (включая данную) и (ii) использования информации, полученной в результате обмена, этими программами, следует обращаться по адресу:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

Такая информация может быть предоставлена на определенных условиях, а в некоторых случаях - и за дополнительную плату.

Описанная в этом документе лицензионная программа и все связанные с ней лицензионные материалы предоставляются IBM в соответствии с условиями Соглашения с заказчиком IBM, Международного соглашения о лицензии на программу IBM или любого другого эквивалентного соглашения.

Данные о производительности и примеры клиентов приведены исключительно в иллюстративных целях. Фактические результаты производительности зависят от конкретных конфигураций и рабочих сред.

Информация о продуктах других компаний была получена от поставщиков этих продуктов, их опубликованных материалов или других общедоступных источников. Компания IBM не проверяла эти продукты и не может подтвердить правильность их работы, совместимость или другие заявленные характеристики продуктов других компаний. По вопросам о возможностях продуктов других компаний следует обращаться к поставщикам этих продуктов.

Заявления относительно будущих намерений IBM могут быть изменены или отозваны без дополнительного уведомления и отражают только текущие цели и задачи.

Все указанные цены IBM являются рекомендуемыми розничными ценами IBM на данный момент и могут быть изменены без предварительного уведомления. Цены дилеров могут быть другими.

Данная информация предназначена только для планирования. Она может быть изменена до выпуска описанных в данном документе продуктов.

Настоящая документация содержит примеры данных и отчетов, применяемых в повседневной деятельности компаний. Для большего сходства с реальностью примеры содержат имена людей, названия компаний, товарных знаков и продуктов. Все эти имена и названия вымышленные. Любые совпадения с реально существующими физическими или юридическими лицами совершенно случайны.

Лицензия на авторские права:

Настоящая документация содержит примеры исходного кода программ, иллюстрирующие приемы программирования в различных операционных системах. Вы имеете право копировать, изменять и распространять эти примеры программ в любой форме без уплаты вознаграждения фирме IBM в целях разработки, применения, сбыта или распространения прикладных программ, соответствующих интерфейсу прикладных программ операционной системы, для которой предназначены эти примеры. Эти примеры не были тщательно и всесторонне протестированы. В связи с этим IBM не может гарантировать их надежность, удобство обслуживания и отсутствие ошибок. Примеры программ предоставляются "КАК ЕСТЬ", без каких-либо гарантий. IBM не несет ответственности за ущерб, который может возникнуть в результате использования эти образцов программ.

Во все копии или фрагменты этих примеров программ, а также программы созданные на их основе, следует добавлять следующее замечание об авторских правах:

© (название вашей компании) (год).

Некоторые фрагменты исходного кода получены из примеров программ фирмы IBM Corp.

© Copyright IBM Corp. _год или годы_.

Замечания о правилах работы с личными данными

Продукты IBM Software, включая решения программного обеспечения как услуг, (“Предложения программного обеспечения”) могут использовать cookie или другие технологии для сбора информации об использовании продукта в целях усовершенствования пользовательского интерфейса, для приспособления взаимодействий к конечному пользователю или для других целей. Во многих случаях Предложениями программного обеспечения собирается информация, в которой невозможно опознать персональные данные. Некоторые из наших Предложений программного обеспечения могут позволить вам собирать опознаваемую персональную информацию. Если это Предложение программного обеспечения использует cookie для сбора опознаваемой персональной информации, то специфическая информация об этом использовании cookie в предложении приведена далее.

Это Предложение программного обеспечения не использует cookie или другие технологии для сбора опознаваемой персональной информации.

Если конфигурации, развернутые для этого Предложения программного обеспечения предоставляют вам как клиенту возможность собирать опознаваемую персональную информацию о конечных пользователях посредством cookie и других технологий, вы должны самостоятельно проконсультироваться с юристом о всех законах, применимых к такому сбору данных, включая требования к уведомлению и согласию.

Более подробная информация об использовании различных технологий, включая cookie, для этих целей, приведена в Политике конфиденциальности IBM (http://www.ibm.com/privacy) и Заявлении IBM о конфиденциальности в Интернет (http://www.ibm.com/privacy/details), а также в разделах “Cookies, Web Beacons and Other Technologies” и “IBM Software Products and Software-as-a-Service Privacy Statement” на странице http://www.ibm.com/software/info/product-privacy.

Товарные знаки

IBM, эмблема IBM и ibm.com являются товарными знаками или зарегистрированными товарными знаками International Business Machines Corp. во всем мире. Названия других продуктов и услуг могут быть товарными знаками IBM и других компаний. Текущий список товарных знаков IBM опубликован на веб-странице Copyright and trademark information по следующему адресу: www.ibm.com/legal/copytrade.shtml.

INFINIBAND, InfiniBand Trade Association и дизайнерские знаки INFINIBAND являются товарными или сервисными знаками INFINIBAND Trade Association.

Linux является зарегистрированным товарным знаком Линуса Торвальдса в США и других странах.

Microsoft, Windows, Windows NT и эмблема Windows являются товарными знаками Microsoft Corporation в США и/или других странах.

Java и все основанные на Java названия и эмблемы являются товарными знаками или зарегистрированными товарными знаками Oracle и/или дочерних компаний.

UNIX - зарегистрированный товарный знак The Open Group в США и других странах.

Индекс

Числа

A

B

C

D

E

F

G

J

L

M

N

P

Г

Д

Ж

З

И

К

Л

М

Н

O

П

P

C

Т

У

Ф

Ц

Ч

Э

Я

Напечатано в Дании