SCO®

SCO OpenServer™ Performance Guide

SCO OpenServer™

SCO OpenServer™
Performance Guide

Document Version: 5.0
1 May 1995

About this book

This book is for administrators of SCO OpenServer™ systems who are interested in investigating and improving system performance. It describes performance tuning for uniprocessor, multiprocessor, and networked systems, including those with TCP/IP, NFS®, and X clients. It discusses how the various subsystems function, possible performance constraints due to hardware limitations, and optimizing system configuration for various uses. Concepts and strategies are illustrated with case studies.

You will find the information you need more quickly if you are familiar with:

- "How this book is organized" (this page)
- "Related documentation" (page 2)
- "Typographical conventions" (page 5)

Although we try to present information in the most useful way, you are the ultimate judge of how well we succeed. Please let us know how we can improve this book (page 6).

How this book is organized

This book tells you:

- what is meant by system performance (page 7)
- how to tune a system (page 13)
- how the configuration of various system components influences the performance of the operating system:
  - Central Processing Units (CPUs) (page 21) for single and multiprocessor systems

  - memory (page 41) including physical (main) memory in Random Access Memory (RAM) and swap areas on disk
  - Input/Output (I/O) (page 71) including hard disks and serial devices
  - networking (page 123) including STREAMS I/O, TCP/IP and NFS
- how you can examine system call activity (page 161) if you are an application programmer

A set of case studies (page 19) illustrates the methodology of system tuning, and the tools that you can use to examine performance.

Appendixes provide additional information about:

- the tools (page 171) that you can use to examine performance
- the kernel parameters (page 185) that you can use to tune performance
- a quick guide to system tuning (page 235)

There is also a glossary (page 243) which explains technical terms and acronyms used throughout the book.

Related documentation

SCO OpenServer systems include comprehensive documentation. Depending on which SCO OpenServer system you have, the following books are available in online and/or printed form. Access online books by double-clicking on the Desktop **Help** icon. Additional printed versions of the books are also available. The Desktop and most SCO OpenServer programs and utilities are linked to extensive context-sensitive help, which in turn is linked to relevant sections in the online versions of the following books. See "Getting help" in the *SCO OpenServer Handbook.*

NOTE When you upgrade or supplement your SCO OpenServer software, you might also install online documentation that is more current than the printed books that came with the original system. For the most up-to-date information, check the online documentation.

Release Notes
contain important late-breaking information about installation, hardware requirements, and known limitations. The *Release Notes* also highlight the new features added for this release.

SCO OpenServer Handbook
provides the information needed to get your SCO OpenServer system up and running, including installation and configuration instructions, and introductions to the Desktop, online documentation, system administration, and troubleshooting.

Graphical Environment Guide
describes how to customize and administer the Graphical Environment, including the X Window System™ server, the SCO® Panner™ window manager, the Desktop, and other X clients.

Graphical Environment help
provides online context-sensitive help for Calendar, Edit, the Desktop, Help, Mail, Paint, the SCO Panner window manager, and the UNIX® command-line window.

Graphical Environment Reference
contains the manual pages for the X server (section X), the Desktop, and X clients from SCO and MIT (section XC).

Guide to Gateways for LAN Servers
describes how to set up SCO® Gateway for NetWare® and LAN Manager Client software on an SCO OpenServer system to access printers, filesystems, and other services provided by servers running Novell® NetWare® and by servers running LAN Manager over DOS, OS/2®, or UNIX systems. This book contains the manual pages for LAN Manager Client commands (section LMC).

Mail and Messaging Guide
describes how to configure and administer your mail system. Topics include **sendmail**, MMDF, **SCO Shell Mail**, **mailx**, and the Post Office Protocol (POP) server.

Networking Guide
provides information on configuring and administering TCP/IP, NFS®, and IPX/SPX™ software to provide networked and distributed functionality, including system and network management, applications support, and file, name, and time services.

Networking Reference
contains the command, file, protocol, and utility manual pages for the IPX/SPX (section PADM), NFS (sections NADM, NC, and NF), and TCP/IP (sections ADMN, ADMP, SFF, and TC) networking software.

Operating System Administrator's Reference
contains the manual pages for system administration commands and utilities (section ADM), system file formats (section F), hardware-specific information (section HW), miscellaneous commands (section M), and SCO Visual Tcl™ commands (section TCL).

Operating System Tutorial
provides a basic introduction to the SCO OpenServer operating system. This book can also be used as a refresher course or a quick-reference guide. Each chapter is a self-contained lesson designed to give hands-on experience using the SCO OpenServer operating system.

Operating System User's Guide
provides an introduction to SCO OpenServer command-line utilities, the SCO Shell utilities, working with files and directories, editing files with the **vi** editor, transferring files to disks and tape, using DOS disks and files in the SCO OpenServer environment, managing processes, shell programming, regular expressions, **awk**, and **sed**.

Operating System User's Reference
contains the manual pages for user-accessible operating system commands and utilities (section C).

PC-Interface Guide
describes how to set up PC-Interface™ software on an SCO OpenServer system to provide print, file, and terminal emulation services to computers running PC-Interface client software under DOS or Microsoft® Windows™.

SCO Merge User's Guide
describes how to use and configure an SCO® Merge™ system. Topics include installing Windows, installing DOS and Windows applications, using DOS with the SCO OpenServer operating system, configuring hardware and software resources, and using SCO Merge in an international environment.

SCO Wabi User's Guide
describes how to use SCO® Wabi™ software to run Windows 3.1 applications on the SCO OpenServer operating system. Topics include installing the SCO Wabi software, setting up drives, configuring ports, managing printing operations, and installing and running applications.

System Administration Guide
describes configuration and maintenance of the base operating system, including account, filesystem, printer, backup, security, UUCP, and virtual disk management.

The SCO OpenServer Development System includes extensive documentation of application development issues and tools.

Many other useful publications about SCO systems by independent authors are available from technical bookstores.

Typographical conventions

This publication presents commands, filenames, keystrokes, and other special elements in these typefaces:

Example:	Used for:
lp or **lp**(C)	commands, device drivers, programs, and utilities (names, icons, or windows); the letter in parentheses indicates the reference manual section in which the command, driver, program, or utility is documented
/new/client.list	files, directories, and desktops (names, icons, or windows)
root	system, network, or user names
filename	placeholders (replace with appropriate name or value)
〈Esc〉	keyboard keys
`Exit program?`	system output such as prompts and messages
yes or **`yes`**	user input
"Description"	field names or column headings (on screen or in database)
open or **open**(S)	library routines, system calls, kernel functions, C keywords; the letter in parentheses indicates the reference manual section in which the file is documented
$HOME	environment or shell variables
SIGHUP	named constants or signals
`buf`	C program structures
`b_b.errno`	C program structure members and variables

How can we improve this book?

What did you find particularly helpful in this book? Are there mistakes in this book? Could it be organized more usefully? Did we leave out information you need or include unnecessary material? If so, please tell us.

To help us implement your suggestions, include relevant details, such as book title, section name, page number, and system component. We would appreciate information on how to contact you in case we need additional explanation.

To contact us, use the card at the back of the *SCO OpenServer Handbook* or write to us at:

Technical Publications
Attn: CFT
The Santa Cruz Operation, Inc.
PO Box 1900
Santa Cruz, California 95061-9969
USA

or e-mail us at:

techpubs@sco.com or ... *uunet!sco!techpubs*

Thank you.

Chapter 1 *What determines performance*

A computer system consists of a finite set of hardware and software components. These components constitute the resources of the system. One of the tasks of the operating system is to share these resources between the programs that are running on the system. Performance is a measure of how well the operating system does this task; the aim of performance tuning is to make it do this task better.

A system's hardware resources have inherent physical limits in the quantity of data they can handle and the speed with which they can do this. The physical subsystems that compose hardware include:

- One or more central processing units (CPUs), and the ancillary processors that support them.
- Memory — both in Random Access Memory (RAM) and as swap space on disk.
- I/O devices including hard and floppy disk drives, tape drives, serial ports, and network cards.
- Networks — both Local Area Networks (LANs) and Wide Area Networks (WANs).

Operating system resources are limited by the hardware resources such as the amount of memory available and how it is accessed. The internal resources of the operating system are usually configurable and control such things as the size of data structures, security policy, standards conformance, and hardware modes.

Examples of operating system resources are:

- The tables that the operating system uses to keep track of users and the programs they are running.
- The buffer cache and other memory buffers that reduce dependence on accessing slow peripheral devices.

If your system is connected to one or more networks, it may depend on remote machines to serve files, perform database transactions, perform calculations, run X clients, and provide swap space, or it may itself provide some of these services. Your system may be a router or gateway if it is connected to more than one network. In such cases, the performance of the network and the remote machines will have a direct influence on the performance of your system.

Hardware factors that influence performance

Your system's hardware has the greatest influence on its performance. It is the ultimate limiting factor on how fast a process will run before it has to start sharing what is available with the operating system and other user processes.

Performance tuning can require you to add hardware or upgrade existing hardware if a system's physical subsystems are unbalanced in power, or insufficiently powerful to satisfy the demands being put on them. There may come a time when, despite your best efforts, you cannot please enough people enough of the time with the hardware resources at your disposal. If so, you will have to go and buy some more hardware. This is one reason why monitoring and recording your system's performance is important if you are not the person spending the money. With the information that you have gathered, you can make a strong case for upgrading your system.

It is important to balance the power of your computer's subsystems with each other; the power of the CPU(s) is not enough in itself. If the other subsystems are slow relative to the available processing power, they will act to constrain it. If they are more powerful, you have possibly overspent, although you should be able to upgrade processing power without much extra expenditure.

There are many hardware factors that can limit the overall system performance:

- The speed and width of the system's address and data buses.
- The model, clock speed, and the size of the internal level-one (L1) memory cache of the system's CPU or CPUs.
- The size of the level-two (L2) cache memory which is external to the CPU. This should be capable of working with all of physical memory.

- The amount of memory, the width of its data path, and its access time. The time that the CPU has to wait for memory to be accessed limits its performance.
- The speed and width of a SCSI bus controlled by a host adapter.
- The width of the data path on peripheral controller cards (32, 16, or 8-bit).
- Whether controllers have built-in cache. This is particularly important for disk and network controllers.
- Access time for hard disks.
- Whether intelligent or dumb serial cards are used; intelligent cards offload much of the work that would otherwise be performed by the CPU.

On multiprocessor machines, the following considerations also become important:

- Write-back L2 cache (for instructions and data) with cache coherency on each CPU to reduce the number of accesses to main memory. This has the benefit of improving CPU performance as well as improving general system performance by reducing contention for the system bus.
- Support for fully distributed interrupts to allow any CPU to service interrupts from I/O devices such as network and disk controllers.
- The memory and I/O subsystems must be as fast as possible to keep up with the demands of the enhanced CPU performance. Use of intelligent peripheral controllers is particularly desirable.

Software factors that influence performance

The way in which applications are written usually has a large impact on performance. If they make inefficient use of processing power, memory, disk, or other subsystems, it is unlikely that you will improve the situation significantly by tuning the operating system.

The efficiency of the algorithms used by an application, or the way that it uses system services, are usually beyond your control unless you have access to source code. Some applications such as large relational database systems provide extensive facilities for performance monitoring and tuning which you should study separately.

- Is it using large numbers of system calls? System calls are expensive in processing overhead and may cause a context switch on the return from the call. You can use **trace**(CP) to discover the system call usage of a program.
- Is it using inefficient **read**(S) and **write**(S) system calls to move small numbers of characters at a time between user space and kernel space? If possible use buffered I/O to avoid this.
- Are formatted reads and writes to disk being used? Unformatted reads and writes are much more efficient for maintaining precision, speed of access, and generally need less disk space.
- Is the application using memory efficiently? Many older applications use disk extensively since they were written in the days of limited core storage and expensive memory.
- What version of **malloc**(S) does the application use (if it uses it at all)? The version in the *libmalloc.a* library allows more control over the allocation of memory than the version in *libc.a*. Memory leakage can occur if you do not call **free**(S) to place blocks of memory back in the **malloc** pool when you have finished with them.
- Does the application group together routines that are used together? This technique (known as localization of reference) tends to reduce the number of text pages that need to be accessed when the program runs. (The system does not load pages of program text into memory when a program runs unless they are needed for the program's execution.)
- Does the application use shared libraries or dynamic linked libraries (DLLs)? The object code of shared libraries can be used by several applications at the same time; the object code of DLLs is also shared and is only loaded when an application needs to access it. Using either type of library is preferable to using statically linked libraries which cannot be shared.
- Does the application use library routines and system calls that are intended to enhance performance? Examples of the APIs provided are:
  - Memory-mapping loads files directly into memory for processing (see **mmap**(S)).
  - Fixed-priority scheduling allows selected time-critical processes to control how they are scheduled and ensure that they execute when they have work to perform. Applications can use the predictable scheduling behavior to improve throughput and reduce contention (see **sched_setparam**(S) and **sched_getparam**(S)).
  - Support for high performance asynchronous I/O, semaphores and latches, and high-resolution timers and spin locks for use by threaded applications (see **aio**(FP), **semaphore**(FP), and **time**(FP)).

Chapter 2

Managing performance

To manage the performance of a system, you normally try to share the available resources equally between its users. However, different users perceive performance according on their own needs and the demands of the applications that they are running. If they use interactive programs, response time is likely to be their main index of performance. Someone interested in performing numeric analysis may only be worried about the turnaround time for off-line batch mode processing. Another person may wish to perform sophisticated image processing in real time and requires quick access to graphics data files. You, as the administrator, are interested in maximizing the throughput of all jobs submitted to the system — in fact, keeping everyone happy. Unfortunately, such differing requirements may be difficult to reconcile.

For example, if you administer a single standalone system, you may decide that your main priority is to improve the interactive response time. You may be able to do this by decreasing the overall workload at peak usage times. This would involve scheduling some work to run as batch jobs at quieter times, or perhaps restricting simultaneous access to your system to a smaller number of users. However, in speeding up your system's response you now have the additional problem of decreased throughput, which results in the completion of fewer jobs, potentially at critical times. In pursuing any particular performance improvement policy there are always likely to be trade-offs, especially in a situation where resources are at a premium.

The next section covers the setting of realistic performance goals as the first step in improving the performance of your computer system. You are then given a method for observing and tuning a system.

Tuning methodology

You can optimize performance by system tuning. This is the process of making adjustments to the way in which various system resources are used so that they correspond more closely to the overall usage patterns of the system. You can improve the overall response time of the system by locating and removing system bottlenecks. You can also customize the various resources to correspond to the needs of an application that is run frequently on the system. Any system tuning that you perform is limited because the performance of an operating system depends closely on the hardware on which it is installed.

To tune a system efficiently, you need a good understanding both of the various system resources, and of how the system is going to be used. This might also involve understanding how different applications use system resources. System tuning is an ongoing process. A well-tuned system may not remain so if the mix of applications and users changes. Once a system has been successfully tuned, you should monitor performance regularly as part of routine system administration. This allows you to make modifications when changes in performance first occur, and not when the performance degrades to the point where the system becomes unusable.

You may be able to extend a system's resources by adding or reconfiguring hardware, but remember that these resources always remain finite. Also you should always bear in mind that there is no exact formula for tuning a system — performance is based on the mixture of applications running on the system, the individuals using them, and your perception of the system's performance.

The flowchart shown in Figure 2-1 (page 15) illustrates the tuning methodology we recommend you follow. Its most important feature is its feedback loop — you may not always get the result you expect when you make changes to your system. You must be prepared to undo your changes so that you can restore your system to its earlier state.

The steps outlined in the methodology are described in the following sections. They are further illustrated by the set of case studies discussed in "Performance tuning case studies" (page 19).

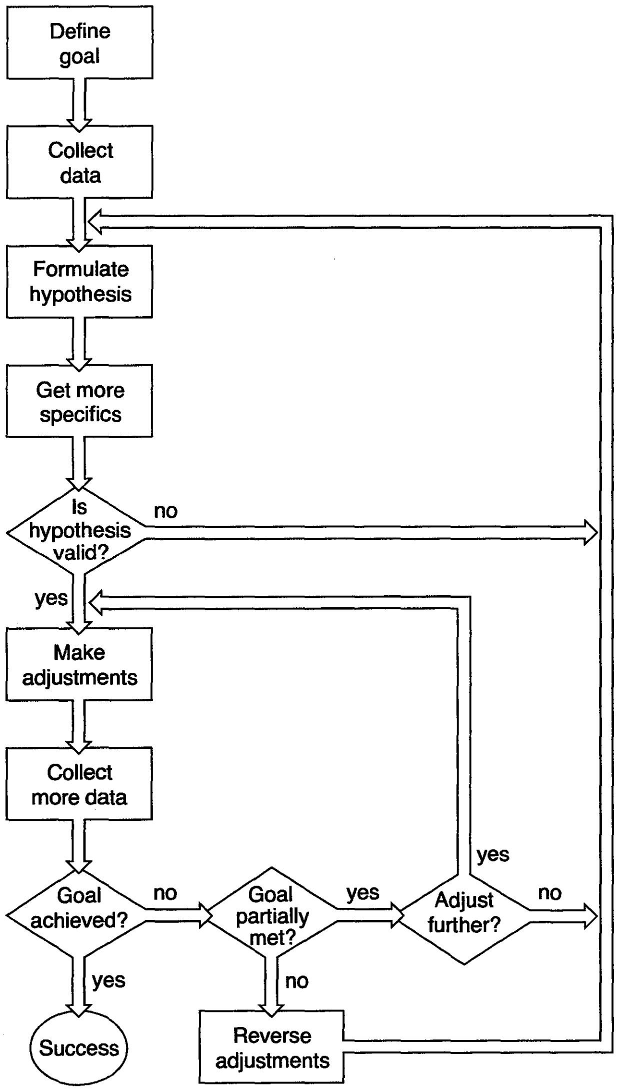


Figure 2-1 Flowchart illustrating the methodology for system performance tuning

Defining performance goals

The first step in tuning a system is to define a set of performance goals. This can range from discovering and removing system bottlenecks in order to improve overall performance, to tuning the system specifically to run a single application, set of applications, or benchmark as efficiently as possible.

The performance goals should be listed in order of priority. Often goals can conflict; for example, a system running a database that uses a large cache might also require a large portion of memory to compile programs during software development. Assigning priority to these goals might involve deciding whether the database performance or the speed of the compilations is more important.

You should attempt to understand all goals as well as possible. If possible, you should note which resources will be affected by each goal. If you specify several goals, it is important that you understand where they might conflict.

Although this guide assumes that you are a system administrator, the goals identified for the tuning of the various subsystems also reflect the perspectives and needs of users, and application developers.

Collecting data

Once you have identified your performance goals, your next step is to determine how the system is performing at present. The aspects of a system's performance that you measure depend on the sort of tasks you expect it to carry out. These are some typical criteria that you might use to judge a system:

- The time taken for an interactive application to perform a task.
- The time taken to process a database transaction.
- The time taken for an application to perform a set number of calculations.

If the system is meant to perform a single function, or run a particular application or benchmark, then you might only look at specific resources. However, it can still be helpful to acquire a sense of the performance of the entire system. If the goals set for the system involve the tuning of applications, then the tuning information provided with the application should be applied before looking at more general system performance.

NOTE It is often possible to improve performance by the careful design and implementation of an application, or by tuning an existing application, rather than by tuning the operating system.

To gain an overview of the system's current performance, you should read and use Appendix A, "Tools reference" (page 171) which discusses the various system resources, and how you can monitor these.

You should collect data over a duration that is long enough for you to be able to establish normal patterns of usage. You should not make decisions that may be based on observations of performance anomalies though your goal may be to smooth these out.

If your goal involves improving the performance of a particular application, you must understand the application's use of systems resources if you suspect that it is not performing as well as it should. Tuning information may be available in the documentation provided with the application. If this is not available, then an indication of how the application uses resources can be gained by gathering information for a period before installing the application, and comparing that information with information gathered while the application is in use.

Formulating a hypothesis

The next step is to determine what is causing the difference between the current performance and your performance goal. You need to understand the subsystems that have an influence on being able to achieve this goal. Begin with a hypothesis, that is, your best informed guess, of the factors that are critical for moving the system toward the goal. You can then use this hypothesis to make adjustments to the system on a trial basis.

If this approach is used then you should maintain a record of adjustments that you have made. You should also keep all the data files produced with the various monitoring commands such as **timex**(ADM), and **sar**(ADM). This is useful when you want to confirm that a side effect noticed after a recent change was caused by that change and did not occur previously.

Getting more specifics

Once you have formulated your hypothesis, look for more specific information. If this information supports the hypothesis, then you can make adjustments to kernel parameters or the hardware configuration to try to improve the performance. If the new information indicates that your hypothesis is wrong then you need to form another.

See Appendix D, "Quick system tuning reference" (page 235) for a description of how to diagnose common performance problems.

Making adjustments to the system

Once it appears that the hypothesis is correct, you can make adjustments to the system. It is vital that you record the parameters that the system had initially, and the changes that you make at each stage. Make all adjustments in small steps to ensure that they have the desired effect. After each adjustment, reassess the system's performance using the same commands that you used to measure its initial performance.

You should normally adjust kernel parameters one at a time so that you can uniquely identify the effect that an adjustment has. If you adjust several things at once, the interaction between them may mask the effect of the change. Some parameters, however, are intended to be adjusted in groups rather than singly. In such a case, always adjust the minimum number of parameters, and always adjust the same set of parameters. Examples of such groups of parameters are **NBUF** and **NHBUF**, and **HTCACHEENTS**, **HTHASHQS** and **HTOFBIAS**.

If your adjustment degrades system performance, retrace your steps to a point where it was at its peak before trying to adjust any other parameters on the system. If your performance goals are not met, you must further evaluate and tune the system. This may mean making changes similar to the ones that you have already made, or you may need consider improving the performance of other subsystems.

If you have attained your performance goals then you can check the system against the lists of desired attributes of well-tuned multiuser or database server systems given in Appendix D, "Quick system tuning reference" (page 235). You should continue to monitor system performance as part of routine system administration to ensure that you recognize and treat any possible future degradation in performance at an early stage.

If you adopt the habit of monitoring performance on a regular basis, you should be able to spot correlations between the numbers recorded and changing demands on the system. Bursts of high system activity during the day, on a particular day of the week, month, or quarter almost certainly reflect the pattern of activity by users, either logged on or running batch jobs. It is up to you to decide how to manage this. You can choose to tune or upgrade the system to cope with peak demand, to reschedule jobs to make use of periods of normally low activity, or both.

Performance tuning case studies

We have provided several case studies that you can use as starting points for your own investigations. Each study is discussed in terms of the five steps described in "Tuning methodology" (page 14):

1. Define a performance goal for the system.
2. Collect data to get a general picture of how the system is behaving.
3. Formulate a hypothesis based on your observations.
4. Get more specifics to enable you to test the validity of your hypothesis.
5. Make adjustments to the system, and test the outcome of these. If necessary, repeat steps 2 to 5 until your goal is achieved.

The case studies have been chosen to represent a variety of application mixes on different systems:

- memory-bound workstation (page 59)
- memory-bound software development system (page 65)
- I/O-bound multiuser system (page 113)
- unbalanced disk activity on a database server (page 118)
- semaphore activity on a database server (page 167)
- network overhead caused by X clients (page 158)

Managing the workload

If a system is sufficiently well tuned for its applications and uses to which it is normally put, you still have a number of options open to you if you are looking for further performance gains. This involves managing the system's workload with the cooperation of the system's users. If they can be persuaded to take some responsibility with you (as the system administrator) for the system's performance then significant improvements can usually be made. Below are some steps that users and administrators can take to alleviate excessive demands on a system without reconfiguring the kernel.

- Move jobs that do not have to run at a particular time of day to off-peak hours. Encourage users to submit jobs using **at**(C), **batch**(C), or **crontab**(C) depending on whether they are one-off (**at** or **batch**) or periodic jobs (**crontab**).
- Collect data on the average system workload and publish it to users so that they are aware of the daily peaks and troughs. If they have the flexibility to choose when to run a program, they will know when they can achieve more work.

- Adjust the default nice value of user processes using the **Hardware/Kernel Manager**. This will set a lower CPU priority for all user processes, and will allow critical jobs with higher priority to use the CPU more frequently.
- Encourage users to reduce the priority of their own processes using **nice**(C) and **renice**(C); this is especially important for those jobs that do not perform much I/O activity — these CPU-intensive jobs are likely to monopolize the available processing time.
- The default action of the Korn shell (**ksh**(C)) is to run background jobs at a reduced priority. Make sure users have not altered this setting in their *.profile* or *.kshrc* files.
- Encourage users to kill unnecessary processes, and to log out when they have finished rather than locking their screen.
- Reduce the maximum number of processes that a user can run concurrently by lowering the value of the kernel parameter **MAXUP**. For example, **MAXUP** set to 20 means that a user can run 19 other processes in addition to their login shell.

If you do not have access to additional hardware and your system is well tuned, you may have to implement some of the above recommendations.

Chapter 3

Tuning CPU resources

Your system hardware contains one or more central processing units (CPUs) plus a host of ancillary processors that relieve the CPU from having to perform certain tasks:

- Math coprocessors perform floating point calculations much more efficiently than software can. The 80486DX™, 80486DX2™, 80486DX4™, and Pentium™ include floating-point capability on the chip itself. Without a floating point coprocessor, the CPU must emulate it using software — this is considerably slower. On systems with an SCO® SMP® License, you can use the **-F** option to **mpsar**(ADM) to monitor how many processes are using floating point arithmetic. This command displays information about the usage of both floating point hardware and software emulation.
- Direct memory access (DMA) controllers handle memory transfer between devices and memory, or memory and memory. Many hardware peripheral controllers on EISA and MCA bus machines have a built-in Bus Master DMA chip that can perform DMA rather than relying on the DMA controller on the motherboard. On MCA bus machines, a chip called a Central Arbitration Control Point (CACP) decides which Bus Master DMA controller gets control of the bus.

  An important limitation of all DMA controllers on ISA and early-series MCA bus machines, and some peripheral controllers on all bus architectures, is that they cannot address more than the first 16MB of memory (24-bit addressing). When the operating system encounters hardware with such limitations, it must instruct the CPU to transfer data between the first 16MB and higher memory.

Some peripheral controllers (including IDE disk controllers) and older SCSI host adapters either cannot perform DMA or the device driver may not support its use. In this case, the operating system instructs the CPU to transfer data between the peripheral and memory on behalf of the hardware. This is known as programmed I/O (PIO).

- Graphics adapters that can take advantage of a local bus architecture (such as VL Bus or PCI) operating at the same speed as the CPU produce a substantial improvement in the performance of the graphics subsystem.
- Universal asynchronous receiver/transmitters (UARTs) control input and output (I/O) on serial lines. Buffering on UARTs enables more efficient use of the CPU in processing characters input or output over serial lines.

  Intelligent serial cards are able to offload much of the character processing that the CPU might otherwise have to perform.
- Programmable interrupt controllers (PICs) handle interrupts from hardware peripheral devices when they are trying to get the attention of the CPU.

The operating system handles these resources for you — reprogramming the various peripheral processor chips to perform tasks on behalf of the CPU.

Operating system states

The operating system can be in one of four states:

executing in user mode
: The CPU is executing the text (machine code) of a process that accesses its own data space in memory.

executing in system mode
: If a process makes a system call in order to perform a privileged task requiring the services of the kernel (such as accessing a disk), then the operating system places the CPU in system mode (also known as kernel mode).

idle waiting for I/O
: Processes are sleeping while waiting for the completion of I/O to disk or other block devices.

idle No processes are ready-to-run on the CPU or are sleeping waiting for block I/O. Processes waiting for keyboard input or network I/O are counted as idle.

The combination of time spent waiting for I/O and time spent idle makes up the total time that the operating system spends idle.

Viewing CPU activity

You can view CPU activity using **sar -u** on single processor systems:

```
23:59:44    %usr    %sys    %wio   %idle
23:59:49       4      24       6      66
23:59:54       7      84       0       9
23:59:59       6      70       1      23

Average        5      59       2      32
```

On systems with an SCO SMP License, use **mpsar -u** to see activity averaged over all the CPUs and **cpusar -u** to report activity for an individual CPU.

`%usr` indicates the percentage of time that the operating system is executing processes in user mode.

`%sys` indicates the percentage of time that the operating system is executing in system mode.

`%wio` indicates the percentage of time that the operating system is idle with processes that could run if they were not waiting for I/O to complete.

`%idle` indicates the percentage of time that the operating system is idle with no runnable processes. On systems with an SCO SMP License, a CPU runs a process called **idle** if there are no other runnable processes.

On systems using SMP, *root* can make a CPU inactive using the **cpuonoff**(ADM) command. The **-c** option displays the number of active and inactive CPUs:

```
$ cpuonoff -c
cpu 1: active
cpu 2: inactive
cpu 3: active
```

The base processor, which cannot be made inactivate, is always indicated by `cpu 1`. An inactive CPU shows 100% idle time with the **cpusar -u** command.

The following sections outline the different process states and how processes can share the same CPU.

Process states

As soon as a process has been created, the system assigns it a state. A process can be in one of several states. You can view the state of the processes on a system using the **ps**(C) command with the **-el** options. The "S" field displays the current state as a single letter.

The important states for performance tuning are:

O *On processor* — the processor is executing on the CPU in either user or system mode.

R *Runnable* — the process is on a run queue and is ready-to-run. A runnable process has every resource that it needs to execute except the CPU itself.

S *Sleeping* — the process is waiting for some I/O event to complete such as keyboard input or a disk transfer. Sleeping processes are not runnable until the I/O resource becomes available.

Figure 3-1 (page 25) represents these process states and the possible transitions between them.

On single CPU systems only one process can run on the CPU at a time. All other runnable processes have to wait on the run queue.

A portion of the kernel known as the scheduler chooses which process to run on the CPU(s). When the scheduler wants to run a different process on the CPU, it scans the run queue from the highest priority to the lowest looking for the first runnable process it can find.

When a process becomes runnable, the kernel calculates its priority and places it on the run queue at that priority. While it remains runnable, the process' priority is recalculated once every second, and its position in the run queue is adjusted. When there are no higher-priority runnable processes on the run queue, the process is placed on the CPU to run for a fixed amount of time known as a time slice.

The operation of the scheduler is more sophisticated for SMP. See "Process scheduling" (page 34) for more information.

For certain mixes of applications, it may be beneficial to performance to adjust the way that the scheduler operates. This is discussed in "Adjusting the scheduling of processes" (page 34).

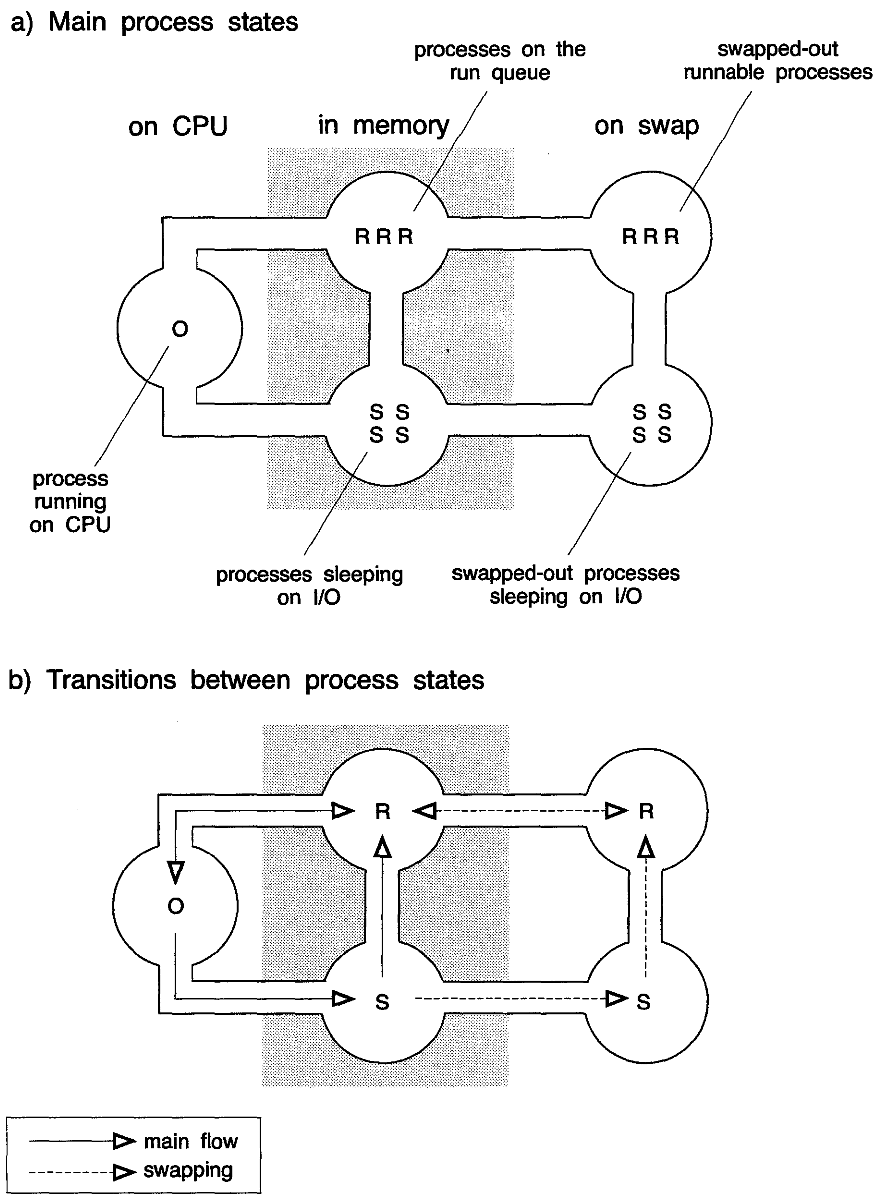

a) Main process states
processes on the run queue
swapped-out runnable processes
on CPU
in memory
on swap
R R R
R R R
O
S S S S
S S S S
process running on CPU
processes sleeping on I/O
swapped-out processes sleeping on I/O
b) Transitions between process states
R
R
O
S
S
main flow
swapping

Figure 3-1 Main process states in a system and the transitions between them

Clock ticks and time slices

The system motherboard has a programmable interval timer which is used as the system clock; this generates 100 clock interrupts or clock ticks per second (this value is defined as the constant **HZ** in the header file */usr/include/sys/param.h*).

The tunable kernel parameter **MAXSLICE** sets the maximum time slice for a process. Its default value is 100 clock ticks (one second). The range of permissible values is between 25 and 100 (between one quarter of a second and one second).

The effect of reducing **MAXSLICE** is to allow each process to run more often but for a shorter period of time. This can make interactive applications running on the system seem more responsive. However, you should note that adjusting the value of **MAXSLICE** may have little effect in practice. This is because most processes will need to sleep before their time slice expires in order to wait for an I/O resource. Even a calculation-intensive process, which performs little I/O, will tend to be replaced on the CPU by processes woken when an I/O resource becomes available.

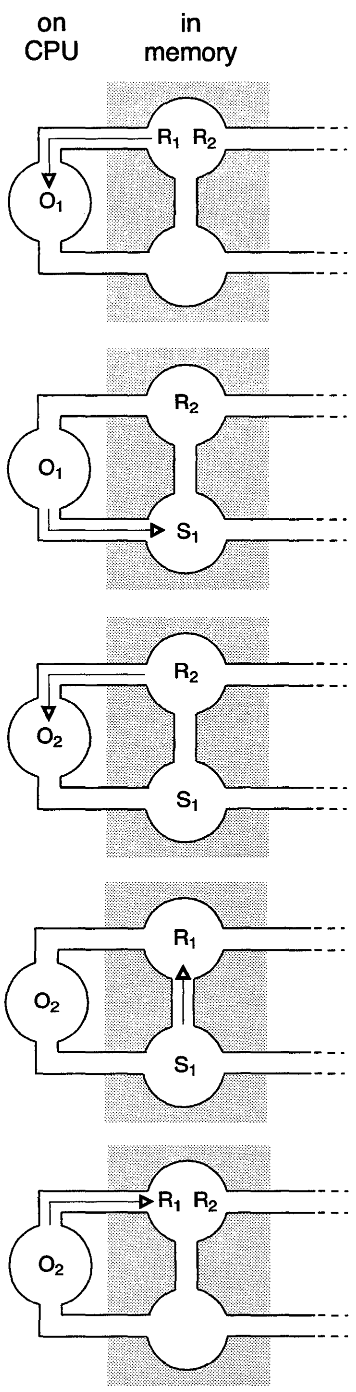
Context switching

A process runs on the CPU until it is context switched. This happens when one of the following occurs:

- The process exits.
- The process uses up its time slice.
- The process requires another resource that is not currently available or needs to wait for I/O to complete.
- A resource has become available for a sleeping process. If there is a higher priority process ready to run, the kernel will run this instead (the current process is preempted).
- The process relinquishes the CPU using a semaphore or similar system call.

The scheduler can only take a process off the CPU when returning to user mode from system mode, or if the process voluntarily relinquishes the CPU from system mode.

If the process has used up its time slice or is preempted, it is returned to the run queue. If it cannot proceed without access to a resource such as disk I/O, it sleeps until the resource is available. Once access to that resource is available, the process is placed on the run queue before being put on the processor. Figure 3-2 (page 27) illustrates this for a process O_1 which goes to sleep waiting for I/O.



a) Runnable process R_1 put on CPU as O_1

b) Process O_1 goes to sleep waiting for I/O as S_1

c) Context switch - runnable process R_2 put on CPU as O_2

d) Process S_1 is woken when resource becomes available; put on run queue as R_1

e) Process O_2 is preempted and put back on run queue as R_2. R_1 is put on CPU next, as shown in figure a, because it has higher priority than R_2

Figure 3-2 Preemption of a process that goes to sleep waiting for I/O

A context switch occurs when the kernel transfers control of the CPU from an executing process to another that is ready to run. The kernel first saves the context of the process. The context is the set of CPU register values and other data that describes the process' state. The kernel then loads the context of the new process which then starts to execute.

When the process that was taken off the CPU next runs, it resumes from the point at which it was taken off the CPU. This is possible because the saved context includes the instruction pointer. This indicates the point in the executable code that the CPU had reached when the context switch occurred.

Interrupts

An interrupt is a notification from a device that tells the kernel that:

- An action such as a disk transfer has been completed.
- Data such as keyboard input or a mouse event has been received.

The kernel services an interrupt within the context of the current process that is running on the CPU. The execution of the current process is suspended while the kernel deals with the interrupt in system mode. The process may then lose its place on the CPU as a result of a context switch. If the interrupt signaled the completion of an I/O transfer, the scheduler wakes the process that was sleeping on that event, and puts it on a run queue at a newly calculated numeric priority. It may or may not be the next process to run depending on this priority.

Calculation of process priorities

A process' priority can range between 0 (lowest priority) and 127 (highest priority). User mode processes run at lower priorities (lower values) than system mode processes. A user mode process can have a priority of 0 to 65, whereas a system mode process has a priority of 66 to 95. Some of the system mode priorities indicate what a process is waiting for. For example, a priority of 81 indicates that a process is waiting for I/O to complete whereas a value of 75 means that it is waiting for keyboard input. The **ps** command with the **-l** option lists process priorities under the `PRI` column.

Processes with priorities in the range 96 to 127 have fixed priority and control their own scheduling.

NOTE You can find a list of priority values in Table A-2, "Priority values" (page 175).

Figure 3-3 (this page) shows the division of process priorities into user mode, system mode, and fixed-priority processes.

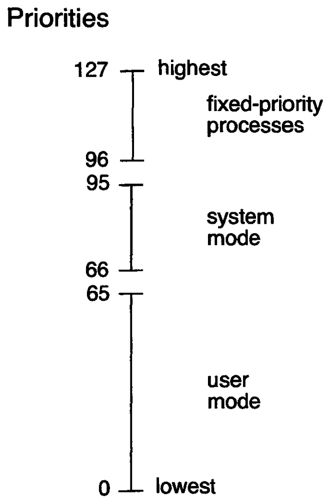


Figure 3-3 System process priorities

The operating system varies the priorities of executing processes according to a simple scheduling algorithm which ensures that each process on the system gets fair access to the CPU. Every process receives a base level priority (of default value 51) when it is created. However, this soon loses any influence on whether a process is selected to run. Note that the priorities of kernel daemons such as **sched**, **vhand**, and **bdflush** are not adjusted. Fixed-priority processes are also exempt — such processes have the ability to adjust their own priority.

The kernel recalculates the priority of a running process every clock tick. The new priority is based on the process' nice value, and how much CPU time the process has used (if any). When the process is taken off the CPU, its lowered priority pushes it down the run queue to decrease the probability that it will be chosen to run in the near future.

A process that manages to run for an entire time slice will have had its priority reduced by the maximum amount.

The kernel recalculates the priorities of all runnable processes (those with a user mode priority less than 65) once every second by successively reducing the negative weighting given to their recent CPU usage. This increases the probability that these processes will be selected to run again in the near future.

The default nice value of a user's process is 20. An ordinary user can increase this value to 39 and in so doing reduce a process' chance of running on the CPU. Processes with low nice values will on average get more CPU time because of the effect the values have on the scheduling algorithm.

There are three ways to control the nice value of a process:

- **nice**(C) reduces the nice value of a new process; *root* can also increase the nice value using this command.
- **renice**(C) reduces the nice value of a process that is already running; *root* can also increase the nice value using this command.
- If the option **bgnice** is set in the Korn shell, it runs background jobs at a nice value of 24. If this option is not set, background jobs run at an equal priority to foreground jobs.

Examining the run queue

Run queue statistics can be seen with **sar -q** on single processor systems or **mpsar -q** on multiprocessor systems:

```
23:59:44 runq-sz %runocc swpq-sz %swpocc
23:59:49     1.7      98     1.5      36
23:59:54     1.0      63     1.0      31
23:59:59     1.0      58     1.0      49

Average      1.3      74     1.2      39
```

`runq-sz` indicates the number of processes that are ready to run (on the run queue) and `%runocc` indicates the percentage of time that the run queue was occupied by at least one process.

See "Identifying CPU-bound systems" (page 38) for a discussion of how to identify if your system is CPU bound.

Multiprocessor systems

The SCO OpenServer system is a multitasking, multiuser operating system, designed to share resources on a computer with a single CPU. It can run on a more powerful multiprocessor system but it cannot use more than one of the available CPUs.

SCO SMP License software adds multiprocessing-specific components to the standard operating system kernel, enabling it to recognize and use additional processors automatically. As SMP is implemented as an extension to, and is completely compatible with the version of the kernel that supports a single CPU. With SCO SMP License software installed, the operating system retains its multitasking, multiuser functionality. There is no impact on existing utilities, system administration, or filesystems. SMP can executes standard OMF (**x.out**), COFF, and ELF binaries without modification.

SMP is modular. As your system requires more processing power, you can add additional processors. For example, two processors give you twice the processing power of a single processor of identical specification in terms of the number of instructions per second that they can execute.

If the operating system can gain extra performance in direct proportion to the number of processors, it is said to exhibit perfect scaling as shown in Figure 3-4 (page 32). In practice, the processors have to compete for other resources such as memory and disk, they have to co-operate in how they handle interrupts from peripherals and from other CPUs, and they may have to wait to gain access to data structures and devices.

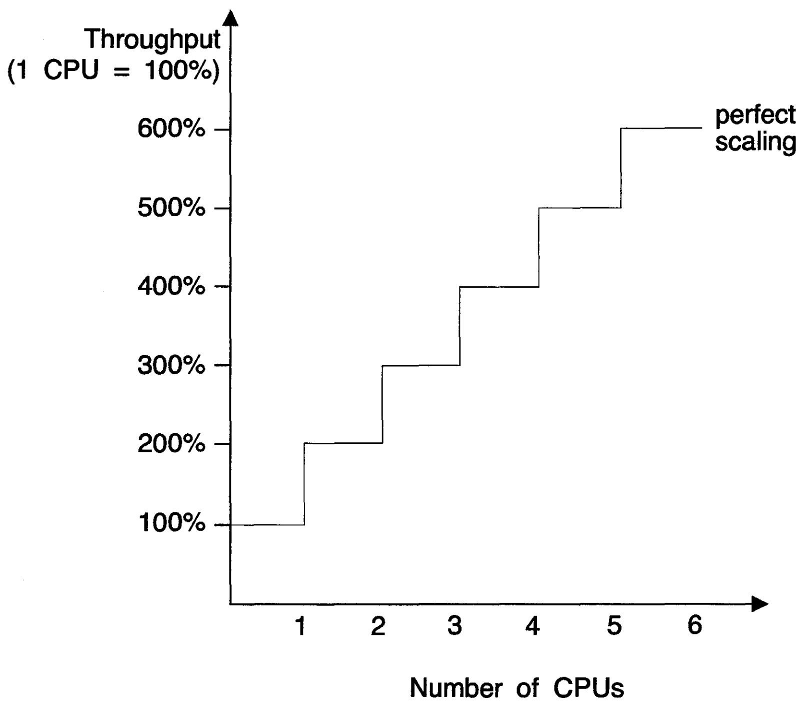


Figure 3-4 Perfect multiprocessor scaling

To ensure good scaling, you should ensure that the memory and I/O subsystems (particularly hard disks) are powerful enough to satisfy the demands that multiple processors put on them. If you do not match the power of your subsystems to that of the processors, your system is likely to be memory or I/O bound, and it will not utilize the potential performance of the processors.

A system will scale well when there are many ready-to-run processes. Multithreaded applications are also well suited to take advantage of a multiprocessing environment.

Support for multiple processors

In SMP, all CPUs can access the same memory, and they all run the same copy of the kernel. As in the single processor version of the operating system, the operating system state on each CPU may be executing in kernel mode, executing in user mode, idle, or idle waiting for I/O.

All processors can run the kernel simultaneously because it is multithreaded; that is, it is designed to run simultaneously on several processors while protecting shared memory structures. Any processor can execute primary kernel functions such as filesystem access, memory and buffer management, distributed interrupt and trap handling, process scheduling, and system calls.

Most standard device drivers provided with the system are also multithreaded. Any unmodified driver or kernel module that does not register itself as multithreaded runs on the base processor.

Figure 3-5 (this page) shows how we can modify the process state diagram introduced in "Process states" (page 24) and apply it to a multiprocessor system. Note that this diagram implies that the kernel not only has to consider when to run a process but also on which CPU to run it.

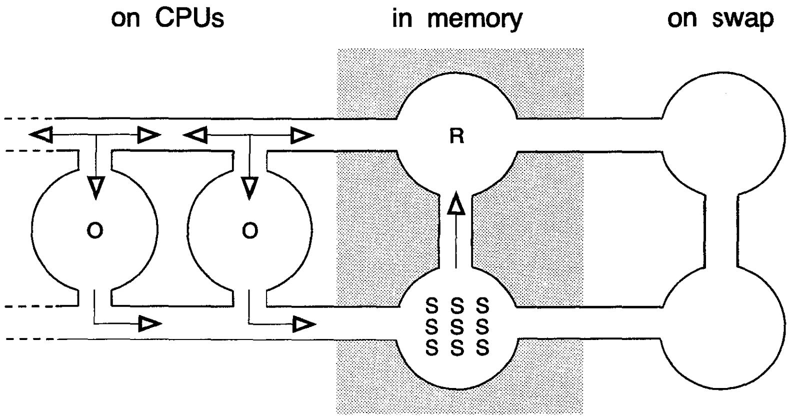


Figure 3-5 Process states on a multiprocessor computer

Using the mpstat load displayer

On systems with an SCO SMP License, the **mpstat** utility visually displays processor activity for each of the processors installed on your system. It allows you to verify at a glance that the system load is balanced across all available processors. See the **mpstat**(ADM) manual page for more information.

Examining interrupt activity on multiprocessor systems

On multiprocessor systems, interrupts sent between the CPUs coordinate and synchronize timing, I/O, and other cooperative activity.

You can use **cpusar -j** to see how active interrupt handling routines are on a particular CPU in a multiprocessor system. If device drivers are not written to be multithreaded they will only run on the base processor. You can examine which device drivers are multithreaded using the **mthread**(ADM) command. You can also use the **displayintr**(ADM) command to see how interrupt handlers are distributed across the system's CPUs and whether they are movable from one CPU to another.

The number of inter-CPU interrupts can be examined using **mpsar -I** to view systemwide activity or **cpusar -I** to examine an individual CPU. The output of these commands depends on your system hardware.

Process scheduling

In a single processor UNIX operating system, the scheduler only concerns itself with when to run a process on the CPU. In a multiprocessor UNIX operating system, the scheduler not only has to consider when to run a process, but also where to run it. Because the kernel runs on all the processors, the process scheduler may be active on any or all of the CPUs. You can adjust how the process scheduler works in order to improve performance as described in "Adjusting the scheduling of processes" (this page).

Adjusting the scheduling of processes

You can configure the process scheduling policy to suit a particular application mix by adjusting the values of a few kernel variables as described in the following sections.

The variables **dopricalc, primove,** and **cache_affinity** control the behavior of priority calculations and the scheduler on both single processor and multiprocessor machines; they are to be found in the file */etc/conf/pack.d/kernel/space.c.*

The variables **preemptive** and **loadbalance** only apply to SMP and can be found in */etc/conf/pack.d/crllry/space.c*. To change the values of these variables, edit the files, then relink and reboot the kernel.

It is not possible to predict the effect of the settings on a particular system. It is likely that you will have to try alternative values to determine whether there is a gain.

For database servers on systems with an SCO SMP License, you may find that setting **preemptive, loadbalance** and **dopricalc** to zero gives a performance improvement.

The following sections describe the effect of adjusting these variables:

- "Controlling priority calculations — dopricalc" (this page)
- "Controlling the effective priority of processes — primove" (page 36)
- "Controlling cache affinity — cache_affinity" (page 37)
- "Controlling process preemption — preemptive" (page 37)
- "Load balancing — loadbalance" (page 37)

Controlling priority calculations — dopricalc

The **dopricalc** variable controls whether the kernel adjusts the priorities of all runnable processes at one-second intervals. Its value has no effect on the recalculation every clock tick of the priority of a process that is currently running.

For some application mixes, such as database servers which have no logged-in users and which make frequent I/O requests, disabling the recalculation of the priorities of ready-to-run processes may improve performance. This is because a process running on a CPU is more likely to continue to run until it reaches the end of its time slice or until it sleeps on starting an I/O request.

The default value of **dopricalc** is 1 which enables the one-second priority calculations. To turn off the calculations, set the value of **dopricalc** to 0, relink the kernel, and reboot the system. This modification will reduce the number of context switches, and may increase the efficiency of the L2 cache. However, it may impair the performance of system if there is a mixture of interactive and CPU-intensive processes. CPU-intensive processes spend all or nearly all of their time in user space; they do not go to sleep waiting for I/O, and so they are unlikely to be context switched except at the end of their time slice. As a consequence, interactive processes may receive less access to the CPU.

Controlling the effective priority of processes — primove

Until now, the discussion of process priorities has assumed that the scheduler uses a process' calculated priority to decide whether the process should be put on the CPU to run. In the default configuration of the kernel, this is effectively true. In fact, the kernel implements the run queue as separate lists of runnable processes for each priority value. The scheduler examines the priority value assigned to each list rather than the priorities of the processes that they contain when looking for a process to run. Provided the kernel assigns processes to the list corresponding to their priority, the lists are invisible. Under some circumstances, it may be beneficial to performance to allow processes to remain in a list after their priority has been changed.

When the priority of a user process is adjusted, the variable **primove** controls whether the kernel moves the process to a higher or lower value priority list. The process will only be moved to a new list if its priority differs from the present list priority by at least the value of **primove**. The effect of increasing **primove** is to make a process remain at a low or high priority for longer. It also means that the operating system has less work to do moving processes between different lists. The default value of **primove** is 0 for compliance with POSIX.1b. This means that any change in a process' priority will cause it to be moved to a different list.

For an example of the use of **primove**, assume that it is given a value of 10. If the priority of a process begins at 51 and rises by at least a value of 10, it is moved to the list corresponding to priority 61. The process does not move between lists until its priority rises by at least the value of **primove**. So if the process' priority rose to 60, it would remain on the priority 50 list. The kernel, however, would still see the process as having a lower priority than another in the priority 55 list. Conversely, a process in the priority 71 list will stay there until its priority falls to 61.

Increasing the value of **primove** makes the kernel less sensitive to process priorities.

Reducing the value of **primove** produces fairer scheduling for all processes but increases the amount of kernel overhead that is needed to manipulate the run queue.

Controlling cache affinity — cache_affinity

By default, the scheduler does not gives preference to a process that last executed on a CPU. The advantage of giving preference to these processes is to improve the hit rate on the level-one (L1) cache and L2 caches. As a consequence, the hardware is less likely to have to reload the caches from memory, an action that could slow down the processor. It also means that the process selected to run does not necessarily have the highest priority.

Cache affinity behavior is controlled by the value of the variable **cache_affinity**. If the value of **cache_affinity** is changed to 1, the kernel gives preference to processes which previously ran on a CPU Valid data and text is more likely to be found in the caches for small processes. If your system tends to run large processes leave **cache_affinity** set to 0.

Controlling process preemption — preemptive

On multiprocessor systems, the scheduler looks for a CPU on which to run a process when that process becomes runnable, or when its time slice has expired. The scheduler first looks for an idle CPU. If it cannot find an idle CPU, it next considers preempting the process on the current CPU if it has a lower priority; it is quicker to preempt the current process as this does not require an interprocessor interrupt. With some application mixes, however, this can increase the number of context switches. For example, when a database server wakes a client, it may be more efficient, in terms of system resources, for the server to continue to run for a period of time after that wakeup.

To prevent the scheduler from preempting the current processor, change the value of **preemptive** to 0.

Load balancing — loadbalance

On multiprocessor systems, the default behavior of the scheduler is to run the highest priority jobs on each of the processors. For example, when a process is woken after a disk transfer completes, the scheduler checks if any CPU is running a process with a lower priority. If so, the processor is instructed to reschedule and run the newly woken process. This load balancing feature is also used when a process is taken off a CPU; it is possible that the process has a higher priority than one on another CPU.

If you change the value of **loadbalance** to 0, the scheduler no longer looks for lower priority processes on other CPUs. This reduces the probability that a process will be preempted. On a system that is performing a reasonable amount of I/O requests, this should reduce the number of context switches and interprocessor interrupts. This provides more processor cycles for executing user applications and should increase overall performance. Processors which are idle can still be selected so idle time is minimized. This adjustment is likely to improve performance where context switching frequency is high, or on database servers where user processes should not be disturbed once they are running. If the system is already spending a significant amount of time idle, it is unlikely that this adjustment will improve performance.

Identifying CPU-bound systems

A system is CPU bound (has a CPU bottleneck) if the processor cannot execute fast enough to keep the number of processes on the run queue consistently low. To determine if a system is CPU bound, run **sar -u** (or **cpusar -u** for each processor on a system with an SCO SMP License) and examine the `%idle` value.

If `%idle` is consistently less than 5% (for all CPUs) on a heavily loaded database server system, then the system may be lacking in processing power. On a heavily loaded system with many logged-in users, a `%idle` value that is persistently less than 20% suggests that the system not be able to cope with a much larger load. Examination of the number of processes on the run queue shows whether there is an unacceptable buildup of runnable processes. If processes are not building up on the run queue, a low idle time need not indicate an immediate problem provided that the other subsystems (memory and I/O) can cope with the demands placed upon them.

Run queue activity can be considered heavy if **sar -q** (**mpsar -q** for SMP) reports that `runq-sz` is consistently greater than 2 (and `%runocc` is greater than 90% for SMP). If low `%idle` values are combined with heavy run queue activity then the system is CPU bound.

If low `%idle` values are combined with low or non-existent run queue activity, it is possible that the system is running CPU-intensive processes. This in itself is not a problem unless an increase in the number of executing processes causes a buildup of numbers of processes on the run queue.

If `%wio` values are consistently high (greater than 15%), this is more likely to indicate a potential I/O bottleneck than a problem with CPU resources. See Chapter 5, "Tuning I/O resources" (page 71) for more information on identifying I/O bottlenecks.

High values of `%wio` may also be seen if the system is swapping and paging. Memory shortages can also lead to a disk I/O bottleneck because the system spends so much time moving processes and pages between memory and swap areas on disk. If the value of `%sys` is high relative to `%usr`, and `%idle` is close to zero, this could indicate that the kernel is consuming a large amount of CPU time running the swapping and page stealing daemons(**sched** and **vhand**). These daemons are part of the kernel and cannot be context switched; this may lead to several processes being stuck on the run queue waiting to run. For details of how to identify and tune memory-bound systems, see Chapter 4, "Tuning memory resources" (page 41) and "Tuning memory-bound systems" (page 52).

The following table summarizes the commands that you can use to determine if a system is CPU bound:

Table 3-1 Identifying a CPU-bound system

Command	Field	Description
sar -u	%idle	percentage of time that the CPU was idle
mpsar -u	%idle	average percentage of time all CPUs are idle (SMP only)
cpusar -u	%idle	percentage of time the specified CPU was idle (SMP only)
[mp]sar -q	%runocc	percentage of time the run queue is occupied
	runq-sz	number of processes on the run queue

See "Tuning CPU-bound systems" (page 40) for a discussion of how to tune CPU-bound systems.

Tuning CPU-bound systems

If it has been determined that the system is CPU bound, there are a number of things that can be done:

- If possible, consider rescheduling the existing job load on your system. If many large jobs are being run at once, rescheduling them to run at different times may improve performance. You should also check the system's - **crontab**(C) files to see if any jobs running at peak times can be scheduled to run at other times.
- If possible, tune the applications so that they use require less CPU power. Consider replacing non-critical applications with ones that require a less powerful system.
- If you have evidence that the system is I/O bound serving interrupts from non-intelligent serial cards, replacing these with intelligent serial cards will offload some of the I/O burden from the CPUs. See "Serial device resources" (page 108) for more details.
- Check if the hard disk controllers in the system are capable of using DMA to transfer data to and from memory. If the CPU has to perform programmed I/O on behalf of the controller, this can limit its performance.
- It is possible that because of a lack of free memory the system is swapping, which could result in a considerable portion of the CPU resources being used to transfer processes back and forth between the disk and memory. To determine if this is the case see the section Chapter 4, "Tuning memory resources" (page 41).
- Upgrade to a faster CPU or CPUs.
- Upgrade to a multiprocessor system from a single processor system. This will help if there are runnable jobs on the run queue or the applications being run are multithreaded.
- Add one or more CPUs to a multiprocessor system.
- Purchase an additional system and divide your processing requirements between it and your current system.

Chapter 4

Tuning memory resources

The SCO OpenServer system is a virtual memory operating system. Virtual memory is implemented using various physical resources:

- Physical memory as RAM chips; sometimes referred to as primary, main, or core memory.
- Program text (machine code instructions) as files within filesystems on disk or ramdisks.
- Swap space consisting of one or more disk divisions or swap files within filesystems dedicated to this purpose. The individual pieces of swap space are known as swap areas. Swap space is also referred to as secondary memory.

Depending on the system hardware, there may also be physical cache memory on the CPU chip itself (level-one (L1) cache), or on the computer's motherboard (level-two (L2) cache), and on peripheral hardware controller cards. If recently accessed data (or, for some L1 and L2 caches, machine instructions) exists in this memory, it can be accessed immediately rather than having to retrieve it from more distant memory.

Write-through caches store data read from memory or a peripheral device; they ensure that data is written synchronously to memory or a physical device before allowing the CPU to continue. Write-back caches retain both read and written data and do not require the CPU to synchronize with data being written.

NOTE Most L2 caches work with a limited amount of main memory. Adding more RAM than the cache can handle may actually make the machine slower. For some machines with a 64KB L2 cache, this only covers the first 16MB of physical memory. See the documentation provided with your computer or motherboard hardware for more details.

Physical memory

Physical memory on the system is divided between the area occupied by the kernel and the area available to user processes. Whenever the system is rebooted the size of these areas, as well as the total amount of physical memory, is logged in the file */usr/adm/messages* under the heading **mem:**, for example:

```
mem: total = 32384k, kernel = 4484k, user = 27900k
```

This shows a system with 32MB of physical memory; the kernel is using just over 4MB of this memory with the remainder being available for user processes.

Physical memory is divided into equal-sized (4KB) pieces known as pages. When a process starts to run, the first 4KB of the program's text (executable machine instructions) is copied into a page of memory. Each subsequent portion of memory that a process requires is assigned an additional page.

When a process terminates, its pages are returned to the free list of unused pages.

Physical memory is continually used in this way unless the number of running processes require more pages of memory than currently exist on the system. In this case the system must redistribute the available memory by either paging out or swapping.

Virtual memory

The operating system uses virtual memory to manage the memory requirements of its processes by combining physical memory with secondary memory (swap space) on disk. The swap area is usually located on a local disk drive. Diskless systems use a page server to maintain their swap areas on its local disk.

The amount of swap space is usually configured to be larger than physical memory; the sum of physical memory and swap space defines the total virtual memory that is available to the system.

Having swap space on disk means that the CPU's access to it is very much slower than to physical memory. Conventionally, the swap area uses an entire division on a hard disk. It is also possible to configure a regular file from within a filesystem for use as swap. Although this is intended for use by diskless workstations, a server can also increase its swap area in this way.

The swap area is used as an additional memory resource for processes which are too large for the available physical user memory. In this way, it is possible to run a process whose entire executable image will not fit into physical memory. However, a process does not have to be completely loaded into physical memory to run, and in most cases is unlikely to be completely loaded anyway.

The virtual address space of a process is divided into separate areas known as regions that it uses to hold its text, data, and stack pages. When a program is loaded, its data region consists of data pages that were initialized when the program was compiled. If the program creates uninitialized data (often known as bss for historical reasons) the kernel adds more pages of memory to the process' data region.

If the operating system is running low on physical memory, it can start to write pages of physical memory belonging to processes out to the swap area. See "Paging" (page 44) and "Swapping" (page 47) for more details.

Figure 4-1 (page 44) illustrates how a process' virtual memory might correspond to what exists in physical memory, on swap, and in the filesystem. The u-area of a process consists of two 4KB pages (displayed here as U_1 and U_2) of virtual memory that contain information about the process needed by the system when the process is running. In this example, these pages are shown existing in physical memory. The data pages, D_3 and D_4, are shown as being paged out to the swap area on disk. The text page, T_4, has also been paged out but it is not written to the swap area as it exists in the filesystem. Those pages which have not yet been accessed by the process (D_5, T_2, and T_5) do not occupy any resources in physical memory or in the swap area.

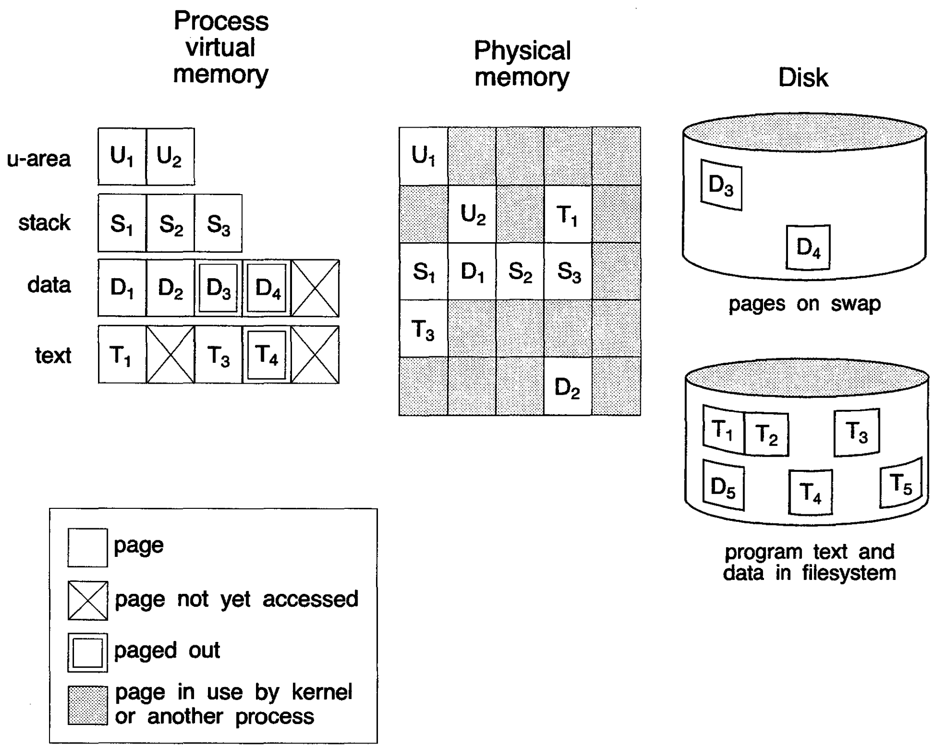


Figure 4-1 How the virtual memory of a process relates to physical memory and disk

Paging

Paging is the process by which the contents of physical memory are moved both to and from swap areas and filesystems. Paging out releases infrequently accessed memory for use by other processes. Paging in brings data or text into memory that a process needs to continue running.

At every clock tick, the kernel checks to see if the number of pages on the free list is below the number specified by the **GPGSLO** kernel parameter. If so, **vhand**, the page stealing daemon, becomes active and begins copying modified data and stack pages to the swap area, starting with least recently used pages. Each page placed on the free list then becomes available for use by other processes. Pages written out to swap must be read back into physical memory when the process needs them again.

Program text and unmodified data pages are added to the free list. Copying them to swap serves no purpose because they can be read directly from a filesystem.

vhand remains active until the number of pages on the free list rises above the number specified by the **GPGSHI** kernel parameter. Both **GPGSLO** and **GPGSHI** are tunable; guidelines on setting values other than the system defaults are given in "Tuning memory-bound systems" (page 52). Figure 4-2 (this page) illustrates **vhand** being run to make **GPGSHI** pages of memory available.

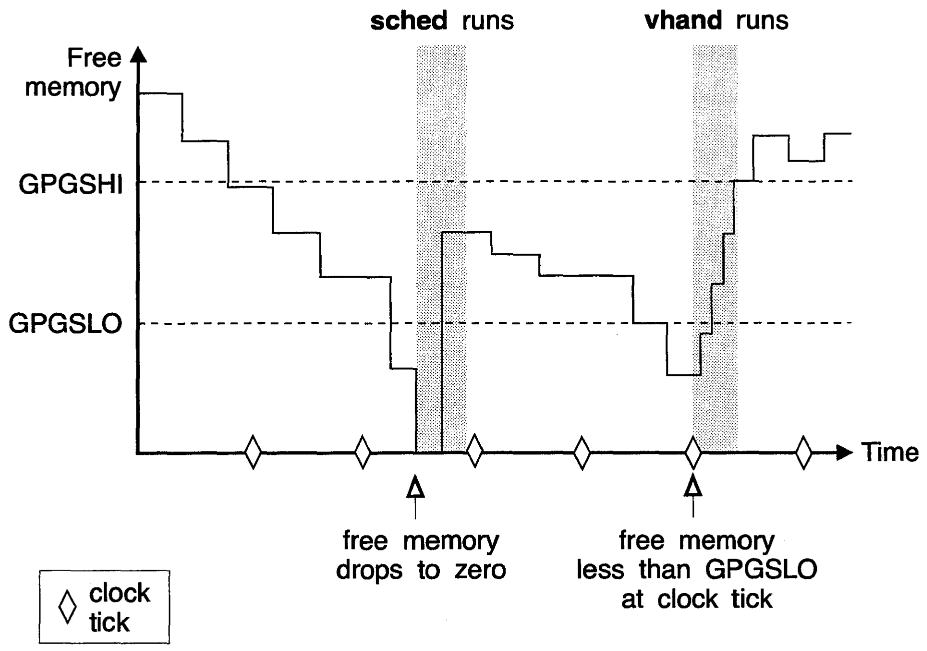


Figure 4-2 When the swapper (sched) and the page stealing (vhand) daemons run to release memory

When a process terminates, its pages are added to the head of the free list. However, the kernel adds pages from a runnable process to the tail of the free list in case they are required subsequently. Processes that are expanding in memory are allocated pages from the head of the free list. In this way, the free list serves both as a source of available pages and a cache of reclaimable pages.

Page faults

A page fault is a hardware event that occurs when a user process tries to access a virtual address that has no physical memory currently assigned to it. The page fault handler attempts to provide a physical page and to load it with the appropriate data. If successful, the process that received the page fault resumes at the instruction that faulted as though the page of memory was always there. Page faults are the opposite of page outs as they involve loading pages into memory rather than copying pages out of memory.

When required, the correct page can be acquired by the page fault handler in several ways:

- The page may be a copy-on-write (COW) page created from a writable data (initialized or uninitialized) or stack page by the forking of a process. Memory is only allocated to such pages when a process first writes to them.
- The page may be on the free list: that is, still in memory even though its contents were paged out. If so, it still contains the required data. The page fault handler unlinks the associated page from the free list, marks it as valid, and resumes the process. Reclaiming a page in this way does not require a disk transfer.
- If the page is paged out and the original page has been reallocated then the page fault handler allocates a page from the head of the free list. If necessary, the process is put to sleep and a disk request is scheduled to retrieve the page's data or text from disk; this inevitably causes a context switch away from the faulting process. Later, after the page has been read, the kernel updates the process' page table, marks the page as being valid, and puts the process on the run queue.
- If a previously unreferenced page is required for use by the process stack or uninitialized data, a fresh page is allocated and filled with zeroes.

Pages belonging to the kernel are usually not paged in or out. Kernel text and most of its associated data structures reside permanently in physical memory while the operating system is running. The only kernel memory that can be swapped or paged are the structures which store internal information about the state of individual processes. These include a process' u-area, Task State Segment (TSS), and page tables.

Swapping

As well as paging, the operating system uses the swapper daemon, **sched**, to free assigned memory pages by copying modified process pages to the swap area.

The normal state of **sched** is to be asleep; when woken it will swap processes out or in or both, depending on the needs of the system. **sched** becomes active and swaps processes out when the amount of free memory in the system drops to zero. Figure 4-2 (page 45) illustrates **sched** being run to make more pages of memory available.

sched checks for processes that are either waiting for an event to complete or that have been stopped by a signal. If a process has been in either of the two states for more than 2 seconds, **sched** moves it from memory to the swap area, and adds the reclaimed pages to the free list. If enough memory is still not available, and **sched** fails to find another process that is sleeping or stopped, it attempts to swap out processes that have been on the run queue or waiting for memory longer than two seconds. Figure 4-3 (this page) shows how **sched** can swap out processes in a system.

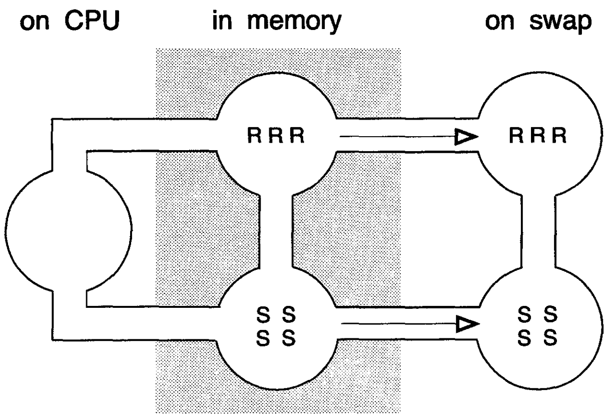


Figure 4-3 The swapper daemon swaps out processes

sched becomes active if **vhand** cannot make enough pages available to satisfy the demands on physical memory. It is generally preferable for a system to page rather than swap because paging creates less disk I/O, and CPU overhead. By increasing the values of **GPGSLO** and **GPGSHI**, a system will start to page memory earlier and is less likely to start swapping suddenly. However, the demands on memory may be such that swapping is inevitable at some stage. See "Tuning memory-bound systems" (page 52) for more details of how to tune this behavior.

sched does not only swap out processes. As is the case for paging activity, swapping between physical memory and the swap area occurs in both directions. **sched** will swap a process in if it has a high enough priority to run.

The operating system only allocates swap space to a process when it swaps out the process. When it swaps a process back in to continue running, it retains the swap space allocation for possible future use. It only frees this when the process completes.

Viewing physical memory usage

The number of pages of physical memory on the free list is shown under `freemem` when using **sar -r** (or **mpsar -r** for SMP):

```
23:59:44 freemem freeswp
23:59:49     390   88712
23:59:54     335   88720
23:59:59     321   88416

Average      349   88732
```

Viewing swap space usage

Information about the usage of the swap areas on your system can be seen using the **swap -l** command:

```
path               dev   swaplo blocks   free
/dev/swap          1,41       0 128000  88712
```

`blocks` shows the total size of a swap area in 512-byte disk blocks. `free` shows the number of blocks on the free list.

The number of unused 512-byte disk blocks in the swap area can be also be seen under the `freeswp` heading using **sar -r** (or **mpsar -r** for SMP):

```
23:59:44 freemem freeswp
23:59:49     390   88712
23:59:54     335   88720
23:59:59     321   88416

Average      349   88732
```

You may find that your system runs out of swap space before `freeswp` drops to zero. This is because the operating system requires that enough swap space be available for the grown data and stack regions of all processes, not just those process' pages that have actually been swapped or paged out.

To discover the number of 4KB pages of swappable virtual memory that the operating system calculates is available, use the following command within an interactive **crash**(ADM) session:

od -d availsmem

For more information about how you can use **crash** to investigate memory usage, see "Monitoring memory allocation" in the *System Administration Guide.*

Viewing swapping and paging activity

The impact of swapping and paging out activity on disk activity can be seen with **sar -w** (or **mpsar -w** for SMP):

```
23:59:44 swpin/s bswin/s swpot/s bswot/s pswch/s
23:59:49    0.00     0.0    0.00     0.0     280
23:59:54    0.00     0.0    0.00     0.0     244
23:59:59    0.00     0.0    0.02     0.3     203

Average     0.00     0.0    0.01     0.1     242
```

The column of interest is `bswot/s`, the average number of pages swapped out per second during the sampling interval. The ratios of pages to transfer requests per second (`bswin/s` to `swpin/s`, and `bswot/s` to `swpot/s`) show how many pages could be moved between memory and disk per average disk transfer.

Swapping activity is also indicated by the size of the swap queue. The swap queue is a queue of runnable processes held in the swap area. Swapped-out processes are queued in an order determined by how long they have been swapped out. The process that has been swapped out for the longest period of time will be the first to be swapped in, as long as it is ready to run.

The values of `swpq-sz` and `%swpocc` displayed by **sar -q** (or **mpsar -q** for SMP) indicate the number of runnable processes on swap, and the percentage of time that the swap areas were occupied by runnable processes:

```
23:59:44 runq-sz %runocc swpq-sz %swpocc
23:59:49     1.7      98     1.5      36
23:59:54     1.0      63     1.0      31
23:59:59     1.0      58     1.0      49

Average      1.3      74     1.2      39
```

You can see paging activity using **sar -p** (or **mpsar -p** for SMP):

```
23:59:44  vflt/s  pflt/s pgfil/s  rclm/s
23:59:49    9.72    2.03    0.00    0.00
23:59:54    0.37    0.18    0.00    0.00
23:59:59    0.00    0.00    0.00    0.00

Average     3.88    0.84    0.00    0.00
```

`vflt/s` is the number of valid pages referenced per second that were not found in physical memory. A referenced page that was previously paged out to swap, or exists as a text or data page in the filesystem is loaded from disk.

`pflt/s` is the number of pages per second required by new processes for their data (both initialized and uninitialized) and stack. New processes created by the **fork**(S) system call do not acquire their own data and stack pages until they or their parent process attempts to write to them. This number also includes the number of illegal attempts to access memory per second.

`pgfil/s` is the number of page references satisfied by reading text and data pages from filesystems.

`rclm/s` is the number of pages per second that the page stealer and swapper daemons (**vhand** and **sched**) have reclaimed and added to the free list. This is an upper limit to the number of pages that are written to the swap area per second in the sampling interval (text and unchanged data are not written to swap as they can be read from the filesystem).

The important column is `rclm/s` which shows if your system is swapping or paging. To confirm this, look at the amount of free memory, `freemem` reported by **sar -r** (or **mpsar -r** for SMP). It is likely that **vhand** has been running if `freemem` has been keeping close to the value of **GPGSHI**.

Another indicator of swapping or paging activity is the cumulative CPU usage shown in the `TIME` column by the **ps -l -p 0,2** command. (**sched** and **vhand** have PIDs 0 and 2 respectively.)

Identifying memory-bound systems

A system is memory bound or has a memory bottleneck if memory on the system is insufficient to keep the pages of all runnable or sleeping processes in physical memory. To determine if your system is memory bound run **sar -p** (or **mpsar -p** for SMP) and look at the value of `rclm/s`. If this column is consistently zero, the system does not have memory problems.

If the number of free pages shown by `freemem` (using **sar -r**) is consistently near or below the value defined for the **GPGSHI** kernel parameter, then the page daemon is probably active. Confirm this by examining the `TIME` usage reported by the **ps -el -p 2** command for the page handling daemon, **vhand**; if this value is increasing, **vhand** is running. If the value of `freemem` is consistently low, this indicates that most of memory is regularly in use. If this activity is accompanied by swapping then the system has memory problems.

A more severe indication of insufficient memory is swapping activity. To check swapping activity use the **sar -w** command. If `bswot/s` shows values that are consistently non-zero, this indicates that the system has a memory problem and is swapping.

Similarly, if the swap queue shows activity, then there are processes being swapped out to make memory available. If the entries for `swpq-sz` and `%swpocc` when running **sar -q** remain blank then no processes are being swapped and memory is sufficient. If `swpq-sz` is greater than zero, then the system has experienced swapping, and there are runnable processes on swap.

In some cases, it may do no harm for a few infrequently-used processes to be swapped out, such as the **getty** processes that are used for the console multiscreens. It does indicate, however, that your system is near to being chronically short of memory if it has to swap whole processes out to free up memory. See "Paging" (page 44) and "Swapping" (page 47) for a description of how the system frees pages of physical memory for use.

Finally, if either **vhand** or **sched** are showing a lot of processing time, indicated by the value of `TIME` when running **ps -l**, then this would suggest a lot of paging and swapping activity.

The following table is a summary of the commands that can be used to determine if a system is memory bound:

Table 4-1 Identifying a memory-bound system

Command	Field	Description
[mp]sar -p	rclm/s	pages added to the free list per second
[mp]sar -q	%swpocc	percentage of time the swap queue is occupied
	swpq-sz	number of processes on the swap queue
[mp]sar -r	freemem	available user memory (4KB pages)
	freeswp	available swap space (512-byte blocks)
[mp]sar -w	bswot/s	average number of pages written to swap per second
swap -l	blocks	total size of the swap area (512-byte blocks)
	free	available swap space (512-byte blocks)
ps -l -p 0	TIME	CPU time used by the swapper, **sched**
ps -l -p 2	TIME	CPU time used by the page stealer, **vhand**

Tuning memory-bound systems

If the system is found to be memory bound there are a number of things that can be done. The most obvious and that which will probably bring the most benefit is to add more physical memory to your system. If this is not possible then a number of alternatives exist:

- Determine if a number of memory intensive processes are being run simultaneously. This can be done by running **ps -el**. The SZ value gives the virtual memory (swappable) size of the process's stack and data (both initialized and uninitialized) regions in 1KB units. If many memory intensive processes are being run simultaneously then rescheduling these jobs to run at alternative times will redistribute the use of memory. To see if any memory-intensive jobs running at peak times can be rescheduled, you should also check the system's **crontab**(C) files.

  It is also possible that some applications programs may have a memory leak and are continuously increasing their size in virtual memory. If you suspect that an application has a memory leak, you should restart the program before its usage of virtual memory starts to make the system swap or page out. You may notice this problem with server processes which run continuously for several weeks.

- If you are writing your own applications, use static shared or dynamic linked libraries to make more efficient use of memory.

You should also ensure that the applications do not have a memory leak.

- Reduce the size of the kernel to free more user memory. This can be done by reducing the size of the buffer cache as discussed in "Increasing memory by reducing the buffer cache size" (page 54).

NOTE If you increase the amount of physical memory in your system to 32MB or more, run the **iddeftune**(ADM) command to increase the values of certain kernel parameters.

If the system appears to be constantly paging, this may be the result of the values of **GPGSLO** and **GPGSHI** being too high. This causes **vhand** to page out pages while a large number of free pages still exist. Lowering these values could delay the onset of paging but might cause the system to begin swapping out whole processes instead if memory drops to zero. If no compromise can be met then the system needs more memory.

If the number of pages on the free list falls below **GPGSLO**, **vhand** begins moving pages out of memory. **vhand** continues to do this until the number of pages on the free list reaches **GPGSHI**.

If the difference between **GPGSLO** and **GPGSHI** is too great, this may cause an I/O bottleneck while the kernel attempts to write the contents of many dirty pages to disk.

If the values of **GPGSLO** and **GPGSHI** are close together, **vhand** will be active for a shorter period of time but more often. If **vhand** is constantly active, its usage of the CPU may degrade performance.

Reducing disk activity caused by swapping and paging

To estimate the impact that paging in from filesystems has on disk activity, multiply the value of `pgfil/s` reported by **sar -p** (or **mpsar -p** for SMP) by 8 to convert from 4KB pages to the number of 512-byte disk blocks read or written per second:

Disk activity due to paging in = 8 * `pgfil/s`

The amount of disk activity caused by swapping and paging out to the swap areas can be estimated from the values of `bswin/s` and `bswot/s` reported by **sar -w** (or **mpsar -w** for SMP):

Disk activity due to swapping = 8 * (`bswin/s` + `bswot/s`)

These values can be compared with the total number number of blocks per second being transferred to and from the disks containing the filesystems and swap areas. Use **sar -d** (or **mpsar -p** for SMP) to report the number of blocks transferred per second (`blks/s`). See "Viewing disk and other block I/O activity" (page 89) for more information about monitoring hard disk activity.

If a high proportion of disk activity is caused by paging in, and this is causing a disk bottleneck, see "Tuning disk I/O-bound systems" (page 92) for suggested ways to cure this.

If swapping and paging out is causing a disk bottleneck, you could create swap areas on several disks to relieve the load on a single disk. If possible, you should try to reduce the memory shortage.

Increasing memory by reducing the buffer cache size

WARNING Reducing the size of the buffer cache to increase the amount of available memory may degrade the system's disk I/O performance.

If **sar -b** (or **mpsar -b** for SMP) shows that the `%rcache` and `%wcache` hit rates are consistently high, memory may be regained for use by user processes by reducing the size of the buffer cache. (See "How the buffer cache works" (page 73) for a description of its operation.)

It is not possible to recommend minimum values for the read and write hit rates. It depends on the amount of extra disk I/O that will be generated and the performance characteristics of the system's disks. Reducing the buffer cache hit rates also means that more processes have to wait for I/O to complete. This increases the total time that processes take to execute and it will also increase the amount of context switching on the system.

You may, for example, decide that you can tolerate reducing current hit rate values of `%rcache` from 95% to 90% and `%wcache` from 65% to 60% provided that your system's disks can cope with the increased demand and also that any deterioration in the performance of applications is not noticeable.

The current number of buffers in use is controlled by the value of the kernel parameter **NBUF**. If this is set to 0, the system determines the number automatically at system startup. The number of buffers is displayed in the startup messages and recorded in the file */usr/adm/messages*.

Adjusting the value of **NBUF** should be done as an iterative process in conjunction with running **sar -b** to look at the buffer cache hit rates. If the number of writes (`bwrit`) is low compared with the number of reads (`bread`), less significance should be attached to the `%wcache` hit rate. You can monitor any resulting increase in disk activity using the **-d** option to **sar** (or **mpsar**) as described in "Viewing disk and other block I/O activity" (page 89).

NOTE If you change the value of **NBUF**, you should also modify the value of the **NHBUF** parameter to an appropriate value. See "Increasing disk I/O throughput by increasing the buffer cache size" (page 75) for more information.

The following table summarizes the commands that you can use to view buffer cache activity:

Table 4-2 Viewing buffer cache activity

Command	Field	Description
[mp]sar -b	bread	number of 1KB blocks read per second from block devices
	lread	number of 1KB blocks read per second from system buffers
	%rcache	percent of disk blocks found in the buffer cache when reading
	bwrit	number of 1KB blocks written per second to block devices from the buffer cache
	lwrit	number of 1KB blocks written per second to system buffers
	%wcache	percent of disk blocks found in the buffer cache when writing

Investigating memory usage by system tables

NOTE In previous releases, you could specify the size of various static data structures in the kernel such as the process, in-core inode, open file, and lock tables. In this release, the operating system dynamically allocates memory to system tables. In this way, they grow over time to accommodate maximum demand. You can specify the maximum size to which a table can grow (for example, the kernel parameter **MAX_PROC** specifies the maximum size of the process table). However, this does not give you a performance gain and may limit your system's functionality if you specify too small a value.

System table usage can be seen with **sar -v** (or **mpsar -v** for SMP):

```
16:10:31 proc-sz ov inod-sz ov file-sz ov lock-sz
16:10:37  61/127  0 160/250  0 177/291  0    2/10
16:10:43  61/127  0 156/250  0 167/291  0    2/10
16:10:48  61/127  0 154/250  0 159/291  0    2/10
```

In each of the size columns, the first number signifies the number of entries currently used in the table and the second signifies the size to which the table has grown since the system was last booted. The table sizes should be monitored over a period of time to determine the upper limits for their grown sizes.

You can also determine the current grown size and the maximum possible sizes of these tables using the **getconf**(C) command.

The following table is a summary of the fields displayed by the **sar -v** and **mpsar -v** commands:

Table 4-3 Viewing the size of system tables

Command	Field	Description
[mp]sar -v	proc-sz	used and grown size of the process table
	inod-sz	used and grown size of the inode table
	file-sz	used and grown size of the file table
	lock-sz	used and grown size of the lock table

For more details on the dynamic kernel tables, see "Table limits" (page 207).

Using graphical clients on low memory systems

If your SCO OpenServer Desktop System is short of memory, you can release memory for use by simplifying the Desktop environment.

If your system is very memory bound, consider making the following changes. These are given in order, from the most to the least effective in releasing memory for use:

- Do not use the Desktop client, **xdt3**(XC), if you do not need to click and drag icons to perform tasks. You can configure the **pmwm** Root menu to list clients that you often use, or you can start clients from the **xterm** or **scoterm** command line. See the *Graphical Environment Guide* for more information about adding clients to or deleting clients from the Root menu.
- Disable **scologin**(XC) from running on the console if a machine is the host for several X terminals, and you want **scologin** to manage only their displays. Disabling the X server from running saves several processes and their associated memory. See the *Graphical Environment Guide* for more information about enabling and disabling **scologin**.
- Run fewer graphical X clients. This reduces overall memory requirements.
- For non-critical tasks where you need only basic functionality, use non-graphical applications or X clients that require less memory.

Tuning X server performance

The X server program controls what is displayed on the Desktop screen, and handles input from the keyboard and mouse. X clients are application programs which either run on the same machine as the server, or on a remote machine over a network connection. The server displays images on the screen on behalf of the clients and it transmits mouse and keyboard input events to the clients on behalf of the user.

The X server program has several options which affect performance. These options can be added to the invocation of **X**(X) or **Xsco**(X) in the file */usr/lib/X11/scologin/Xservers*:

-bs If specified, disables support for backing store.

Backing store is a buffer used to store the entire contents of a window. It allows the X server to redraw the entire window rather than requiring the application (X client) to do this. Disabling backing store can save a significant amount of memory but redrawing windows will cause clients to expend more CPU time. This will impact the CPU usage of the machine on which the client is running. If the client is running remotely, it may also generate significantly more network traffic while it redraws the window. This can also cause a noticeable delay while it does this.

-nice ***value***

Specifies the X server's **nice** ***value*** in the range 0 to 39. The default **nice** ***value*** is 0 which gives the most responsive performance by the mouse or other pointing device. See "Calculation of process priorities" (page 28) for a discussion of **nice**.

-nompxlock

If specified, allows the X server to run on any CPU on multiprocessor systems where the video drivers are multithreaded. This can enhance performance on systems with an SCO SMP License by reducing the load on the base processor.

-su If specified, disables support for save-unders.

Save-unders are temporary buffers that store the contents of windows that other windows, menus, pop-ups, and so on may obscure. Disabling save-unders requires the clients to expend more CPU time redrawing portions of windows, and adds to network load for remote clients.

Kernel parameters that affect the X Window System

Although the following kernel parameters do not directly affect performance, they are important for the correct operation of the X Window System™, the Desktop, and X terminals. You may be unable to start an X client if you do not enough of these resources configured. See Appendix B, "Configuring kernel parameters" (page 185) for details of how to change the value of kernel parameters.

MAX_PROC
: Set the value of this parameter to 0 to allow the process table to grow dynamically.

MAXUP
: Limits the number of processes that the system will run on behalf of a user. Each window requires at least one process. Note that local applications running in a window may start several additional processes.

NOFILES
: Specifies the maximum number of files that a process (including the X server) can have open simultaneously. This limits the number of X clients that can be started because the X server opens a file descriptor to each client. In addition, the X server requires about 10 file descriptors in order to read fonts, color maps, and so on.

NSPTTYS
: Allow at least as many pseudo terminals as the number of windows that will be opened for use on the console, by X terminals, and X clients. If necessary, you can change the number of pseudo terminals configured for use using the **mkdev ptty** command.

NSTREAM
: Allow at least four stream heads for the X server plus four for each client connection. **NSTREAM** should be approximately four times the value of **NSPTTYS**.

NSTRPAGES
: This parameter controls the maximum number of 4KB pages of memory that can be dynamically allocated for use by STREAMS messages. Allow at least 125 pages plus 125 pages for each X terminal that is supported.

NUMSP
: Two stream pipes are needed for the X server plus two for each local X client. **NUMSP** should be approximately twice the value of **NSPTTYS**.

For more information about tuning STREAMS resources, see "STREAMS resources" (page 123).

Case study: memory-bound workstation

In this example, a user is given a workstation running the SCO OpenServer Desktop System to use. The machine has had one previous user who may have made undocumented changes to the system's configuration. The workstation's new owner is given the *root* password and is made responsible for its day-to-day administration. The performance of the machine seems generally adequate although it does become noticeably slower when several different applications are started at the same time.

System configuration

The configuration of the system is:

- Uniprocessor 80486DX running at 33MHz.
- ISA bus.
- 24MB of RAM.
- 48MB of swap space.
- One 434MB IDE disk drive
- One 16-bit Ethernet card with a 16KB buffer.
- VESA bus graphics adapter with 1MB of Dynamic Random Access Memory (DRAM).
- SVGA monitor configured to run at a display resolution of 1024x768 with 256 colors.

The user's home area and applications are accessed via NFS-mounted filesystems on a file server maintained by the company's Information Services department. The local area network is lightly loaded. There are occasional bursts of NFS traffic when users access remote files. There are no funds available for upgrading the workstation.

Defining a performance goal

The user wishes to become familiar with how their workstation has been set up and to improve its performance if possible. They have only a few hours available to perform this task.

Collecting data

The user collects the following settings for kernel parameters by running the **Hardware/Kernel Manager**:

GPGSLO	200 — the number of memory pages available when the page stealing daemon, **vhand**, becomes active to release pages for use.
GPGSHI	300 — the target number of pages for **vhand** to release for use.
MAXUP	25 processes per user are allowed.
NBUF	3000 1KB buffers are requested at system startup.
NHBUF	256 buffer hash queues are reserved.
NSTREAM	256 stream heads are configured.
NSTRPAGES	maximum 500 4KB pages of memory can be dynamically allocated for use by STREAMS message buffers.

The user examines the file */usr/adm/messages* and notes the following information:

- The kernel requires 7MB of memory.
- The buffer cache occupies 3MB of kernel memory.
- There are 17MB of memory available for user processes.

In addition the user notes the following facts from various configuration files on the system:

- The TCP/IP interface definition for the Ethernet card is defined to use back-to-back packets and full frames in the file */etc/tcp*.
- Four NFS **biod** daemons are configured to run in the file */etc/nfs*.

Next, the user starts up all the usual applications that they run on the desktop — **scomail**, **xclock**, editing files in two windows, browsing WWW using Mosaic, viewing **scohelp**, running two sessions on remote machines, and running a word processor and spreadsheet. They are unable to start any more processes than this and a message is displayed to this effect. They then switch to another multiscreen, log in as *root*, and start to record **sar** data at 30-second intervals to a temporary file:

```
sar -o /tmp/sar_op 30 120
```

They then continue to use the system for an hour before examining the results.

Formulating a hypothesis

There are several things that immediately strike the user as less than optimal about this system configuration:

- Only 25 processes are available for each user even though this system only has one user. This is probably the reason why only a limited number of windows could be started.
- The number of hash queues configured (**NHBUF**) is much lower than the number of system buffers (**NBUF**). There is approximately 1 hash queue for every 12 buffers; this means than each queue will contain 12 buffers on average. On a single processor system, the system will automatically allocate at least one hash queue for every two buffers if **NHBUF** is set to 0.
- The user suspects that too much memory is allocated to buffers that could more usefully be allocated to user processes. Most disk access is remote and will cause most loading on the NFS file server. NFS remote writes to files are write through (synchronous) and not cached on either the client or the server machines. Remote reads are cached locally and have unknown requirements. As most of the applications that the user runs are idle while not being used, it is unlikely that the STREAMS subsystem is severely loaded.

Getting more specifics

The **sar -u** report is extracted from the file */tmp/sar_op*:

sar -u -f /tmp/sar_op

This report shows the system's usage of the CPU:

```
09:00:00    %usr    %sys    %wio   %idle
. . .
09:15:30       6       5       1      88
09:15:00       5       4       1      90
09:16:30       6       3       0      91
. . .
```

It is apparent that the system spends most of its time idle with plenty of spare processing capacity. The low waiting on I/O (`%wio`) figures do not indicate any bottleneck in the I/O subsystems.

Memory investigation

The user next runs **sar -r** to examine the system's usage of memory:

```
09:05:00 freemem freeswp
...
09:15:30     314   73166
09:16:00     302   72902
09:16:30     308   72888
...
```

This shows that there is plenty of swap space (`freeswp`) but that the system is running low on physical memory (`freemem` is close to **GPGSHI**) so the page handling (**vhand**) and the swapper (**sched**) daemons may be active. (See Chapter 4, "Tuning memory resources" (page 41) for more information about the circumstances under which these daemons become active.)

The **sar -q** report shows that no runnable processes are swapped out (no value is displayed for `swpq-sz`):

```
09:05:00 runq-sz %runocc swpq-sz %swpocc
...
09:15:30     1.3       2
09:16:00     1.0       3
09:16:30     1.1       2
...
```

Running **sar -w**, the `swpot/s` field is greater than zero; this is evidence that the system is swapping out to the swap area:

```
09:05:00 swpin/s bswin/s swpot/s bswot/s pswch/s
...
09:15:30    0.05     0.2    0.01     0.2      72
09:16:00    0.07     0.5    0.02     0.4      55
09:16:30    0.03     0.2    0.01     0.3      43
...
```

The system does not appear to be very short of resources apart from memory for user processes. There is plenty of spare CPU capacity and no immediately apparent problem with I/O.

It is possible that a user process is grabbing too much memory for itself. In this instance, running the command **ps -el** shows that no process has a swappable virtual memory size (`SZ` field) greater than the `X` server, and most are much smaller. When tuning a system, it is always worth checking to see which processes are using most swappable memory (`SZ` field) and most time (`TIME` field).

The next step is to see if the amount of memory used by the buffer cache can be reduced.

I/O investigation

The user runs **sar -b** to investigate buffer cache hit rates:

```
09:05:00 bread/s lread/s %rcache bwrit/s lwrit/s %wcache pread/s pwrit/s
...
09:15:30      1      17      97       6      18      68       0       0
09:16:00      1      25      95       1       3      67       0       0
09:16:30      1      18      96       3       9      67       0       0
...
```

The hit rates are quite high at approximately 96% for reads and 67% for writes. The numbers of blocks being transferred is quite small. As most of the files being accessed are remote, local disk activity should be low apart from paging in of program text and data, and any paging out activity. This is investigated using **sar -d**:

```
09:05:00 device   %busy   avque   r+w/s   blks/s   avwait   avserv
...
09:15:30 wd-0      1.42    2.39    2.13     6.82     7.50    10.42

09:16:00 wd-0      2.03    2.97    1.37     3.64     7.00    13.78

09:16:30 wd-0      1.16    2.29    1.70     5.95     9.48    12.23

...
```

The disk appears not to be busy so it should be able to cope with a decreased cache hit rate. If memory is released by decreasing the size of the buffer cache, this may also lessen any paging out activity and so decrease disk activity.

STREAMS usage investigation

Finally, the user runs **netstat -m** to investigate how STREAMS is using memory:

```
streams allocation:
                              config  alloc   free    total    max   fail
streams                          256    113    143     6270    124      0
queues                           566    394    172    16891    404      0
mblks                            271    102    169   179326    283      0
buffer headers                   442    391     51   155964    475      0
class  1,     64 bytes          1288    276      8    50289   1288      0
class  2,    128 bytes           796    171     25    18668    796      0
class  3,    256 bytes           364     50     14     9174    364      0
class  4,    512 bytes           132     12     20     3334    132      0
class  5,   1024 bytes            54      5      9     1904     54      0
class  6,   2048 bytes            84     62     22     1622     84      0
class  7,   4096 bytes             8      8      0      293      8      0
class  8,   8192 bytes             1      0      1      113      1      0
class  9,  16384 bytes             1      0      1       21      1      0
class 10,  32768 bytes             0      0      0        0      0      0
class 11,  65536 bytes             0      0      0        0      0      0
class 12, 131072 bytes             0      0      0        0      0      0
class 13, 262144 bytes             0      0      0        0      0      0
class 14, 524288 bytes             0      0      0        0      0      0
total configured streams memory:2000.00KB
streams memory in use: 205.29KB
maximum streams memory used: 686.34KB
```

This report shows that the kernel's peak usage of memory for STREAMS was about 700KB. Its current usage of physical memory is about 200KB which is well below the maximum 2MB of memory that can be dynamically allocated.

The usage of stream heads reported in the `streams` column shows that the 256 configured for use are sufficient. The number configured could be reduced using the **NSTREAM** parameter but this releases only 80 bytes of memory per stream head.

Making adjustments to the system

Firstly, the user increases **MAXUP** to 128 as this is only a configuration limitation. They will now be able to run many more processes than before.

To release more memory for use by user processes, they reduce the memory allocated to the buffer cache to 1MB by setting **NBUF** to 1024. The number of hash queues, determined by the value of **NHBUF**, is increased to 512 — that is, half the value of **NBUF**.

After making these changes, the kernel is relinked, and the system rebooted. The new size of the kernel has decreased by approximately 2MB to 5MB. This releases 2MB of memory for user processes.

The user continues to monitor the system in everyday use, particularly noting the impact of the changes on memory, buffer cache, and disk usage.

Case study: memory-bound software development system

This case study examines a system in which a large number of software developers are looping through the process of edit-compile-run a program. It is therefore a system on which a relatively small number of CPU-intensive jobs are constantly run. These are likely to be jobs that require a considerable amount of I/O, memory, and CPU-time. This is because compilers are usually large programs that access many files, create large data structures, and can be a source of memory contention problems.

System configuration

The system's configuration is as follows:

- Uniprocessor 80486DX2 running at 50MHz.
- EISA bus.
- 24MB of RAM.
- Two 1GB SCSI-2 hard disks.
- 48MB swap space on the root disk only.
- One intelligent 16-port serial card.
- 16 ASCII terminals.
- One Ethernet network card with 16KB buffer and 16-bit wide data path.

The system is isolated from other machines on the company's LAN with the connection primarily being used for e-mail traffic. No remote filesystems are mounted or exported.

Defining a performance goal

At installation, the system administrator knew the type of work that the machine would need to process, and so set the goal of maximizing I/O throughput. This was achieved by tuning the system so that the amount of time the machine spent performing disk I/O would be as low as possible. The system administrator also set up an entry in *root*'s **crontab** file to record system data at one-minute intervals during the working day:

```
* 8-18 * * 1-5 /usr/lib/sa/sa1
```

Recently, through observation and complaints from others, it has become apparent that the system is slowing down: in particular, the system's users have experienced slow response time. The goal is to restore the system to its initial performance.

Collecting data

The system administrator runs **sar** to examine the statistics that have been collected for the system. The following output is an example showing the CPU utilization at peak load when the system was first tuned:

```
08:00:00    %usr    %sys    %wio   %idle
...
14:06:00      53      25       2      20
14:07:00      55      23       1      21
14:08:00      52      20       3      25
...
```

Examining the situation now shows a much different pattern of usage:

```
08:00:00    %usr    %sys    %wio   %idle
...
10:51:00      35      37      28       0
10:52:00      29      44      26       1
10:53:00      32      38      30       0
...
```

The `%wio` figure is high (consistently greater than 15%) which indicates a possible I/O bottleneck. The cause of this could be related to the demands of the applications being run, or it could also be caused by swapping activity if the system is short of memory.

Formulating a hypothesis

The **sar -u** report shows that the system is spending a greater proportion of its time waiting for I/O and in system mode.

If the system is memory bound, this may also be causing a disk I/O bottleneck. Alternatively, if the problem was predominantly I/O based, the slowness could be caused by uneven activity across the system's disks or by slow controllers and disk drives being unable to keep up with demand. Another possibility is that the buffer cache may not be large enough to cope with the number of different files being compiled and the number of libraries being loaded.

If the problem is lack of memory, it could be that the system is constantly paging and swapping. Paging out to the swap areas need not be a major cause of performance degradation, but swapping out is usually an indication that there is a severe memory shortage. As a consequence, disk I/O performance can degrade rapidly if the disks are busy handling paging and swapping requests. In this way, high memory usage can lead very quickly to disk I/O overload. It also requires the kernel to expend more CPU time handling the increased activity. Preventing memory shortages helps to improve disk I/O performance and increases the proportion of CPU time available to user processes.

Getting more specifics

To confirm the hypothesis that the system is memory bound, the system administrator next examines the performance of the memory and I/O subsystems.

Memory investigation

The system administrator uses **sar -r** to report on the number of memory pages and swap file disk blocks that are currently unused:

```
08:00:00 freemem freeswp
...
10:51:00      44    2056
10:52:00      42    1720
10:53:00      41    1688
...
```

Since the number of free pages in the `freemem` column is consistently near the value defined for the **GPGSHI** kernel parameter (40), the page stealing daemon is probably active. Sharp drops in the amount of free swap space, `freeswp`, also indicate that the stack and modified data pages of processes are being moved to disk.

The average value of `freemem` indicates that there is only about 200KB of free memory available on the system. This is very low considering that the total physical memory size is 24MB. The system is also dangerously close to running out of swap as the average value of `freeswp` indicates that there is only about 910KB of space left on the swap device. The shortage of swap space is even more apparent when the **swap -l** command is run:

```
path                dev   swaplo blocks   free
/dev/swap           1,41       0  96000   1688
```

Only 1688 disk blocks remain unused out of 64000 configured on the swap device.

More evidence of swapping is found by running **sar -q**:

```
08:00:00 runq-sz %runocc swpq-sz %swpocc
...
10:51:00     2.7      98     1.0      36
10:52:00     2.0      63     3.0      31
10:53:00     2.0      58     1.0      49
...
```

The non-zero values in the `swpq-sz` and `%swpocc` indicate that processes are ready-to-run which have been swapped out.

To see evidence of swapping activity, the administrator uses **sar -w**:

```
08:00:00 swpin/s bswin/s swpot/s bswot/s pswch/s
...
10:51:00    0.52    12.1    1.01    19.2      72
10:52:00    1.21    22.5    3.02    37.4      55
10:53:00    0.71    15.2    0.97     7.3      83
...
```

The values of `swpot/s` and `bswot/s` are both well above zero. This shows that the system was frequently swapping during the sampling period.

The evidence confirms the comments of the users, and the suspicions of the administrator, that the system has a memory shortage.

I/O investigation

To investigate further, the system administrator uses **sar -b** to display statistics about the buffer cache:

```
08:00:00 bread/s lread/s %rcache bwrit/s lwrit/s %wcache pread/s pwrit/s
...
10:51:00     239     723      67       7      16      58       0       0
10:52:00     448    1280      65      10      22      56       0       0
10:53:00     374    1100      66      11      25      57       0       0
...
```

"rcache" at 66% and "wcache" at 57% indicate low hit rates in the buffer cache. Although these figures are low, the priority must be to reduce the memory shortage in the system. Tuning the buffer cache hit rate must be left to a later stage as increasing the buffer cache size will further reduce the amount of available memory.

Finally, the system administrator checks disk I/O performance using **sar -d**:

```
08:00:00 device   %busy    avque    r+w/s    blks/s   avwait   avserv
...
10:51:00 Sdsk-0   85.42    1.89     39.39    166.28   80.26    25.24

10:52:00 Sdsk-0   86.00    1.79     38.73    163.64   82.35    25.87
         Sdsk-1   10.01    1.16     12.37     23.11    3.24    20.19

10:53:00 Sdsk-0   87.00    1.92     38.07    171.95   78.32    26.32

...
```

The value of "avque" for the *root* disk (`Sdsk-0`) is consistently greater than 1.0 and is continually more than 80% busy. This indicates that the device is spending too much time servicing transfer requests, and that the average number of outstanding requests is too high. This activity is the combined result of the paging and swapping activity, and disk access by user processes. The second SCSI disk (`Sdsk-1`) does not contain a swap area and is much less active.

Making adjustments to the system

Since the system is both memory and I/O bound, it is likely that disk I/O performance is being made worse by the constant paging and swapping, so the sensible approach is to attack the memory problems first. As the system is almost completely out of memory and swap space, increasing these resources is a priority.

There are several ways to increase the amount of memory available to user processes on this system:

- Purchase more memory — though there may be inherent resistance to this, it can be a cost-effective solution in terms of time saved.
- See if any large processes such as servers are running that could run on less-loaded machines. If this were a system that supported workstations, some users might be running their X clients on the server instead of on their workstation.
- Some users may be running programs that require an unreasonably large amount of memory. It may be possible to tune the programs to reduce their demands on memory. Alternatively, such programs could be run when the system is more lightly loaded.

Solutions that will not help include:

- Reducing the size of the buffer cache would probably not help in this example because the system already has a disk I/O problem. On a system where the buffer cache hit rates were high, some memory could be regained by reducing the buffer cache size.
- Creating another swap area on the second system disk (`Sdsk-1`) might help to spread the I/O load caused by swapping and paging but it will not help to reduce the system overhead on the CPU.
- Minimizing the number of processes running on the machine at any one time by appropriate scheduling. This is not a reasonable solution in a production environment such as this where users expect a rapid response.

Chapter 5
Tuning I/O resources

Input/output (I/O) is the process of transferring data from memory to a device, from a device to memory, or from one device to another. Most I/O usually occurs between memory and the system's hard disk drives, though on some multiuser systems, it might be between the system and terminals connected via the network or serial lines. If the speed at which peripheral devices can access and communicate data to the system is relatively slow, the operating system may spend most of its time idle waiting for I/O with many processes asleep until the I/O completes.

The following sections contain information about the monitoring and tuning of various I/O subsystems:

- "Subsystems that affect disk and other I/O" (this page)
- "The mechanics of a disk transfer" (page 87)
- "Tuning virtual disk performance" (page 100)
- "Serial device resources" (page 108)
- Chapter 6, "Tuning networking resources" (page 123)

Subsystems that affect disk and other I/O

There are two methods of transferring data between memory and disk:

- Normal I/O uses **read**(S) and **write**(S) system calls. These can go through the buffer cache using the block device interface or direct to disk using the raw device interface. The **read** and **write** calls block the process until they complete — that is, they are synchronous.

- Asynchronous I/O (AIO) allows non-blocking access to raw disk devices. This allows processes to carry out other tasks while the kernel performs the I/O requests. If possible, you should enable AIO for database systems if they support it; this maximizes transaction throughput and minimizes delays.

  The two forms of AIO supported are the **aio**(HW) driver and the POSIX.1b **aio** functions.

  See "Viewing AIO activity" (page 162) for information about how to monitor AIO activity.

  Synchronous I/O operations to the raw disk device force the process requesting the operation to wait for it to complete. Database applications typically use synchronous I/O to ensure the integrity of the data being written to disk. For example, the journal logs that a database uses to recover in the event of system failure are written to disk using synchronous I/O.

To make the transfer of data between memory and disk more efficient, the system maintains a buffer cache of most recently accessed disk data. This reduces the amount of disk I/O that the system needs to perform. See "How the buffer cache works" (page 73) for a description of its operation.

In a similar way, the system maintains a namei cache (for translating names to inodes) of most recently used filenames in order to speed up the location of files in filesystems. In fact, it uses a separate namei cache for AFS, EAFS, and HTFS™ filesystems (all based on the **ht** driver) from the namei cache it uses for DTFS™ filesystems (which is based on the **dt** driver). See "How the namei cache works" (page 81) for a description of their operation.

Finally, the multiphysical buffers use a small pool of memory (generally 160KB to 256KB in size). They are used for various purposes as described in "How multiphysical buffers are used" (page 84).

For a description of how to monitor the activity of block devices including disks, see "Viewing disk and other block I/O activity" (page 89).

Disk I/O and networked filesystem (such as SCO® NFS®) performance are affected by filesystem fragmentation and other filesystem-related factors as described in "Filesystem factors affecting disk performance" (page 94).

How the buffer cache works

On a typical system approximately 85% of disk I/O can be avoided by using the buffer cache, though this depends on the mix of jobs running. The buffer cache is created in an area of kernel memory and is never swapped out. Although the buffer cache can be regarded as a memory resource, it is primarily an I/O resource due to its use in mediating data transfer. When a user process issues a **read** request, the operating system searches the buffer cache for the requested data. If the data is in the buffer cache, the request is satisfied without accessing the physical device. It is quite likely that data to be read is already in the buffer cache because the kernel copies an entire block containing the data from disk into memory. This allows any subsequent data falling within that block to be read more quickly from the cache in memory, rather than having to re-access the disk. The kernel also performs read-ahead of blocks on the assumption that most files are accessed from beginning to end.

The data area of each buffer for filesystems other than DTFS is 1KB which is the same size as a filesystem logical block and twice the typical physical disk block size of 512 bytes. DTFS filesystems use buffers with data areas in multiples of 512 bytes from 512 bytes to 4KB.

If data is written to the disk, the kernel first checks the buffer cache to see if the block, containing the data address to be written, is already in memory. If it is, then the block found in the buffer cache is updated; if not, the block must first be read into the buffer cache to allow the existing data to be overwritten.

When the kernel writes data to a buffer, it marks it as dirty. This means that the buffer must be written to disk before the buffer can be re-used. Writing data to the buffer cache allows multiple updates to occur in memory rather than having to access the disk each time. Once a buffer has aged in memory for a set interval it is flushed to disk by the buffer flushing daemon, **bdflush**.

The kernel parameter **NAUTOUP** specifies how long a buffer can be dirty before it is written to disk from the buffer cache. The default value for **NAUTOUP** is 10 seconds, and ranges between 0 and 60. It does not cause a buffer to be written precisely at **NAUTOUP** seconds, but at the next buffer flushing following this time interval.

Although the system buffer cache significantly improves overall system throughput, in the event of a system power failure or a kernel panic, data remaining in the buffer cache but which has not been written to disk may be lost. This is because data scheduled to be written to a physical device will have been erased from physical memory (which is volatile) as a consequence of the crash.

The default flushing interval of the buffer flushing daemon, **bdflush**, is 30 seconds. The kernel parameter **BDFLUSHR** controls the flushing interval. You can configure **BDFLUSHR** to take a value in the range 1 to 300 seconds.

If your system crashes, you will lose **NAUTOUP** + (**BDFLUSHR**/2) seconds of data on average. With the default values of these parameters, this corresponds to 25 seconds of data. Decreasing **BDFLUSHR** will increase data integrity but increase system overhead. The converse is true if you increase the interval.

Apart from adjusting the aging and flushing intervals, you can also control the size of the buffer cache. The kernel parameter **NBUF** determines the amount of memory in kilobytes that is available for buffers. If you are using the DTFS filesystem, the value of **NBUF** does not correspond to the actual number of buffers in use. The default value of **NBUF** is 0; this causes the kernel to allocate approximately 10% of available physical memory to buffers.

The size of the buffer cache in kilobytes is displayed when the system starts up and in the file */usr/adm/messages*. Look for a line of the form:

```
kernel: Hz = 100, i/o bufs = numberk
```

If there are any buffers in memory above the first 16MB, the line may take the form:

```
kernel: Hz = 100, i/o bufs = numberk  (high bufs = numberk)
```

The amount of memory reserved automatically for buffers may be not be optimal depending on the mix of applications that a system will run. For example, you may need to increase the buffer cache size on a networked file server to make disk I/O more efficient and increase throughput. You might also find that reducing the buffer cache size on the clients of the file server may be possible since the applications that they are running tend to access a small number of files. It is usually beneficial to do this because it increases the amount of physical memory available for user processes.

How you can investigate the effectiveness of the buffer cache is the subject of "Viewing buffer cache activity" (page 75).

For more information on tuning the size of the buffer cache see:

- "Increasing disk I/O throughput by increasing the buffer cache size" (page 75)
- "Increasing memory by reducing the buffer cache size" (page 54)

Appendix B, "Configuring kernel parameters" (page 185) tells you how you can use the **configure**(ADM) utility to change the values of kernel parameters such as **NAUTOUP**, **BDFLUSHR**, and **NBUF**.

Viewing buffer cache activity

Buffer cache activity can be seen using **sar -b** (or **mpsar -b** for SMP):

```
23:59:44 bread/s lread/s %rcache bwrit/s lwrit/s %wcache pread/s pwrit/s
23:59:49      38     723      95       4      16      75       0       0
23:59:54      38    1280      97       4      22      81       0       0
23:59:59      37    1100      97       2      25      91       0       0

Average       37    1006      96       3      20      83       0       0
```

The buffer cache read hit rate, `%rcache`, indicates the percentage by volume of data read from disk (or any block device) where the data was already in the buffer cache.

The buffer cache write hit rate, `%wcache`, indicates the percentage by volume of data written to disk (or any block device) where the block in which the data was to be written was already in the buffer cache.

NOTE For all filesystems other than DTFS, `%rcache` and `%wcache` are also equal to the percentage of read and write requests satisfied using the buffer cache.

`bread/s` indicates the average number of kilobytes per second read from the block devices (including disk drives) into the the buffer cache.

`bwrit/s` indicates the average number of kilobytes per second written from the buffer cache to block devices by the buffer flushing daemon, **bdflush**.

Increasing disk I/O throughput by increasing the buffer cache size

If the read and write buffer cache hit rates (`%rcache` and `%wcache`) reported by **sar -b** (or **mpsar -b** for SMP) show consistently low values, you can improve disk I/O performance by increasing the size of the buffer cache. This is particularly worth doing if the number of kilobytes of data transferred per second between the buffer cache and disk (`bread/s` + `bwrit/s`) is high. You can also examine the benefit to disk I/O performance using **sar -d** as described in "Viewing disk and other block I/O activity" (page 89). This should show improved `%busy`, `avque`, and `avwait` figures for disks containing regularly accessed filesystems as the buffer cache size is increased. Even if the impact on disk I/O is not significant, requesting processes benefit by not having to perform as many waits because of cache misses.

You should also note that increasing the size of the buffer cache directly reduces the amount of memory available for user processes. If free memory is reduced, the system may be more susceptible to paging out and swapping. If you increase the buffer cache size, you should monitor paging and swapping as well as buffer cache activity.

See Chapter 4, "Tuning memory resources" (page 41) for information on monitoring paging and swapping.

If a compromise cannot be reached between these resources and the applications being run cannot be tuned to reduce disk access, then the only alternative is to add either more memory or improve the throughput of the disk drives.

The following table is a summary of the commands that can be used to view buffer cache activity:

Table 5-1 Viewing buffer cache activity

Command	Field	Description
[mp]sar -b	%rcache	percentage by volume of data read from block devices satisfied using the buffer cache
	%wcache	percentage by volume of data written to block devices satisfied using the buffer cache

To increase the size of the buffer cache first determine the number of I/O buffers as outlined in the subsection "Viewing buffer cache activity" (page 75). The number of buffers can then be changed by modifying the **NBUF** kernel parameter.

It is not possible to recommend values of `%rcache` and `%wcache` for which you should aim. The values depend to a great extent on the mix of applications that your system is running, the speed of its disk subsystems, and on the amount of memory available. Lower limits can be quoted such as 90% for `%rcache` and 65% for `%wcache`, but you should not assume that these are ideal for your system. Ideal values would be 100% for both hit rates but you are unlikely to see these on a real system.

The maximum possible value of `%rcache` depends on how often new files are accessed whose data has not already been cached. Applications which read files sporadically or randomly will tend to have lower values for `%rcache`. If files are read which are not then subsequently re-read, this has the additional disadvantage of removing possibly useful buffers from the cache for reading and writing.

The effectiveness of caching blocks for write operations depends on how often applications need to modify data within the same blocks and how long buffers can remain dirty before the data is flushed to disk. The average time that data remains in memory before being flushed is **NAUTOUP** + (**BDFLUSHR** / 2). This is 25 seconds given the default values of these parameters.

If applications tend to write to the same blocks on a time scale that is greater than this, the same buffers will be flushed to disk more often. If applications append to files but do not modify existing buffers, the write hit rate will be low and the newly written blocks will tend to remove possibly useful buffers from the cache. If you are running such applications on your system, increasing the buffer cache size may adversely affect system performance whenever the buffer flushing daemon runs. When this happens, applications may appear to stop working temporarily (*hang*) although most keyboard input will continue to be echoed to the screen. Applications such as **vi**(C) and **telnet**(TC) which process keyboard input in user mode may appear to stop accepting key strokes. The kernel suspends the activity of all user processes until the flushing daemon has written the dirty buffers to disk. On a large buffer cache, this could take several seconds. To improve this situation, spread out the disk activity over time in the following ways:

- Decrease the value of **BDFLUSHR** so that the flushing daemon runs more often. This will reduce the peak demand on disk I/O at the possible expense of a slight increase in context switching activity.
- Decrease the value of **NAUTOUP** so that fewer dirty buffers accumulate in the cache. Potentially useful data remains in the buffers that have been marked clean until they are reused. Do not reduce **NAUTOUP** too much or caching may become ineffective.
- Use caching disk controllers (with battery backup if you are concerned about the integrity of your data).
- Some applications such as database management systems provide their own buffer caching strategy. This usually operates through the raw disk device and so does not use the operating system buffer cache.

Figure 5-1 (page 78) shows how the buffer cache read and write hit rates might increase as the number of buffers is increased. There are several points to notice here:

- You cannot independently tune the read and write hit rates (`%rcache` and `%wcache`). If the number of kilobytes of data read per second into the buffer cache from disk (`bread/s`) is much higher than the number written to disk (`bwrit/s`), you should attach more significance to the value of `%rcache`. On most systems, you will find that there is more data read from than written to disk.

- Increasing the value of **NBUF** has most effect for low cache hit rates — for high cache hit rates, the curves start to level off (saturate) and you need a large increase in **NBUF** to produce a small increase in the hit rate. For example, to increase the read hit rate (`%bread`) from 90% to 95%, a relative increase of 5.6%, you might need to double the value of **NBUF**. Although the read hit rate increases by only 5.6%, the amount of data that needs to be read from disk has been reduced by 50%. If disk I/O is a problem and your system is also not short of memory, you may consider it worthwhile to increase the size of the buffer cache.

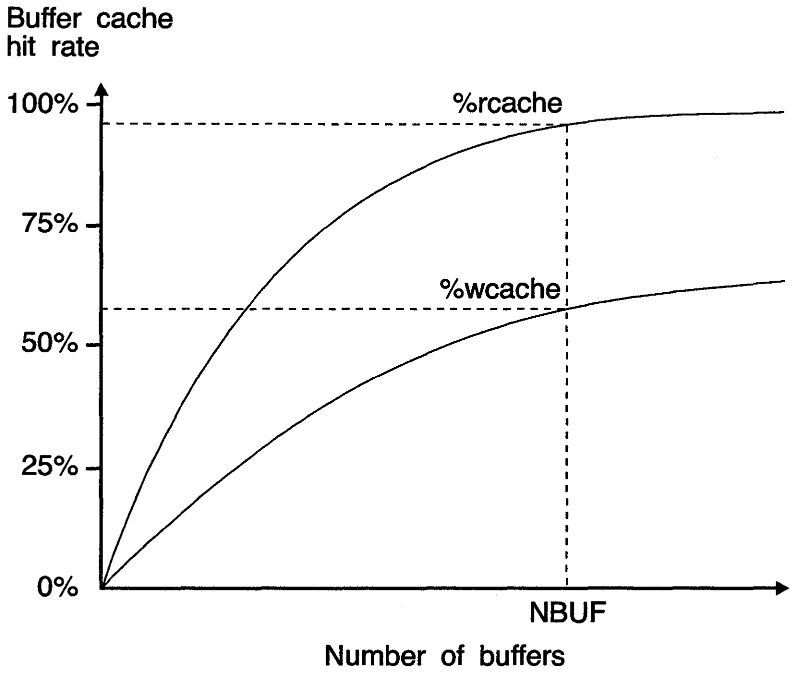


Figure 5-1 How cache hit rates depend on the number of buffers

Note that when you modify the value of **NBUF**, you may also need to set the value of **NHBUF** (the number of hash queues) as described in "Tuning the number of buffer cache hash queues" (page 80).

If your system has a large amount of memory and shows no swapping or significant paging out activity at peak load, you may wish to try increasing the size of the buffer cache. Provided that you do not allocate too much memory to buffers (so causing the system to page out and swap), this should reduce I/O activity and improve the interactive performance of applications. You should do this as an iterative process while monitoring the buffer cache hit rate and the amount of physical memory available to user processes.

If the amount of free memory drops drastically and the system begins to page out and swap, you should reduce the size of the buffer cache. See Chapter 4, "Tuning memory resources" (page 41) for more information.

Overriding the size of the buffer cache at boot time

You can use the **nbuf** bootstring to set a different size for the buffer cache when the system is booted. The value supplied as the argument to **nbuf** overrides the value of **NBUF** configured into the kernel. For example, the following command to **boot**(HW) sets the buffer cache size to 150KB in addition to using the default bootstring:

```
defbootstr nbuf=150
```

If **NHBUF** is set to 0, the number of hash queues will automatically be adjusted for the new buffer cache size.

See "Using bootstrings" in the *SCO OpenServer Handbook* for more information.

Positioning the buffer cache in memory

Peripheral controllers that support 32-bit addressing are capable of DMA transfers into memory above 16MB. In this case, the size of the buffer cache is its only important feature. You can tell whether SCSI host adapters support 32-bit addressing, among other performance-enhancing features, by examining the initialization message that their device driver outputs on the system console when the system starts and the kernel is loaded. For example, the following message:

```
%adapter  0x8000-0x8CDC 11       -       type=eiad ha=0 id=7 fts=std
```

shows that an Adaptec AHA-174x SCSI host adapter is installed which supports 32-bit DMA addresses (`fts=..d`). See "Boot time messages from host adapter drivers" in the *SCO OpenServer Handbook* for more information.

The distribution of the buffers also becomes important for controllers that only support 24-bit addressing — these can only access the first 16MB of memory. The **PLOWBUFS** tunable parameter specifies the percentage of the cache buffers that the system will try to place below the 16MB boundary at boot time. During system startup, the distribution of buffers between memory above and below 16MB is displayed:

```
kernel: Hz = 100, i/o bufs = 32768k (high bufs = 25232k)
```

To enable transfers of data between peripherals and memory above 16MB, the system makes use of multiphysical buffers situated in the lowest 16MB of memory. The amount of memory allocated for these buffers is controlled by the value of the kernel parameter **NMPBUF**. On systems with 24-bit controllers, you should ensure that as much of the buffer cache as possible lies in the first 16MB of memory. For more information see "How multiphysical buffers are used" (page 84).

Tuning the number of buffer cache hash queues

To speed look-up access, buffers are linked onto hash queues. The number of these is controlled by the **NHBUF** parameter. The average number of buffers per queue is the total number of buffers, **NBUF**, divided by the number of hash queues. The greater the number of hash queues, the fewer the number of buffers that will exist on any given hash queue on average.

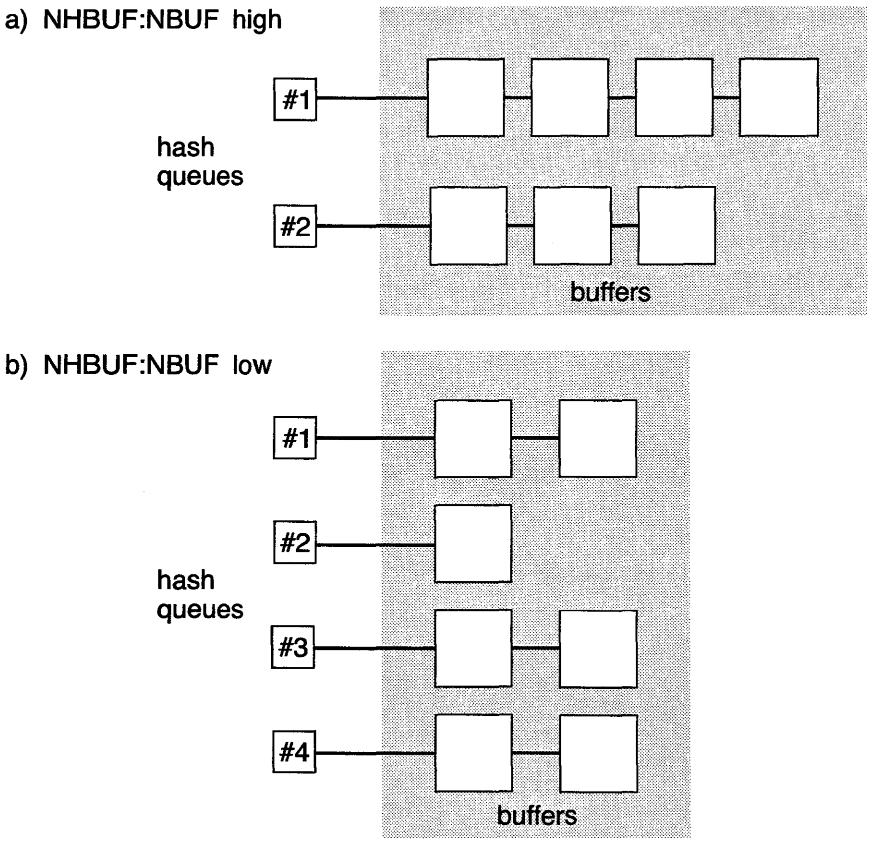


Figure 5-2 Keeping the ratio of hash queues to buffers low reduces contention for SMP

If you set **NHBUF** to 0, the system configures the number of buffer cache hash queues automatically at startup. On a uniprocessor system, the system sets the number of hash queues to the nearest power of 2 that is greater than or equal to half the value of **NBUF**. This should be treated as a recommended lower bound for **NHBUF**; you may find that setting **NHBUF** to a higher value gives better performance.

NOTE The value that you assign to **NHBUF** must be a power of 2; 512, 1024, and 2048 are examples.

On a multiprocessor system, the system sets the number of hash queues to the nearest power of 2 that is greater than or equal to twice the value of **NBUF**. The reason for this can be seen from Figure 5-2 (page 80). On a system with an SCO SMP License, the kernel running on one CPU locks the entire hash queue when it accesses a buffer. The kernel running on another CPU that wants to access the same hash queue must wait until it is released. Such contention can be avoided by keeping the ratio of hash queues to buffers low. For example, if the value of **NBUF** is 32000 on a system with an SCO SMP License, you should set **NHBUF** to at least 65536.

How the namei cache works

In order to find a file referenced by a given pathname, each of the components of the pathname must be read to find the subsequent component. For example, take the file */etc/passwd* when used in a command such as:

```
cat /etc/passwd
```

In order to find the file *passwd,* the *root* directory (/) must first be found on the disk. Then the entry for the pathname component *etc* is used to locate that directory. The *etc* directory is read from the disk and used to locate the file *passwd.* The file *passwd* can then be read from the disk.

All of the above steps use Index Nodes or inodes. A file in a filesystem is represented by an inode which records its type, size, permissions, location, ownership, and access and modification dates. To locate the file's data, the inode also stores the block number (or numbers) of the disk blocks containing the data. Note that the inode does not contain the name of the file. Another file, a directory, stores the filename together with the corresponding inode number. In this way, several directory entries (or filenames) may refer to the same inode; these are known as hard links.

When a command accesses a pathname, such as */etc/passwd,* the process of translating name to inode to data block has to be carried out for every component of the pathname before the file's data can be located. If a pathname component is a directory, such as */etc,* the data blocks pointed to by its inode contain a map of filenames to inodes. This map is searched for the next pathname component, and this process continues until the final name component is reached. All inodes can be looked up in the inode table stored in memory, or if not present there, at the head of the filesystem on disk where a linear list of inodes is kept. The in-core inode table stores additional information so that the kernel accesses the correct device if more than one filesystem exists.

Converting pathnames to inode numbers is a time-consuming process. It may require several disk accesses to read the inodes corresponding to the components of a directory pathname. The namei cache is used to reduce the number of times the disk must be accessed to find a file. Provided that a name component is less than 14 characters long, its name, inode number, and its parent inode number are placed in the namei cache located in memory. Pathname components longer than 14 characters are never cached. When a command wishes to open a file, the kernel first looks in the namei cache for each pathname component in turn. If it cannot find a component there, it retrieves the directory information from disk into the buffer cache and adds the entry to the namei cache if possible.

Viewing namei cache activity

The effectiveness of the system's namei caches can be seen using **sar -n**:

```
23:59:44  H_hits Hmisses (%Hhit)  D_hits Dmisses (%Dhit)
23:59:49     869       0  (100%)       5       0  (100%)
23:59:54    1091       6  ( 99%)       3       1  ( 75%)
23:59:59     832       0  (100%)       0       1  (  0%)

Average      931       2  (100%)       8       2  ( 80%)
```

NOTE **sar -n** shows two sets of statistics; one for the namei cache used by AFS, EAFS, S51K, and HTFS filesystems (**ht** driver-based), the other for the namei cache used by DTFS filesystems (**dt** driver-based).

`H_hits` indicates the number of pathname components that were found in the namei cache for **ht** driver-based filesystems.

`Hmisses` indicates the number of pathname components that were not found in the namei cache for **ht** driver-based filesystems and subsequently required that the information be read from one of the disk drives.

`%Hhit` gives the percentage of the total number of pathname components referenced that were found in the namei cache for **ht** driver-based filesystems.

`D_hits`, `Dmisses`, and `%Dhit` provide corresponding information for DTFS filesystems.

Separate statistics are provided for **ht** and **dt** driver-based filesystems because separate sets of kernel parameters are provided to tune them. See "Reducing disk I/O by increasing the size of the namei cache" (page 83) for more information.

Reducing disk I/O by increasing the size of the namei cache

There are three kernel parameters that control the performance of the namei cache for AFS, EAFS, and HTFS filesystems:

HTCACHEENTS
: Sets the number of entries in the namei cache. Increasing the size of the namei cache reduces the number of disk accesses required to find inodes associated with pathname components.

HTHASHQS
: Sets the number of hash queues used to access the namei cache. Having more hash queues speeds finding a pathname component in the cache.

HTOFBIAS
: Controls how long the system spends searching for a free entry in the namei cache before deleting and recycling an existing entry.

The kernel parameters **DTCACHEENTS**, **DTHASHQS**, and **DTOFBIAS** fulfill the same purpose for DTFS filesystems.

Follow these steps to tune the namei cache for HTFS, AFS, and EAFS filesystems:

1. If the `%Hhit` value shown by **sar -n** is consistently low (for example, less than 65%), increase **HTCACHEENTS** until `%Hhit` increases to a satisfactorily high value (for example, more than 90%). As a rough guide, **HTCACHEENTS** should be approximately three times the grown size of the in-core inode table reported by the `inod-sz` field of **sar -v**.
2. Make **HTHASHQS** a prime number that is at least half the value of **HTCACHEENTS**. (A prime number has no factors other than itself and 1. You can test whether a number is prime using **factor**(C).)
3. Do not change **HTOFBIAS** unless application programs on your system tend to access the same files. For example, the C compiler generates temporary files that are written by one pass of the compiler and read by another. An environment where compilation is a major activity may benefit from increasing **HTOFBIAS**.

   As a general rule, do not set **HTOFBIAS** greater than one tenth of the value of **HTCACHEENTS**. If you set the value of **HTOFBIAS** low, the names of open files have less special caching priority, and are more likely to be dropped from the cache. If you set **HTOFBIAS** high, open files keep their names in the cache for longer but the operating system has to spend more time looking for a free entry in the cache.

Repeat these steps over a number of days and use **sar -n** to monitor the namei cache's performance. Each time you change the parameter values, relink the kernel and reboot your system for the new values to take effect.

NOTE Poor name-lookup performance can result if the value of **HTCACHEENTS** is less than twice the grown size of the in-core inode table, and **HTOFBIAS** is greater than one tenth the value of **HTCACHEENTS**.

To tune the performance of the DTFS filesystem namei cache, follow the same procedure but examine the hit rate shown by the `%Dhit` field of **sar -n** and adjust the values of the kernel parameters **DTCACHEENTS, DTHASHQS,** and **DTOFBIAS** instead.

See Appendix B, "Configuring kernel parameters" (page 185) for details of how to change the values of kernel parameters.

How multiphysical buffers are used

The multiphysical buffer pool is an area of memory that can be allocated to various tasks associated with moving data between memory and physical devices:

- 16KB scatter-gather buffers are used to transfer contiguous blocks of data on disk to and from the buffer cache. This mechanism is normally only used if the disk controller does not support scatter-gather in hardware.

  If no buffers are available, a process will normally sleep.
- 1KB copy request buffers (or copy buffers) are used to move data to and from buffers in the buffer cache that lie in memory above 16MB. Copy buffers are necessary for DMA and peripheral controllers that cannot address memory above 16MB as shown in Figure 5-3 (page 85). If a copy buffer is not available, a process sleeps until one becomes available.
- 4KB transfer buffers are used for moving data between memory and peripheral devices on behalf of applications whose data may lie in memory above the first 16MB. These buffers are only necessary for DMA and peripheral controllers that cannot address memory above 16MB. If no transfer buffers are available, the process is put to sleep.

The value of the **NMPBUF** kernel parameter controls the number of 4KB memory pages used for scatter-gather, copy request, and transfer buffers. The number of pages of memory reserved for these buffers is tuned automatically if **NMPBUF** is set to 0. You can check if sufficient memory has been assigned as described in "Tuning the number of multiphysical buffers" (page 86).

The system dynamically allocates memory to the following data structures which are used in performing certain I/O operations:

- Scatter-gather buffer headers are used to control scatter-gather requests if the disk hardware supports scatter-gather. If no headers are available, the requests are sent to the disk controller one at a time.
- Control blocks are used to send raw disk I/O (AIO) requests including access to swap space. If no control blocks are available, the process is put to sleep.

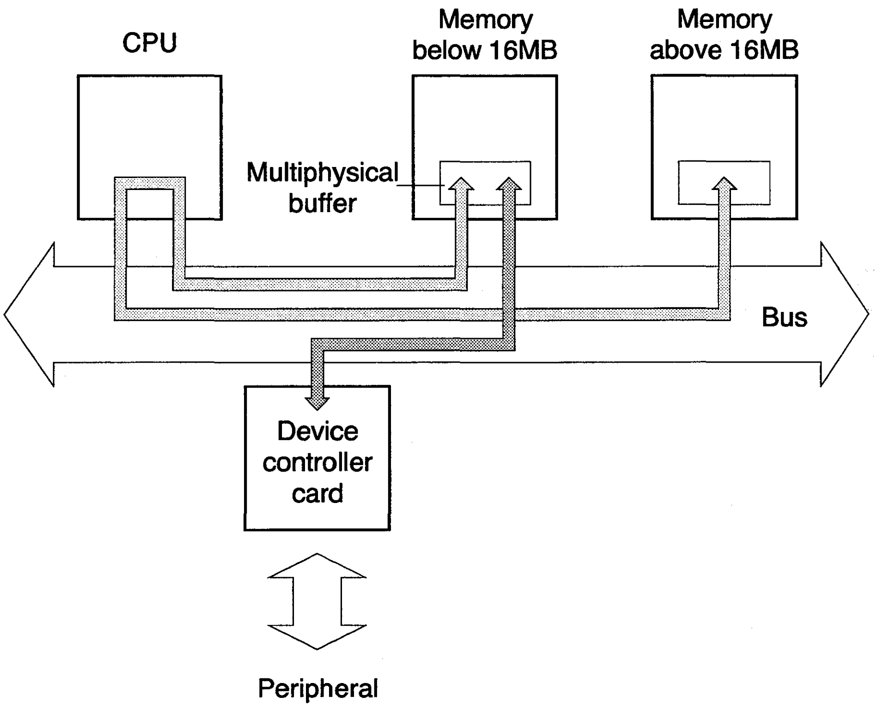


Figure 5-3 How the system uses multiphysical buffers to overcome 24-bit addressing limitations

Tuning the number of multiphysical buffers

There is a possible performance impact from having too little memory available for multiphysical buffers. When the system requires a scatter-gather, transfer, or copy buffer when none is available, it will put the process requiring the resource to sleep until one is available. The number of 4KB pages of memory available for these buffers is determined by the value of the kernel parameter **NMPBUF**.

Appendix B, "Configuring kernel parameters" (page 185) tells you how you can use the **configure**(ADM) utility to change the value of **NMPBUF**.

Use **sar -B** and **sar -h** (or their **mpsar** equivalents for SMP) to see if enough multiphysical buffers are configured. The output from **sar -B** reports on the system's use of copy buffers:

```
23:59:44  cpybufs/s slpcpybufs/s
23:59:49       0.00          0.00
23:59:54       0.00          0.00
23:59:59       0.00          0.00

Average        0.00          0.00
```

The `slpcpybufs/s` column shows how many times per second a process has to sleep waiting for a copy buffer.

sar -h reports on the usage of the other multiphysical buffers:

```
23:59:44 mpbuf/s  ompb/s mphbuf/s omphbuf/s  pbuf/s spbuf/s dmabuf/s sdmabuf/s
23:59:49    0.10    0.00     0.21      0.00    0.06    0.00     0.04      0.00
23:59:54    0.09    0.00     6.22      0.00    0.18    0.00     0.07      0.00
23:59:59    0.20    0.00     0.54      0.00    0.05    0.00     0.03      0.00

Average     0.13    0.00     2.32      0.00    0.10    0.00     0.05      0.00
```

The columns `ompb/s` and `sdmabuf/s` show how many times the system ran short of scatter-gather and DMA transfer buffers per second.

If **sar -B** and **sar -h** report non-zero values for `slpcpybufs/s`, `sdmabuf/s`, or `ompb/s`, increase the value of the kernel parameter **NMPBUF** by at least the maximum value reported for the sum of the following quantities:

NMPBUF * (`slpcpybufs/s` / (4 * `cpybufs/s`))
NMPBUF * (4 * `ompb/s` / `mpbuf/s`)
NMPBUF * (`sdmabuf/s` / `dmabuf/s`)

For example, suppose **sar -B** and **sar -h** reported the following shortfall in the availability of multiphysical buffers at a particular point in time:

```
12:00:00  cpybufs/s slpcpybufs/s
12:05:00       4.01         1.17

12:00:00 mpbuf/s  ompb/s mphbuf/s omphbuf/s  pbuf/s spbuf/s dmabuf/s sdmabuf/s
12:05:00    3.05    0.30     5.11      1.21    2.89    0.54     7.62      1.54
```

In this case, if the current value of **NMPBUF** is 40, you should increase it by:

```
40*((1.17/(4*4.01))+(4*0.30/3.05)+(1.54/7.62))
```

This evaluates to 27 when rounded up to the nearest whole number. If you wish to err on the safe side, double the value of **NMPBUF** to 80. This will require an extra 160KB of memory (40 4KB pages).

The following table summarizes the commands that you can use to view multiphysical buffer activity:

Table 5-2 Viewing multiphysical buffer activity

Command	Field	Description
[mp]sar -B	cpybufs/s	copy buffer usage per second
	slpcpybufs/s	sleeps waiting for copy buffer per second
[mp]sar -h	%mpbuf/s	scatter-gather buffer usage per second
	ompb/s	number of times scatter-gather buffers were not available per second
	dmabuf/s	DMA transfer buffers usage per second
	sdmabuf/s	sleeps waiting for DMA transfer buffers per second

The mechanics of a disk transfer

A typical write sequence between a process on the CPU and a hard disk involves many stages. When the buffer flushing daemon, **bdflush**, flushes the buffer cache to move data to disk, it queues the write request on the appropriate disk controller. Note that a **write** to the raw disk device queues the request on the disk controller without passing through the buffer cache.

Until now everything that has taken place has been comparatively rapid in its execution (ignoring the scheduled delays in the system). As soon as the disk is physically written to (or read from) we encounter several orders of magnitude reduction in speed. The appropriate disk head needs to seek across the disk to position itself over the correct track. The disk platter rotates so that the head writes to the correct sector.

If the amount of data is larger than a disk block and the blocks for the file to be written to are sufficiently fragmented, the seek/rotate cycle needs to be repeated until the write sequence is complete.

A hard disk's performance is characterized by its access time; this is the sum of the seek time taken by the head to move to the correct track, the rotational latency while the disk rotates until the head is over the desired block, and the transmission time needed to transfer data to or from the block as shown in Figure 5-4 (this page).

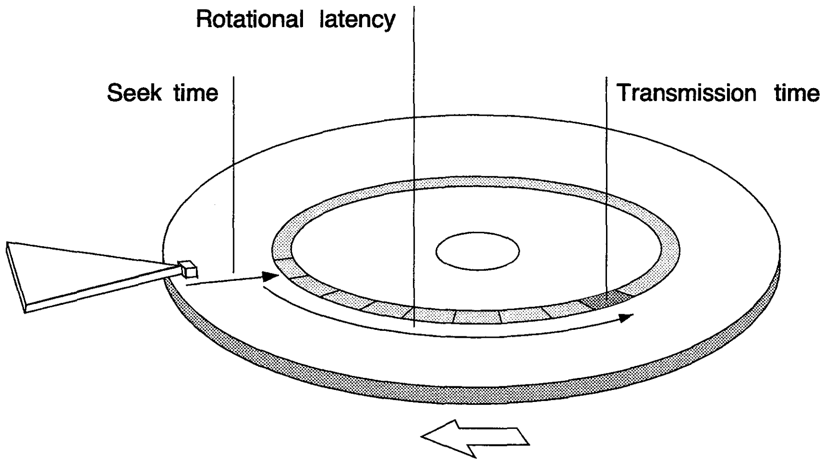


Figure 5-4 The performance characteristics of a hard disk

These activities take a finite time to complete and are usually quoted as average figures. The peak transfer rate, which is also commonly quoted as a measure of disk performance, depends directly upon these activities. However, peak transfer rates are unusual, especially on disks which are used close to their maximum capacity. These suffer from greater fragmentation of data than emptier disks. If you have already purchased your disks, then the only use of the above values is in deciding which disks you should use to store your files.

Another factor that you can take into consideration if you know the geometry of your disks is the layout of filesystems across the disk surface. Modern disks often have more blocks per track in the outer cylinders than in the inner tracks. When reading contiguous blocks, data from the outer tracks is transferred faster than from inner tracks. The innermost tracks, however, have lower latency times when reading relatively few blocks.

If the applications on your system access large sequential files, such as database journal logs, graphics images, fonts and PostScript®, you may see a gain in disk performance in putting these files in filesystems on the outer tracks of your disks.

If you intend to use **divvy**(ADM) to re-arrange the divisions on a disk partition, consult the documentation for your drive to see whether block 0 is at the center or the outside of the disk. Most disk manufacturers place it at the center but this is not necessarily the case for all types of hard disk. You will also need to make backups of any existing filesystems on the disk and restore these after making the changes to the disk layout.

Viewing disk and other block I/O activity

The activity of block devices installed on the system, including floptical, floppy and hard disk drives, CD-ROM and SCSI tape drives, can be examined using **sar -d** (or **mpsar -d** for SMP). This example shows the activity for a single SCSI disk:

```
23:59:44  device  %busy    avque    r+w/s   blks/s   avwait   avserv
23:59:49  Sdsk-0  99.42     4.18    39.39   166.28    80.26    25.24
23:59:54  Sdsk-0 100.00     4.18    38.73   163.64    82.35    25.87
23:59:59  Sdsk-0 100.00     3.98    38.07   171.95    78.32    26.32

Average   Sdsk-0  99.89     4.12    38.78   167.21    80.32    25.76
```

`device` shows the name of the device whose activity is being reported. In this example, the device is the first SCSI disk in the system.

`%busy` indicates the percentage of time that the system was transferring data to and from the device.

`avque` indicates the average number of requests pending on the device including any on the device itself. This number is usually greater than the number of processes waiting to access the device if scatter-gather read ahead is being performed on behalf of a filesystem.

`avwait` represents the average time in milliseconds that the request waits in the driver before being sent to the device.

`avserv` represents the average time in milliseconds that it takes a request to complete. The length of time is calculated from the time that the request was sent to the device to the moment that the device signals that it has completed the request. Note that `avserv` values vary considerably according to the type of disk and any caching on the disk controller.

`r+w/s` is the number of read and write transfers from and to the disk, and `blks/s` is the number of 512-byte blocks transferred per second. These two values can be used to calculate the average size of data transfers using the formula:

Average size of data transfer = `blks/s` / `r+w/s`

Identifying disk I/O-bound systems

A system is I/O bound, or has an I/O bottleneck, if the peripheral devices (hard disk, tape, and so on) cannot transfer data as fast as the system requests it. This causes processes to be put to sleep, "waiting for I/O", and leaves the CPU(s) idle for much of the time. To determine if the system is disk I/O bound run **sar -u** (or **cpusar -u** for SMP) and look at the `%wio` value. This displays the percentage of time that each CPU spends waiting for I/O to complete while there are no runnable processes. If this value is high then it is possible that I/O is not keeping up with the rest of the system. (You should not always assume that there is a problem with disks; for example, `%wio` might be high because a tape drive is being accessed.) Other indications of a disk I/O bottleneck can be seen using **sar -d** (or **mpsar -d** for SMP). Note that **sar -d** can be also be used to view the activity of block I/O devices including hard disk drives, SCSI tape drives, and floppy disks.

If the values for `%busy` and `avque` are both consistently high then the devices cannot keep up with the requests to transfer data. Devices such as floppy disks and some older types of tape drive are inherently slow. As these devices are generally infrequently used — for system backup, software installation, and so on — there is little that performance tuning can usefully accomplish.

The value of `blks/s` displayed by **sar -d** can be combined with `%busy` to give an indication of the maximum I/O throughput of a disk, and may suggest where a I/O bottleneck can occur:

Maximum disk throughput (KB/s) = `blks/s` * 50 / `%busy`

High values of the ratio of `avwait` to `avserv` also suggest that the device is saturated with requests.

If the number of transfers, `r+w/s`, is high but the amount of data being transferred, `blks/s`, is low, it may be possible to modify the application to transfer larger amounts of data less frequently. This should reduce the number of requests for the disk and reduce contention for it.

The read and write hit rates (`%rcache` and `%wcache`) shown by **sar -b** should show high values. If these values fall, the system is having to access blocks on disk (or other block devices) rather than in the buffer cache. If this happens, increasing the size of the buffer cache may help to alleviate a disk I/O bottleneck.

A low hit rate for the namei cache could lead to the disk being accessed more often in order to convert pathname components to inode numbers. If **sar -n** displays results showing that `%Hhit` or `%Dhit` is consistently low then the namei cache for the corresponding filesystem type is too small. It is not possible to give a general definition of what is a low value since this depends on the application mix that you run on your system. Because the performance of the namei cache does not depend linearly on its size, you will find that improving cache hit rates that are already high requires a significantly greater cache size. For example, you might have to increase the size of the namei cache by 30% in order to increase the namei cache hit rate from 90% to 95% giving a relative gain of 5.6%.

Note that namei caching is only performed for pathname components that are 14 characters or less in length. To take advantage of the namei cache, you should use directory names and filenames that are less than 15 characters long.

The following table is a summary of the commands that can be used to determine if a system is I/O bound:

Table 5-3 Identifying an I/O-bound system

Command	Field	Description
[cpu]sar -u	%wio	percent of time that the CPU spends waiting for I/O with no processes to run
[mp]sar -b	bread/s	average number of blocks read into the buffer cache per second
	bwrit/s	average number of blocks written from the buffer cache per second
	%rcache	percent of read disk block requests satisfied by reading the buffer cache
	%wcache	percent of write disk block requests satisfied by writing to the buffer cache
[mp]sar -d	avque	average number of requests on queue waiting for device per second

(Continued on next page)

Table 5-3 Identifying an I/O-bound system
(Continued)

Command	Field	Description
	avserv	average time transfer takes to complete in milliseconds
	blks/s	number of 512 byte blocks transferred to and from the device per second
	%busy	percent of time device was servicing transfer request
	r+w/s	number of read/write transfers to device per second
[mp]sar -n	%Hhit	AFS, EAFS and HTFS filesystem (**ht** driver) namei cache hit rate as a percentage
	%Dhit	DTFS filesystem (**dt** driver) namei cache hit rate as a percentage

Tuning disk I/O-bound systems

If the system is I/O bound because of disk activity, there are a number of things that can be done:

- Replace the existing disks with faster versions.
- Filesystems that are used to hold temporary files can be implemented as ramdisks in memory. (See **ramdisk**(HW) for more information.) This has the disadvantage of taking memory away from applications but it can be extremely effective in improving I/O throughput.
- Upgrade the disk controller to a type that supports block or track caching, and scatter-gather read/writes.
- For SCSI disks, upgrade the host adaptor to one that supports caching, scatter-gather, and tagged command queuing. Where possible, use fast SCSI subsystems with wide data paths.
- If the system is running a disk-intensive application such as a database, having multiple host adapters (for SCSI), disk controllers and disks will help speed up access to data by reducing contention.
- Spread filesystems and swap areas across different disks and/or buses to help spread the load. Alternatively, you can use hardware RAID or virtual disk software to balance the load across several disks. For a comparison of the various configurations that are available using virtual disks see "About virtual disks" in the *System Administration Guide.*

You may find that the performance of the system can be improved slightly by increasing the values of the **BDFLUSHR** and **NAUTOUP** kernel parameters. This will reduce the number of times the disk will be accessed because blocks can be updated more often in memory before they are written to the disk. The inherent risk is that more data will be lost if the system crashes because it will be longer since it was last written to the disk. It is considered good practice to protect mission-critical systems against power failure using a UPS or similar device.

Various disk organization strategies are discussed in "Overcoming performance limitations of hard disks" (page 96) which includes suggestions for optimizing your current hardware configuration.

Disk manufacturers implement various hardware and firmware (software in the disk controller) strategies to improve disk performance. These include track caching and varying the number of disk blocks per track across the disk surface. Usually, you have no control over such features.

SCSI disk driver request queue

The SCSI disk driver maintains a queue of disk requests to be sent to the disk controller. The length of this queue for each disk is controlled by the **SDSKOUT** parameter. If a process sends too many requests to the driver, it will put the process to sleep until a free slot in the queue is available. See "Tuning the number of SCSI disk request blocks" (this page) for details of how to monitor and tune the performance of SCSI disk request queuing.

Tuning the number of SCSI disk request blocks

The **sar -S** command (or its equivalent **mpsar -S** for SMP) reports the usage of SCSI request blocks:

```
23:59:44 reqblk/s oreqblk/s
23:59:49    12.56      0.00
23:59:54     3.04      0.00
23:59:59    25.45      0.00

Average     13.68      0.00
```

`reqblk/s` is the average number of request blocks used per second. `oreqblk/s` is the number of times per second that processes were put to sleep because not enough request blocks were available. If `oreqblk/s` is greater than zero, increase the value of **SDSKOUT** by at least the maximum value reported for the following quantity:

SDSKOUT * `oreqblk/s` / `reqblk/s`

The following table summarizes the commands that you can use to view SCSI request block activity:

Table 5-4 Viewing SCSI request block activity

Command	Field	Description
[mp]sar -S	reqblk/s	number of SCSI request blocks allocated per second
	oreqblk/s	number of times per second that the system ran out of SCSI request blocks

Filesystem factors affecting disk performance

Traditional UNIX filesystems use inodes to reference file data held in disk blocks. As files are added and deleted from the filesystem over time, it becomes increasingly unlikely that a file can be allocated a contiguous number of blocks on the disk. This is especially true if a file grows slowly over time as blocks following its present last block will probably become allocated to other files. To read such a file may require many head seek movements and consequently take a much longer time time than if its blocks were written one after another on the disk.

AFS, EAFS, and HTFS filesystems try to allocate disk blocks to files in clusters to overcome fragmentation of the filesystem. Fragmentation becomes more serious as the number of unallocated (free) disk blocks decreases. Filesystems that are more than 90% full are almost certainly fragmented. To defragment a filesystem archive its contents to tape or a spare disk, delete the filesystem and then restore it.

On inode-based filesystems, large files are represented using single, double, and even triple indirection. In single indirection, a filesystem block referenced by an inode holds references to other blocks that contain data. In double and triple indirection, there are respectively one and two intermediate levels of indirect blocks containing references to further blocks. A file that is larger than 10 filesystem blocks (10KB) requires several disk operations to update its inode structure, indirect blocks, and data blocks.

Directories are searched as lists so that the average time to find a directory entry initially increases in direct proportion to the total number of entries. The blocks that a directory uses to store its entries are referenced from its inode. Searching for a directory entry therefore becomes slower when indirect blocks have to be accessed. The first 10 direct data blocks can hold 640 14-character filename entries. The namei cache can overcome some of the overhead that would result from searching large directories. It does this by providing efficient translation of name to inode number for commonly-accessed pathname components.

You can increase the performance of HTFS filesystems by disabling checkpointing and transaction intent logging. To do this for an HTFS *root* filesystem, use the **Hardware/Kernel Manager** or **configure**(ADM) to set the values of the kernel parameters **ROOTCHKPT** and **ROOTLOG** to 0. Then relink the kernel and reboot the system. For other HTFS filesystems, use the **Filesystem Manager** to specify no logging and no checkpointing or use the **-onolog,nochkpt** option modifiers with **mount**(ADM). The disadvantage of disabling checkpointing and logging is that it makes the filesystem metadata more susceptible to being corrupted and potentially unrecoverable in the case of a system crash. Full filesystem checking using **fsck**(ADM) will also take considerably longer.

For more information on these subjects see "Maintaining filesystem efficiency" in the *System Administration Guide* and "How the namei cache works" (page 81).

Overcoming performance limitations of hard disks

One area where you are likely to experience performance limitations is with I/O from and to hard disks. These are heavily used on most systems, and accessing data on them is much slower than is the case with main memory. The time taken to access main memory is typically many thousands of times less than that taken to access data on disk. The solution is to try to arrange for the data that you want to be in a memory cache when you need it, not on disk. The cache may be one maintained by the operating system, though many applications such as databases manage their own caching strategies in user space. The situation is helped further by modern disks and disk controllers which implement cache memory in hardware.

"Increasing disk I/O throughput by increasing the buffer cache size" (page 75) describes how you can tune the buffer caching provided for access through the interface to block devices such as hard disks.

"Viewing namei cache activity" (page 82) describes how to tune the **namei** cache. This is the cache that the system maintains to avoid disk access when mapping filenames to inode numbers.

Not all activity on disk involves access to filesystems. Examples are swapping and paging to swap space, and the use of raw disk partitions by many database management systems. It is worth examining disk transfer request activity to discover how busy a system's disks are at the lowest level. "Viewing disk and other block I/O activity" (page 89) describes how you can monitor the activity of block I/O in a system not only for block-structured media such as hard disk, CD-ROM, floppy and floptical disks, but also for SCSI tape drives.

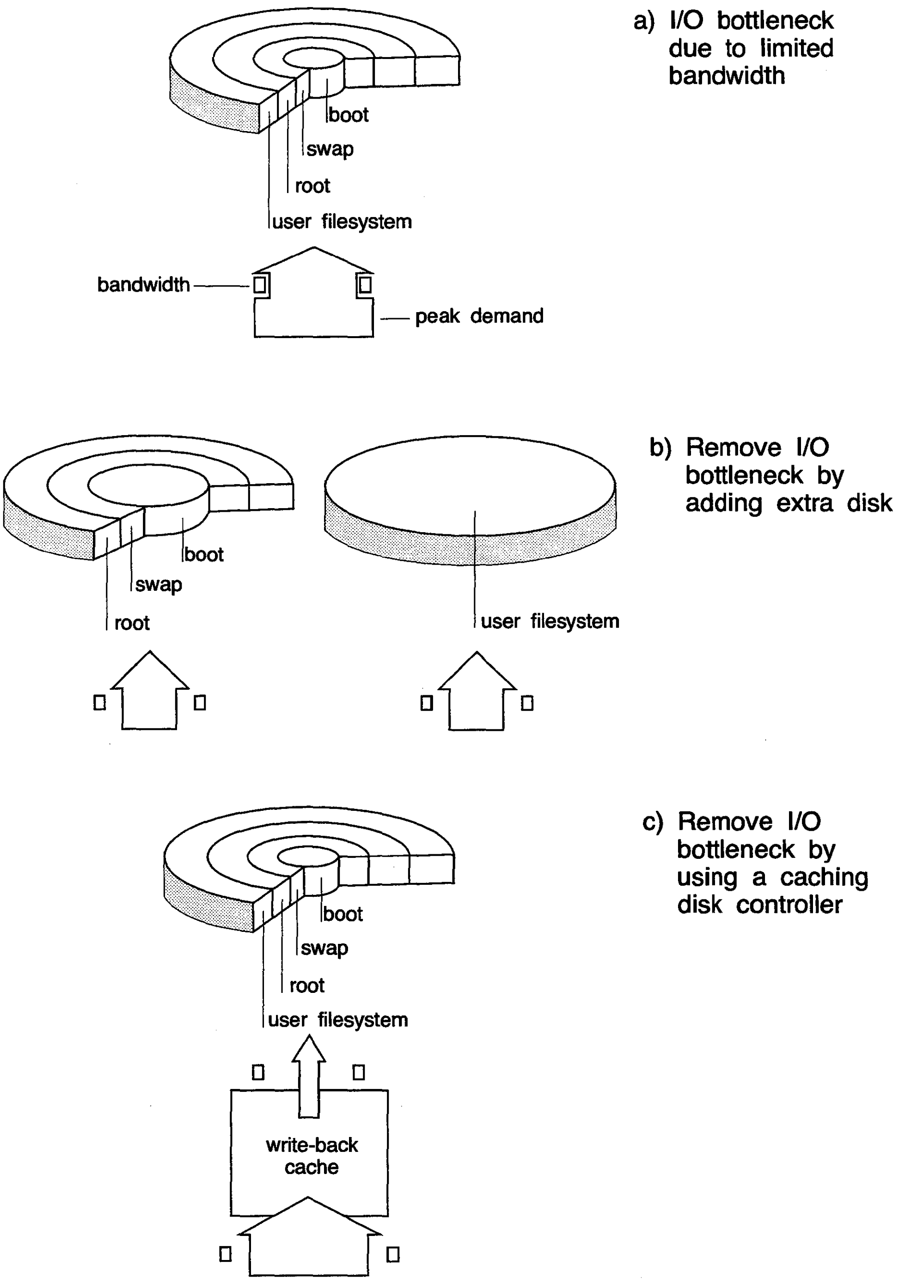

a) I/O bottleneck due to limited bandwidth
boot
swap
root
user filesystem
bandwidth
peak demand
b) Remove I/O bottleneck by adding extra disk
boot
swap
root
user filesystem
c) Remove I/O bottleneck by using a caching disk controller
boot
swap
root
user filesystem
write-back cache

Figure 5-5 Curing a disk I/O bottleneck caused by limited bandwidth

Comparison of I/O activity allows you to see if activity between different disks is unbalanced. In itself, this is not a problem unless the bandwidth of a particular disk is limiting throughput. Figure 5-5 (page 97) shows a system where a disk I/O bottleneck is cured by the addition of an extra disk or a caching disk controller. Adding an extra disk is likely to be successful unless the bandwidth limitation occurs elsewhere, for example, in the disk controller. Adding a caching controller is likely to succeed where a disk is having difficulty coping with peak demand. A write-back cache should be backed up by a UPS to guard against mains power failure and the consequent data loss that would occur.

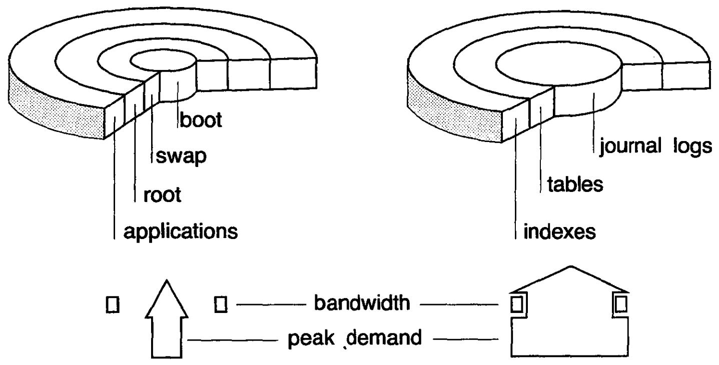
Balancing activity between disks may sometimes be achieved by simply moving a filesystem between two disks. A disk I/O bottleneck may occur if applications software and a user filesystem coexist on the same disk. This may lead to large access times as the disk heads are consistently sweeping across the entire disk. One solution is to move the applications software to other disk(s). The documentation for the applications may provide guidelines for this.

It is often unwise to move software or user filesystems onto the hard disk containing the *root* filesystem. Depending on how you use the system, this can be one of the most heavily-used disks.

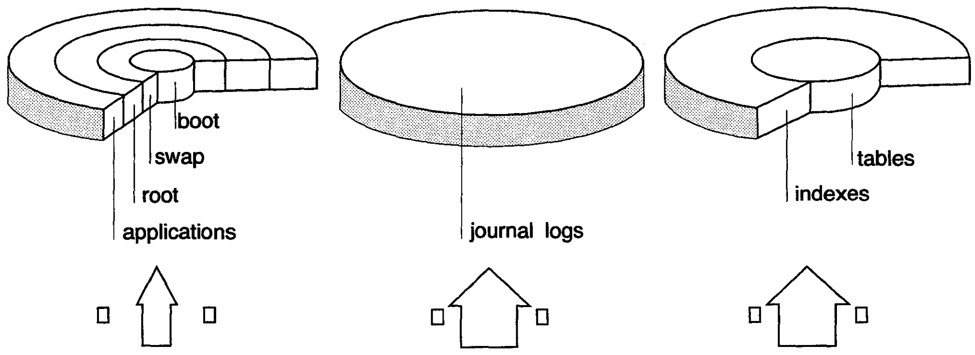
A common source of disk bottlenecks on relational database servers occurs when the journal logs (used if the system has to recover from a crash) share the same disk as database tables and indexes. The journal logs are constantly updated and the disks containing them are usually the busiest on the system. The journal logs are also written sequentially so keeping them on separate disks reduces seek time. Figure 5-6 (page 99) shows how a disk dedicated for use by the journal logs might be added to a system in order to remove a bottleneck.

Another method for rebalancing disk activity is to create a virtual disk from several existing disks. See Chapter 8, "Administering virtual disks" in the *System Administration Guide* for more information. If a striped virtual disk is configured, this can increase disk throughput by spreading I/O activity evenly across the physical disks. Although the performance of disks containing a database journal log would benefit from striping, it is not advisable to stripe other filesystems across the same disks. Similarly, if there are several journal logs, these should be kept on physically separate disks.

If you have configured one or more virtual disks on your system, see "Tuning virtual disk performance" (page 100) for more information about how to tune them.



a) I/O bottleneck due to unbalanced disk activity on a database server



b) Rebalance I/O activity by moving logs to a separate disk

Figure 5-6 Curing a disk I/O bottleneck caused by unbalanced disk I/O activity

Tuning virtual disk performance

When configuring a virtual disk array, you need to balance the I/O performance (maximum throughput) you expect from it against its inherent resilience to hardware failure. The various configurations can be arranged from best to worst under these criteria as shown in the following table:

Table 5-5 Comparison of simple and virtual disk configurations

Rating	I/O performance	Resilience
Best	RAID 10	mirror
	stripe	RAID 10
		RAID 53
	RAID 5, 53	RAID 4, 5 (hot spare)
	RAID 4	RAID 4, 5 (no hot spare)
	mirror	
	concatenated	simple
Worst	simple	concatenated, stripe

The performance rankings are based on equal virtual disk storage capacity for each configuration. You should also note that increasing the amount of redundancy in a virtual disk array increases its resilience but it also increases the number of disks and therefore the cost. If you use several host adapters to provide resilience against one of these failing, this will also add to the cost of setting up a virtual disk array. The best all round performer for I/O throughput and resilience is a RAID 10 array.

Mirrored disks (RAID 1) offer the best resilience to hardware failure (especially if the disks are on separate buses or controllers). They usually have a lower maximum I/O throughput than striped (RAID 0) or RAID 4 and 5 arrays. However, you should note that a RAID 1 array can outperform RAID 4 and 5 if the predominant pattern of operation is to write data to disk.

To get the best performance from a virtual disk array, your system's CPU power and memory (RAM) capacity should be capable of handling the extra overhead incurred by the virtual disk driver (**vdisk**) in processing disk requests. We recommend that your system has at least a 486™ processor running at 50MHz. The amount of memory that the **vdisk** driver uses depends on:

- the number of virtual disk arrays configured
- which RAID levels are used; RAID 4, 5 and 53 are the most memory hungry
- the number of disk requests (jobs) to be processed simultaneously
- the value of the kernel tunables which control the size of the internal data structures used by **vdisk**

On a lightly loaded system with two or three virtual disks, **vdisk** requires up to 500KB of memory for its internal data structures. On a busy file server with several virtual disks which have been tuned to optimize I/O throughput, **vdisk** requires at least 1MB of memory. Since the kernel allocates physical memory to **vdisk** dynamically, this memory does not show up in the size of the kernel when it initializes. If your system is short of real memory, it may start to swap or page out memory belonging to processes when **vdisk** is heavily loaded.

You should also note that using host adapters or disk controllers that are not capable of performing DMA above the first 16MB of memory (that is, not 32-bit controllers) will increase CPU overhead for all disk I/O (not just to virtual disks) for systems with more than 16MB of RAM.

If you mix hard disks which have different access times, this can cause an I/O bottleneck on the slow disks. To avoid this, ensure that all the disks in a virtual disk array have similar performance characteristics. The way that you assign disk pieces between disks can also cause a bottleneck. Physical disks that contain more disk pieces than others are likely to be more heavily loaded. This may occur if you allow the layout of your virtual disk arrays to overlap partially.

See the following for more information on tuning virtual disk performance:

- "Performance considerations for RAID 4 and 5" (page 102)
- "Choosing a cluster size" (page 103)
- "Balancing disk load in virtual disk arrays" (page 105)
- "Tuning virtual disk kernel parameters" (page 106)

For more information about virtual disk configuration and administration, see Chapter 8, "Administering virtual disks" in the *System Administration Guide*.

Performance considerations for RAID 4 and 5

An extra performance consideration for RAID 4 and 5 configurations is that their I/O throughput becomes progressively worse as the ratio of the number of writes to reads increases. If the ratio of writes to reads is very high, the performance of RAID 4 and 5 can drop below that of a RAID 1 (mirrored) array.

RAID 4 and 5 arrays perform poorly on writes because of the extra disk operations required to update the parity information. The operating system performs the following sequence of events when writing a cluster to a RAID 4 and 5 array:

1. Read the original cluster data.
2. Read the original parity.
3. Subtract the original data from the parity.
4. Add the new data to the parity.
5. Write the new parity.
6. Write the new data.

As a result, a write requires a total of four disk accesses and two calculations on the parity.

Three-piece RAID 4 or 5 arrays use two pieces for data and the remaining piece for parity. This allows an optimization to be made which avoids reading the parity piece. The sequence of events performed by the operating system when writing becomes:

1. Read the data piece other than the one to be updated.
2. Generate parity from other data piece and new data.
3. Write the new parity.
4. Write the new data.

In this case, the write involves three disk operations and one parity calculation. To take advantage of this optimization, we recommend that you use three-piece RAID 5 arrays when constructing a RAID 53 array.

Choosing a cluster size

The cluster size that is chosen for striped configurations (RAID 0, 4, and 5) has a large affect on the performance of a virtual disk array. In a multiuser environment, you should adjust the cluster size to be equal to the predominant request size of the applications that are accessing the array. In this context, applications can include the operating system buffer cache, or relational database buffering mechanisms. It does not necessarily mean user programs which access a filesystem.

In a multiuser environment, the aim is for each request to affect only a single data piece. In this way, disk activity will be minimized and spread evenly between the disks. If a request begins halfway through a cluster, it will need to access two disks in the array. The effect of this will be to increase the overall disk I/O and reduce job throughput. Such a request is known as a *split job.* If this type of request occurs frequently, it may be worthwhile increasing the cluster size so that more requests fit in a single cluster. However, if you make the cluster size too large, contention between processes for access to individual clusters will also increase disk I/O and decrease job throughput.

Sometimes it is beneficial to make the dominant I/O size equal to the stripe size of a virtual disk array. Examples are a single application performing synchronous I/O, or a single-user system. This improves throughput because I/O requests are performed in parallel across all the disks in an array. The highest throughput is obtained when write requests are the same size as the stripe and are aligned on its boundaries. On RAID 4 and 5 arrays, such *full stripe* writes enhance performance because no old data needs to be read in order to generate parity.

A relational database server would appear to be an ideal application for full-stripe I/O. However, such applications often provide their own facilities for disk load-balancing and protection against disk failure. For the best possible performance, use these features in preference to virtual disk arrays. However, you should note that tuning such applications can be time consuming. Using virtual disk arrays provides a quicker and more easily configurable method of obtaining a performance improvement over single simple disks.

The **vdisk** driver keeps counts of the type of requests which have been made to an array. You can examine these counts using the **-ps** options to **dkconfig**(ADM).

In this example, **dkconfig** is used to examine the request statistics for the virtual disk array */dev/dsk/vdisk3*:

```
/dev/dsk/vdisk3:    16384 iosz    397195 reads    153969 writes    551164 io
    piece  1 /dev/dsk/2s1         404172 reads    140260 writes    544432 io
    piece  2 /dev/dsk/3s1         350326 reads    137769 writes    488095 io
    piece  3 /dev/dsk/4s1         382089 reads    135147 writes    517236 io
    piece  4 /dev/dsk/5s1         365069 reads    135808 writes    500877 io
    Job Types:
              Full Stripe              0 reads         0 writes
              Group               174463 reads     94708 writes
              Cluster             332476 reads    119464 writes
              Split Jobs          260919
    IO Sizes:
                                   16384 bytes    205180 io
                                    1024 bytes     94662 io
                                    2048 bytes     73729 io
                                    3072 bytes     48287 io
                                    8192 bytes     14454 io
                                    4096 bytes        23 io
                                   12288 bytes         4 io
                                   13312 bytes         3 io
                                    5120 bytes         2 io
                                   10240 bytes         1 io
                                    7819 resets to IO size statistics
```

The counts include:

- The dominant request size (`iosz`).
- The number of requests (reads, writes and total) made to each piece of an array.
- The number of each type of job:

  `Full Stripe`
  : read and write requests which span an entire stripe

  `Group`
  : read and write requests which occupy more than a whole cluster but less than a stripe

  `Cluster`
  : read and write requests which fit into a single cluster

  `Split Jobs`
  : number of requests which start or end in the middle of a cluster and extend over a cluster boundary

- Statistics about the number of requests (`io`) of different sizes. Use these to help you determine the optimum cluster size for your application.

The **vdisk** driver maintains a count of the ten most frequent request sizes. The counts are approximate because the driver cannot record the size of every request. When a request is made of a size which has not previously been counted, the driver throws away the counts for the five least frequently used sizes. It then uses one of the freed slots to record the new size. The driver also keeps a count of the number of times it has to reset the counters in this way. If the number of resets is a large proportion of the total number of requests, then the applications probably use a wide range of I/O sizes. In such cases, it is difficult to choose a cluster size.

If you are using the block device to access a disk array, such as when using an array for a filesystem, you may often achieve the best performance by setting the cluster size to 32 (16KB) or greater. This is because the buffer cache reads ahead 16KB.

In the example shown above, the system was running a benchmark to perform a stress test of a filesystem implemented on a RAID 5 array with a cluster size of 32. The dominant request size was 16KB as expected for access via the buffer cache but there were a comparable number of split jobs. In this case, better performance might be achieved by increasing the cluster size to 40 or 48.

Balancing disk load in virtual disk arrays

In general, virtual disk arrays tend to load disks evenly. However, it is worth using **dkconfig**(ADM) to check that the read and write counts to each of an array's pieces are even. If they are not, try changing the cluster size or the number of pieces in the array.

If a physical disk is a member of more then one array, use **sar -d** to check that the disks are being used evenly as described in "Viewing disk and other block I/O activity" (page 89). If the load is unbalanced, try rearranging the disks between the arrays.

Tuning virtual disk kernel parameters

The virtual disk driver has a number of tunable kernel parameters which affect the behavior of arrays. These parameters are listed in "Virtual disks" (page 200).

The **vdisk** driver collects a number of statistics to help you tune the values of these parameters. Use the **-Tp** options to **dkconfig**(ADM) to view these statistics.

In this example, **dkconfig** is used to examine these statistics for the virtual disk array */dev/dsk/vdisk3*:

```
Global Virtual Disk Buffer Pool
    Maximum number of buffer headers        306
    Number of free buffer headers           216
Global Virtual Disk Job Pool
    Maximum number of jobs                  200
    Number of free jobs                     177
Limits for each redundant virtual disk
    Maximum number of jobs                  100
    Maximum number of piece pool entries    100
    Maximum number of async writes           20
    Total number of hash table entries      256

/dev/dsk/vdisk3:
    Number of free jobs                      57
    Number of free piece pool entries        57
    Number of async writes                    5
    Number of async sleeps                    1
    Number of hash table tests          2005571
    Number of hash table sleeps          124824
```

The statistics show:

- The usage of buffer headers in the virtual disk buffer pool.

  The **vdisk** driver maintains its own pool of buffer headers independently of the buffer cache. When it requires a buffer header and one is not available in the buffer pool, the kernel dynamically allocates another 16KB of memory for buffer headers. It does not deallocate this memory since the buffer headers can be reused. Each buffer header requires 214 bytes of memory. In the example above, the **vdisk** driver is using 64KB of memory for buffer headers.

  There are no kernel parameters that control the size of the buffer pool. In practice, its size depends on the number of virtual disks and **VDUNITJOBS** (the maximum number of job structures that can be allocated to each virtual disk).

- The usage of job structures in the virtual disk job pool.

  When the **vdisk** driver is initialized, it allocates memory to the job structures in the job pool that is shared by all the system's virtual disks. The maximum number of available job structures is controlled by the **VDJOBS** kernel parameter. If the system runs out of free job structures, the **vdisk** driver will report "vdisk - job pool is empty" on the console. A process requiring a job structure will sleep until one becomes available.

- The limit imposed on the number of job structures that can be allocated to each redundant virtual disk array (RAID 1, 4, and 5).

  The **vdisk** driver restricts the number of job structures that a virtual disk can use. This is to ensure that the virtual disks have an equal share of the job pool. If no more jobs are available, the driver reports "vdisk - job queue is full". A process requiring a job will sleep until one becomes available.

  The maximum number of job structures per virtual disk is controlled by the kernel parameter **VDUNITJOBS**. The output from **dkconfig -Tp** shows the current number of free job structures for each virtual disk. If this value is near 0, increase the value of **VDUNITJOBS**.

- The limit imposed on the number of piece pool entries in the virtual disk piece pool for each redundant virtual disk array (RAID 1, 4, and 5).

  When a virtual disk is configured, the **vdisk** driver allocates memory to be used for the virtual disk's *piece pool*. A piece pool entry contains a piece structure for each virtual disk. Since a piece pool entry may be needed for each for each job structure, the kernel parameter **VDUNITJOBS** also defines the number of entries in each virtual disk's piece pool. If a virtual disk's piece pool becomes exhausted, the driver reports "vdisk - piece pool is empty". A process requiring an entry from the piece pool will then sleep until one becomes available.

  The output from **dkconfig -Tp** shows the current number of free piece pool entries for each disk piece. If this number is consistently near 0, increase the value of **VDUNITJOBS**.

- The maximum number of outstanding asynchronous writes to the other half of a RAID 1 mirror, or to the parity device of a RAID 4 or 5 array.

  The **vdisk** driver maintains a count of the outstanding asynchronous writes for each virtual disk. If this count exceeds the value of the kernel parameter **VDASYNCMAX**, processes attempting to write to the virtual disk will sleep until the count has been reduced. For each virtual disk, the output from **dkconfig -Tp** shows the current number of outstanding asynchronous writes and the number of times that processes had to sleep waiting for the count to be reduced. If sufficient memory is available, increase the value of **VDASYNCMAX** to prevent processes having to sleep.

If enough system buffers are available, increasing the value of **VDASYNCMAX** to 32 can increase write performance on RAID 1 configurations.

- The total number of entries in the hash table that controls access to the virtual disk stripes in each RAID 4 and 5 array.

  A hash table is used to prevent more than one process accessing a given virtual disk stripe at a time. However, since each hash entry protects more than one stripe, processes may contend for a hash table entry even when they want to access different stripes and there is no danger of data corruption.

  If a process tests a hash table entry and finds that another process may already be accessing a stripe, it sleeps until the entry is available. The output from **dkconfig -Tp** shows the number of sleeps and tests for each disk piece. If the number of sleeps is relatively high and your system is not short of memory, increase the value of the **VDHASHMAX** kernel parameter to reduce contention for hash table entries. If there is no appreciable decrease in the ratio of sleeps to tests, it is likely that the applications using the virtual disks are contending for the same stripes.

  NOTE You can use the **-z** option to **dkconfig** to zero out the hash table test and sleep counters.

  The size of each hash table entry is 24 bytes so that the default hash table size of 1024 entries requires 24KB of memory.

Serial device resources

The operation of the terminal and serial device drivers for non-intelligent serial I/O cards is shown in Figure 5-7 (page 109). Characters entered at a keyboard are received over a serial line and detected by the universal asynchronous receiver/transmitter (UART) on the serial card. On non-intelligent serial cards that use either a 8250 or a 16450 UART, a hardware interrupt is generated for every character that is received. The 16550 UART can buffer up to 16 characters and so does not generate as many interrupts per character.

The serial I/O (**sio**) driver's interrupt handling routine transfers the character(s) from the UART to the input buffer of the serial driver for every interrupt it receives. If the input stream of characters is not continuous, the 16550 will time out and generate an interrupt anyway. This prevents the interactive response experienced by a user being affected by the presence of the buffer.

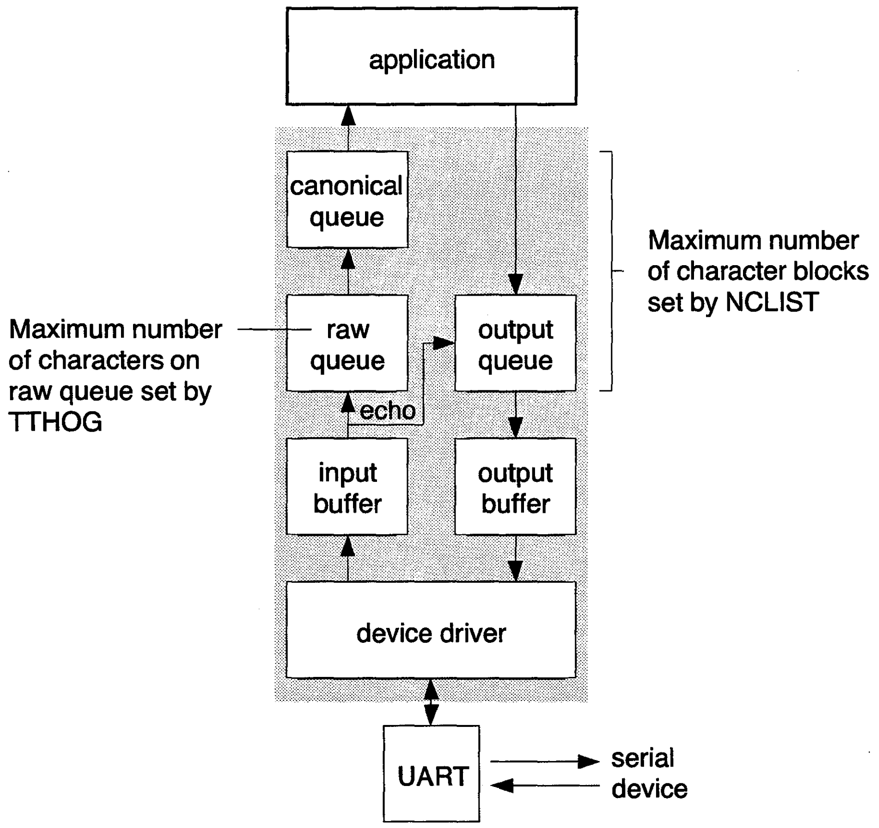


Figure 5-7 Processing of input and output characters by the character list-based terminal driver

The input characters are next moved to a character block buffer in the terminal driver (**tty**) before being moved to the raw input queue. If echoing is enabled, input characters are copied to the output queue as shown in Figure 5-7 (this page). This enables characters typed by a user to be displayed on their screen. If the line is set to canonical mode, the **tty** driver processes the characters looking for newline, keyboard signals such as ERASE (erase character), INTR (interrupt), and EOF (end-of-file), and international keyboard mapping. The processed characters are put on the canonical queue from where they can be copied to the user's process. If the terminal is set to raw mode, no characters are processed — every character typed is passed to the application.

Output characters from the application program are copied to the output queue before being moved through various output buffers, onto the UART and back to the terminal screen for display.

The main resource that you can tune is the number of character blocks used to implement the character lists used by the terminal driver. Each character block (**cblock**) consists of a 64-byte buffer and an 8-byte header. A character list (**clist**) is formed from a linked list of character blocks. It grows by adding another block to its end. The total number of character blocks that are available for use is controlled by the kernel parameter **NCLIST**. If necessary, you can increase the number of available character blocks using **setconf**(ADM) but this change only remains in force until you next reboot the kernel. Use **configure**(ADM) to make permanent changes to the number of character blocks configured.

Tuning serial device resources

You can use the **sar -y** and **sar -g** commands (or their **mpsar** equivalents for SMP) to examine the performance of the terminal (**tty**) and serial I/O drivers.

sar -y reports character processing and hardware interrupt activity in the **tty** and serial drivers:

```
23:59:44 rawch/s canch/s outch/s rcvin/s xmtin/s mdmin/s
23:59:49      10      10      25      11      11       0
23:59:54      12      10      34      14      14       0
23:59:59       8       8      12       8       8       0

Average       10       9      24      11      11       0
```

For dumb serial cards using 8250 UARTs, the ratio of the number of interrupts received from the serial card per second, `rcvin/s`, to the number of raw characters received per second, `rawch/s`, should be close to 1. Non-intelligent (dumb) serial cards that use the 16550 UART can buffer up to 14 characters before generating an interrupt. In this case, the ratio could be as low as 0.07 but will usually be nearer 1 if continuous input is not arriving. Intelligent serial cards move many more characters per interrupt and so cause much less loading of the CPU.

If the users of your system are logged in over serial lines, you should monitor `rawch/s` and `rcvin/s` over time to gain an impression of typical values for the rates of raw character input and interrupt activity for the serial card(s) on your system. If you are using intelligent multiport cards, these are usually supplied with software that you can use to diagnose their performance.

You should also examine the values of `rawch/s` and `rcvin/s` if you suspect that a bad line or chattering terminal is generating spurious interrupts. If a serial device is being used with modem control, you should also examine the number of modem interrupts per second shown by `mdmin/s`.

The CPU has to handle all interrupts that it receives from serial cards. If the number it has to handle is very high, it will be interrupt bound and may be unable to allocate sufficient time to running user applications. In extreme cases, characters may be lost if interrupts arrive while other interrupts are still waiting to be processed. This is known as interrupt overrun.

When using fast modems, too many characters may arrive in the UART's buffer while the interrupt handler is trying to process them. This is referred to as high interrupt latency. In such a case, the buffer may lose the last character to arrive. To prevent this, you need to decrease the number of characters in the buffer that will cause the UART to generate an interrupt (the receive interrupt trigger level). You should also increase the values of the kernel parameters **NCLIST** and **TTHOG** to match the increased demand on the terminal and serial drivers. **TTHOG** controls how many characters are allowed to be on the raw queue of the terminal driver (see Figure 5-7 (page 109)) before the driver will automatically dispose of them. You must also increase the number of character list blocks, defined by **NCLIST**, to allow the raw queue to grow to such a size. See "Configuring high-speed modems" in the *SCO OpenServer Handbook* for more details.

sar -g reports interrupt overruns and lost characters in the serial I/O (**sio**) driver, and any shortage of character list buffers in the **tty** driver:

```
23:59:44 ovsiohw/s ovsiodma/s ovclist/s
23:59:49      0.00       0.00      0.00
23:59:54      0.00       0.00      0.00
23:59:59      0.00       0.00      0.00

Average       0.00       0.00      0.00
```

`ovsiohw/s` shows the number of interrupt overruns per second. If this value is greater than zero and your system supports many users logged in over serial lines, you should consider upgrading to intelligent multiport cards or network terminal concentrators.

`ovsiodma/s` shows the number of times per second that the serial driver lost input characters because there was insufficient space in the receiver cache. If this value is greater than zero, you may need to decrease the interrupt trigger level and increase the values of **NCLIST** and **TTHOG**.

`ovclist/s` shows the number of times per second that the serial driver ran out of character list buffers. If this value is greater than zero, examine the number of these buffers using the command:

getconf KERNEL_CLISTS

Increase the current number of character list buffers using:

setconf KERNEL_CLISTS ***number***

until `ovclist/s` drops back to zero. Use **configure**(ADM) to change **NCLIST** to this new value.

The following table is a summary of the commands that can be used to view terminal and serial driver activity:

Table 5-6 Viewing serial and terminal driver activity

Command	Field	Description
[mp]sar -y	rawch/s	number of characters per second handled on the raw input queue
	rcvin/s	number of interrupts per second notifying that hardware has received input
	mdmin/s	number of modem interrupts per second
[mp]sar -g	ovsiohw/s	number of serial driver interrupt overruns per second
	ovsiodma/s	number of times per second that the serial driver lost input characters
	ovclist/s	number of times per second that the system ran out of character list buffers

Case study: I/O-bound multiuser system

A multiuser system in a company serves approximately 30 employees running a variety of packages including a simple database application (non-relational), an accounting package, and word processing software. At peak usage, there are complaints from users that response is slow and that characters are echoed with a noticeable time delay.

System configuration

The system configuration is:

- Uniprocessor 80486DX2 running at 66MHz.
- EISA bus.
- 32MB of RAM.
- 64MB of swap space.
- **NBUF** set to 3000.
- Two 1GB SCSI-2 hard disks.
- One 16-port and one 8-port non-intelligent serial card using 16450 UARTs.
- 22 ASCII terminals and PCs running terminal emulation software.
- One V.42 fax modem.

Defining a performance goal

The system administrator is tasked with improving the interactive performance of the system. Funds are available for upgrading the machine's subsystems if sufficient need is demonstrated. Any change to the system must be undertaken with minimal disruption to the users.

Collecting data

The administrator ensures that system accounting is enabled using **sar_enable**(ADM), and produces reports of system activity at five-minute intervals during the working week by placing the following line in *root*'s - **crontab**(C) file:

```
0  8-18  *  *  1-5  /usr/lib/sa/sa2  -s 8:00  -e 18:00  -i 300  -A
```

The administrator notes the times at which users report that the system response is slow and examines the corresponding operating system activity in the report (effectively using **sar -u**):

```
08:00:00    %usr    %sys    %wio   %idle
...
11:10:00      42      46       4       8
11:15:00      40      49       6       5
11:20:00      38      50       7       5
11:25:00      41      47       5       7
...
```

The system is spending a large amount of time in system mode and little time idle or waiting for I/O.

The length of the run queue shows that an unacceptably large number of user processes are lined up for running (**sar -q** statistics):

```
08:00:00 runq-sz %runocc swpq-sz %swpocc
...
11:10:00     4.3      85
11:15:00     7.8      98
11:20:00     5.0      88
11:25:00     3.5      72
...
```

An acceptable number of processes on the run queue would be two or fewer.

At times when the system response seems acceptable, the system activity has the following pattern:

```
08:00:00    %usr    %sys    %wio   %idle
...
16:40:00      55      20       0      25
16:45:00      52      25       2      21
16:50:00      59      20       1      20
16:55:00      54      21       2      23
...
```

This shows that the system spends little time waiting for I/O and a large proportion of time in user mode. The `%idle` figure shows more than 20% spare CPU capacity on the system. The run queue statistics also show that user processes are getting fair access to run on the CPU:

```
08:00:00 runq-sz %runocc swpq-sz %swpocc
...
16:40:00     1.0      22
16:45:00     2.1      18
16:50:00     1.6       9
16:55:00     1.1      12
...
```

Formulating a hypothesis

From the CPU utilization statistics, it looks as though the system is occasionally spending too much time in system mode. This could be caused by memory shortages or too much overhead placed on the CPU by peripheral devices. The low waiting on I/O figures imply that memory shortage is not a problem. If the system were swapping or paging, this would usually generate much more disk activity.

The administrator next examines the performance of the memory, disk and serial I/O subsystems to check on their performance.

Getting more specifics

The memory usage figures for the period when the proportion of time spent in system mode (`%sys`) was high show the following pattern (**sar -r** statistics):

```
08:00:00 freemem freeswp
...
11:10:00    1570  131072
11:15:00    1612  131072
11:20:00    1598  131072
11:25:00    1598  131072
...
```

The value of **GPGSHI** for this system is 300 and none of the swap space is allocated to processes — there is no apparent evidence of swapping or paging to disk. This is confirmed by examining the reports for **sar -w**:

```
08:00:00 swpin/s bswin/s swpot/s bswot/s pswch/s
...
11:10:00    0.04     0.2    0.00     0.0      51
11:15:00    0.02     0.1    0.00     0.0      63
11:20:00    0.00     0.0    0.00     0.0      56
11:25:00    0.01     0.1    0.00     0.0      66
...
```

The zero values for `swpot/s` and `bswot/s` indicate that there was no swapping out activity.

Examining the **sar -q**, **sar -r** and **sar -w** reports at other times shows occasional short periods of paging activity but these are correlated with batch payroll runs. It should be possible to reduce the impact of these on the system by rescheduling the jobs to run overnight.

The administrator next examines the buffer cache usage statistics for the same period (**sar -b** statistics):

```
08:00:00 bread/s lread/s %rcache bwrit/s lwrit/s %wcache pread/s pwrit/s
...
11:10:00      27     361      93       5      16      68       0       0
11:15:00      35     320      89       7      22      66       0       0
11:20:00      22     275      92       5      15      65       0       0
11:25:00      22     282      96       9      27      67       0       0
...
```

These figures show hit rates on the buffer cache of about 90% for reads and 65% for writes. Approximately 30KB of data (`bread/s` + `bwrit/s`) is being read from or written to disk per second.

Disk performance is examined next using the statistics provided by **sar -d**:

```
08:00:00 device %busy    avque    r+w/s    blks/s   avwait   avserv
11:10:00 Sdsk-0  0.91    3.70      2.37    13.15    12.42    4.60
         Sdsk-1 25.01    1.62     11.39    55.21     3.26    5.30

11:15:00 Sdsk-0  0.57    2.58      1.37     6.98    13.05    8.26
         Sdsk-1 24.10    1.43     10.93    50.42     3.11    7.23

11:20:00 Sdsk-0  0.81    2.42      1.98    11.01     9.55    6.72
         Sdsk-1 21.77    1.85      6.05    39.11     4.54    5.37

11:25:00 Sdsk-0  0.76    3.90      2.00     9.52    14.18    4.89
         Sdsk-1 20.24    2.07      5.83    34.87    10.60    9.91
```

These results show that the busiest disk (Sdsk-1) has acceptable performance with a reasonably short request queue, acceptable busy values, and low wait and service times. The pattern of activity on the root disk (Sdsk-0) is such that the request queue is longer since requests are tending to arrive in bursts. There is no evidence that the system is disk I/O bound though it may be possible to improve the interactive performance of some applications by increasing the buffer cache hit rates.

Making adjustments to the system

Based on the evidence given above, the system would benefit from increasing the number of buffers in the buffer cache. Although the system does not show much sign of being disk I/O bound (**sar -u** shows `%wio` less than 15% at peak load), applications are placing a reasonably heavy demand on the second SCSI disk (Sdsk-1). This will affect the interactive response of programs which have to sleep if the data being requested cannot be found in the buffer cache. As the system does not appear to be short of memory at peak load, the system administrator may wish to experiment with doubling the size of the buffer cache by setting **NBUF** to 6000. Based on the evidence from **sar -r** that approximately 6MB (1500 4KB pages) of memory are free at peak load, doubling the size of the buffer cache will reduce this value to about 3MB. If the size of the buffer cache is increased, the system should be monitored to see:

- if the proportion of time spent waiting for I/O decreases (`%wio` reported by **sar -u**)
- if the buffer cache hit rates improve (`%rcache` and `%wcache` reported by **sar -b**)
- if disk activity on the second SCSI disk decreases appreciably as a result of increasing the number of buffers (`%busy` reported by **sar -d**)
- if the system is becoming short of memory (**sar -r** reports that `freemem` is dropping near to or below the value of **GPGSHI**)

If increasing the size of the buffer cache starts to make the system swap or page out intensively, the administrator should either reduce its size again or make more memory available in other ways.

If the interactive performance of applications is still less than desired, another possibility is to use intelligent serial I/O cards to relieve the processing overhead on the CPU. The serial multiport cards use 16450 UARTs and were previously used in two less powerful systems. It is possible that the CPU is spending too much time moving characters out to the serial lines on behalf of the serial cards. The CPU will do this whenever the applications need to refresh terminal screens to update database forms, word processor displays and so on.

Case study: unbalanced disk activity on a database server

This case study examines a database management system that uses raw I/O; many databases provide the ability to perform I/O through the raw device rather than using block I/O through the buffer cache.

The system in this study is not used to run any other applications or for developing applications. The study uses a benchmark written in-house to tune the system's performance. It is understood that a benchmark is only an approximation to the demands that will be placed upon the database in real life.

System configuration

The system configuration is:

- Multiprocessor — four 80486DX2 processors running at 66MHz.
- EISA bus.
- 64MB of RAM.
- 64MB of swap space — note that this system has been configured with a swap space that is smaller than the minimum of twice the amount of RAM (128MB) that is normally recommended.
- Six 1GB SCSI-2 hard disk drives.
- One Bus Mastering DMA Ethernet network card with a 16KB buffer and 32-bit wide data path.
- Two terminal concentrators supporting a total of 128 users on a variety of ASCII terminals and personal computers running terminal emulation software.

Additional software that has been installed on the system includes SCO SMP License, a relational database management system, and transaction compression software.

The configuration of the disk drives is:

Sdsk-0	boot, swap, root, and applications divisions
Sdsk-1 through Sdsk-5	database tables, indexes, and journal logs configured as RAID 5 (striped with parity) to balance disk load

Defining a performance goal

The database supports the main activities of an organization, and transaction response time is critical. Desirable performance goals for such a system are:

- Do not allow the system to start swapping or paging out.
- Minimize time spent waiting for I/O.
- Maximize the time that the database management processes spend on the CPUs.
- Minimize system overhead.
- Minimize contention for resources between the CPUs.
- Ensure that sufficient kernel resources are available to the database management software (such as IPC facilities, STREAMS, and so on).
- Maximize the amount of physical memory that is available for the database's internal work areas.

Collecting data

The database administrator runs **mpsar** to collect system statistics every 5 minutes while running the benchmark:

mpsar -o /tmp/sar_op 300 25

After the benchmark run finishes, **cpusar -u** is run to examine the activity of the operating system on each CPU. For example, on the base processor, the command is:

cpusar -P 1 -u -f /tmp/sar_op

The output below illustrates the status of the first processor; reports for the other processors recorded similar results:

```
02:30:00    %usr    %sys    %wio   %idle
...
02:55:00      34      14      25      27
03:00:00      32      11      32      25
03:05:00      27       8      41      24
03:10:00      31      11      32      25
...
```

Formulating a hypothesis

The information provided by **cpusar** indicates that the processors are spending a significant amount of time waiting for I/O. Since it is observed that the disk drive busy lights are on virtually all of the time, this indicates that the disks may not be keeping up with the rest of the system. It is known that the benchmark does not test the networking I/O subsystems since it generates activity only on the server. For this reason, network I/O can be eliminated as a possible cause.

From the available evidence, it looks as if the problem is in the disk subsystem. Sometimes an I/O problem is a symptom of an underlying memory shortage. A more likely cause of the poor I/O performance is the distribution of activity among the system's disks. However, it is a good policy to eliminate memory shortage as the origin of the performance degradation.

Getting more specifics

In this example, there is a high amount of RAM, so it is unlikely that memory is the root of the problem. To make certain, the administrator runs **mpsar -r** to check on the amount free memory and swap while the benchmark was being run:

```
02:30:00 freemem freeswp
...
02:55:00    2012  131072
03:00:00    2004  131072
03:05:00    2098  131072
03:10:00    2009  131072
...
```

There was approximately 8MB of free memory and no swap space used during the period when the benchmark was run.

The administrator now concentrates on examining disk usage. It is unlikely that buffer cache usage is causing a problem as this has already been tuned for the root and applications filesystems. The database itself does not use the buffer cache as it uses asynchronous I/O access to disk.

The **mpsar -d** command is run to show disk usage during the benchmark:

```
02:30:00 device  %busy    avque    r+w/s    blks/s    avwait   avserv
...
02:55:00 Sdsk-0   0.91     2.11    12.37     23.11     3.24     2.91
         Sdsk-1  85.24     4.13    39.93    155.12    79.62    25.44
         Sdsk-2  78.02     4.14    41.10    160.82    63.50    20.21
         Sdsk-3  85.11     4.36    34.54    167.54    80.85    24.05
         Sdsk-4  89.34     4.42    37.00    156.91    85.26    24.92
         Sdsk-5  83.59     4.48    40.41    159.34    60.11    17.26

03:00:00 Sdsk-0   0.76     2.42     8.37     13.64     8.35     5.88
         Sdsk-1  90.10     4.42    42.37    156.67    65.71    19.19
         Sdsk-2  85.42     5.38    39.09    160.81    79.96    18.24
         Sdsk-3  88.29     4.62    37.65    163.57    83.27    23.00
         Sdsk-4  85.99     5.29    41.11    166.28    79.21    18.45
         Sdsk-5  99.54     4.21    43.09    170.21    56.76    17.67

03:05:00 Sdsk-0   0.70     4.28     8.07     11.95     7.32     2.23
         Sdsk-1  88.01     4.03    40.01    167.76    63.27    20.91
         Sdsk-2  89.18     3.82    38.97    156.08    65.44    23.20
         Sdsk-3  93.02     4.53    39.55    163.20    80.56    22.81
         Sdsk-4  85.33     4.07    37.03    154.31    78.70    25.66
         Sdsk-5  91.14     4.87    41.34    164.40    59.02    15.24

03:10:00 Sdsk-0   0.79     2.94     9.60     16.23     6.30     3.67
         Sdsk-1  87.75     4.20    40.77    159.85    69.53    21.85
         Sdsk-2  83.82     4.52    39.72    159.24     69.63   20.55
         Sdsk-3  88.81     4.50    37.25    164.77     81.56   23.29
         Sdsk-4  86.89     4.59    38.38    159.17     81.06   23.01
         Sdsk-5  91.42     4.52    41.61    164.65     58.63   15.98
...
```

These results indicate that the system is I/O bound in the disk subsystem. The average service time (`avserv`) on the root disk (Sdsk-0) is much lower than that on the other disks — even though they are physically identical. There is no significant variation between activity on disks Sdsk-1 through Sdsk-5 — this is as expected given their RAID configuration which is designed to balance activity across several disks.

Making adjustments to the system

The large service times to the disks containing the database tables, indexes, and journal logs seem to indicate that large disk seek times are having an impact here.

Database journal logs are written to sequentially so it may impair disk performance if they are striped across the same disks as the tables and indexes. The effect will be to mix requests to write to the logs with those to access the tables and indexes. The requests will then be spread across all the disks in the array, losing all coherency between requests to write to the logs. Instead, each log should be placed on a dedicated disk so that the next block to be written to will usually be immediately available without having to move the drive head. This also makes sense for reducing load on the other disks when you consider that most writes on a database server tend to be to the logs.

The recommendation for reconfiguring this system would be to move the journal logs to dedicated disks. Since requests to access tables and indexes are generally random, the maximum performance from the disks that they occupy will probably be obtained by leaving these in a RAID 5 configuration. Journal logs on a database server are usually assigned to a dedicated mirrored disk configuration, such as RAID 1, to ensure data integrity.

Once the disk layout has been reconfigured, the benchmark should be run again to test the hypothesis. If the performance of the dedicated journal log disks is still poor, the disks should be upgraded to ones with lower access times. These disks will also benefit from write-back caching provided that the integrity of the data is protected by a backup power supply.

Database performance tuning

There is little to be gained from tuning the operating system's buffer cache for a database management system that uses raw I/O. However, the database management system may maintain its own buffer caches which you can tune independently. Refer to the performance tuning documentation that was supplied with the database management system for more information. Some databases also maintain profiling files that indicate how many jobs are going to each device. You may also find this useful in balancing the load on the system.

Use of shared data segments

If your database caches its internal information in IPC shared memory segments, you may need to increase the shared memory kernel parameters **SHMMAX** and **SHMMNI**. See "Shared memory" (page 218), "Shared memory parameters" (page 218) and your database documentation for more information.

Chapter 6 *Tuning networking resources*

Networking protocol stacks and the programs which run over them place additional burdens on your system's resources, including CPU and memory. This chapter describes the areas of concern for the network administrator, the tools used to diagnose performance problems, and procedures used to enhance network performance for STREAMS, SCO TCP/IP, SCO NFS, and LAN Manager Client Filesystem. See the following sections for a discussion of tuning these and other subsystems:

- "STREAMS resources" (this page)
- "TCP/IP resources" (page 131)
- "NFS resources" (page 142)
- "LAN Manager Client Filesystem resources" (page 154)
- "Other networking resources" (page 157)

See also:

- "Introduction to networking" in the *Networking Guide*

STREAMS resources

The X Window System, networking services such as TCP/IP and NFS, applications that use streams pipes, and certain device drivers use STREAMS to perform I/O.

The STREAMS I/O system was designed to provide a simultaneous two-way (full duplex) connection between a process running in user space and a device driver (or pseudo-device driver) linked into the kernel. The topmost level within the kernel with which the user process communicates is known as the stream head.

Using STREAMS has the advantage that it allows the processing of I/O between an application and a device driver to be divided into a number of functionally distinct layers such as those required by network architectures that implement TCP/IP or the Open Systems Interconnection (OSI) 7-layer model.

The STREAMS I/O mechanism is based on the flow of messages from the stream head to a device driver, and in the opposite direction, from the device driver to the stream head. Messages that are passed away from the stream head toward the driver are said to be traveling downstream; messages going in the opposite direction are traveling upstream. Between the stream head and the driver, there may be a number of stream modules which process messages in addition to passing them to the next module. Each type of module is implemented as a separate driver linked into the kernel. For example, the **udp** driver implements the network module that applies the UDP protocol to messages. Each module has two separate queues for processing upstream and downstream-bound messages before handing them to the next module.

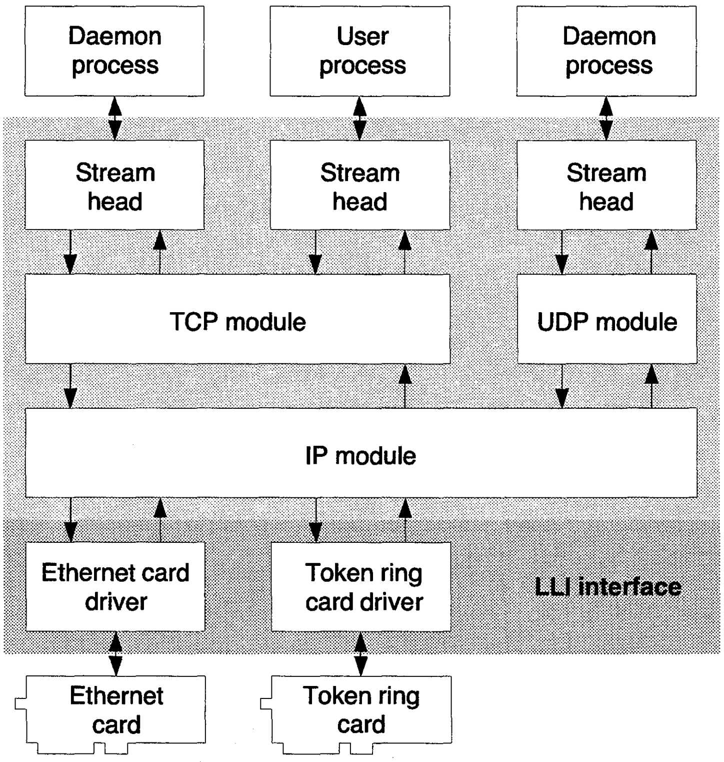


Figure 6-1 Implementation of networking protocols using STREAMS

A network protocol stack is built by pushing successive protocol modules below the stream head. A protocol stack that implements the TCP/IP networking protocols can be built by pushing an Internet Protocol (IP) module, and a Transmission Control Protocol (TCP) module below the stream head. Modules can also be multiplexed so that a module can talk to several stream heads, drivers or other modules. Figure 6-1 (page 124) shows:

- Two stream heads talking to the same TCP module.
- The TCP and User Datagram Protocol (UDP) transport layer modules both connected to the underlying IP module.
- Two different network adapter drivers interfaced to the IP module; this is necessary on systems that act as routers between two networks that use the Internet Protocol.

Note that the diagram simplifies the Link Layer Interface (LLI) for clarity. This layer consists of the Data Link Provider Interface (DLPI) and the SCO MAC Driver Interface.

For a more complete picture of the available protocol stacks and drivers, see "Network hardware drivers" in the *Networking Guide.*

Figure 6-2 (page 126) shows how the TCP/IP protocol stack encapsulates data from an application to be sent over a network that uses Ethernet as the physical layer. The Transport layer module adds a header to the data to convert it into a TCP segment or a UDP packet. The Internet layer module turns this into an IP datagram, and then passes it to the network driver which adds a header and CRC trailer. The resulting Ethernet frame is then ready for transmission over the physical medium.

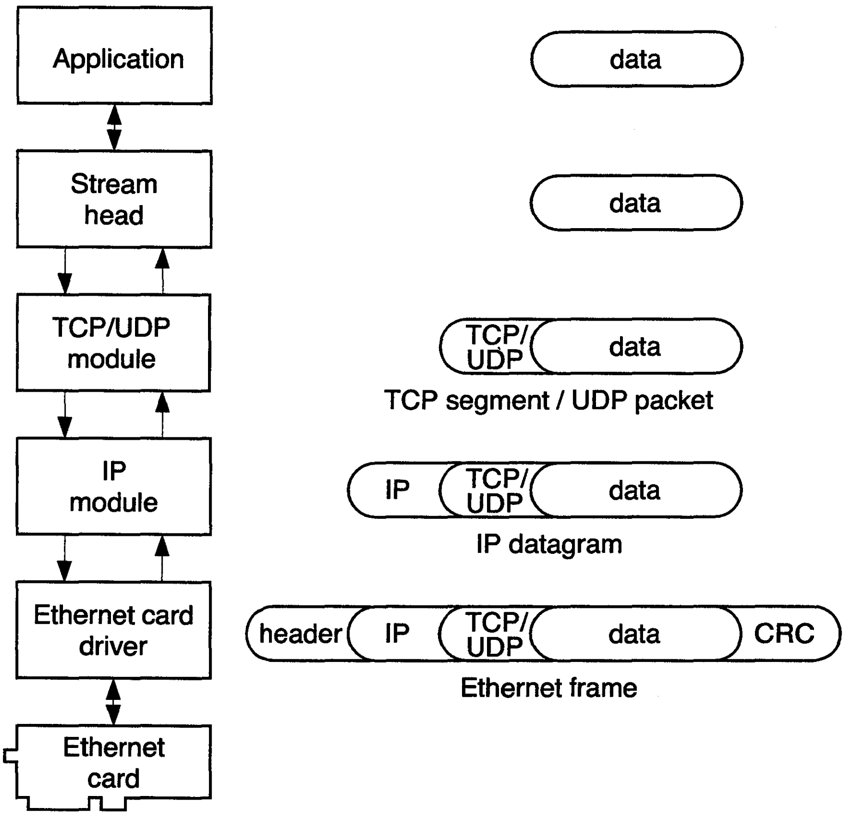


Figure 6-2 Creating an Ethernet frame by successive encapsulation

To retrieve data from an Ethernet frame, the inverse process is applied; the received information is passed as a message upstream where it is processed by successive modules until its data is passed to the application. If the information is received by a router between two networks, the message will only travel upward as far as the Internet layer module from one network adapter before being passed back down to a different network adapter.

Figure 6-3 (page 127) shows protocol stacks on two machines linked via a physical connection technology such as Ethernet, Token Ring or Fiber Distributed Data Interface (FDDI). Applications appear to have a direct or virtual connection; they do not need to know how connection is established at the lower levels.

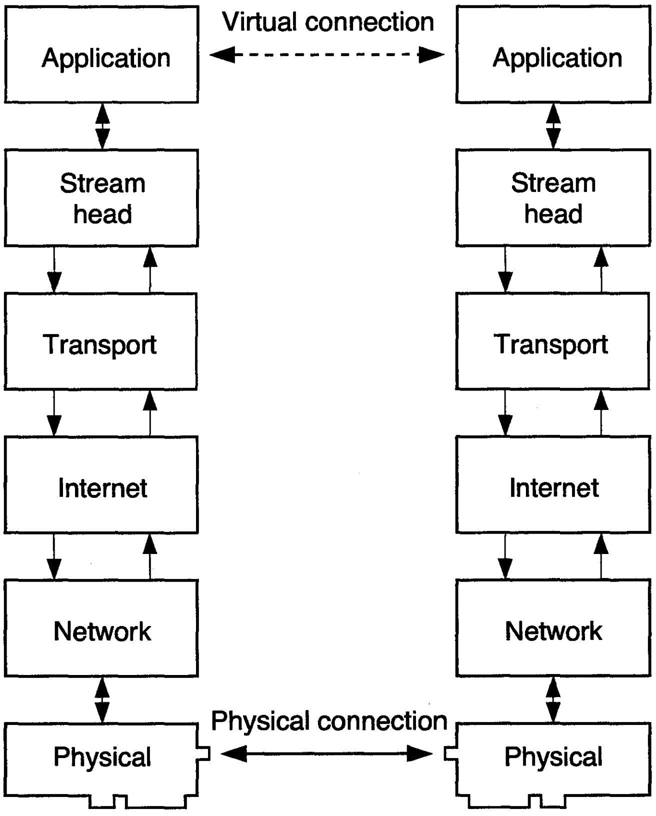


Figure 6-3 Virtual and physical connections over a network

The primary usage of memory by the STREAMS subsystem is for building messages. Figure 6-4 (page 128) illustrates how a message is created from pieces dynamically allocated from the memory reserved for use by STREAMS. Each message consists of a fixed-size message header and one or more buffer headers attached to buffers. The buffers come in several different sizes and contain the substance of the message such as data, **ioctl** control commands (see **ioctl**(S), and **streamio**(M)), acknowledgements, and errors.

Message buffers are available in 15 sizes or classes:

- 16-byte class 0 buffers are stored within the buffer header.
- Class 1 to class 14 buffers, ranging in size of powers of 2 from 64-byte to 512KB, use a separately allocated buffer structure pointed to by the buffer header.

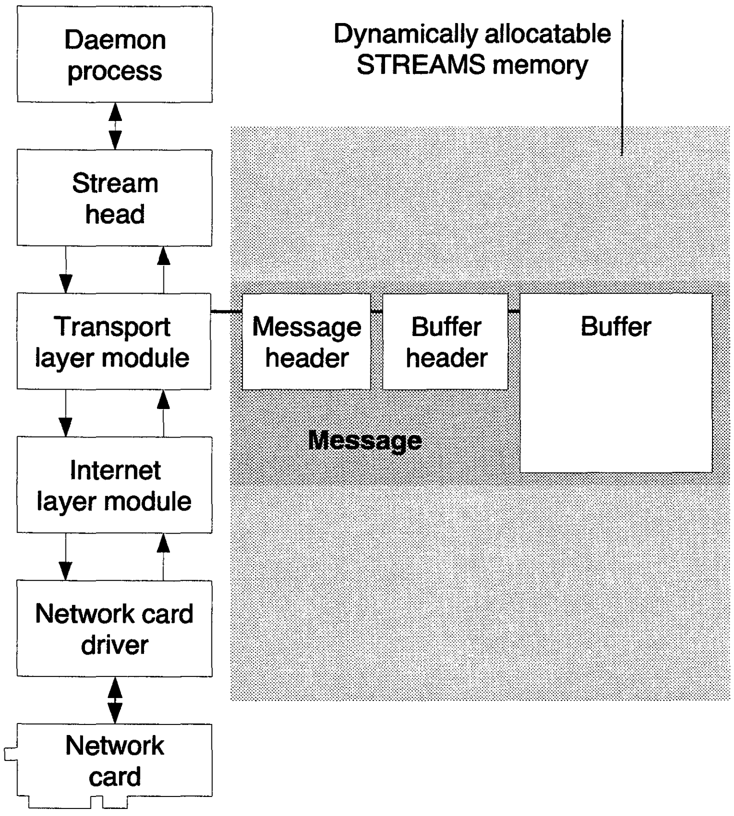


Figure 6-4 Memory structures used by STREAMS messages

Three kernel parameters are important for the configuration of STREAMS: **NSTRPAGES**, **STRSPLITFRAC**, and **NSTREAM**.

NSTRPAGES controls the total amount of physical memory that can be made available for messages. The kernel can dynamically allocate up to **NSTRPAGES** pages of memory for message headers, buffer headers, and buffers. If a message needs a buffer which is not currently available on the free list of buffers, a new buffer is dynamically allocated for use from memory. If more than **STRSPLITFRAC** percent of **NSTRPAGES** is in use and a suitable buffer is not available on the free list, the kernel will try to split a larger buffer for use and only allocates more memory if this fails.

The default value of **STRSPLITFRAC** is 80%; if you set this value lower, STREAMS will use less memory which will tend to become fragmented more quickly. When this happens, unallocated STREAMS memory exists as many small non-contiguous pieces which are unusable for large buffers. The STREAMS daemon, **strd**, manages memory on behalf of the STREAMS subsystem. If **strd** runs, it expends CPU time in system mode in order to release pages of STREAMS memory for use (this is known as garbage collection).

NSTREAM controls the number of stream heads that can be used. One stream head is needed for each application running on your machine that uses STREAMS to establish connections. Applications that use stream pipes require two stream heads per pipe.

Examples of applications that use stream heads are:

- Remote login and file transfer programs such as **ftp**(TC), **rcmd**(TC), **rcp**(TC), **rlogin**(TC), and **telnet**(TC).
- Remote X clients.
- The power management daemon, **pwrd**(ADM), and each APM or UPS driver that talks to it.

NOTE **NSTREAM** should be set to at least 256 on systems running SCO OpenServer software which mount several remote filesystems or invoke remote X clients. A program will not run if it cannot obtain a stream head and will output a message such as:

```
NOTICE: program: out of streams
```

If you see such a message, increase the value of **NSTREAM**, relink the kernel, and reboot.
Each configured stream head requires 80 bytes of memory. Apart from this overhead, the value of **NSTREAM** has no effect on performance.

See also:

- "Networking protocol stacks" in the *Networking Guide*

Monitoring STREAMS performance

Your SCO OpenServer system uses the STREAMS mechanism to support TCP/IP and other network protocols such as IPX/SPX. You should ensure that you provide an appropriate number of STREAMS resources for TCP/IP and IPX/SPX; without them, performance may suffer or the system may hang.

Run the **netstat -m** command to display STREAMS memory usage:

```
streams allocation:
                          config   alloc    free     total    max     fail
streams                     160      84      76       215      87       0
queues                      452     394      58       496     414       0
mblks                       271     102     169     49326     183       0
buffer headers              442     391      51      5964     395       0
class  1,      64 bytes      64       0      64     20289      44       0
class  2,     128 bytes      96       0      96      8668      72       0
class  3,     256 bytes      64       7      57      7174      63       0
class  4,     512 bytes      32       8      24      1334      25       0
class  5,    1024 bytes       4       0       4       904       3       0
class  6,    2048 bytes     104      62      42       622     103       0
class  7,    4096 bytes       8       8       0        93       8       0
class  8,    8192 bytes       1       0       1        13       1       0
class  9,   16384 bytes       1       0       1         1       1       0
class 10,   32768 bytes       0       0       0         0       0       0
class 11,   65536 bytes       0       0       0         0       0       0
class 12,  131072 bytes       0       0       0         0       0       0
class 13,  262144 bytes       0       0       0         0       0       0
class 14,  524288 bytes       0       0       0         0       0       0
total configured streams memory: 2000.00KB
streams memory in use: 185.98KB
maximum streams memory used: 334.43KB
```

Tuning STREAMS usage

For each data structure used, the important column is the `fail` column shown by **netstat -m**. If this is non-zero for the number of stream heads configured (shown as the value in the row labeled `streams` under the `config` column), increase the value of **NSTREAM** using **configure**(ADM) as described in "Using configure to change kernel resources" (page 189) and "STREAMS" (page 209).

The amount of memory currently in use by STREAMS, and the maximum amount used since the system was started are shown at the bottom of the output from **netstat -m**.

The figure for the total memory configured for use by STREAMS represents an upper limit to the amount of memory that can be dynamically allocated for use.

If there are several non-zero entries in the `fail` column and the amount of memory in use by STREAMS is almost the same as the total amount of memory configured for STREAMS, increase the value of **NSTRPAGES**. This parameter controls the number of 4KB pages of physical memory that can be dynamically allocated for use by STREAMS.

NOTE In this release, memory used for STREAMS message headers, buffer headers, and buffers is dynamically allocated from memory. There is no need to tune the numbers of these resources individually.

The following table summarizes the commands that you can use to examine STREAMS usage:

Table 6-1 Examining STREAMS performance

Command	Field	Description
netstat -m	fail	number of times a STREAMS resource was unavailable

TCP/IP resources

The TCP/IP protocol suite consists of the Transmission Control Protocol (TCP), the Internet Protocol (IP), and other protocols described in "TCP/IP" in the *Networking Guide*. The TCP/IP protocol suite is implemented using STREAMS. You should ensure that sufficient STREAMS resources are available for networking to function correctly as described in "STREAMS resources" (page 123).

See also:

- "Networking protocol stacks" in the *Networking Guide*

Tuning TCP/IP performance

The IP protocol stack is configured to maximize performance on all supported network adapters. If desired, you can further adjust performance parameters for each network interface using the **ifconfig**(ADMN) command as described in "Using ifconfig to change parameters for a network card" (page 225). This command allows you to adjust:

- The send and receive TCP window for an interface. These windows are used by two communicating systems to negotiate the amount of data that can be sent before an acknowledgement is required. The default values of these windows are set to optimize performance on a local area network (LAN). If you are using a high bandwidth, and high latency connection such as a satellite link, increase the values of these parameters to increase throughput on the link. The maximum value that you can set is 64KB.
- The maximum segment size (MSS) rounding parameter. This is a boolean value; if set to 1, TCP negotiates the largest segment size that can be transmitted in the maximum transmission unit of the physical network. This is also referred to as using full frames. If set to 0, the MSS is rounded down to the nearest power of 2. For Ethernet, this corresponds to 1KB.

With modern Ethernet hardware, you should use full frames to maximize the amount of data per Ethernet frame. On older Ethernet cards with small buffers and narrow data paths, rounding down should be selected to enable the data in the Ethernet frame to be moved into the card's buffer more efficiently.

Token Ring networks have a much larger MTU than Ethernet; full frames should always be used.

- One-packet mode. This should be set for older network adapters whose small buffers cannot handle back-to-back streams of packets.
- Time-to-live. If it is known that a substantial number of network hops will be necessary for a packet to reach its destination, increase this parameter.

You can adjust systemwide TCP/IP parameters using **inconfig**(ADMN) as described in "Using inconfig to change global TCP/IP parameters" (page 226).

Problems with TCP/IP may be experienced if:

- There are insufficient STREAMS resources. These may be investigated as described in "STREAMS resources" (page 123).
- There is too much activity on the network. Break the network into smaller subnetworks, or move network-intensive client-server applications onto dedicated machines. These issues are considered further in "Configuring network topology for performance" (page 137).
- There is intermittent loss of connection due to the network being incorrectly configured physically. See "Testing network connectivity" (page 136) for ways of testing this.

Other performance considerations for TCP/IP include:

- Using the Internet routing discovery daemon (**irdd**) instead of **routed** or **gated** as described in "Configuring routing for performance" (page 140).
- Altering the functionality of the domain name server to decrease system load, or to balance the load between the network and the local machine as described in "Configuring DNS name service for performance" (page 141).
- Ensuring that serial line communications (SLIP and PPP) operate at peak performance as described in "Tuning SLIP performance" (page 135) and "Tuning PPP performance" (page 136).

The main tool for investigating the performance of TCP/IP is **netstat**(TC) as described in "Monitoring TCP/IP performance" (page 133).

See also:

- "Troubleshooting TCP/IP" in the *Networking Guide*

Monitoring TCP/IP performance

The most useful command for examining TCP/IP performance (and that of other protocol stacks) is **netstat**(TC). This command displays the contents of various networking-related data structures held in the kernel. We have already encountered the **-m** option to **netstat** in "Monitoring STREAMS performance" (page 129) where it allowed us to see how STREAMS resources were allocated inside the kernel.

The command **netstat -i** displays the status of the system's network interfaces. (To view only a single interface, specify this using the **-I** option.) The output from this command has the following form:

```
Name  Mtu   Network    Address     Ipkts Ierrs    Opkts Oerrs  Collis
sme0  1500  reseau     paris      996515     0   422045    42       0
lo0   2048  loopback   loopback    25436     0    25436     0       0
```

The important fields are `Ierrs`, `Oerrs`, and `Collis`.

`Ierrs` is the number of received packets that the system recognized as being corrupted. This usually indicates faulty network hardware such as a bad connector, incorrect termination (on Ethernet), but it may also be caused by packets being received for an unrecognized protocol. For network adapters with small buffers, it may mean that they have been saturated by end-to-end streams of packets. In this case, you should switch the network interface to one-packet mode using the **ifconfig**(ADMN) command as described in "Using ifconfig to change parameters for a network card" (page 225).

`Oerrs` is the number of errors that occurred while the system was trying to transmit a packet. This generally indicates a connection problem. On Ethernet, it may also indicate a prolonged period of time during which the network is unusable due to packet collisions.

`Collis` is the number of times that the system (connected to a network using Ethernet as its physical medium) detected another starting to transmit while it was already transmitting. Such an event is called a packet collision. The ratio of the number of collisions to the number of output packets transmitted gives a indication of the loading of the network. If the number of `Collis` is greater than 10% of `pkts` for the most heavily used systems on the network, you should investigate partitioning the network as described in "Configuring network topology for performance" (page 137).

Networks implemented using Token Ring and FDDI technology use a different protocol to communicate at the physical layer and do not experience packet collisions. The value in the `Collis` field should be zero for such networks.

See "Troubleshooting TCP/IP" in the *Networking Guide* for a full discussion of these issues.

NOTE You can also use the **ndstat**(ADM) command to obtain similar information to that displayed by **netstat -i**.

The following table summarizes the commands that you can use to examine the performance of TCP/IP:

Table 6-2 Examining TCP/IP performance

Command	Field	Description
netstat -i	Ipkts	number of network packets received
	Ierrs	number of corrupted network packets received
	Opkts	number of network packets transmitted
	Oerrs	number of errors while transmitting packets
	Collis	number of packet collisions detected

Configuring TCP/IP daemons for performance

If TCP/IP is configured, your system runs the */etc/rc2.d/S85tcp* script each time it goes to multiuser mode. (Note that this file is a link to */etc/tcp*.) This script starts several TCP/IP daemons. If configured to run, the following daemons may affect performance:

gated handles routing and supports a variety of routing protocols.

irdd provides Internet routing discovery.

routed handles routing by default. **routed** may be commented out of */etc/tcp* if your system uses **irdd**(ADMN) to maintain routing information. Note that all systems to which you are networked must be able to handle **icmp**(ADMP) routing. See Chapter 11, "Configuring Internet Protocol (IP) routing" in the *Networking Guide* for a full discussion of the **gated, irdd,** and **routed** daemons.

named provides Domain Name Service (DNS). **named** has many performance implications. See "Configuring DNS name service for performance" (page 141) for information on configuring DNS to use **named**.

rwhod provides the remote **who** facility, see **rwho**(TC). **rwhod** is commented out of */etc/tcp* for performance reasons. Uncommenting this daemon generates additional network traffic as the daemon queries the system for user and uptime information and broadcasts this data to the network.

snmpd implements the simple network management protocol (SNMP). **snmpd** runs by default. It generates additional packets during startup and other unusual system events. It monitors and responds to SNMP traffic from other machines. If you do not want SNMP running on your system, use the **SNMP Agent Manager** to turn off the SNMP agent as described in "Configuring SNMP with the SNMP Agent Manager" in the *Networking Guide.*

Tuning SLIP performance

To maximize performance of a connection over a SLIP link, do the following:

- Select Van Jacobson (VJ) TCP/IP header compression using the **+c** option to **slattach**(ADMN) if the incoming connection expects this to be set or it can automatically detect compressed packets.
- Select automatic detection of TCP/IP header compression using the **+e** option to **slattach** if the incoming connection uses compressed packets.

NOTE If both ends of a connection use automatic detection, header compression will not be used. At least one end must explicitly choose to use compression using the **+c** option.

- Use the **-m** ***mtu*** option to **slattach** to set the maximum transmission unit for the link. The default value is 296 bytes — 40 bytes for the header plus 256 bytes of data. Increase this value if you are not using the link as an interactive connection. For example, if you are transferring data, you might set this to 1064. The minimum possible value is 42 bytes. The maximum suggested value is 1536 bytes.

For a complete discussion of using SLIP, see Chapter 5, "Configuring the Serial Line Internet Protocol (SLIP)" in the *Networking Guide.*

Tuning PPP performance

To maximize performance of a connection over a PPP link, do the following:

- Enable hardware flow control on the modem being used. It must be connected to a modem control port such as */dev/tty1A* or */dev/tty2A*.
- Use the **Network Configuration Manager** (see Chapter 25, "Configuring network connections" in the *SCO OpenServer Handbook*) to turn on Van Jacobson (VJ) TCP/IP header compression, enable the maximum number (16) of VJ compression slots, and enable compression of these slots. See Chapter 4, "Configuring the Point-to-Point Protocol (PPP)" in the *Networking Guide* for more information about other compression features that you can enable.
- Set the maximum receive unit (MRU) to 296 for interactive inbound or outbound connections. Set this higher, to 1064 for example, if the link is only being used to transfer data. The maximum suggested value is 1536 bytes.

You can also edit the file */etc/ppphosts* to configure these parameters; see **ppphosts**(SFF) for more information.

For a complete discussion of using PPP, see Chapter 4, "Configuring the Point-to-Point Protocol (PPP)" in the *Networking Guide*.

Testing network connectivity

The **ping**(ADMN) command is useful for seeing if a destination machine is reachable across a local area network (LAN) or a wide area network (WAN). If you are *root*, you can use the flood option, **-f**, on a LAN. This sends a hundred or more packets per second and provides a stress test of the network connection. For every packet sent and received, **ping** prints a period (.) and a backspace respectively. If you see several periods being printed, the network is dropping packets.

If you want to find out how packets are reaching a destination and how long this takes, use the **traceroute**(ADMN) command. This provides information about the number of hops needed, the address of each intermediate gateway, and the maximum, minimum and average round trip times in milliseconds. On many hop connections, you may need to increase the maximum time-to-live (TTL) and wait times for the probe packets that **traceroute** sends out. To do this, use the **-m** and **-w** options.

See also:

- Chapter 10, "Testing connectivity with other sites" in the *Networking Guide*

Configuring network topology for performance

The types and capabilities of Ethernet network technology (as defined by the *IEEE 802.3* standard) are shown in the following table:

Table 6-3 Ethernet network technologies

Type and alternative names	Topology and medium	Maximum segment length	Maximum number of nodes per segment
10Base5, Thick-Net	linear, 50 ohm 10mm coaxial cable terminated at both ends	500m	100
10Base2, ThinNet, CheaperNet	linear, 50 ohm 5mm coaxial cable terminated at both ends	185m	30
10Base-T, twisted pair	star, unshielded twisted pair	100m	2

For Ethernet technologies that use a linear network topology, the cable must not have any branches or loops and it must be correctly terminated at both ends.

To attach nodes to the network, 10Base5 connects drop cables to vampire taps directly attached to the coaxial cable or to transceiver boxes placed in line with the cable.

10Base2 T-piece connectors must be connected directly to the coaxial terminal of the network card — that is, you cannot use a coaxial cable as a drop cable.

If you want to extend the length of an Ethernet cable segment, there are three ways of doing this:

- Repeaters retransmit network packets (including any electrical noise) and connect network segments at the physical layer. They do not separate network traffic but they can be used for connecting different network media, for example, to connect 10Base2 and 10Base5.
- Bridges also connect network segments at the physical layer but they can be used to filter selected traffic between network segments.
- Routers connect networks that use the same networking protocols. For TCP/IP, the connection is made at the level of the IP layer. Routers can control whether packets are forwarded between network segments.

Monitor the network regularly for packet collisions as described in "Monitoring TCP/IP performance" (page 133) or use a network activity tester (commonly called a *sniffer*) if you have access to one. If the proportion of collisions to packets sent is greater than 10%, your network is probably overloaded. Some networks may be able to struggle along on at collision rates as high as 30% but this is rarely acceptable.

If there are a large number of input or output errors, suspect the network hardware of causing problems. Reflected signals can be caused by cable defects, incorrect termination, or bad connections. A network cable analyzer can be used to isolate cable faults and detect any electrical interference.

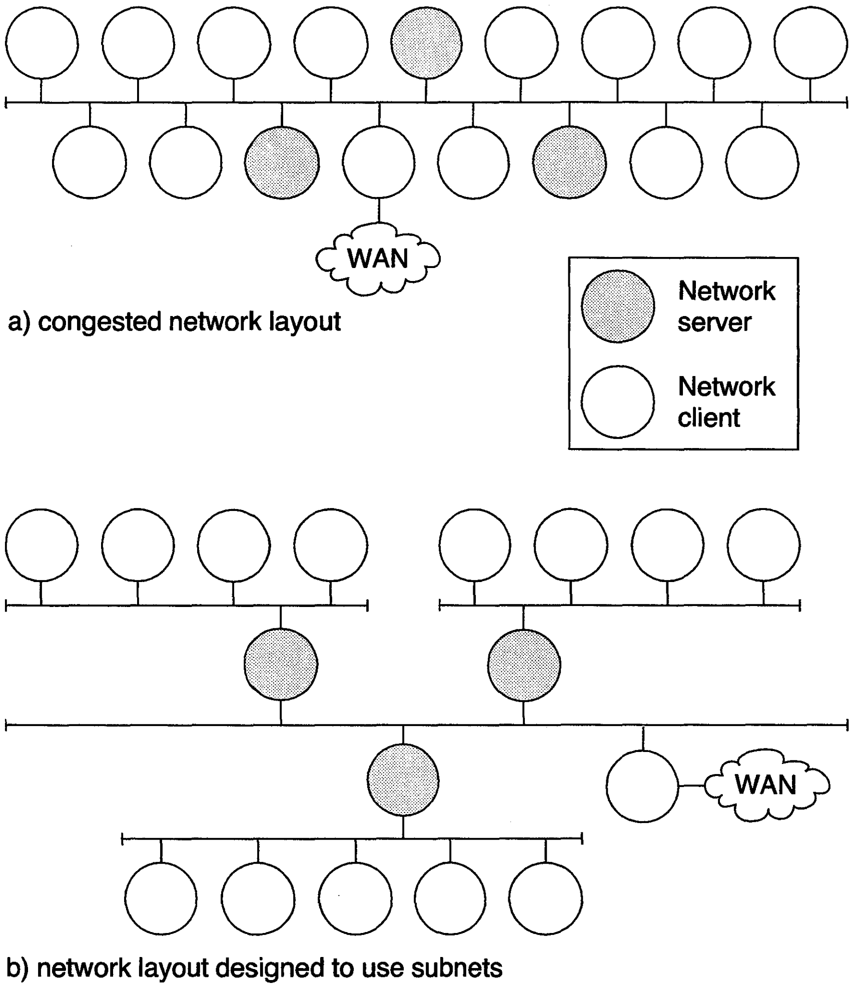

WAN
a) congested network layout
Network server
Network client
WAN
b) network layout designed to use subnets

Figure 6-5 Dividing a network into subnetworks to reduce network traffic

To reduce network loading, consider dividing it into separate networks (subnets) as shown in Figure 6-5 (page 138). This diagram shows how a network could be divided into three separate subnets. Routers connect each subnet to a backbone network. This solution only makes sense if you can group clients with individual servers by the function they perform. For example, you could arrange that each subnet corresponds to an existing department or project team within an organization. The clients dependent on each server should live on the same subnet for there to be a gain in network performance. If many machines are clients of more than one server, this layout may actually make the situation worse as it will impose an additional load on the servers acting as routers.

An alternative would be to use bridges to connect the network segments though this may be a more expensive solution. A potential problem with this is that if a bridge fails, the connection between the two segments is severed.

By connecting subnets using more than one router, you can provide an alternative route in case of failure of one of the routers. Another problem with using bridges is that they are intended to partially isolate network segments — they are not a solution if you want to provide open access to all available services.

Design the layout of subnets to reflect network usage. Typically, each subnet will contain at least one server of one or more of the following types:

- File server providing access to networked filesystems.
- Database server providing access to a database.
- Compute server providing intensive numeric calculations.
- Page server providing swap space for diskless clients.
- Bootstrap server enabling X terminals and diskless clients to boot over the network.
- Host server for X terminals.
- Master or slave Network Information Services (NIS) servers for clients or copy-only servers.

Some machines may also be expected to run X client processes for X servers running on X terminals and workstations. Applications such as desktop publishing and PostScript previewers transfer large amounts of data across the network. If possible, you may find it preferable to confine running such applications to dedicated workstations on the network.

If you run client-server applications across repeaters, bridges, or routers, you should be aware that this will impose additional delay in the connection. This delay is usually least for repeaters, and greatest for routers.

See also:

- Chapter 3, "Administering TCP/IP" in the *Networking Guide*
- "Creating subnets" in the *Networking Guide*
- "Troubleshooting TCP/IP" in the *Networking Guide*

Configuring routing for performance

There are few performance issues concerned with routing. Choice of routes outside your system is not generally in your control so this discussion only considers routing within an autonomous network.

Most networks use the Routing Information Protocol (RIP) for internal routing. RIP uses a metric for choosing a route based on distance as a number of hops. This metric is not optimal in certain circumstances. For example, it would choose a path to the desired destination over a slow serial link in preference to crossing an Ethernet and a Token Ring. You can increase the hop count on the slow interface advertised in the */etc/gateways* file to overcome this limitation. The RIP protocol is available with both the **routed**(ADMN) and **gated**(ADMN) routing daemons.

Most networks tend to use **routed** as it requires no configuration. However, we recommend that you only use RIP for simple network topologies. The Open Shortest Path First (OSPF) protocol is better suited than RIP for complex networks with many routers because it has a more sophisticated routing metric. It can also group networks into areas. The routing information passed between areas uses an abstracted form of internal routing information to reduce routing traffic. OSPF is only available using the **gated** routing daemon.

You can use the Internet Router Discovery (IRD) protocol for routing within networks in autonomous systems. This is not a true routing protocol but it allows hosts connected to a multicast or broadcast network to discover the IP addresses of routers using ICMP messages. Routers can also use the protocol to make themselves known. The **irdd**(ADMN) daemon uses the IRD protocol and is normally configured to run by default in addition to **routed**.

You can minimize the routing traffic on your network by configuring:

- Non-routing hosts to use only the IRD protocol.
- Interior routers to use IRD, and either RIP or OSPF.
- Exterior routers of an autonomous system to use an exterior routing protocol such as BGP or EGP.

For a full discussion of the various protocols, the daemons that use them, and how to configure these daemons, see Chapter 11, "Configuring Internet Protocol (IP) routing" in the *Networking Guide.*

Configuring DNS name service for performance

The Domain Name Service server included with TCP/IP can operate in a number of modes, each of which has its own performance implications.

A primary or secondary DNS nameserver maintains and accesses potentially large databases, answers requests from other servers and clients, and performs zone transfers. Both network traffic and memory are impacted.

There are several ways in which you can influence the performance of primary and secondary DNS nameservers:

- Choose appropriate machines to serve as primary and secondary nameservers. Such machines should be stable, have a large amount of memory, and a low system load.
- Choose the appropriate number of secondary (redundant) nameservers to ensure against failure. Be careful, however, that you do not overload the network by having too many secondary servers, or any one machine by having too few.
- Configure time-to-live (***ttl***) values in the standard resource records (RRs) of the zone file so that cached data does not expire too quickly necessitating further data transfer.
- Schedule zone file transfers for network slack times if your zone contains many secondary servers. You can do this by changing the version number (***serial***) of the zone file on the master server. Kill **named** with **SIGHUP** (for example, using the command **kill -s HUP $(cat /etc/named.pid)** if you use the Korn shell) to make it re-read the *named.boot* file. Then kill **named** with **SIGHUP** on each secondary server to make it request a full zone transfer. Note that you would normally use the ***refresh*** fields of the Start of Authority (SOA) record to control the frequency of zone refreshes.

A caching-only DNS nameserver maintains and accesses a potentially large cache. Because a caching-only server may answer many of its own requests, memory is impacted more highly than network traffic. If the machine has limited memory, you should strongly consider turning the machine into a DNS client using the resolver configuration file, */etc/resolv.conf*.

A DNS client pushes all resolution requests onto one or more DNS servers on the network; none are handled locally. This puts the burden of resolution on the network and on the nameservers listed in *resolv.conf*. It also means that **named** does not run and, therefore, does not add to the system load. In the case where the local machine has limited memory and response time over the network ranges from adequate to excellent, this configuration is desirable from a performance standpoint. If network response time is slow and memory is not limited, consider re-configuring the system as a caching-only server.

See also:

- Chapter 6, "Configuring the Domain Name Service" in the *Networking Guide*

NFS resources

The Network File System (NFS) software allows one computer (an NFS client) attached to a network to access the filesystems present on the hard disk of another computer (an NFS server) on the network. An NFS client can mount the whole or part of a remote filesystem. It can then access the files in this filesystem almost as if they were present on a local hard disk.

The speed of access to data is designed to approach that achievable using the server's hard disk directly. The performance of NFS is limited by:

- The maximum throughput of the underlying network technology. For example, Ethernet is capable of transferring up to approximately 1.25MB/s.
- The performance of the server's hardware subsystems including network adapters, and hard disks. Network adapters on an NFS server should have high specifications. They should at least have a 16KB I/O buffer and 32-bit data path, and, if possible, they should be capable of performing Bus Master DMA.
- The configuration of the buffer cache, STREAMS, and TCP/IP on the server.
- The number of daemons running on the server to service requests from network clients.

- The loading of the network which determines the effective throughput to and from the client.
- Demands for network file access to the server from other clients on the network.
- The performance of the client's subsystems including its buffer cache, STREAMS resources, TCP/IP configuration, and the network adapter.
- Other processes running on the client and server that compete for resources with NFS.

In practice, NFS servers feel the greatest stress. NFS client performance is directly dependent on the performance of the server and the network so you should examine these subsystems first if you experience poor client response.

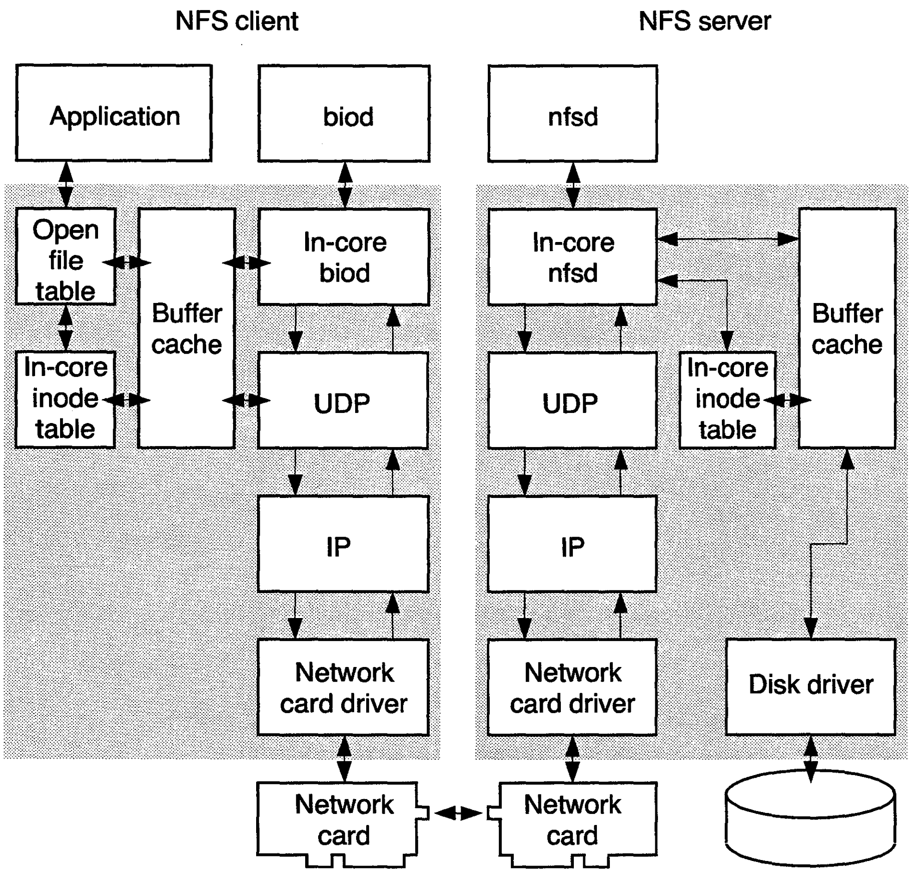


Figure 6-6 Schematic diagram of how NFS works

The mechanism by which NFS is implemented is illustrated in Figure 6-6 (page 143). This figure shows a simplified version of how NFS operates and also illustrates the main features that affect performance. The NFS client is shown at the left-hand side; the NFS server with its local disk to the right. All the subsystems in the path traced between the application program running on the client and the disk on the server directly affect performance.

As for TCP/IP, NFS has been configured in this release to maximize performance. However, you may be able to further increase performance based on your system's needs.

Because NFS depends directly on STREAMS and TCP/IP resources, its performance is directly influenced by these subsystems. See "STREAMS resources" (page 123) and "TCP/IP resources" (page 131) for more information about examining the performance of these subsystems.

See also:

- Chapter 15, "Configuring the Network File System (NFS)" in the *Networking Guide*

Monitoring NFS performance

The **nfsstat**(NADM) command reports statistics on NFS performance. Use the **-c** option to display client statistics:

```
Client rpc:
calls    badcalls retrans  badxid   timeout  wait     newcred
336033   50       413      418      299      0        0
peekeers badresps
0        0

Client nfs:
calls    badcalls nclget   nclsleep
335617   0        336033   0
. . .
```

The important fields for client performance are contained in the remote procedure call (RPC) statistics:

`badcalls`
: The number of times that an RPC call failed due to an error such as a timeout or an interrupted connection.

  Note that on soft-mounted filesystems, a request is retransmitted a limited number of times before it is reported as a failed RPC call. The value of `badcalls` is only incremented for the final failed attempt; previous failures increase the value of `retrans`. All requests that fail due to a timeout are recorded in `timeout`.

`retrans`

The number of requests for service that the client had to retransmit to servers. If the value of `badxid` is small, the network is probably dropping packets rather than the servers being slow.

The value of `retrans` should not be more than 5% of the value shown for `calls` in the NFS statistics.

`badxid`

The number of responses from servers for which the client has already received a response. If a client does not receive a response to a request within a time period, it retransmits the request. It is possible that the server may service the original request. In such a case, the client receives more than one response to a request. The value of `badxid` is incremented for every unexpected response.

If the value of `badxid` is approximately equal to `retrans`, one or more servers probably cannot service client requests fast enough. A server may not be running enough **nfsd** daemons or it may be insufficiently powerful to satisfy the clients' requests. See "Configuring NFS daemons" (page 147) for more information.

If the value of `badxid` is also approximately equal to `timeout`, you can increase the timeout value specified by the **timeo mount**(ADM) keyword in the file */etc/default/filesys* or in the appropriate map file if you are using **automount**(NADM). This will allow the servers more time to respond to requests.

`timeout`

The number of calls that timed out waiting for response from a server.

`wait`

The number of requests that had to wait for an available client handle. If this is non-zero, there are insufficient **biod** daemons running on the client. See "Configuring NFS daemons" (page 147) for more information

Use the **-s** option to display server statistics:

```
Server rpc:
calls       badcalls    nullrecv    badlen      xdrcall
57972       0           0           0           0
. . .
```

If the values of `badlen` and `xdrcall` are non-zero, the network is corrupting packets.

See also:

- "Troubleshooting NFS" in the *Networking Guide*

The following table summarizes the commands that you can use to examine the performance of NFS:

Table 6-4 Examining NFS performance

Command	Field	Description
nfsstat -c	badcalls	number of RPC call failures by client
	badxid	number of unnecessary repeated responses received by client from servers
	retrans	number of repeated requests by client to servers
	timeout	number of calls that timed out on client
	wait	number of calls that had to wait for a client handle
nfsstat -s	badlen	number of corrupted RPC requests received by server
	xdrcall	number of corrupted data headers received by server
nfsstat -z		zero out statistics

Tuning NFS performance

If NFS is configured, your system runs the */etc/rc2.d/S89nfs* script each time it goes to multiuser mode. (Note that this file is a link to */etc/nfs*.) This script starts several NFS daemons. If configured to run, the following daemons may affect performance:

biod Runs on clients to handle access to remote filesystems.

nfsd Runs on servers to handle access by remote clients to local filesystems.

pcnfsd Runs on servers to handle access by remote clients that use the DOS, OS/2, or Macintosh operating systems and run PCNFS.

If the output from the **sar -u** command (see "Identifying disk I/O-bound systems" (page 90)) shows that an NFS client is spending a significant proportion of time waiting for I/O to complete (`%wio` is consistently greater than 15%), and this cannot be attributed to local disk activity (a disk is busy if **sar -d** consistently shows `avque` greater than 1 and `%busy` greater than 80%), then the performance of NFS may be causing an I/O bottleneck. See "Tuning NFS client performance" (page 147) for more information.

Tuning NFS client performance

Read performance by an NFS client is influenced by several factors:

- The **biod** daemon processes running on the client provide read-ahead of disk blocks into the client machine's buffer cache as described in "Tuning the number of biod daemons on a client" (page 149). This only improves performance if the applications tend to perform sequential reads through files. The client's buffer cache should be tuned to enhance the read hit rate as described in "Increasing disk I/O throughput by increasing the buffer cache size" (page 75). If the data can be found in the client's buffer cache, this avoids the overhead of going out to the remote filesystem for the data. Note that there is no guarantee that the data is the most current available unless file locking is used
- The number of **nfsd** daemon processes running on the server to service requests from clients as described in "Tuning the number of nfsd daemons on a server" (page 148).

Write performance by an NFS client is affected if you choose to use non-standard asynchronous writes as described in "Configuring asynchronous or synchronous writes" (page 151). You should tune the server's buffer cache size to increase the write hit rate as described in "Increasing disk I/O throughput by increasing the buffer cache size" (page 75).

There is no benefit to NFS client write performance in tuning the write hit rate on the server if you opt to use default synchronous writes through its buffer cache.

NFS server performance can be further improved by using a disk controller with a write-back (rather than a write-through) cache. This runs the risk of losing data unless its integrity is protected using a UPS.

Configuring NFS daemons

If your system does not serve clients running PCNFS, comment out the following lines in */etc/rpcinit* that start **pcnfsd**:

```
[ -x /etc/pcnfsd ] && {
        echo " pcnfsd\c"
        pcnfsd &
}
```

When NFS is next started, **pcnfsd** will not run. This will not affect performance to any great extent apart from removing an unwanted process from the system.

The daemons **mountd, portmap, statd,** and **lockd** are needed for the operation of NFS.

The following sections discuss how to tune the number of **biod** and **nsfd** daemons running on clients and servers:

- "Tuning the number of nfsd daemons on a server" (this page)
- "Tuning the number of biod daemons on a client" (page 149)

Tuning the number of nfsd daemons on a server

Like **biods, nfsd** daemons provide processes for the scheduler to control — the bulk of the work dealing with requests from clients is performed inside the kernel. Each **nfsd** is available to service an incoming request unless it is already occupied. The more **nfsds** that are running, the faster the incoming requests can be satisfied. There is little context switching overhead with running several **nfsds** as only one sleeping daemon is woken when a request needs to be served.

If you run more **nfsds** than necessary, the main overhead is the pages of memory that each process needs for its u-area, data, and stack (program text is shared). Unused **nfsd** processes will sleep; they will be candidates for being paged or swapped out should the system need to obtain memory.

If too few **nfsds** are running on the server, or its other subsystems, such as the hard disk, cannot respond fast enough, it will not be able to keep up with the demand from clients. You may see this on clients if several requests time out but the server can still service other requests. If you run the command **nfsstat -c** on the clients, its output provides some information about the server's performance as perceived by the client:

```
Client rpc:
calls    badcalls retrans  badxid   timeout  wait      newcred
336033   50       413      418      299      0         0
. . .
```

If `badxid` is non-zero and roughly equal to `retrans`, as is the case in this example, the server is not keeping up with the clients' requests.

If you run too few **nfsds** on a server, the number of messages on the request queue builds up inside the upstream networking protocol stack in the UDP module. In extreme cases, you could consume all memory on the server reserved for use by STREAMS; this would cause applications using STREAMS to fail. Use **netstat -m** to examine STREAMS usage on the server as described in "Monitoring STREAMS performance" (page 129).

You can also use the command **netstat -s -p udp** to examine how many system failures due to shortage of STREAMS memory have occurred in the UDP module:

```
udp:
        0 incomplete headers
        0 bad data length fields
        0 bad checksums
        62 bad ports
        438014 input packets delivered
        0 systems errors during input
        417038 packets sent
```

To change the number of **nfsds** that are configured to run, edit the following lines in the file */etc/nfs* on the server:

```
[ -x /etc/nfsd ] && {
        umask 000
        echo " nfsd(xnumber)\c"
        nfsd number &
        umask $oldmask
}
```

When NFS is next started on the client, ***number*** **nfsds** will run.

Tuning the number of biod daemons on a client

On an NFS client system, you do not need to run any **biod** processes for applications to access remote filesystems. The **biods** handle read-ahead and write-behind on remote filesystems in order to improve performance. When reading, they send requests to read disk blocks ahead of that currently requested. When writing, they take over responsibility for handling writing the block to the remote disk from the application. The **biod** processes visible using **ps**(C) are merely convenient handles used by the process scheduler to control NFS client operation — the majority of the work dealing with the **read** and **write** requests is dealt with inside the kernel.

If no **biods** are running, the application's performance will suffer as a result. When it writes to the remote filesystem, the **write** system call will block until the data has been written to the disk on the server. When it reads from the remote filesystem, it is unlikely to find the blocks in the buffer cache.

From this, you might deduce that running an extra copy of **biod** will always enhance NFS performance on the client. For example, if four **biods** are running, each of these can perform asynchronous **writes** without applications programs having to wait for these to complete. If an application requires access to the remote filesystem while the **biods** are busy, it performs this itself. The limit to performance enhancement comes from the fact that each **biod**'s disk requests impose a load on the server. **nfsd** daemons, the buffer cache, and disk I/O on the server will all come under more pressure if more **biod** daemons are run on the clients.Network traffic will also increase as will the activity of the networking protocol stacks on both the server and its clients. The default number of **biod** processes run on a client is four. To see if the number running on your system is adequate, use the **ps -ef** command and examine the elapsed CPU time used by the **biods** under the `TIME` column. Note that the results are only meaningful if your system has been operating under normal conditions for several hours.

If **nfsstat -c** on the client shows a `wait` for client handle value of zero and if the `TIME` value for at least one of the **biods** is substantially less than the others, then there are probably enough daemons running. If several **biods** show low `TIME` values, it should be safe to reduce their number to one more than the number showing high `TIME` values.

If all the `TIME` values are high, increase the number of **biods** by two, and continue to monitor the situation.

If you are *root*, you can reduce the number of **biods** running by killing them with **kill**(C). You can also start extra **biods** running using the command **/etc/biod**.

To change the number of **biods** that are configured to run, edit the following lines in the file */etc/nfs* on each client:

```
[ -x /etc/biod ] && {
        echo " biod(xnumber)\c"
        biod number &
}
```

When NFS is next started on the client, ***number*** **biods** will run.

Configuring asynchronous or synchronous writes

One way of improving NFS performance is to prevent applications and **biod** daemons from performing synchronous **write** calls to the remote disk. The mechanism used is controlled by the value of the kernel variable `nfs_server_async_writes` set in the file */etc/conf/pack.d/nfs/space.c* on the server. This variable can take three values:

0 Enables slow synchronous writes; write the data blocks through the buffer cache to the disk one at a time.

1 Enables fast synchronous writes (the default); the data blocks are first written to the server's buffer cache. A file sync operation then flushes the data to disk.

2 Enables asynchronous writes; write the data to the server's buffer cache and rely on the server's buffer flushing daemon to write the data to disk. This minimizes the time that processes on the client have to wait for the write operation to complete.

To change the way that blocks are written to disk on the server, edit the *space.c* file and change the value of `nfs_server_async_writes` to select the desired behavior. Relink the kernel and reboot the system as described in "Relinking the kernel" in the *SCO OpenServer Handbook.*

WARNING Asynchronous writes do not conform to the NFS V2 standard. There is a possibility that data may be lost irretrievably.

If you choose to use asynchronous writes, this will improve performance but it increases the risk that data can be lost without reporting an error to the client. The client can receive notification of a successful write while the data is still in the server's buffer cache. If the server's disk goes down because of a power failure or other fault, there is a risk that the data may not have been written to disk. You can protect against this to some extent using:

- An uninterruptible power supply (UPS) to maintain power to the server's buffer cache, disk cache, and disk in case of power failure.
- Virtual disk management or hardware RAID array to ensure data integrity and to protect against disk failure.

See also:

- "NFS server and client daemons" in the *Networking Guide*

Configuring NFS to use TCP

For NFS filesystems mounted over a high latency, high bandwidth connection such as a wide area network (WAN), there are benefits in using TCP as the transport protocol rather than UDP. With TCP, you can define large send and receive windows to be set on an interface as described in "Tuning TCP/IP performance" (page 131). This allows a large amount of data to be sent before requiring an acknowledgement. On a noisy connection, it is preferable to use TCP because it performs packet error detection and correction; UDP relies on the application to correct errors.

To define NFS to support TCP as a transport protocol on a server, edit */etc/nfs*. Use the **-d** and **-t** options to **nfsd**(NADM) to allocate the number of **nfsd** daemons that support each protocol. For example, to define six **nfsds** to use UDP and two to use TCP, change the lines that starts the **nfsd** daemons to read:

```
[ -x /etc/nfsd ] && {
        umask 000
        echo " nfsd(UDPx6,TCPx2)\c"
        nfsd -u 6 -t 2 &
        umask $oldmask
}
```

Specify the **mount**(ADM) option modifier **tcp** on each client for the remote NFS filesystem that you want to mount using TCP. This must be added to the options defined for the **mntopts** keyword in the file */etc/default/filesys* (see *filesys*(F) for more information). The following is an example of such an entry:

```
bdev=nfs_svr:/remote \
        mountdir=/remote_mnt fstyp=NFS \
        fsck=no fsckflags= \
        init=yes initcmd="sleep 2" \
        mntopts="bg,soft,tcp" \
        rcmount=yes rcfsck=no mountflags=
```

If you use **automount**, you can specify the option modifiers in the *auto.master* configuration map as described in **automount**(NADM).

These changes will not take effect until NFS is next started on the server and clients.

Configuring IP to maximize NFS performance

By default, NFS transfers data in 8KB blocks. If the network is Ethernet-based and full frames are being used, six Ethernet frames are required to transmit these blocks. If the data in the frames is rounded down to 1KB, eight frames are required to transmit the data. If your network adapter can handle full frames and back-to-back packets, it should already be configured as such as described in "Using ifconfig to change parameters for a network card" (page 225).

Configuring mount options to maximize NFS performance

If the network adapter on an NFS client cannot handle full frames and back-to-back packets, reduce the NFS read and write transfer sizes below the default of 8KB. To do this, specify the **mount**(ADM) option modifiers **rsize** and **wsize** for each mounted filesystem. These must be added to the options defined for the **mntopts** keyword in the file */etc/default/filesys* (see *filesys*(F) for more information). The following is an example of such an entry reducing the read and write transfer sizes to 1KB (1024 bytes):

```
bdev=nfs_svr:/remote \
        mountdir=/remote_mnt fstyp=NFS \
        fsck=no fsckflags= \
        init=yes initcmd="sleep 2" \
        mntopts="bg,soft,rsize=1024,wsize=1024" \
        rcmount=yes rcfsck=no mountflags=
```

If you use **automount**, you can specify these option modifiers in the *auto.master* configuration map as described in **automount**(NADM).

Performance considerations when using automount

If you use **automount**(NADM) to mount remote filesystems automatically on demand, you should consider the following performance implications:

- **automount** is single threaded. Mount requests can be delayed by another request that has been made to a slow or inactive NFS server. For more information see "Troubleshooting NFS" in the *Networking Guide.*
- Making multiple requests to an automounted filesystem can cause high system overhead. The kernel forces a context switch to **automount** to look up the pathname for each request. This can happen, for example, in a shell script that repeatedly copies files between a local and a remote filesystem, or between two remote filesystems (see also "Unnecessary automounts" in the *Networking Guide*).
- Direct **automount** maps require two mount table entries for each automounted filesystem; if you use indirect maps, one mount table entry is used by **automount** plus one for each automounted filesystem. By default, the number of mount table entries is determined dynamically, so there is no need to change the **MAX_MOUNT** kernel parameter to a value other than 0 (see also "Direct and indirect mounting" in the *Networking Guide*).

- Login time can increase significantly for **csh**(C) users who include many automounted filesystems in their path. The C shell adds command pathnames to its internal hash table when the path variable is set. As a consequence, **automount** mounts each automounted remote filesystem that is listed.

See also:

- Chapter 16, "Configuring the NFS automounter" in the *Networking Guide*
- "When to use automount" in the *Networking Guide*
- "Troubleshooting automount" in the *Networking Guide*

Performance considerations when using NIS

The Network Information Service (NIS) supplements NFS and provides a distributed database of commonly accessed administration files. A master NIS server holds information files needed by all machines on the network centrally; examples of these files are */etc/passwd*, */etc/group*, and */etc/services*. Whenever this information is updated, it is pushed out to slave servers and copy-only servers to ensure that it is updated globally.

NIS clients, which may be diskless, request information from servers whenever needed. This may be quite a common occurrence. For example, a command such as **ls -l** requires access to information held in the files */etc/passwd* and */etc/group* so that it can display the user and group ownership of files. If you are running NIS clients on your network, you should be aware that a proportion of network traffic will be caused by NIS clients requesting such information.

LAN Manager Client Filesystem resources

LAN Manager Client software allows your system to mount and access DOS, NT, and OS/2 filesystems on LAN Manager servers. See Chapter 4, "Administering and using LAN Manager Client" in the *Guide to Gateways for LAN Servers* for more information.

As for NFS, the performance of LMCFS is enhanced if the client system can take advantage of read-aheads and asynchronous writes. The size of the read-ahead buffer (in bytes) is controlled by the value of the **rawsize** option modifier supplied to **mount**(ADM). Data that is read-ahead is discarded if it is not used within the time set by the value (in tenths of a second) of the **udttl** option modifier. Asynchronous writes may be enabled using the **async** option modifier. This carries an inherent risk of possible data loss should the server crash before the data has been written to disk.

Tuning LAN Manager Client Filesystem performance

If the output from the **sar -u** command (see "Identifying disk I/O-bound systems" (page 90)) shows that a LAN Manager client is spending a significant proportion of time waiting for I/O to complete (`%wio` is consistently greater than 15%), and this cannot be attributed to local disk activity (a disk is busy if **sar -d** consistently shows `avque` greater than 1 and `%busy` greater than 80%), then the performance of the LAN Manager Client Filesystem (LMCFS) may be causing an I/O bottleneck.

The kernel parameters that control the behavior of LMCFS are described in "LAN Manager Client Filesystem parameters" (page 223). These parameters can only be adjusted using **idtune**(ADM) as described in "Using idtune to reallocate kernel resources" (page 190).

There are three areas where you can examine the performance of LAN Manager clients:

- "Examining possible network or server problems" (this page)
- "Examining the usage of server message blocks and lminodes" (page 156)
- "Examining the performance of each mounted filesystem" (page 156)

Examining possible network or server problems

The **-v** option to the **vcview**(LMC) command indicates possible network problems that may be affecting LMCFS performance:

```
. . .
MaxXmt MaxRcv MaxMux TxCnt  TxErr RxCnt  RxErr Conns Retrans Reconns
4096   4096   50     32131  5     34023  21    1     15      4
. . .
```

If the transmission error rate (100*`TxErr`/`TxCnt`) or reception error rate (100*`RxErr`/`RxCnt`) is high (greater than 10%) or either of these these rates is increasing, the network or the server may be overloaded.

Similarly, if the number of retransmissions (`Retrans`) or reconnections (`Reconns`) is increasing, this may also indicate that the network or the server is overloaded.

Examining the usage of server message blocks and lminodes

The command **lmc stats** (see **lmc**(LMC)) displays the usage of server message block (SMB) data buffers, request slots, and LAN Manager inodes (lminodes):

```
                alloc    maxalloc avail    fail     hiprifail
SMB buffers:    49       102      1024     0        0
SMB req slots:  49       60       256      0        -
. . .
SMB sync reads 840 (8.31% of total reads)

lminode alloc failures 0
```

If insufficient SMB data buffers or request slots are configured, processes will wait until more become available.

Increase the value of the kernel parameter **LMCFS_NUM_BUF** if the `fail` column displays a non-zero value for SMB data buffers.

Increase the value of the kernel parameter **LMCFS_NUM_REQ** if the `fail` column displays a non-zero value for SMB request slots.

Increase the value of the kernel parameter **LMCFS_LMINUM** if `lminode alloc failures` shows a non-zero value.

If the proportion of synchronous reads shown by `SMB sync reads` is high, this can have a significant negative impact on performance. You can increase the size of the read-ahead buffer using the **rawsize** option modifier to **mount**. This data is discarded if it is not used within the time set by the **udttl** option modifier to **mount**.

Examining the performance of each mounted filesystem

The command **lmc mntstats** (see **lmc**(LMC)) shows statistics for each mounted LAN Manager filesystem:

```
NT/TMP mounted on /mnt, user-based, asynch
rbsize rawsize wbsize awwsize timeout retrans udttl old r/a Broken oplocks
8192   16384   8192   16384   300     5       50    0       0
```

`old r/a` shows the number of read-ahead blocks that have been discarded because the client did not make use of the data quickly enough. If the value of `old r/a` is increasing, either increase the value of **udttl** or decrease the size of **rawsize**. Which action you should take depends on how long you are willing to see the data age.

`Broken oplocks` shows the number of opportunistic locks that were relinquished. This shows contention due to several clients accessing the same files on the server.

Other networking resources

SCO OpenServer networking services also support the following networking protocols and software:

- The Internetwork Packet Exchange (IPX) protocol provides a connectionless datagram service similar to UDP except that the data packet size is limited to 500 bytes. IPX also plays a similar role to the IP protocol in TCP/IP by providing a message service for modules above it in the protocol stack, including SPX.

  The Sequenced Packet Exchange (SPX) protocol is a connection-oriented service similar to TCP except that it does not support a configurable send and receive window size; an acknowledgement is required for every packet that is sent.

  For testing IPX/SPX network connectivity, the command **nping**(PADM) is available. This provides similar information to that provided by the TCP/IP command **ping**(ADMN).

  For more information about IPX/SPX, see "IPX/SPX" in the *Networking Guide.*

- SCO Gateway for NetWare software re-exports filesystems mounted on Novell® NetWare® servers. Its operation depends fundamentally on the configuration of IPX/SPX. For more information, see "Accessing NetWare servers with SCO Gateway for NetWare" in the *Networking Guide.*

- NetBEUI is another type of protocol stack similar to TCP/IP and IPX/SPX. It is included with Microsoft® LAN Manager for SCO® Systems.

- The NetBIOS interface allows applications such as LAN Manager to send messages over TCP/IP.

- SCO OSI is a protocol stack that implements the Open Systems Interconnection 7-layer model.

Like the TCP/IP protocol, all these networking resources depend on the availability of sufficient STREAMS and CPU resources, and the performance of the network and network adapter cards. See "STREAMS resources" (page 123) for more information. You can use the **netstat** or **ndstat** commands to examine the number of errors and packet collisions associated with each network interface. See "Monitoring TCP/IP performance" (page 133) for more information.

Case study: network overhead caused by X clients

In this study, users report that the response time to key presses and mouse movement is occasionally very slow during the working day. They also report that resizing and moving windows is also very slow at certain times. This performance is unacceptable for the interactive applications being run — word processing, spreadsheets, e-mail, desktop publishing, and graphics processing.

System configuration

The system's configuration is as follows:

- Ethernet-based (10Base2) LAN.
- 10 X terminals — used for running applications that place little processing and memory load on the main file server.
- 12 PC-based workstations running the SCO OpenServer Desktop System — these run applications locally rather than on the file server.
- File server — a two Pentium 50MHz multiprocessor machine with a high specification network interface card.
- Two host servers for the X terminals — both servers are based on the 80486DX2 running at 66MHz.

From regular performance monitoring, it is known that there is no performance problem caused by X clients running on the two host servers. Users are encouraged to run applications locally if they have workstations.

Defining a performance goal

The system administrator is asked to investigate the source of the problem, and suggest means to improve the response time.

Collecting data

The system administrator runs the **netstat -i** command on several workstations to record networking statistics throughout the working day. A sampling interval of one minute is specified and the output is written to a file in the */tmp* directory:

```
netstat -i 60 > /tmp/netstat_op
```

The administrator runs the command on several workstations to try to eliminate the possibility that faulty network interface cards are the cause of the problem.

The recorded output shows occasional short periods when the network is overloaded (for clarity, only the statistics for the network interface xxx0 are shown in this example):

```
    input  (xxx0)      output
packets errs  packets errs  colls
...
110     0     101     0     0
78      0     66      0     0
85      1     75      2     23
180     2     123     1     42
120     1     55      1     18
87      0     67      0     2
67      0     54      0     0
...
```

At these times, the numbers of input and output errors are non-zero, and the number of collisions approaches 30% of output packets. The same behavior is observed on all the workstations on which statistics were gathered.

If the periods of heavy loading are excluded, the frequency of packet collisions approaches 0%.

Formulating a hypothesis

From the results of running **netstat**, the system administrator suspects that some applications must be moving large amounts of data across the network. Careful examination of the figures shows that the network is overloaded approximately 5% of the time. Periods of high loading generally last only a few minutes and seem to occur in bursts. Such behavior is typical if large files are transferred using NFS. It is unlikely to be the result of network traffic caused by remote X clients as these are run locally where possible. Possible culprits are programs used to preview PostScript and graphics image files, DTP packages, and screen-capture utilities.

Getting more specifics

With the cooperation of several users, the administrator monitors network performance using **netstat** over a period of 30 minutes. During this period the users run the suspect applications to load and manipulate large files across the network. The outcome of this investigation is that graphics image previewers and screen-capture utilities seem to cause the most network overhead. The files being viewed or created are often several megabytes in size.

Making adjustments to the system

There are several things that can be done to reduce the peak load on the network:

- Encourage users to save and load graphics images to and from the local disk on the workstation they are using. These files may then be copied to the file server when the network is less busy.
- Run screen capture and graphics preview utilities on dedicated workstations rather than on X terminals.
- Splitting the network into several subnets might help if the nodes on the network can easily be divided into logically distinct groups. However, this may cause more CPU overhead on the file server if it is is used as the router between the subnets. This solution is more expensive and may make the problem worse if the wrong network topology is chosen.

Chapter 7

Tuning system call activity

This chapter is of interest to application programmers who need to investigate the level of activity of system calls on a system.

System calls are used by programs and utilities to request services from the kernel. These can involve passing data to the kernel to be written to disk, finding process information and creating new processes. By allowing the kernel to perform these services on behalf of an application program, they can be provided transparently. For example, a program can **write** data without needing to be concerned whether this is to a file, memory, or a physical device such as disk or tape. It also prevents programs from directly manipulating and accidentally damaging system structures.

System calls can adversely affect performance because of the overhead required to go into system mode and the extra context switching that may result.

Viewing system call activity

System call activity can be seen with **sar -c** (or **mpsar -c** for SMP):

```
23:59:44 scall/s sread/s swrit/s  fork/s  exec/s  rchar/s  wchar/s
23:59:49     473       9       0    0.09    0.12   292077      421
23:59:54     516      13       3    0.03    0.03   367668      574
23:59:59     483      13       3    0.01    0.02   366992      566

Average      489      12       2    0.04    0.06   338280      512
```

`scall/s` indicates the average number of system calls per second averaged over the sampling interval. Also of interest are `sread/s` and `swrit/s` which indicate the number of **read**(S) and **write**(S) calls, and `rchar/s` and `wchar/s` which show the number of characters transferred by them.

If you are an applications programmer and the SCO OpenServer Development System is installed on your system, you can use **prof**(CP) to examine the results of execution profiling provided by the **monitor**(S) function. This should show where a program spends most of its time when it is executing. You can also use the **trace**(CP) utility to investigate system call usage by a program.

Identifying excessive read and write system call activity

Normally, **read** and **write** system calls should not account for more than half of the total number of system calls. If the number of characters transferred by each **read** (`rchar/s` / `sread/s`) or **write** (`wchar/s` / `swrit/s`) call is small, it is likely that some applications are reading and writing small amounts of data for each system call. It is wasteful for the system to spend much of its time switching between system and user mode because of the overhead this incurs.

It may be possible to reduce the number of **read** and **write** calls by tuning the application that uses them. For example, a database management system may provide its own tunable parameters to enable you to tune the caching it provides for disk I/O.

Viewing process fork and exec activity

`fork/s` and `exec/s` show the number of **fork**(S) and **exec**(S) calls per second. If the system shows high **fork** and **exec** activity, this may be due to it running a large number of shell scripts. To avoid this, one possibility is to rewrite the shell scripts in a high-level compiled language such as C.

Viewing AIO activity

If applications are using asynchronous I/O (AIO) to disk, you can use the **-O** option to **sar**(ADM) (or **mpsar**(ADM) for SMP) to examine the performance of AIO requests. The values reported include the number of AIO read and write requests per second, and the total number of 1KB blocks (both read and write) being handled per second. The `%direct` column of the report shows the percentage of AIO requests that are passed directly to the disk driver by the POSIX.1b **aio** functions defined in the Software Update for Database Systems (SUDS) library. Other AIO requests are handled by the **aio**(HW) driver.

Viewing IPC activity

You can use the **sar -m** command (or **mpsar -m** for SMP) to see how many System V interprocess communication (IPC) message queue and semaphore primitives are issued per second. Note that you can also use the **ipcs**(ADM) command to report the status of active message queues, shared memory segments, and semaphores.

Semaphore resources

Semaphores are used to prevent processes from accessing the same resource, usually shared memory, at the same time.

The number of System V semaphores configured for use is controlled by the kernel parameter **SEMMNS**.

If the `sema/s` column in the output from **sar -m** shows that the number of semaphore primitives called per second is high (for example, greater than 100), the application may not be using IPC efficiently. It is not possible to recommend a value here. What constitutes a high number of semaphore calls depends on the use to which the application puts them and the processing power of the system running the application.

System V semaphores are known to be inefficient and adversely affect the performance of multiprocessor systems. This is because:

- They increase contention between processors — this reduces scaling and prevents the available CPU power being used effectively.
- They increase activity on the run queues as several processes sleeping on a semaphore may be woken when its state changes — this increases system overhead.
- They increase the likelihood of context switching — this also increases system overhead.

If you are an applications programmer, consider using the SUDS library routines instead; these implement more efficient POSIX.1b semaphores. The number of POSIX.1b semaphores configured for use is controlled by the kernel parameter **SEM_NSEMS_MAX**.

Some database management systems may use a sleeper driver to synchronize processes. (This may also be referred to as a post-wait driver.) If this is not enabled, they may revert to using less efficient System V semaphores. See the documentation provided with the database management system for more information.

For more information on the kernel parameters that you can use to configure semaphores, see "Semaphores" (page 216) and "Semaphore parameters" (page 217).

Messages and message queue resources

Messages are intended for interprocess communication which involves small quantities of data, usually less than 1KB. Between being sent and being received, the messages are stored on message queues. These queues are implemented as linked lists within the kernel.

Under some circumstances, you may need to increase resources allocated for messages and message queues above the default values defined in the *mtune*(F) file. Note that the kernel parameters defined in *mtune* set system-wide limits, not per-process limits.

Follow the guidelines below when changing the kernel parameters that control the configuration of message queues:

- Each process that calls **msgget**(S) with either of the flags **IPC_CREAT** or **IPC_PRIVATE** set obtains an ID for a new message queue.
- The total number of available message headers (**MSGTQL**) must be less than or equal to 16383. This limits the total number of messages system-wide because each unread message must have a header.
- The total number of segments configured for use (**MSGSEG**) must be less than or equal to 32768. This limits the total number of messages system-wide because each message consists of at least one segment.
- The size of each message segment (**MSGSSZ**) is specified in bytes and must be a multiple of 4 in the range 4 to 4096. Each message is allocated enough segments to hold it; any remaining space in the last segment allocated to a message is unused. A small value of **MSGSSZ** is suitable for systems which will send and receive many small messages. A large value is suitable if messages are fewer and larger. Small segments require more processing overhead by the kernel as it keeps track of them; large segments can be wasteful of memory.
- The total amount of memory reserved for use by message data is controlled by the product of the number of segments and the segment size:

  MSGSEG * MSGSSZ

  This value must be less than or equal to 128KB (131072 bytes).
- Increase the size of the map used for managing messages (**MSGMAP**) if a large number of small messages are processed. Typically, you should set the map size to half the number of memory segments configured (**MSGSEG**). Do not increase **MSGMAP** to a value greater than that of **MSGSEG**.
- The amount of message data allowed in an individual queue (**MSGMNB**) must be less than or equal to 64KB - 4 bytes (that is, less than or equal to 65532 bytes).
- The maximum length of an individual message is limited by the value of **MSGMAX**. Although the recommended maximum is 8192 bytes (8KB), the kernel can support messages up to 32767 bytes in length. Note, however, that the message size may also be limited by the value of **MSGMNB**.

The following table shows how to calculate the maximum values for these parameters based on the value of **MSGSSZ**. Note that **MSGSSZ** must be a multiple of 4 in the range 4 to 4096:

Table 7-1 Calculation of maximum value of message parameters

Parameter	Maximum value
MSGMAP	131072 / MSGSSZ
MSGMAX	32767
MSGMNB	65532
MSGMNI	1024
MSGSEG	131072 / MSGSSZ
MSGTQL	MSGMNB / MSGSSZ

For more information on the kernel parameters that you can use to configure message queues, see "Message queues" (page 213) and "Message queue parameters" (page 215).

Shared memory resources

Shared memory is an extremely fast method of interprocess communication. As its name suggests, it operates by allowing processes to share memory segments within their address spaces. Data written by one process is available immediately for reading by another process. To prevent processes trying to access the same memory addresses at the same time, known as a race condition, the processes must be synchronized using a mechanism such as a semaphore.

The maximum number of shared-memory segments available for use is controlled by the value of the kernel parameter **SHMMNI**. The maximum size in bytes of a segment is determined by the value of the kernel parameter **SHMMAX**.

For more information on the kernel parameters that you can use to configure shared memory, see "Shared memory" (page 218) and "Shared memory parameters" (page 218).

SUDS library spin locks and latches

If your application uses spin locks and latches from the SUDS library to synchronize processes, you can use the **-L** option to **sar**(ADM) (or **mpsar**(ADM) for systems with an SCO SMP License) to view their activity.

These latches allow processes to spin or sleep while waiting to acquire a latch. Alternatively, a process can be made to sleep if it has been spinning for a given time period without being able to acquire a latch. This prevents it spending an unnecessarily long time spinning. It is efficient for a process to spin for a short time to avoid the system overhead that a context switch would cause. Process that wait a long time for a latch should sleep to avoid wasting CPU time.

See the **sar**(ADM) manual page for more information about the latch activity reported by the **-L** option.

The following table summarizes the commands that can be used to determine if a system is suffering under heavy system call activity:

Table 7-2 Viewing system call activity

Command	Field	Description
[mp]sar -c	scall/s	total number of all system calls per second
	sread/s	read system calls per second
	swrit/s	write system calls per second
	fork/s	fork system calls per second
	exec/s	exec system calls per second
	rchar/s	characters transferred by read system calls per second
	wchar/s	characters transferred by write system calls per second
ipcs -a		status of System V IPC facilities
[mp]sar -m	msg/s	message queue primitives per second
	sema/s	semaphore primitives per second
[mp]sar -O	%direct	percentage of AIO requests using the POSIX.1b **aio** functions

Reducing system call activity

Reducing most system call activity is only possible if the source code for the programs making the system calls is available. If a program is making a large number of **read** and **write** system calls that each transfer a small number of bytes, then the program needs to be rewritten to make fewer system calls that each transfer larger numbers of bytes.

Other possible sources of system call activity are applications that use interprocess communication (semaphores, shared memory, and message queues), and record locking. You should ensure that the system has sufficient of these resources to meet the demands of the application. Most large applications such as database management systems include advice on tuning the application for the host operating system. They may also include their own tuning facilities, so you should always check the documentation that was supplied with the application.

Case study: semaphore activity on a database server

In this study, a site has installed a relational database on a multiprocessor system. The database gives the choice of using System V semaphores or the sleeper driver (sometimes called the post-wait driver) to synchronize processes. The object is to investigate which of these options will maximize the number of transactions that can be processed per second and the response time for the user.

System configuration

The system's configuration is as follows:

- Multiprocessor — 2 Pentium 60MHz processors.
- EISA bus.
- 96MB of RAM.
- 96MB of swap space.
- 14GB of hard disk (two arrays of seven 1GB SCSI-2 disks).
- One Bus Mastering DMA Ethernet network card with a 16KB buffer and 32-bit wide data path.

The database server does not act as host machine to any users directly; instead there are five host machines connected to the LAN which serve an average of 100 users each.

Defining a performance goal

The performance goal in this study is to compare the performance of the database when using System V semaphores and when using the sleeper driver.

NOTE To configure the sleeper driver into the kernel, change the second field of the line in the file */etc/conf/sdevice.d/sleeper* to read "Y". Then relink and reboot the kernel.

Collecting data

To monitor the performance, an in-house benchmark is used for an hour with the system configured to use System V semaphores, and then with it using the sleeper driver. The benchmark measures the minimum, maximum, and average transaction times and the total throughput in transactions per second.

The result of running the benchmark is that the best performance is achieved using the sleeper driver.

Formulating a hypothesis

When the database is using System V semaphores, the system may be spending too much time in kernel mode executing semaphore calls. The benchmark run using the sleeper driver gives better results because it is an enhancement specifically aimed at improving the performance of relational databases. It allows an RDBMS to synchronize built-in processes without the high overhead of switching between user mode and system mode associated with System V semaphores.

Getting more specifics

To test the hypothesis, **mpsar -u** is used to display the time that the system spent in system mode while each benchmark was being run. For the benchmark using the sleeper driver, typical results were:

```
13:55:00    %usr    %sys    %wio   %idle
...
14:20:00      75      20       2       3
14:25:00      72      23       1       4
14:30:00      69      24       5       2
14:35:00      77      19       4       0
...
```

The averaged performance of all the CPUs was excellent with low percentages spent in system mode, idle waiting for I/O, or idle.

For the run using semaphores, the results were:

```
16:08:00    %usr    %sys    %wio   %idle
...
16:48:00      55      38       6       0
16:43:00      59      32       7       2
16:58:00      61      34       4       1
16:53:00      58      38       2       2
...
```

The system spends more time in system mode and waiting for I/O when System V semaphores are used. The benchmark results indicate that transaction throughput and response time are approximately 10% better when the sleeper driver is used.

Making adjustments to the system

The database is configured to use the sleeper driver as this provides the best performance for the benchmark. The system should be monitored in everyday use to evaluate its performance under real loading.

Vendors of the database management systems are continually improving their products to use more sophisticated database technologies. If you upgrade the database management system to a version that supports POSIX.1b semaphores, you may need to evaluate if these should be used instead of the sleeper driver.

Appendix A

Tools reference

A variety of tools are available to monitor system performance or report on the usage of system resources such as disk space, interprocess communication (IPC) facilities, and pipes:

df	Reports the amount of free disk blocks on local disk divisions. See "df — report disk space usage" (page 172) and **df**(C) for more information. Also see the descriptions of the related commands: **dfspace**(C) and **du**(C).
ipcs	Reports the status of System V interprocess communication (IPC) facilities — message queues, semaphores, and shared memory. See **ipcs**(ADM) for more information.
netstat	Reports on STREAMS usage and various network performance statistics. It is particularly useful for diagnosing if a network is overloaded or a network card is faulty. See **netstat**(TC) for more information. See also **ndstat**(ADM) which reports similar information.
nfsstat	Reports NFS statistics on NFS servers and clients. It is particularly useful for detecting problems with NFS configuration. See **nfsstat**(NADM) for more information.
ping	Can be used to test connectivity over a network. See **ping**(ADMN) for more information.
pipestat	Reports on the usage of ordinary and high performance pipes. See **pipe**(ADM) for more information.
ps	Reports on processes currently occupying the process table. See "ps — check process activity" (page 173) and **ps**(C) for more information.
sar	Samples the state of the system and provides reports on various system-wide activities. See "sar — system activity reporter" (page 176) and **sar**(ADM) for more information.

swap Reports on the amount of available swap space or configures additional swap devices. See "swap — check and add swap space" (page 179) and **swap**(ADM) for more information.

timex Reports on system resource usage during the execution of a command or program. See "timex — examine system activity per command" (page 180) and **timex**(ADM) for more information. See also the description of the related command, **time**(C).

traceroute
Traces the route that network packets take to reach a given destination. See **traceroute**(ADMN) for more information.

vmstat Reports on process states, paging and swapping activity, system calls, context switches and CPU usage. See "vmstat — virtual memory statistics" (page 181) and **vmstat**(C) for more information.

df — report disk space usage

When attempting to achieve optimal performance for the I/O subsystem, it is important to make sure that the disks have enough free space to do their job efficiently. The **df**(C) command, and its close relative **dfspace**(C), enable you to see how much free space there is. The following example shows the output from **df** and **dfspace** on the same system:

```
$ df
/         (/dev/root       ):   37872 blocks   46812 i-nodes
/u        (/dev/u          ):  270814 blocks   36874 i-nodes
/public   (/dev/public     ):  191388 blocks   55006 i-nodes
/london   (wansvr:/london  ):  149750 blocks       0 i-nodes
$ dfspace
/         :     Disk space:  18.49 MB of 292.96 MB available ( 6.31%).
/u        :     Disk space: 132.23 MB of 629.98 MB available (20.99%).
/public   :     Disk space:  93.45 MB of 305.77 MB available (30.56%).
/london   :     Disk space:  73.12 MB of 202.56 MB available (36.10%).

Total Disk Space: 317.29 MB of 1431.29 MB available (22.17%).
$ df -v
Mount Dir  Filesystem              blocks      used      free   %used
/          /dev/root               600000    562128     37872     93%
/u         /dev/u                 1290218   1019404    270814     79%
/public    /dev/public             626218    434830    191388     69%
/london    wansvr:/london          414858    265108    149750     63%
```

The **-i** option to **df** also provides additional information about the number of free and used inodes.

dfspace is a shell script interface to **df**. Without options, it presents the filesystem data in a more readable format than **df**. When used with its options, **df** provides more comprehensive information than **dfspace**.

In the above example, there are three local filesystems:

- */dev/root*
- */dev/u*
- */dev/public*

and one remote filesystem:

- *wansvr:/london*

All of these local filesystems have adequate numbers of blocks and inodes remaining for use. You should aim to keep at least 15% of free space on each filesystem. This helps to prevent fragmentation which slows down disk I/O. In the above example there are no problems with the filesystems */dev/u* and */dev/public* which are less than 85% used. The *root* filesystem (*/dev/root*), however, is 93% full. This filesystem is relatively static apart from the temporary file storage directories */tmp* and */usr/tmp*. In the configuration shown, there is very little free space in these directories. Possible solutions are to create divisions to hold these directories on other disks, or increase the size of the *root* filesystem.

du(C) is another command that can be used to investigate disk usage. It differs from **df** and **dfspace** because it reports the number of 512-byte blocks that files and directories contain rather than the contents of an entire filesystem. If no path is specified, **du** reports recursively on files and directories in and below the current directory. Its use is usually confined to sizing file and directory contents.

ps — check process activity

The **ps**(C) command obtains information about active processes. It gives a "snapshot" picture of what processes are executing, which is useful when you are trying to identify what processes are loading the system. Without options, **ps** gives information about the login session from which it was invoked. If you use **ps** as user *root*, you can obtain information about all the system's processes. The most useful options are as follows:

Table A-1 ps options

Option	Reports on:
-e	print information on all processes
-f	generate a full listing
-l	generate a long listing (includes more fields)
-u	print information on a specified user (or users)

With various combinations of the above options you can, amongst other things, find out about the resource usage, priority and state of a process or groups of processes on the system. For example, below is an extract of output after typing **ps -el**:

```
 F S   UID   PID  PPID  C PRI NI  ADDR  SZ    WCHAN TTY      TIME CMD
31 S     0     0     0  0  95 20  1f21   0 f0299018 ?        0:00 sched
20 S     0     1     0  0  66 20   252  40 e0000000 ?       30:37 init
31 S     0     2     0  0  95 20   254   0 f00c687c ?        0:01 vhand
31 S     0     3     0  0  81 20   256   0 f00be318 ?        5:19 bdflush
. . .
20 S     0   204     1  0  76 20   416  96 f023451a ?        1:56 cron
20 S     0   441     1  0  75 20   972  44 f01076b8 03       0:00 getty
20 S 20213  8783     1  0  73 20  1855  48 f011bae4 006      0:04 ksh
20 S 13079 25014 24908  0  75 20  155c  48 f010ee28 p4       0:01 ksh
20 R 13079 25016 24910 22  36 20   506 144 f010ed58 p2       0:03 vi
20 S 12752 27895 26142  0  73 20   7b0  40 f011f75c 010      0:00 sh
20 Z 13297 25733 25153  0  51 20                             0:00 <defunct>
20 R 13297 26089 25148 45  28 20   8a8  48 f012123c p12      0:01 ksh
20 S 12752 26142     1  0  73 20  1ce2  48 f01214ec 010      0:04 csh
20 R 12752 28220 27898 55  25 20  1e16 188 f010f6b0 p25      0:01 email
20 S 12353 27047 25727  0  73 20  161c  44 f012179c p13      0:00 ksh
20 O 13585 28248 28205 36  37 20   cc9  92          p23      0:00 ps
20 S 20213 28240  8783  0  75 20   711 140 f01156f8 006      0:00 vi
. . .
```

The field headed `F` gives information about the status of a process as a combination of one or more octal flags. For example, the **sched** process at the top has a setting of 31 which is the sum of the flags 1, 10 and 20. This means that the **sched** process is part of the kernel (1), sleeping at a priority of 77 or more (10), and is loaded in primary memory (20). The priority is confirmed by consulting the `PRI` field further along the line which displays a priority of 95. In fact both **sched** (the swapper) and **vhand** (the paging daemon) are inactive but have the highest possible priority. Should either of them need to run in the future they will do so at the context switch following their waking up as no other process will have a higher priority. For more information on the octal flags displayed and their interpretation see **ps**(C).

The `S` column shows the state of each process. The states shown in the example: S, R, O and Z mean sleeping (waiting for an event), ready-to-run, on the processor (running) and zombie (defunct) respectively. There is only one process running, which is the **ps** command itself (see the penultimate line). Every other process is either waiting to run or waiting for a resource to become available. The exception is the zombie process which is currently terminating; this entry will only disappear from the process table if the parent issues a **wait**(S) system call.

The current priority of a process is also a useful indicator of what a process is doing. Check the value in the `PRI` field which can be interpreted as shown in the following table:

Table A-2 Priority values

Priority	Meaning
95	swapping/paging
88	waiting for an inode
81	waiting for I/O
80	waiting for buffer
76	waiting for pipe
75	waiting for tty input
74	waiting for tty output
73	waiting for exit
66	sleeping — lowest system mode priority
65	highest user mode priority
51	default user mode priority
0	lowest user mode priority

Looking back at the above **ps** output you can see, for example, that the getty process has a priority of 75, as it is (not surprisingly) waiting for some keyboard input. Whereas priority values between 66 and 95 are fixed for a specific action to be taken, anything lower than 66 indicates a user mode process. The running process in the above example (**ps**) is at priority 37 and is therefore in user mode.

The `C` field indicates the recent usage of CPU time by a process. This is useful for determining those processes which are making a machine slow currently.

The `NI` field shows the nice value of a process. This directly affects the calculation of its priority when it is being scheduled. All processes in the above example are running with the default nice value of 20.

The `TIME` field shows the minutes and seconds of CPU time used by processes. This is useful for seeing if any processes are CPU hogs, or runaway, gobbling up large amounts of CPU time.

The SZ field shows the swappable size of the process's data and stack in 1KB units. This information is of limited use in determining how much memory is currently occupied by a process as it does not take into account how much of the reported memory usage is shared. Totaling up this field for all memory resident processes will not produce a meaningful figure for current memory usage. It is useful on a per process basis as you can use it to compare the memory usage of different versions of an application.

NOTE If you booted your system from a file other than */unix* (such as */unix.old*), you must specify the name of that file with the **-n** option to **ps**. For example, **ps -ef -n unix.old**.

sar — system activity reporter

sar(ADM) provides information that can help you understand how system resources are being used on your system. This information can help you solve and avoid serious performance problems on your system.

The individual **sar** options are described on the **sar**(ADM) manual page.

For systems with an SCO SMP License, **mpsar**(ADM) reports systemwide statistics, and **cpusar**(ADM) reports per-CPU statistics.

The following table summarizes the functionality of each **sar**, **mpsar**, and **cpusar** option that reports an aspect of system activity:

Table A-3 sar, cpusar, and mpsar options

Option	Activity reported
-a	file access operations
-A	summarize all reports
-b	buffer cache
-B	copy buffers
-c	system calls
-d	block devices including disks and all SCSI peripherals
-F	floating point activity (**mpsar** only)
-g	serial I/O including overflows and character block usage
-h	scatter-gather and physical transfer buffers
-I	inter-CPU interrupts (**cpusar** and **mpsar** only)
-j	interrupts serviced per CPU (**cpusar** only)
-L	latches

(Continued on next page)

Table A-3 sar, cpusar, and mpsar options
(Continued)

Option	Activity reported
-m	System V message queue and semaphores
-n	namei cache
-O	asynchronous I/O (AIO)
-p	paging
-q	run and swap queues
-Q	processes locked to CPUs (**cpusar** and **mpsar** only)
-r	unused memory and swap
-R	process scheduling
-S	SCSI request blocks
-u	CPU utilization (default option for all **sar** commands)
-v	kernel tables
-w	paging and context switching
-y	terminal driver including hardware interrupts

How sar works

System activity recording is disabled by default on your system. If you wish to enable it, log in as *root*, enter the command **/usr/lib/sar/sar_enable -y**, then shut down and reboot the system. See **sar_enable**(ADM) for more information.

Once system activity recording has been started on your system, it measures internal activity using a number of counters contained in the kernel. Each time an operation is performed, this increments an associated counter. **sar**(ADM) can generate reports based on the raw data gathered from these counters. **sar** reports provide useful information to administrators who wish to find out if the system is performing adequately. **sar** can either gather system activity data at the present time, or extract historic information collected in data files created by **sadc**(ADM) (System Activity Data Collector) or **sa1**(ADM).

If system activity recording has been started, the following **crontab** entries exist for user *sys* in the file */usr/spool/cron/crontabs/sys*:

```
0 * * * 0-6 /usr/lib/sa/sa1
20,40 8-17 * * 1-5 /usr/lib/sa/sa1
```

The first **sa1** entry produces records every hour of every day of the week. The second entry does the same but at 20 and 40 minutes past the hour between 8 am and 5 pm from Monday to Friday. So, there is always a record made every hour, and at anticipated peak times of activity recordings are made every 20 minutes. If necessary, *root* can modify these entries using the **crontab**(C) command.

The output files are in binary format (for compactness) and are stored in */usr/adm/sa*. The filenames have the format *sadd,* where ***dd*** is the day of the month.

Running sar

To record system activity every ***t*** seconds for ***n*** intervals and save this data to ***sar_data,*** enter **sar -o *datafile t n*** on a single processor system, or **mpsar -o *datafile t n*** on a multiprocessor system.

For example, to collect data every 60 seconds for 10 minutes into the file */tmp/sar_data* on a single CPU machine, you would enter:

sar -o /tmp/sar_data 60 10

To examine the data from ***datafile,*** the **sar**(ADM) command is:

sar [***option*** ...] [**-f *datafile***]

and the **mpsar**(ADM) and **cpusar**(ADM) commands are:

mpsar [***option*** ...] [**-f *datafile***]

cpusar [***option*** ...] [**-f *datafile***]

Each ***option*** specifies the aspect of system activity that you want to examine. ***datafile*** is the name of the file that contains the statistics you want to view. For example, to view the **sar -v** report for the tenth day of the most recent month, enter:

sar -v -f /usr/adm/sa/sa10

You can also run **sar** to view system activity in "real time" rather than examining previously collected data. To do this, specify the sampling interval in seconds followed by the number of repetitions required. For example, to take 20 samples at an interval of 15 seconds, enter:

sar -v 15 20

As shipped, the system allows any user to run **sar** in real time. However, the files in the */usr/adm/sa* directory are readable only by *root*. You must change the permissions on the files in that directory if you want other users to be able to access **sar** data.

With certain options, if there is no information to display in any of the relevant fields after a specified time interval then a time stamp will be the only output to the screen. In all other cases zeros are displayed under each relevant column.

When tuning your system, we recommend that you use a benchmark and have the system under normal load for your application.

swap — check and add swap space

Swap space is secondary disk storage that is used when the system considers that there is insufficient main memory. On a well-configured system, it is primarily used for processing dirty pages when free memory drops below the value of the kernel parameter GPGSLO. If memory is very short, the kernel may swap whole processes out to swap. Candidates for swapping out are processes that have been waiting for an event to complete or have been stopped by a signal for more than two seconds. If a process is chosen to be swapped out then its stack and data pages are written to the swap device. (Initialized data and program text can always be reread from the original executable file on disk).

The system comes configured with one swap device. Adding additional swap devices with the **swap**(ADM) command makes more memory available to user processes. Swapping and excessive paging degrade system performance but augmenting the swap space is a way to make more memory available to executing processes without optimizing the size of the kernel and its internal data structures and without adding physical memory.

The following command adds a second swap device, */dev/swap1*, to the system. The swap area starts 0 blocks into the swap device and the swap device is 16000 512-byte blocks in size.

```
swap -a /dev/swap1 0 16000
```

Use the **swap -l** command to see statistics about all the swap devices currently configured on the system. You can also see how much swap is configured on your system at startup by checking **nswap**. This is listed in the configuration and diagnostic file */usr/adm/messages* as a number of 512-byte blocks.

Running the **swap -a** command adds a second swap device only until the system is rebooted. To ensure that the second swap device is available every time the system is rebooted, use a startup script in the */etc/rc2.d* directory. For example, you could call it *S09AddSwap*.

In this release, a swap area can also be created within a filesystem to allow swapping to a file. To do this, you must marry a block special device to a regular file. For more information, see **swap**(ADM) and **marry**(ADM).

timex — examine system activity per command

timex(ADM) times a command and reports the system activities that occurred on behalf of the command as it executed. Run without options it reports the amount of real (clock) time that expired while the command was executing and the amount of CPU time (user and system) that was devoted to the process. For example:

```
# timex command command_options
real    6:54.30
user      53.98
sys       14.86
```

Running **timex -s** is roughly equivalent to running **sar -A**, but it displays system statistics only from when you issued the command until the command finished executing. If no other programs are running, this information can help identify which resources a specific command uses during its execution. System consumption can be collected for each application program and used for tuning the heavily loaded resources. Other information is available if the process accounting software is installed; see **timex**(ADM) for more information.

NOTE To enable process accounting, log in as *root*, enter the command **/usr/lib/acct/acct_enable -y**, then shutdown and reboot the system. See **acct_enable**(ADM) for more information.

timex belongs to a family of commands that report command resource usage. It can be regarded as an extension to **time**(C) which has no options and produces output identical to **timex** without options. If you wish to use **time** then you must invoke it by its full pathname as each of the Bourne, Korn and C shells have their own built-in version. The output from each of the shell built-ins varies slightly but is just as limited. The C shell, however, does add in average CPU usage of the specified command.

vmstat — virtual memory statistics

vmstat(C) is a useful tool for monitoring system performance but is not as comprehensive as **sar**. **vmstat** gives an immediate picture of how a system is functioning. It enables you to see if system resources are being used within their capacity.

vmstat's default output concentrates on four types of system activity — process, paging/swapping, system and CPU activity. If a timing interval is specified then **vmstat** produces indefinite output until you press ⟨Del⟩. Consider the following example for the command **vmstat 5**:

```
PROCS    PAGING                                                     SYSTEM   CPU
r  b  w  frs     dmd sw cch fil pft frp pos pif pis rso rsi  sy  cs  us su id

1 126  0 64000   0   0   0   0   0   0   0   0   0   0   0   59  34   0  3 97
0 127  0 64000   8   0   0   0   0   0   0   0   0   0   0   47  22   0  2 98
1 126  0 64000   0   0   0   0   0   0   0   0   0   0   0   45  16   0  2 98
0 127  0 64000   0   0   0   0   0   0   0   0   0   0   0   86  23   1  5 94
0 127  0 64000   0   0   0   0   0   0   0   0   0   0   0   24  12   0  1 99
1 129  0 64000  10   0  15   0  55   0   0   0   0   0   0 1369  43  19 42 39
0 130  0 64000   0   0   0   0   0   0   0   0   0   0   0  277  36   2  6 92
0 130  0 64000   0   0   0   0   0   0   0   0   0   0   0   78  26   0  1 99
0 130  0 64000   0   0   0   0   0   0   0   0   0   0   0  117  36   0  1 99
0 130  0 64000   0   0   0   0   0   0   0   0   0   0   0  138  46   0  2 98
0 130  0 64000   0   0   0   0   0   0   0   0   0   0   0  144  51   1  2 97
. . .
```

In this case **vmstat** displays data at regular intervals. Each display representing an average of the activity over the preceding five second interval.

The `PROCS` heading encompasses the first three fields of output:

`r` number of processes on the run queue

`b` number of processes blocked waiting for a resource

`w` number of processes swapped out

During the sample period there were no swapped out processes, hardly any processes on the run queue, and between 126 and 130 blocked processes. Any process which was ready to run would not spend much time on the run queue. This conclusion is reinforced by the value of `id` under the `CPU` heading which shows that the system is almost 100% idle most of the time.

The `PAGING` heading encompasses both paging and swapping activity on the system. The operating system does not preallocate swap space to running processes. It only allocates swap space to processes that have been swapped at least once; this space is only relinquished when such a process terminates. It does, however, decrease its internal count of available swappable memory.

In the above example, the amount of free swap space (frs) remains a constant 64000 (roughly 32MB in 512-byte units). Because this is the amount of swap originally configured for this system, no swapping or paging out to disk occurred during the sampling period. This is confirmed by the zero value of the w field. The fields from pos to rsi also show that no processes or regions were swapped in or out during the time that **vmstat** was running.

There is a brief amount of paging activity on the sixth line of output. One or more processes attempted to access pages that were not currently valid. To satisfy the demand for these pages, the kernel obtained them from the page cache (cch) in memory or from filesystems on disk but not from swap (sw).

If a process invokes the **fork**(S) system call, this creates an additional copy, or child process, of the original process. The new process shares the data or stack regions of its parent. The pages in these regions are marked copy-on-write (COW). This is to avoid wasting CPU and memory resources because the usual purpose of a fork is for either the parent or child process to execute a new command in place of itself. If, instead, the parent or child process tries to write to a page marked COW, this generates a protection fault (pft) causing the page fault handler in the kernel to make a copy of the page.

The dmd field accounts for a combination of demand zero pages (those created and initialized with zeros for data storage) and demand fill pages (those created and filled with text).

System call (sy) and context switching activity (cs) can also be seen under the SYSTEM heading.

The **-s** option to **vmstat** reports statistics about paging activity since the system was started or in a specified time interval:

```
  64000 free swap space
  12222 demand zero and demand fill pages
  25932 pages on swap
  44589 pages in cache
  28719 pages on file
  33791 protection fault
  84644 pages are freed
     23 success in swapping out a process
      0 fail in swapping in a process
     22 success in swapping in a process
     98 swapping out a region
     64 swapping in a region
 457461 cpu context switches
1870524 system calls
```

Lines showing large values for `pages on swap`, `success in swapping out a process`, `success in swapping in a process`, `swapping out a region`, and `swapping in a region` may indicate that excessive swapping or paging is degrading performance.

The **-f** option to **vmstat** provides information about the number of forks (that is, new processes created) since the system was started or in a specified time interval. For example, to monitor how many fork system calls are being invoked every second, use the command **vmstat -f 1**:

```
0 forks
0 forks
2 forks
1 forks
0 forks
. . .
```

A

Appendix B

Configuring kernel parameters

Kernel parameters control the allocation of various kernel resources. These resources are constantly being used, released and recycled, and include:

buffers	Recently used data is cached in memory; buffers increase performance by reducing the need to read data from disk. Buffers also allow efficient transfer of data by moving it in large units.
table entries	Space in system tables that the kernel uses to keep track of current tasks, resources, and events.
policies	Governing such things as security, and conformance to various standards.

Other parameters are used to indicate control the behavior of device drivers or the available quantity of special resources such as the number of multiscreens or semaphores.

Each resource limit is represented by a separate kernel parameter. The limit imposed by a parameter can be decreased or extended, sometimes at the expense of other resources. Deciding how to optimize the use of these resources is one aspect of kernel performance tuning.

For a description of the tools available for examining and changing parameters, see "Configuration tools" (page 188).

For a description of the various kernel parameters that you can change using the **configure**(ADM) utility or via the **Hardware/Kernel Manager**, see "Kernel parameters that you can change using configure" (page 191).

For a description of the various kernel parameters that you can only change from the command line using the **idtune**(ADM) utility, see "Using idtune to reallocate kernel resources" (page 190).

See "Using configure to change kernel resources" (page 189) for a description of how to run the **configure**(ADM) utility.

If you have TCP/IP installed on your system, see Appendix C, "Configuring TCP/IP tunable parameters" (page 225).

If you are using the LAN Manager Client Filesystem (LMCFS), see "LAN Manager Client Filesystem parameters" (page 223).

When to change system parameters

Among the cases in which you may need to reallocate system resources are:

- You install additional physical memory and thus have greater memory resources to allocate.
- Persistent error messages are being displayed on the system console indicating that certain resources are used up, such as inodes or table entries.
- The system response time is consistently slow, indicating that other resources are too constrained for the system to operate efficiently (as when too little physical memory is installed).
- Resource usage needs to be tailored to meet the needs of a particular application.

If one of your performance goals is to reduce the size of the kernel (usually because the system is paging excessively or swapping), first concentrate on tunable parameters that control large structures. The following table lists a small subset of kernel tunable parameters and indicates the cost (or benefit) in bytes of incrementing (or decrementing) each parameter by a single unit. For example, if **NCLIST** set to 200, this requires 200 times 72 bytes, or approximately 14KB of memory.

Parameter	Number of bytes per unit parameter
DTCACHEENTS	44
DTHASHQS	8
HTCACHEENTS	44
HTHASHQS	8
NBUF	1024
NCLIST	72 (64 for the buffer + 8 for the header)
NHBUF	8
NHINODE	8
NMPBUF	4096
MSGMAP	8
NSPTTYS	246
NSTREAM	80 (52 for the STREAMS header + 28 for the extended header)
MAX_INODE	76 per entry added to the dynamic in-core inode table
MAX_PROC	344 per entry added to the dynamic process table
MAX_FILE	12 per entry added to the dynamic open file table
MAX_REGION	76 per entry added to the dynamic region table

Dynamic table parameters such as **MAX_PROC** usually have their values set to 0. Each table grows in size as more entries are needed. The memory overhead of the grown kernel table can be found by multiplying the values shown above by the number of table entries reported by **getconf**(C). For example, from the Korn shell, you can find the current size of the process table by entering:

```
let nproc=344*$(getconf KERNEL_PROC)
echo "Size of process table in bytes is $nproc"
```

Specialized applications often require the reallocation of key system resources for optimum performance. For example, users with large databases may find that they need more System V semaphores than are currently allocated.

Most of the tunable parameters discussed in this chapter are defined in */etc/conf/cf.d/mtune*. This file lists the default, maximum and minimum values respectively of each of the parameters specified. To change the values of specific tunable parameters manually, use the appropriate tool as described in "Configuration tools" (page 188).

Configuration tools

The following tools are available for examining and/or changing tunable parameters:

configure A menu-driven program that allows you to examine and modify the value of tunable kernel parameters. This program is also accessible via the **Hardware/Kernel Manager**. See "Using configure to change kernel resources" (page 189) and **configure**(ADM) for more information.

getconf This utility reports configuration-dependent values for various standards and for dynamic kernel tables; use **setconf** to modify temporarily those values that relate to dynamic kernel tables. See "Examining and changing configuration-dependent values" (page 223) and **getconf**(C) for more information.

idtune Modify the values of some tunable parameters (defined in */etc/conf/cf.d/mtune*) that cannot be modified with **configure**. See "Using idtune to reallocate kernel resources" (page 190) and **idtune**(ADM) for more information.

iddeftune Run this command to modify the values of certain tunable parameters if you increase the amount of physical memory (RAM) to more than 32MB. See **iddeftune**(ADM) for more information.

ifconfig Reconfigure the TCP/IP protocol stack belonging to a single network interface. See "Using ifconfig to change parameters for a network card" (page 225) and **ifconfig**(ADMN) for more information.

inconfig Reconfigure default TCP/IP settings for *all* network interfaces. See "Using inconfig to change global TCP/IP parameters" (page 226) and **inconfig**(ADMN) for more information.

Network Configuration Manager
Examine, configure, or modify network protocol stacks (*chains*). The **Network Configuration Manager** is the graphical version of **netconfig**(ADM). See Chapter 25, "Configuring network connections" in the *SCO OpenServer Handbook* for more information.

setconf Increase dynamic kernel table sizes, or decrease maximum size of dynamic kernel tables. The new value only remains in force until the system is next rebooted. See "Examining and changing configuration-dependent values" (page 223) and **setconf**(ADM) for more information.

Using configure to change kernel resources

The **configure**(ADM) utility is a menu-driven program that presents each tunable kernel parameter and prompts for modification.

To change a kernel parameter using **configure,** do the following:

1. Enter the following commands as *root* to run **configure**:

   ```
   cd /etc/conf/cf.d
   ./configure
   ```

2. The **configure** menu displays groups of parameter categories; their individual meanings are discussed in "Kernel parameters that you can change using configure" (page 191).

   Choose a category by entering the number preceding it. The resources in that category are displayed, one by one, each with its current value. Enter a new value for the resource, or to retain the current value, press ⟨Enter⟩. After all the resources in the category are displayed, **configure** returns to the category menu prompt. Return to the Main Menu to choose another category or exit **configure** by entering " q ".

   NOTE The software drivers associated with a parameter must be present in the kernel for the setting of the parameter to have any effect.

3. After you finish changing parameters, link them into a new kernel and reboot your system as described in "Relinking the kernel" in the *SCO OpenServer Handbook.*

   NOTE If you wish to set the values of parameters defined in */etc/conf/cf.d/mtune* from a shell script, you should use the **idtune**(ADM) command as described in "Using idtune to reallocate kernel resources" (page 190).

Using idtune to reallocate kernel resources

You cannot use **configure** to change some kernel parameters because they are not generally considered to need adjusting. If you do need to alter such a parameter, log in as *root* and use the **idtune**(ADM) command:

```
cd /etc/conf/cf.d
/etc/conf/bin/idtune resource value
```

resource is the name of the tunable parameter in uppercase as it appears in */etc/conf/cf.d/mtune* (see *mtune*(F)). ***value*** is the parameter's new value. After changing the parameter values, relink the kernel, shut down and reboot the system as described in "Relinking the kernel" in the *SCO OpenServer Handbook.*

You can use the **-f** option to **idtune** to force it to accept a value outside the range specified by the minimum and maximum values defined in *mtune.* If necessary, you can also use the **-min** and **-max** options to write new minimum and maximum values to the *mtune* file.

WARNING The **configure** and **idtune** commands write new values defined for kernel parameters to */etc/conf/cf.d/stune* (see *stune*(F)). Do not edit *mtune* itself as it can be a valuable reference.

The following sections describe the parameters that can only be tuned using **idtune**:

- "Boot load extension parameters" (page 222)
- "Buffer cache free list" (page 195)
- "Hardware and device driver parameters" (page 222)
- "Memory management parameters" (page 197)
- "Message queue parameters" (page 215)
- "Semaphore parameters" (page 217)
- "Shared memory parameters" (page 218)
- "STREAMS parameters" (page 213)
- "System parameters" (page 219)
- "LAN Manager Client Filesystem parameters" (page 223)

Kernel parameters that you can change using configure

The tunable parameters that you can change using **configure**(ADM) are grouped into two sets of categories depending on whether they affect system performance or configuration:

Performance tunables

- "Buffer management" (page 192)
- "Processes and paging" (page 195)
- "TTYs" (page 197)
- "Name cache" (page 198)
- "Asynchronous I/O" (page 199)
- "Virtual disks" (page 200)

Configuration tunables

- "User and group configuration" (page 201)
- "Security" (page 203)
- "TTY and console configuration" (page 204)
- "Filesystem configuration" (page 205)
- "Table limits" (page 207)
- "STREAMS" (page 209)
- "Message queues" (page 213)
- "Event queues" (page 216)
- "Semaphores" (page 216)
- "Shared memory" (page 218)
- "Miscellaneous system parameters" (page 219)
- "Miscellaneous device drivers and hardware parameters" (page 220)

Buffer management

The following tunables may be used to tune the performance of your system's buffers.

NBUF

The amount of memory in 1KB units allocated for use by the system buffer cache at boot time. The system buffer cache is memory used as a temporary storage area between the disk and user address space when reading to or writing from mounted filesystems.

If **NBUF** is set to the default of 0, the system calculates the size of the buffer cache automatically.

The size of the buffer cache is displayed as "kernel i/o bufs" at boot time, and is recorded along with other configuration information in */usr/adm/messages*. The hit rate on the buffer cache increases as the number of buffers is increased. Cache hits reduce the number of disk accesses and thus may improve overall disk I/O performance. Study the **sar -b** report for statistics about the cache hit rate on your system. See "Increasing disk I/O throughput by increasing the buffer cache size" (page 75) for more information.

The system buffer cache typically contains between 300 and 600 buffers, but may contain 8000 or more buffers on a large server system. The maximum possible number of buffers is 450000. On HTFS, EAFS, AFS, and S51K filesystems, each buffer uses 1KB of memory plus a 72-byte header. Having an unnecessarily large buffer cache can degrade system performance because too little space is available for executing processes.

If you are using the DTFS filesystem, buffers are multiples of 512 bytes in size ranging from 512 bytes to 4KB. The number of buffers in the buffer cache is not constant in this case and varies with demand.

For optimal performance, you should adjust the number of hash queues (**NHBUF**) when you adjust the value of **NBUF**.

NHBUF

Specifies how many hash queues to allocate for buffer in the buffer cache. These are used to search for a buffer (given a device number and block number) rather than have to search through the entire list of buffers. This value of **NHBUF** must be a power of 2 ranging between 32 and 524288. Each hash queue costs 8 bytes of memory. The default value of **NHBUF** is 0 which sets the number of hash queues automatically:

- On single processor machines, **NHBUF** is set to the power of 2 that is less than or equal to half the value of **NBUF**.
- On multiprocessor machines, **NHBUF** is set to the power of 2 that is greater than or equal to twice the value of **NHBUF**. This reduces the likelihood of contention between processors wanting to access the same hash queue.

NMPBUF

Number of 4KB pages of memory used for the following types of multi-physical buffers:

- 16KB scatter-gather buffers (also known as cluster buffers). These are used to perform transfers of contiguous blocks of data on disk to and from the buffer cache.
- 4KB transfer buffers. These are used as intermediate storage when moving data between memory and peripheral devices with controllers that cannot access memory above 16MB.
- 1KB copy request buffers. These are used as intermediate storage when moving data between the buffer cache and peripheral devices with controllers that cannot access memory above 16MB.

NMPBUF should be set larger than 40 for machines with more than 16MB of memory and many users. The maximum possible size is 512.

If the value of **NMPBUF** is set to zero (default), the kernel determines a suitable value automatically at startup. In this case, it sets the value of **NMPBUF** in the range 40 to 64 depending on the amount of available memory.

PLOWBUFS

Amount of buffer cache that is contained in the first 16MB of RAM. It is expressed as a percentage, and should be as high as possible if the controllers for the peripheral devices (such as the disks) in your system cannot perform DMA to memory above the first 16MB (24-bit addressing controllers). If possible, set **PLOWBUFS** to 100 to eliminate the need to copy between buffers above 16MB and the copy buffers (see **NMPBUF**).

To ascertain if a SCSI host adapter can access memory above the first 16MB (32-bit addressing), consult the initialization message for its driver in the file */usr/adm/messages*. If the string `fts=` is followed by one or more characters including a `d`, the controller is 32-bit, otherwise it is 24-bit.

The default value of **PLOWBUFS** is 30, and can range between 1 and 100%. You need only change this parameter if your system has more than 16MB of RAM.

PUTBUFSZ

Specifies the size of the circular buffer, **putbuf**, that contains a copy of the last **PUTBUFSZ** characters written to the console by the operating system. The contents of **putbuf** can be viewed by using **crash**(ADM). The default and minimum value is 2000; the maximum is 10000.

NHINODE

Specifies the size of the inode hash table which must be a power of 2. It ranges from 64 to 8192 with a default value of 128.

BDFLUSHR

Specifies the rate for the **bdflush** daemon process to run, checking the need to write the filesystem buffers to the disk. The range is 1 to 300 seconds. The value of this parameter must be chosen in conjunction with the value of **NAUTOUP**. For example, it is nonsensical to set **NAUTOUP** to 10 and **BDFLUSHR** to 100; some buffers would be marked dirty 10 seconds after they were written, but would not be written to disk for another 90 seconds. Choose the values for these two parameters considering how long a dirty buffer may have to wait to be written to disk and how much disk-writing activity will occur each time **bdflush** becomes active. For example, if both **NAUTOUP** and **BDFLUSHR** are set to 40, buffers are 40 to 80 seconds old when written to disk and the system will sustain a large amount of disk-writing activity every 40 seconds. If **NAUTOUP** is set to 10 and **BDFLUSHR** is set to 40, buffers are 10 to 50 seconds old when written to disk and the system sustains a large amount of disk-writing activity every 40 seconds. Setting **NAUTOUP** to 40 and **BDFLUSHR** to 10 means that buffers are 40 to 50 seconds old when written, but the system sustains a smaller amount of disk writing activity every 10 seconds. With this setting, however, the system may devote more overhead time to searching the block lists.

WARNING If the system crashes with **BDFLUSHR** set to 300 (its maximum possible value) then 150 seconds worth of data, on average, will be lost from the buffer cache. A high value of **BDFLUSHR** may radically improve disk I/O performance but will do so at the risk of significant data loss.

NAUTOUP

Specifies the buffer age in seconds for automatic filesystem updates. A system buffer is written to disk when the **bdflush** daemon process runs and the buffer has been scheduled for a write for **NAUTOUP** seconds or more. This means that not all write buffers will be flushed each time **bdflush** runs. This enables a process to perform multiple writes to a buffer but fewer actual writes to a disk. This is because **bdflush** will sometimes run less than **NAUTOUP** seconds after certain buffers were written to. These will remain scheduled to be written until the next appropriate flush.

The ratio of writes between physical memory to kernel buffer and buffer to disk will tend to increase (that is, fewer actual disk writes) if the ratio between the flush rate **BDFLUSHR** and **NAUTOUP** decreases. Specifying a smaller limit increases system reliability by writing the buffers to disk more frequently and decreases system performance. Specifying a larger limit increases system performance at the expense of reliability. The default value is 10, and ranges between 0 (flush all buffers regardless of how short a time they were scheduled to be written) and 60 seconds.

Buffer cache free list

NOTE This parameter is not tunable using **configure**(ADM); you must use the **idtune**(ADM) command instead as described in "Using idtune to reallocate kernel resources" (page 190).

BFREEMIN

Sets a lower limit on the number of buffers that must remain in the free list. This allows some (possibly useful) blocks to remain on the free list even when a large file is accessed. If only **BFREEMIN** buffers remain on the freelist, a process requiring one or more buffers may sleep until more become available. The value of **BFREEMIN** is usually set to the default and minimum value of 0; the maximum value is 100. You may see an improvement in the buffer cache read and write hit rates reported by **sar -b** if you set the value of **BFREEMIN** to the smaller of **NBUF**/10 or 100. An improvement in performance is most likely on machines that are used primarily for media copying, **uucp** transfers, and running other applications that are both quasi-single-user and access many files.

Processes and paging

The tunable parameters **GPGSLO** and **GPGSHI** determine how often the paging daemon **vhand** runs. **vhand** can only run at clock ticks and it is responsible for freeing up memory when needed by processes. It uses a "least recently used" algorithm as an approximation of process working sets, and it writes out pages to disk that are not modified during a defined time period.

GPGSLO

Specifies the low value of free memory pages at which **vhand** will start stealing pages from processes. Normally, **GPGSLO** is tuned to a value that is about 1/16 of pagable memory. Increase the value to make the **vhand** daemon more likely to become active; decrease the value to make it less likely to become active.

The value of **GPGSLO** must be a positive whole number greater than or equal to 0 and less than or equal to 200. Its value must also be less than that of **GPGSHI**.

If **GPGSLO** is too large a fraction of the pages that are available, **vhand** becomes active before memory starts to become really short and useful pages may be paged out. If **GPGSLO** is too small, the system may run out of memory altogether between clock ticks. If this happens, the swapper daemon **sched** runs to swap whole processes out to disk.

GPGSHI

Specifies the high value of free memory pages at which **vhand** will stop stealing pages from processes. Normally **GPGSHI** is set to a value that is about 1/10 of pagable memory.

The value of **GPGSHI** must be a positive whole number greater than or equal to 1 and less than or equal to 300. Its value must also be greater than that of **GPGSLO**.

If the interval between **GPGSLO** and **GPGSHI** is too small, there will be a tendency for **vhand** to be constantly active once the number of free pages first drops below **GPGSLO**. If the interval is too large, a large amount of disk activity is required to write pages to disk.

MINARMEM

Threshold value that specifies the minimum amount (in pages) of physical memory that is available for the text and data segments of user processes. (Available physical memory for user processes is shown by the command **od -d availrmem** in **crash**(ADM).) The default and minimum is 25; the maximum is 40 pages.

If there is ever insufficient physical memory available to allocate to STREAMS or kernel memory allocated resources, an application may fail or hang, and the system will display the following message on the console:

```
CONFIG: routine - n resident pages wanted
```

If you see this message, it is likely that your system has insufficient RAM.

MINASMEM

Threshold value that specifies the minimum size (in pages) that available virtual memory is allowed to reach. (Available virtual memory is shown by the command **od -d availsmem** in **crash**(ADM).) More swap space or physical memory must be added to the system if it runs out of virtual memory. In the case of adding swap space, this can be done dynamically using swap-to-file. If system performance is still poor because it is swapping or paging out excessively, add more RAM to the system. The default and minimum is 25; the maximum is 40 pages. If this limit is exceeded, the following message is displayed on the console:

```
CONFIG: swapdel - Total swap area too small (MINASMEM = number exceeded)
```

If there is ever insufficient physical memory available to allocate to STREAMS or kernel memory allocated resources, an application may fail or hang, and the system will display the following message on the console:

```
CONFIG: routine - n swappable pages wanted
```

If you see this message, increasing the value of **MINASMEM** may help but it is more likely that your system has insufficient memory or swap space.

MAXSLICE

Specifies in clock ticks the maximum time slice for user processes. After a process executes for its allocated time slice, that process is suspended. The operating system then dispatches the highest priority process from the run queue, and allocates to it **MAXSLICE** clock ticks. **MAXSLICE** must be a value from 25 to 100; the default is 100.

SPTMAP

Determines the size of the map entry array used for managing kernel virtual address space. The default value is 200; the minimum and maximum values are 100 and 500.

Memory management parameters

NOTE This group of parameters is not tunable using **configure**(ADM); you must use the **idtune**(ADM) command instead as described in "Using idtune to reallocate kernel resources" (page 190).

MAXSC

Specifies the maximum number of pages that are swapped out in a single operation. The default and maximum value is 8.

MAXFC

Maximum number of pages that are added to the free list in a single operation. The default and maximum value is 8.

TTYs

The following parameters control various data structure sizes and other limits in character device drivers provided with the operating system.

NCLIST

Specifies the number of character list buffers to allocate. Each buffer contains up to 64 bytes of data. The buffers are dynamically linked to form input and output queues for the terminal lines and other slow-speed devices. The average number of buffers needed for each terminal is in the range of 5 to 10. Each entry (buffer space plus header) costs 72 bytes.
When full, input and output characters dealing with terminals are lost, although echoing continues, and the following message is displayed on the console:

```
CONFIG: Out of clists (NCLIST = number exceeded)
```

The default and minimum value of **NCLIST** is 120, and the maximum is 16640.

For users logged in over serial lines with speeds up to 9600 bps, the recommended setting of **NCLIST** is 10 times the maximum number of users that you expect will log in simultaneously. You should also increase the **TTHOG** parameter; this controls the effective maximum size of the raw input queue for fast serial lines.

Since each buffer is 64 bytes in size, you should increase **NCLIST** by **TTHOG** divided by 64 and multiplied by the number of fast serial lines, as shown in the following table:

TTHOG value	Increase NCLIST by
2048	32 * number of fast serial lines
4096	64 * number of fast serial lines
8192	128 * number of fast serial lines

TTHOG
Sets the effective size of the raw queue of the tty driver. The default and minimum value is 256 bytes; the maximum is 8192 bytes. Increasing the value of this parameter allows more unprocessed characters to be retained in the tty buffer, which may prevent input characters from being lost if the system is extremely busy.

If you are using sustained data transfer rates greater than 9600 bps, you should increase **TTHOG** to 2048 or 4096 bytes depending on the demands of the application. You must also increase the value of **NCLIST** to match the increased value of **TTHOG**.

Name cache

The following parameters control the performance of the namei caches that are used to speed the translation of filenames to inode numbers.

Parameters beginning with **HT** control the namei cache used with HTFS, EAFS, and AFS filesystems (all based on the **ht** filesystem driver).

HTCACHEENTS
Number of name components in the **ht** namei cache. It must have a value of between 1 and 4096; the default is 256. The recommended value for diverse workgroups is to make **HTCACHEENTS** large, roughly three times the maximum grown size of the in-core inode table reported by **sar -v**.

HTHASHQS
Number of hash queues for the **ht** namei cache. **HTHASHQS** must be a prime number between 1 and 8191; the default is 61. The recommended value of **HTHASHQS** for diverse workgroups is to make it at least half the size of **HTCACHEENTS**.

HTOFBIAS
Determines the bias towards keeping the names of open files in the **ht** namei cache. It must have a value of between 1 and 256; the default is 8. The higher that you make the value of **HTOFBIAS**, the longer the names will remain in the cache. A value of 0 means that the names have no special caching priority.

Parameters beginning with **DT** control the namei cache used with DTFS filesystems (based on the **dt** filesystem driver).

DTCACHEENTS
: Number of name components in the **dt** namei cache. It must have a value of between 1 and 4096; the default is 256. The recommended value for diverse workgroups is to make **DTCACHEENTS** large, roughly three times the maximum grown size of the in-core inode table reported by **sar -v**.

DTHASHQS
: Number of hash queues for the **dt** namei cache. **DTHASHQS** must be a prime number between 1 and 8191; the default is 61. The recommended value of **DTHASHQS** for diverse workgroups is to make it at least half the size of **DTCACHEENTS**.

DTOFBIAS
: Determines the bias towards keeping the names of open files in the **dt** namei cache. It must have a value of between 1 and 256; the default is 8. The higher that you make the value of **DTOFBIAS**, the longer the names will remain in the cache. A value of 0 means that the names have no special caching priority.

Asynchronous I/O

The asynchronous I/O feature supports asynchronous I/O operations on raw disk partitions. It must be added to the kernel using the **mkdev aio** command for these parameters to have any effect (see **aio**(HW) for more information).

NAIOPROC
: Size of the AIO process table that determines the number of processes that may be simultaneously performing asynchronous I/O. The range of values is between 1 and 16; the default is 5. When the AIO process table overflows, the following message is displayed on the console:

```
CONFIG: aio_memlock - AIO process table overflow (NAIOPROC = number exceeded)
```

NAIOREQ
: Size of the AIO request table that determines the maximum number of pending asynchronous I/O requests. The range of values is between 5 and 200; the default is 120. When the AIO request table overflows, the following message is displayed on the console:

```
CONFIG: aio_breakup - AIO request table overflow (NAIOREQ = number exceeded)
```

NAIOBUF
: Size of the AIO buffer table that determines number of asynchronous I/O buffers. This should always be set to the same value as **NAIOREQ**. When the AIO buffer table overflows, the following message is displayed on the console:

```
CONFIG: aio_breakup - AIO buffer table overflow (NAIOBUF = number exceeded)
```

NAIOHBUF
Number of internal asynchronous hash queues. The range of values is between 1 and 50; the default is 25.

NAIOREQPP
Maximum number of asynchronous I/O requests that a single process can have pending. The default value is 120, meaning that a single process can potentially exhaust all asynchronous I/O resources. The range of values is between 30 and 200.

NAIOLOCKTBL
Number of entries in the internal kernel table for asynchronous I/O lock permissions. The range of values is between 5 and 20; the default is 10. If there are many entries in the */usr/lib/aiomemlock* file, this value may need to be increased. When the AIO lock table overflows, the following message is displayed on the console:

```
CONFIG: aio_setlockauth - AIO lock table overflow (NAIOLOCKTBL = number exceeded)
```

Virtual disks

The following parameters control the performance of virtual disk arrays if these are configured on your system.

VDUNITMAX
The maximum number of virtual disks that can be configured. This parameter defines the size of several structures used by the **vd** driver. On systems where the number of virtual disks is likely to be constant, set **VDUNITMAX** equal to the number of virtual disks. The default value is 100; the minimum and maximum values are 5 and 256.

VDJOBS
The maximum number of virtual disk jobs that can exist in the global job pool. The default value is 200; the minimum and maximum values are 100 and 400.

VDUNITJOBS
The maximum number of job structures and piece pool entries for each virtual disk in the system. A piece pool entry contains a piece structure for each disk piece in a virtual disk array. For example, a piece pool entry for a three-piece RAID 5 array contains three piece structures. Each job structure is 88 bytes in size. Each piece structure is 84 bytes in size. The default value of **VDUNITJOBS** is 100; the minimum and maximum values are 50 and 200.

VDHASHMAX
The size of the hash table used for protecting the integrity of data during read, modify, and write operations. Each hash table entry requires 24 bytes of memory. The value of **VDHASHMAX** must be a power of 2; the minimum and maximum values are 512 and 8192. The default value is 1024.

VDASYNCPARITY
Controls whether writes to the parity device on RAID 4 and 5 devices are performed asynchronously. The default is 1 (write asynchronously). If set to 0, the system waits for all I/O to complete.

VDASYNCWRITES
Controls whether writes to the other half of a RAID 1 device (mirror) are performed asynchronously. The default is 1 (write asynchronously). If set to 0, the system waits for I/O on both halves of a mirror to complete.

VDASYNCMAX
Sets the maximum number of outstanding asynchronous writes for RAID 1, 4 and 5 configurations in asynchronous mode (that is, **VDASYNCWRITES** or **VDASYNCPARITY** are set to 1). The default value is 20; the minimum and maximum values are 20 and 64.

VDWRITEBACK
Enables write-back caching. This increases the throughput of a virtual disk by writing data asynchronously during the last phase of a read-modify-write job. The default value is 0 (do not use write-back caching). If set to 1, write-back caching is enabled.

WARNING Enabling write-back caching may compromise the integrity of the data if the system crashes. Use this feature only at your own discretion.

VDRPT
The interval in seconds between error conditions being reported. The default value is 3600; the minimum and maximum values are 0 and 86400 seconds. If set to 0, errors are only reported when detected.

User and group configuration

The following parameters control resources that are specific to individual users or groups.

NOFILES
Specifies the maximum number of open files for each process. Unless an application package recommends that **NOFILES** be changed, the default setting should be left unaltered.

The Bourne, C and Korn shells all use three file table entries: standard input, standard output, and standard error (file descriptors 0, 1, and 2 respectively). This leaves the value of **NOFILES** minus 3 as the number of other open files available for each process. If a process requires up to three more than this number, then the standard files must be closed. This practice is not recommended and must be used with caution, if at all. If the configured value of **NOFILES** is greater than the maximum (11000) or less than the minimum (60), the configured value is set to the default (110), and a message is sent to the console.

Unless an application package recommends that **NOFILES** be changed, the default setting should be left as is.

ULIMIT

Specifies in 512-byte blocks the size of the largest file that an ordinary user can write. The default value is 2097151; that is, the largest file an ordinary user can write is approximately 1GB (one gigabyte). A lower limit can be enforced on users by changing the value of **ULIMIT** in the file */etc/default/login*; see **login**(M).

The **ULIMIT** parameter does not apply to reads; any user can read a file of any size.

MAXUP

Specifies how many concurrent user processes an ordinary user is allowed to run. The entry is in the range of 15 to 16000, with a default value of 100 processes. This value should be at least 10% smaller than the value of **MAX_PROC** (or the maximum grown size of the process table reported by **sar -v** if **MAX_PROC** is set to 0). This value is determined by the user identification number, not by the terminal. For example, the more people that are logged in on the same user identification, the quicker the default limit would be reached.

MAXUMEM

Maximum size of a process' virtual address space in 4096-byte pages. The allowed range of values is between 2560 and 131072; the default is 131072 pages (512MB). If you decrease this value and a process will not start due to lack of memory, its parent shell reports one of the messages: "Too big" or "Not enough space".

NGROUPS

Maximum number of simultaneous supplemental process groups per process. The value of **NGROUPS** can be set to any integral value from 0 to 128; the default value is 8.

NGROUPS maps to the POSIX.1 runtime value **NGROUPS_MAX** for which the minimum value allowed by FIPS is 8. To retain FIPS and XPG4 compliance, you must restrict the value of **NGROUPS** to be greater than or equal to 8.

CMASK

The default mask used by **umask**(S) for file creation. By default this is zero, meaning that the **umask** is not set in the kernel. The range of values is between 0 and 0777. See **chmod**(C)and **umask**(C) for an explanation of setting absolute mode file permissions.

CHOWN_RES

Controls system-wide **chown** kernel privilege (formally known as the **chown** kernel authorization) on all filesystems that set the POSIX.1 constant **_POSIX_CHOWN_RESTRICTED** (also defined in *X/Open CAE Specification, System Interfaces and Headers, Issue 4, 1992*). See **getconf**(C) for more information.

If set, **CHOWN_RES** prevents all users except *root* from changing ownership of files on all filesystems that support **_POSIX_CHOWN_RESTRICTED**. The default value of **CHOWN_RES** is 0 (not set) which causes the restriction not to be enforced.

You can also use the **chown** kernel privilege to control users' privilege to change file ownership. If **chown** kernel privilege is removed, some XPG4-conformant applications may fail if they use interprocess communication (semaphores, shared memory, and message passing). You should only set **chown** kernel privilege in this way if you require C2-level security.

IOV_MAX

Maximum size of the I/O vector (`struct iovec`) array (number of non-contiguous buffers) that can be used by the **readv**(S) (scatter read) and **writev**(S) (gather write) system calls. The default value is 512; the minimum and maximum values are 16 and 1024.

Security

The security profile (High, Improved, Traditional, or Low) can be selected as discussed in "Changing the system security profile" in the *System Administration Guide*. The security parameters can be set to modify the behavior of the security features and to ensure compatibility with utilities that expect traditional UNIX system behavior. Each of these parameters can be set to 0 (off) or 1 (on).

SECLUID

Controls the enforcement of login user ID (LUID). Under SCO's implementation of C2 requirements, every process must have an LUID. This means that processes that set UIDs or GIDs, such as the printer scheduler (**lpsched**), must have an LUID set when started at system startup in */etc/rc2.d/S80lp*. This can cause problems with **setuid** programs. When the security default is set to a profile other than "High", enforcement of LUID is relaxed and **setuid** programs do not require an LUID to run.

B

SECSTOPIO

Controls whether the kernel implements the **stopio**(S) system call. When **SECSTOPIO** is set to 1, the kernel acts on **stopio**(S) calls; when it is set to 0, the kernel ignores **stopio** calls. The **stopio** system call is used under C2 to ensure that a device is not held open by another process after it is reallocated. This means that other processes attempting to access the same device may be killed.

stopio(S) is used by **initcond**(ADM), which is called by **getty**(M) immediately before starting user interaction and by **init**(M) immediately after an interactive session has terminated.

SECCLEARID

Controls the clearing of SUID/SGID bits when a file is written. Under C2 requirements, the set user ID (SUID or **setuid**) and set group ID (SGID or **setgid**) bits on files must be cleared (removed) when a file is written. This prevents someone from replacing the contents of a **setuid** binary. This can cause problems with programs that do not expect this behavior. In the "Low" security profile, SUID and SGID bits are not cleared when files are written.

The following table summarizes the initial settings of the security parameters for each security profile.

Parameter	Low	Traditional	Improved	High
SECLUID	off	off	off	on
SECSTOPIO	off	on	on	on
SECCLEARID	off	on	on	on

TTY and console configuration

The multiscreen parameters determine the number of console multiscreens that can run simultaneously on the system. Each multiscreen requires about 4 to 8KB of memory depending on the number of lines (25 or 43). If you need to save memory and are not using multiscreens heavily, set **NSCRN** to 4 and **SCRNMEM** to 16 or 32. When you do this, you must also **disable**(C) multiscreens 5-12 (*tty5* to *tty12*) or **getty** will generate warning messages when the system goes to multiuser mode. **NSCRN** and **SCRNMEM** can be set to smaller values than this if you are sure that you need fewer multiscreens.

TBLNK

Controls the console screen saver feature on VGA consoles (only). It is the number of seconds before the screen blanks to save wear on the monitor. **TBLNK** can have a value of 0 to 32767, with 0 (default) disabling screen blanking.

NSCRN

The number of console multiscreens. A value of 0 configures this value at boot time. The maximum value is 12.

SCRNMEM

Number of 1024-byte blocks used for console screen memory. A value of 0 (the default) configures this value at boot time based on the amount of memory installed. The range of values is between 9 and 128. Each multiscreen uses from 4 to 8KB of memory, so when using a non-zero value for this parameter, make **SCRNMEM** equal to 4 or 8 times the value of **NSCRN**.

NSPTTYS

Number of pseudo-ttys on the system. The default value is 16; the minimum and maximum values are 1 and 256. Each NSPTTYS requires 246 bytes of memory. This parameter should only be altered using the **mkdev ptty** command which also creates the additional device nodes. Pseudo-ttys are not related to console multiscreens; they are used for features such as serial multiscreens **mscreen**(M), for shell windows, and for remote logins.

NUMXT

Number of layers a subdevice can configure to support bitmapped display devices such as the **BLIT** or the **AT&T** 5620 and 730 terminals. The range of values is between 1 and 32; the default is 3. When this number is exceeded, the following message is displayed on the console:

```
CONFIG: xtinit - Cannot allocate xt link buffers (NUMXT = number exceeded)
```

Note that the **xt** driver must have been linked into the kernel using the **mkdev layers** command or the **Hardware/Kernel Manager** in order to use these display devices.

NUMSXT

Number of shell layers (**shl**(C)) a subdevice can configure. The range of values is between 1 and 32; the default is 6.

Note that the **sxt** driver must have been linked into the kernel using the **mkdev shl** command or the **Hardware/Kernel Manager** in order to use shell layers.

Filesystem configuration

The following parameters control the configuration of different filesystem types.

MAXVDEPTH

Maximum number of undeletable (versioned) files allowed in the DTFS and HTFS filesystems. A value of 0 disables versioning; the maximum value is 65535. This parameter can be overridden when the filesystem is mounted.

MINVTIME
Minimum time before a file is made undeletable (versioned) in the DTFS and HTFS filesystems. If set to 0, a file is always versioned (as long as **MAXVDEPTH** is greater than 0); if set to a value greater than 0, the file is versioned after it has existed for that number of seconds. The maximum value is 32767.

This parameter can be overridden when the filesystem is mounted.

ROOTCHKPT
If set to 0, disable checkpointing in a *root* HTFS filesystem; if set to 1 (default), enable checkpointing.

ROOTLOG
If set to 0, disable transaction intent logging in a *root* HTFS filesystem; if set to 1 (default), enable logging.

ROOTSYNC
If set to 0 (default), disable file synchronization on close on a *root* DTFS filesystem; if set to 1, enable synchronization on close.

ROOTNOCOMP
If set to 1, disable compression in a *root* DTFS filesystem; if set to 0 (default), enable compression.

ROOTMAXVDEPTH
Maximum number of undeletable (versioned) files on a *root* DTFS or HTFS filesystem. A value of 0 disables versioning.

ROOTMINVTIME
Minimum time before a file is made undeletable (versioned) on a *root* DTFS or HTFS filesystem. If set to 0, a file is always versioned (as long as - **ROOTMAXVDEPTH** is greater than 0); if set to a value greater than 0, the file is versioned after it has existed for that number of seconds.

DOSNMOUNT
Maximum number of mounted **DOS** filesystems. The range of values is between 0 and 25; the default is 4.

DOSNINODE
Maximum number of open inodes for **DOS** filesystems. The range of values is between 0 and 300; the default is 40.

Table limits

The following parameters control the allocation of memory to dynamic kernel tables.

TBLPAGES
The maximum number of pages of memory for dynamic tables. The range of values is between 10 and 10000; the default is 0 which means that the kernel configures the value based on the amount of memory available at system startup.

TBLDMAPAGES
The maximum number of pages of "dmaable" memory for dynamic tables. The range of values is between 10 and 1000 pages; the default is 100.

TBLLIMIT
The percentage of **TBLPAGES** or **TBLDMAPAGES** to which a single table may grow. The range of values is between 10 and 100%; the default is 70.

TBLSYSLIMIT
The percentage of memory allowed for dynamic tables if **TBLPAGES** is set to 0. The range of values is between 10 and 90%; the default is 25.

TBLMAP
The size of the dynamic table virtual space allocation map. The range of values is between 50 (default) and 500.

The following parameters control the maximum grown sizes of dynamic kernel tables. If set to 0, the maximum possible size defaults to the value shown by **getconf**(C) provided that sufficient **TBLPAGES** of memory have been allocated. For example, the command **getconf KERNEL_MOUNT_MAX** displays the maximum possible size of the mount table.

MAX_DISK
The maximum number of disk drives attached to the system. When the Diskinfo table overflows, the following message is displayed on the console:

```
CONFIG: dk_name - Diskinfo table overflow (MAX_DISK = number exceeded)
```

The minimum and maximum configurable values of **MAX_DISK** are 1 and 1024; the default value of 0 means that the kernel determines the number of disk drives dynamically.

MAX_INODE
Specifies the maximum number of inode table entries that can be allocated. Each table entry represents an in-core inode that is an active file such as a current directory, an open file, or a mount point. Pipes, clone drivers, sockets, semaphores and shared data also use inodes, although they are not associated with a disk file. The number of entries used depends on the number of opened files.

The minimum and maximum configurable values of **MAX_INODE** are 100 and 64000; the default value of 0 means that the in-core inode table grows dynamically.

Each open file requires an inode entry in the in-core inode table. If the inode table is too small, a message similar to the following is displayed on the console:

```
CONFIG: routine - Inode table overflow (MAX_INODE = number exceeded)
```

When the inode table overflows, the specific request is refused. Although not fatal to the system, inode table overflow may damage the operation of various spoolers, daemons, the mailer, and other important utilities. Abnormal results and missing data files are a common result.

If the system consistently displays this error message, use **sar -v** to evaluate whether your system needs tuning. The `inod-sz` value shows the number of inode table entries being used and the number of entries that have been allocated for use by the table.

MAX_PROC

Specifies the maximum number of process table entries that can be allocated. Each table entry represents an active process. The number of entries depends on the number of terminal lines available and the number of processes spawned by each user. If the process table is full, the following message appears on the console and in the file */usr/adm/messages*:

```
CONFIG: newproc - Process table overflow (MAX_PROC = number exceeded)
```

The minimum and maximum values of **MAX_PROC** that can be set are 50 and 16000; the default value is 0 which means that the process table grows dynamically. The `proc-sz` values shown by **sar -v** show how many process table entries are being used compared to those that have been dynamically allocated.

MAX_FILE

Specifies the maximum number of open file table entries that can be allocated. Each entry represents an open file.

The minimum and maximum values of **MAX_FILE** that can be set are 100 and 64000; the default value is 0 which means that the file table grows dynamically.

When the file table overflows, the following warning message is displayed on the system console:

```
CONFIG: falloc - File table overflow (MAX_FILE = number exceeded)
```

This parameter does not control the number of open files per process; see the description of **NOFILES** parameter.

MAX_REGION

Specifies the maximum number of region table entries that can be allocated. Most processes have three regions: text, data, and stack. Additional regions are needed for each shared memory segment and shared library (text and data) attached. However, the region table entry for the text of a "shared text" program is shared by all processes executing that program. Each shared-memory segment attached to one or more processes uses another region table entry.

The minimum and maximum values of **MAX_REGION** that can be set are 500 and 160000; the default value is 0 which means that the region table grows dynamically.

If you do configure **MAX_REGION**, as a general rule you should set its value to slightly more than three times greater than **MAX_PROC**. When the region table overflows, the following message is displayed on the console:

```
CONFIG: allocreg - Region table overflow (MAX_REGION = number exceeded)
```

MAX_MOUNT

Specifies the maximum number of mount table entries that can be allocated. Each entry represents a mounted filesystem. The *root* filesystem (*/*) is always the first entry. When full, the **mount**(S) system call returns the **EBUSY** error code.

The minimum and maximum values of **MAX_MOUNT** that can be configured are 4 and 4096; the default value of 0 means that the kernel grows the size of the mount table dynamically.

MAX_FLCKREC

Specifies the maximum number of lock table entries that can be allocated. This determines the number of file regions that can be locked by the system. The "lock-sz" value reported by **sar -v** shows the number of entries that are being used in comparison to the number that have been allocated.

The minimum and maximum values of **MAX_FLCKREC** that can be configured are 50 and 16000; the default value is 0 which means that the kernel grows the size of the record lock table dynamically according to the needs of the applications running on your system.

STREAMS

STREAMS is a facility for UNIX system communication services. It supports the implementation of services ranging from complete networking protocol suites (such as TCP/IP and IPX/SPX) to individual device drivers. STREAMS defines standard interfaces for character I/O. The associated mechanism is simple and open-ended, consisting of a set of system calls, kernel resources and kernel routines.

STREAMS use system resources that are limited by values defined in kernel configuration modules. Depending on the demand that you and other system users place on these resources, your system could run out of STREAMS resources if you do not first reset the allocations in the kernel configuration modules.

Running out of some STREAMS resources (such as those controlled by the **NSTREAM** parameter) generates kernel configuration error messages. STREAMS message buffers are dynamically allocated from memory up to a limit set by the value of the kernel parameter **NSTRPAGES**. This parameter sets the maximum number of pages of physical memory that can be dynamically allocated for use by STREAMS.

Before changing the STREAMS parameters **NSTREAM** or **NSTRPAGES**, you should check the current usage of STREAMS resources using the **strstat** command of the **crash**(ADM) utility or **netstat**(TC) with the **-m** option.

The following tunable parameters are associated with STREAMS processing:

NSTREAM

Number of stream head (**stdata** and **estdata**) data structures configured. One of each structure is needed for each stream opened, including both streams currently open from user processes and streams linked under multiplexers. The allowed range of values is between 1 and 512; the default is 32. The recommended configuration value is highly application-dependent, but a value of 256 usually suffices on a computer for running a single transport provider with moderate traffic. On Open Desktop, each X client also uses a pair of **stdata** and a pair of **estdata** structures. You should set **NSTREAM** to at least 256 on systems that are running X clients. When the number of stream head structures is exceeded, the following message is displayed on the console:

```
CONFIG: stropen - Out of streams (NSTREAM = n exceeded)
```

NSTRPAGES

The maximum number of pages of virtual memory that can be allocated dynamically for use by STREAMS message buffers. The allowed range of values is between 0 and 8000 pages; the default is 500.

If **NSTRPAGES** pages of virtual memory are not available when STREAMS are initialized at startup, the system displays the following message on the console for each STREAMS table that is affected:

```
CONFIG: strinit - Cannot alloc STREAMS name table \
    (NSTRPAGES = n too big)
```

If more buffers are requested than there are available pages of physical memory to create them, the system displays the following message on the console:

```
CONFIG: allocb - Out of streams memory (NSTRPAGES = n exceeded)
```

Extra memory is allocated temporarily for high priority buffers only. The system will then try to reduce STREAMS memory usage until it is less than **NSTRPAGES**.

NOTE Memory used by STREAMS for buffers is fully dynamic; memory can be freed as well as allocated.

The value of **NSTRPAGES** does not affect the size of the kernel at system startup although the size of the kernel will grow and shrink over time as pages of memory are allocated for use by STREAMS and subsequently released.

STRSPLITFRAC
Sets the percentage of **NSTRPAGES** above which the system tries to create buffers by splitting larger buffers that are on the free list. Below this limit, the system tries to allocate new pages of memory to create the buffers. **STRSPLITFRAC** can range between between 50 and 100 (percent); the default is 80. If you set **STRSPLITFRAC** lower than this, the system will use less memory for STREAMS but the memory that is used will tend to become fragmented and the kernel will require more CPU time to manage it.

NSTREVENT
Initial number of stream event structures configured. Stream event cells are used for recording process-specific information in the **poll** system call. They are also used in the implementation of the STREAMS **I_SETSIG ioctl** and in the kernel **bufcall** mechanism. A rough minimum value to configure would be the expected number of processes to be simultaneously using **poll** times the expected number of STREAMS being polled for each process, plus the expected number of processes expected to be using STREAMS concurrently. The default and minimum value is 256; the maximum is 512. Note that this number is not necessarily a hard upper limit on the number of event cells that are available on the system (see **MAXSEPGCNT**).

MAXSEPGCNT
The maximum (4KB) page count for stream events. If this value is 0 (minimum), only the amount defined by **NSTREVENT** is available for use. If the value is not 0 and if the kernel runs out of event cells, it will under some circumstances attempt to allocate an extra page of memory from which new event cells can be created. **MAXSEPGCNT** places a limit on the number of pages that can be allocated for this purpose. Once a page is allocated for event cells, however, it cannot be recovered later for use elsewhere. The default value is 1 and the maximum 32.

STRMSGSZ
Maximum allowable size of the data portion of any STREAMS message. This should usually be set just large enough to accommodate the maximum packet size restrictions of the configured STREAMS modules. If it is larger than necessary, a single **write** or **putmsg** can consume an inordinate number of message headers. The range of values is between 4096 and 524288; the default value of 16384 is sufficient for existing applications.

NUMSP
Determines the number of STREAMS pipe devices (*/dev/spx*, see **spx**(HW)) supported by the system. The default value is 64; the maximum and minimum values are 1 and 256. Administrators do not normally need to modify this parameter unless certain applications state that they require it.

NUMTIM
Maximum number of **timod**(M) STREAMS modules that can be pushed by the Transport Layer Interface (TLI) onto a stream head. This parameter limits the number of streams that can be opened. The default value is 16 but various protocol stacks (for example, **TCP**, **LMU**, or **NETBIOS**) may require its value to be set to 32, 64, or 128. Administrators do not normally need to modify this parameter.

NUMTRW
Maximum number of **timod**(M) STREAMS modules that can be pushed by the Transport Layer Interface (TLI) onto a stream head in order that the stream will accept **read**(S) and **write**(S) system calls. This parameter effectively limits the number of streams onto which the module can be pushed. The default value is 16 but various protocol stacks (for example, TCP, LMU, or NETBIOS) may require its value to be set to 32, 64, or 128. Administrators do not normally need to modify this parameter.

See "STREAMS parameters" (page 213) for a description of the STREAMS parameters that can only be tuned using **idtune**(ADM).

STREAMS parameters

NOTE This group of parameters is not tunable using **configure**(ADM); you must use the **idtune**(ADM) command instead as described in "Using idtune to reallocate kernel resources" (page 190).

NMUXLINK
: Number of stream multiplexer links configured. One link structure is required for each active multiplexer link (STREAMS **I_LINK ioctl**) in networking protocol stacks such as those used to implement TCP/IP and NFS. Each PPP link also requires such a structure. The number needed is application-dependent; the default value is 192. The minimum and maximum configurable values are 1 and 4096.

NSTRPUSH
: Maximum number of modules that may be pushed onto a stream. This prevents an errant user process from consuming all of the available queues on a single stream. The default possible value is 9. In practice, applications usually push at most four modules onto a stream.

NLOG
: Number of minor devices to be configured for the log driver; the active minor devices are 0 through (**NLOG**−1). The only value of 3 services an error logger (**strerr**) and a trace command (**strace**), with one left over for miscellaneous usage.

STRCTLSZ
: Maximum allowable size of the control portion of any STREAMS message. The control portion of a **putmsg** message is not subject to the constraints of the minimum/maximum packet size, so the value entered here is the only way of providing a limit for the control part of a message. The only possible value of 1024 is more than sufficient for existing applications.

Message queues

The following tunable parameters are associated with interprocess communication message queues:

MSGMAP
: Specifies the number of entries in the memory map for messages. An entry in the message map table says that **MSGSEG** / **MSGMAP** memory segments are free at a particular address.

MSGMAP measures how fragmented you expect your map to get. Its value can be small if you always send a few large messages, or it can be large if you send a lot of small messages. The suggested value for **MSGMAP** is approximately half the value of **MSGSEG**; this allocates two message segments per map entry. If the value of **MSGMAP** is set equal to **MSGSEG**, long messages may become totally fragmented with their component segments being randomly scattered across the map.

Do not set **MSGMAP** to a value greater than that of **MSGSEG**. The range of configurable values is from 4 to 32768; the default value is 512. Each entry costs 8 bytes.

MSGMAX
Maximum size of a message in bytes. The minimum value is 128, the default value is 8192 bytes, and the maximum possible size the kernel can process is 32767 bytes.

MSGMNB
Maximum number of bytes of memory that all the messages in any one message queue can occupy. The default value is 8192; the maximum and minimum values are 128 bytes and 65532 bytes.

MSGSEG
Number of **MSGSSZ** segments of memory allocated at kernel startup for holding messages. Therefore a total of **MSGSEG*****MSGSSZ** bytes of memory are allocated for messages.

NOTE The amount of memory allocated for messages must not exceed 128KB.

If **MSGSEG** is set at 0, then the kernel will auto-configure the values of **MSGSEG**, **MSGMAX**, and **MSGMNB**. For most memory configurations, **MSGSEG** is set to 1024, and **MSGMAX** and **MSGMNB** are both set to **MSGSEG*****MSGSSZ**.

The **IPC_NOWAIT** flag can be passed into many of the **msg** system calls. If this flag is passed, then the system calls will fail immediately if there is no space for a message. If this flag is not passed, then the system calls will sleep until there is room for the message.

To determine adequate values for each of the parameters, compute the maximum size and number of messages desired, and allocate that amount of space. For example, if the system will have at most 40 messages of 1KB each pending, then **MSGTQL** should be set to 40, and **MSGSEG** is computed as:

- 40 messages of 1K each = 40KB total message space.
- Divide total message space by **MSGSSZ** to get **MSGSEG**. If **MSGSSZ**=8 bytes, then **MSGSEG** = 40*1024/8 = 5120.

The default value of **MSGSEG** is 1024; the minimum and maximum values are 32 and 32768.

See "Message queue parameters" (this page) for a description of the message queue parameters that can only be tuned using **idtune**(ADM).

Message queue parameters

The following parameters are associated with System V IPC message queues.

NOTE This group of parameters is not tunable using **configure**(ADM); you must use the **idtune**(ADM) command instead as described in "Using idtune to reallocate kernel resources" (page 190).

MSGMNI

Maximum number of different message queues allowed system-wide. The default value of **MSGTQL** is 50; the minimum and maximum values are 1 and 1024. You should not normally need to adjust the value of this parameter.

MSGTQL

Number of system message headers that can be stored by the kernel; that is, the maximum number of unread messages at any given time. Each header costs 12 bytes. The default value of **MSGTQL** is 1024; the minimum and maximum values are 32 and 16383. You should not normally need to adjust the value of this parameter unless an application needs a large number of messages.

MSGSSZ

Size in bytes of the memory segment used for storing a message in a message queue.

A message that is shorter than a whole number multiple of memory segments will waste some bytes. For example, an 18-byte message requires three message segments if **MSGSSZ** is set to 8 bytes. In this case, 6 bytes of memory are unused, and unusable by other messages.

The product of the values of **MSGSSZ** and **MSGSEG** determines the total amount of data that can be present in all message queues on a system. This product should not be greater than 128KB.

The default value of **MSGSSZ** is 8 bytes; the minimum and maximum values are 4 bytes and 4096 bytes. The configured value of **MSGSSZ** must be divisible by 4. You should not normally need to adjust the value of this parameter.

Event queues

The following parameters control the configuration of the event queues.

EVQUEUES
Maximum number of open event queues systemwide. Each **EVQUEUES** costs 88 + (2 * **EVDEVSPERQ**) bytes of memory. The range of values is between 1 and 256; the default is 8.

EVDEVS
Maximum number of devices attached to event queues systemwide. Each **EVDEVS** costs 48 bytes of memory. The range of values is between 1 and 256; the default is 16. When the event table overflows, the following message is displayed on the console:

```
CONFIG: event - Event table full (EVDEVS = number exceeded)
```

EVDEVSPERQ
Maximum number of devices for each event queue. The range of values is between 1 and 16; the default is 3. When the event channel overflows, the following message is displayed on the console:

```
CONFIG: event - Event channel full (EVDEVSPERQ = number exceeded)
```

Semaphores

The following tunable parameters are associated with interprocess communication semaphores:

SEMMAP
Size of the control map used to manage semaphore sets. The default and minimum value is 10; the maximum is 100. Each entry costs 8 bytes.

SEMMNI
Number of semaphore identifiers in the kernel. This is the number of unique semaphore sets that can be active at any given time. The default and minimum value is 10; the maximum is 300. Each entry costs 32 bytes.

SEMMNU
Number of semaphore undo structures in the system. The size is equal to 8*(**SEMUME** + 2) bytes. See "Semaphore parameters" (page 217) for a definition of **SEMUME**. The range of values is between 10 and 100; the default is 30.

XSEMMAX
Size of the XENIX® semaphore table that determines the maximum number of XENIX semaphores allowed systemwide. The minimum value for **XSEMMAX** is 20, the maximum value is 90, and the default value is 60. When the XENIX semaphore table overflows, the following message is displayed on the console:

```
CONFIG: xsem_alloc - XENIX semaphore table overflow (XSEMMAX = number exceeded)
```

See "Semaphore parameters" (this page) for a description of the semaphore parameters that can only be tuned using **idtune**(ADM).

Semaphore parameters

NOTE This group of parameters is not tunable using **configure**(ADM); you must use the **idtune**(ADM) command instead as described in "Using idtune to reallocate kernel resources" (page 190).

SEM_NSEMS_MAX
Maximum number of POSIX.1b semaphores available for use on the system (provided by the SUDS library). The default value is 100; the minimum and maximum configurable values are 1 and 255 respectively.

The following parameters are associated with System V IPC semaphores only:

SEMMSL
Maximum number of semaphores for each semaphore identifier. The default and minimum value is 25; the maximum value is 60.

SEMOPM
Maximum number of semaphore operations that can be executed for each **semop**(S) call. The default value is 10; each entry costs 8 bytes.

SEMUME
Number of undo entries for each process. The default value is 10.

SEMVMX
Maximum value a semaphore can have. The default value is 32767.

SEMAEM
Maximum value for adjustment on exit, alias **semadj**. This value is used when a semaphore value becomes greater than or equal to the absolute value of **semop**, unless the program has set its own value. The default value is 16384.

SEMMNS
Number of semaphores in the system. The default and minimum value is 60; the maximum value is 300. Each entry costs 8 bytes.

Shared memory

The following tunable parameters are associated with interprocess communication shared memory:

SHMMAX
Maximum shared-memory segment size. The range of values is between 131072 and 80530637 bytes; the default value is 524288 bytes.

SHMMIN
Minimum shared-memory segment size. The default value is 1 byte.

XSDSEGS
Maximum number of XENIX special shared-data segments allowed system wide. The range of values is between 1 and 150; the default is 25. When the XENIX shared data table overflows, the following message is displayed on the console:

```
CONFIG: xsd_alloc - XENIX shared data table overflow (XSDSEGS = number exceeded)
```

XSDSLOTS
Number of slots for each XENIX shared data segment. The maximum number of XENIX special shared data segment attachments system wide is **XSDSEGS*XSDSLOTS**. The range of values is between 1 and 10; the default is 3.

See "Shared memory parameters" (this page) for a description of the shared memory parameters that can only be tuned using **idtune**(ADM).

Shared memory parameters

NOTE The following parameter is not tunable using **configure**(ADM); you must use the **idtune**(ADM) command instead as described in "Using idtune to reallocate kernel resources" (page 190).

SHMMNI
Maximum number of shared-memory identifiers systemwide. The minimum and default value is 100; the maximum is 2000. Each entry costs 52 bytes.

Miscellaneous system parameters

The following parameters control the size of the configuration string buffer, and the size of the kernel profiler symbol table.

MAX_CFGSIZE
: Maximum size of configuration information saved by the **tab**(HW) driver. This is the maximum size of information available using */dev/string/cfg* as described on the **string**(M) manual page. If this limit is exceeded, the following message is displayed on the console:

```
CONFIG: string: Configuration buffer full (MAX_CFGSIZE = number exceeded)
```

MAX_CFGSIZE ranges from 256 to 32768 bytes; the default is 1024 bytes.

PRFMAX
: Sets the maximum number of text symbols that the kernel profiler, */dev/prf*, can properly process. The range of values is between 2048 and 8192; the default is 4500. See **profiler**(ADM) for information about the kernel profiler.

System parameters

NOTE This group of parameters is not tunable using **configure**(ADM); you must use the **idtune**(ADM) command instead as described in "Using idtune to reallocate kernel resources" (page 190).

NODE
: System name. The value of **NODE** must not be greater than eight characters. The default value is "scosysv".

TIMEZONE
: Specifies the timezone in units of minutes different from Greenwich Mean Time (GMT). Note that the value specifies the system default timezone and not the value of the **TZ** environment variable. **TIMEZONE** can have a value from -1440 (east of GMT) to 1440 (west of GMT); the default is 480.

DSTFLAG
: Specifies the **dstflag** described for the **ctime**(S) system call. A value of 1 indicates Daylight Savings Time applies locally, zero is used otherwise.

KDBSYMSIZE
: Size of the kernel debugger symbol table in bytes. (This parameter is only useful if a debugger is linked into the kernel.) It must have a value of between 50000 and 500000; the default is 300000.

NCPYRIGHT
: Defines the maximum number of strings used to store some vendor driver copyright messages that may be displayed on the console when the system is booted. Modifying this parameter is unlikely to affect the display of most copyright messages.

Miscellaneous device drivers and hardware parameters

The following parameters control the configuration of various device drivers and hardware behavior.

CTBUFSIZE

Size of the tape buffer in kilobytes. This static buffer is allocated by the QIC-02 cartridge tape device driver (**ct**) when it is initialized at system startup. This parameter should have a value of between 32 and 256. Set this parameter to 0 if the **ct** driver is linked into the kernel but you either do not have or do not use a cartridge tape drive. The following are values that this parameter can take in various circumstances:

32KB
: bare minimum: this is insufficient to stream

64KB
: minimum to allow streaming (good for systems with little memory) or little tape use (if tape I/O performance is not critical)

96KB
: reduce to this at first if the default uses too much memory

128KB
: default: this offers good tradeoff performance between I/O and memory

192KB
: increase to this at first if the default provides poor I/O performance

256KB
: maximum size

NOTE The SCSI tape device driver (**Stp**) allocates a statically configured 128KB buffer for each device which is not controlled by this parameter. All SCSI tape drives including SCSI cartridge tape drives use the **Stp** driver.

SDSKOUT

Maximum number of simultaneous requests that can be queued for each SCSI disk. The SCSI disk driver (**Sdsk**) will sleep if no request blocks are available. The default value of this parameter is 4; the minimum and maximum values are 1 and 256. You should set **SDSKOUT** higher if the **-S** option to **sar**(ADM) (or **mpsar**(ADM) for SMP) reports that the system is running out of request blocks.

DMAEXCL

Specifies whether simultaneous DMA requests are allowed. Some computers have DMA chips that malfunction when more than one allocated channel is used simultaneously. **DMAEXCL** is set to 0 by default to allow simultaneous DMA on multiple channels. Set its value to 1 if this causes a problem.

KBTYPE

Determines the logical character protocol used between the keyboard and the keyboard driver. This tunable is set by default to 0 for **XT** scancodes and is recommended; a value of 1 specifies **AT** scancodes which are recognized by the console driver but not by the X server or by DOS emulators. All **AT**-compatible keyboards support both modes.

VGA_PLASMA

Set to 1 if an IBM® PS/2® model P70 or P75 VGA plasma display is present; set to 0 (default) if not.

NSHINTR

Maximum number of devices sharing the same interrupt vector. This has a default value of 8; the minimum and maximum values are 2 and 20. You should not normally need to modify this parameter.

DO387CR3

Controls the setting of high-order bits of Control Register 3 (CR3) when an 80387™ math coprocessor is installed. Because of design defects in early versions of the Intel® 80387™ chip (B1 stepping), this math coprocessor may not operate correctly in some computers. The problem causes a CPU to hang when DMA, paging, or coprocessor accesses occur. You can work around this problem by changing the **DO387CR3** parameter from the default value of 0 (switched off) to 1.

WARNING Do not set this parameter to 1 on 80486™ or Pentium™ machines.

DOWPCR0

If set, the kernel uses the write protection bit in Control Register 0 (CR0) to enable write protection in kernel mode. The default value is 1 which sets this parameter. This parameter is effectively disabled on machines which contain one or more 80386™ CPUs which do not support this feature.

MODE_SELECT

No effect. Mode-select checking on parallel (printer) ports can be adjusted on a per-printer basis using the `pa_tune[]` array defined in *<sys/paconf.h>* and documented in the file */etc/conf/pack.d/pa/space.c*.

Hardware and device driver parameters

NOTE This group of parameters is not tunable using **configure**(ADM); you must use the **idtune**(ADM) command instead as described in "Using idtune to reallocate kernel resources" (page 190).

NAHACCB
: Number of mailboxes available for the Adaptec 154X/164X host adapter driver to talk to other Adaptec hardware. The higher the number, the less likely it is that the driver has to sleep. It is not normally necessary to modify this parameter.

NEMAP
: Specifies the maximum number of **mapchan**(M) I/O translation mappings that can be in effect at the same time. The default value of this parameter is 10.

NKDVTTY
: Number of virtual terminals (8) supported by the console keyboard driver. Administrators should not modify this parameter.

Boot load extension parameters

NOTE This group of parameters is not tunable using **configure**(ADM); you must use the **idtune**(ADM) command instead as described in "Using idtune to reallocate kernel resources" (page 190).

EXTRA_NDEV
: Number of extra device slots in **`fmodsw[]`**, **`io_init[]`**, and **`io...[]`**. It defines the number of slots reserved in the device driver tables for Boot Time Loadable Drivers (BTLDs).

EXTRA_NEVENT
: Number of extra event slots. It defines the number of slots reserved in the event driver tables for BTLDs.

EXTRA_NFILSYS
: Number of extra types of filesystem. It defines the number of extra types of filesystem that can be loaded using BTLDs.

MAX_BDEV
: Maximum number of block devices (**bdevcnt** is at least this value). It defines the minimum number of entries in **`bdevsw[]`**, the block device switch table.

MAX_CDEV
: Maximum number of character devices (**cdevcnt** is at least this value). It defines the minimum number of entries in **`cdevsw[]`**, the character device switch table.

LAN Manager Client Filesystem parameters

NOTE The LAN Manager Client Filesystem (LMCFS) adds several kernel parameters to the *mtune* file that are not tunable using **configure**(ADM); you must use the **idtune**(ADM) command instead as described in "Using idtune to reallocate kernel resources" (page 190).

LMCFS_BUF_SZ
: Determines the maximum amount of data that LMCFS can transmit or receive in a single network packet. The default value is 4096 bytes.

LMCFS_LMINUM
: Controls the number of allocatable inodes. The default value is 150; the maximum value is 600. Set this value higher if users have many LMCFS files open simultaneously.

LMCFS_NUM_BUF
: Sets the number of server message block (SMB) data buffers used by LMCFS. The default value is 256; the maximum value is 8192. The size of each buffer is determined by **LMCFS_BUF_SZ**.

LMCFS_NUM_REQ
: Constrains the number of simultaneous SMB requests that can be made on the network. The default value is 64; the maximum value is 1024. This parameter should be set to at least one quarter of the value of **LMCFS_NUM_BUF**.

Examining and changing configuration-dependent values

getconf allows you to inspect the values of configuration-dependent variables for various standards, and the values of dynamic kernel table parameters. Below is an example of the use of **getconf**:

```
$ getconf NZERO
20
$ getconf CLK_TCK
100
```

This indicates that the default process priority on the system is 20 and the system clock runs at 100 ticks per second.

Path variables, such as **NAME_MAX** which defines the maximum filename length, depend on the filesystem type and therefore the pathname. These examples show the values of **NAME_MAX** for an HTFS and a XENIX filesystem:

```
# getconf NAME_MAX /htfs_filesystem
255
# getconf NAME_MAX /xenix_filesystem
14
```

For a complete list of the variable names to use with the command see **getconf**(C).

If you are logged in as *root*, you can use the **setconf**(ADM) command to change a subset of the configuration dependent parameters. Using **setconf**, you can increase the current size of the dynamic kernel tables or decrease their maximum possible size. You can also dynamically increase the number of character buffers available for use by the serial driver, for example:

```
setconf KERNEL_CLISTS 1024
```

The maximum possible number of such buffers that you can allocate is controlled by the **KERNEL_CLISTS_MAX** parameter.

NOTE Any change that you make using **setconf** remains in force only until the system is next rebooted. Use the **Hardware/Kernel Manager** or **configure** to make the change permanent.

Appendix C

Configuring TCP/IP tunable parameters

You can adjust the configuration parameters for TCP/IP using the **ifconfig**(ADMN) and **inconfig**(ADMN) utilities as described in the following sections:

- "Using ifconfig to change parameters for a network card" (this page)
- "Using inconfig to change global TCP/IP parameters" (page 226)

If you need to change STREAMS resources, you must use the **configure**(ADM) command as described in "Using configure to change kernel resources" (page 189).

Using ifconfig to change parameters for a network card

You can use the **ifconfig**(ADMN) command to reconfigure performance parameters for a single network interface. If you wish to make this change permanent you must edit the entry for the interface in the */etc/tcp* script.

The **metric, onepacket,** and **perf** parameters affect performance.

metric can be used to artificially raise the routing metric of the interface used by the routing daemon, **routed**(ADMN). This has the effect of making a route using this interface less favorable. For example, to set the metric for the **sme0** interface to 10, enter:

```
/etc/ifconfig sme0 inet metric 10
```

onepacket enables one-packet at a time operation for interfaces with small buffers that are unable to handle continuous streams of back-to-back packets. This parameter takes two arguments that allow you to define a small packet size, and the number of these that you will permit in the receive window.

This deals with TCP/IP implementations that can send more than one packet within the window size for the connection. Set the small packet size and count to zero if you are not interested in detecting small packets. For example, to set one-packet mode with a small packet threshold of one small packet of 512 bytes on the **e3A0** interface, enter:

```
/etc/ifconfig e3A0 inet onepacket 512 1
```

To turn off one-packet mode for this interface, enter:

```
/etc/ifconfig e3A0 inet -onepacket
```

perf allows you to tune performance parameters on a per-interface basis. The arguments to **perf** specify the receive and send window sizes in bytes, and whether TCP should restrict the data in a segment to a multiple of 1KB (a value of 0 restricts; 1 uses the full segment size).

The following example sets the receive and send window size to 4KB, and uses the maximum 1464-byte data size available in an Ethernet frame:

```
/etc/ifconfig sme0 inet perf 4096 4096 1
```

NOTE Segment truncation does not change the size of the Ethernet frame; this is fixed at 1530 bytes.

Using inconfig to change global TCP/IP parameters

As *root*, you can use the **inconfig**(ADMN) command to change the global default TCP/IP configuration values.

NOTE Any global performance parameters that you set using **inconfig** are overridden by per-interface values specified using **ifconfig**.

For example, to enable forwarding of IP packets, you would enter:

```
inconfig ipforwarding 1
```

inconfig updates the values of the parameters defined in */etc/default/inet* and those in use by the currently executing kernel. You do not need to reboot your system for these changes to take effect; **inconfig** dynamically updates the kernel with the changes you specify. Before doing so, it verifies that the values you input are valid. If they are not, the current values of the parameters are retained.

See "TCP/IP parameters" (page 227) for a description of the TCP/IP parameters that you can tune using **inconfig**.

TCP/IP parameters

The parameters that control the operation of TCP/IP are defined in the file */etc/default/inet*.

The parameters are grouped according to function:

- "Address Resolution Protocol (ARP) parameters" (this page)
- "Asynchronous half-duplex (ASYH) line connection parameters" (page 228)
- "Internet Control Message Protocol (ICMP) parameters" (page 228)
- "Internet Group Management Protocol (IGMP) parameters" (page 229)
- "Configuring the in-kernel network terminal (IKNT) driver" (page 229)
- "Internet Protocol (IP) parameters" (page 229)
- "Message block control logging (MBCL) parameters" (page 232)
- "NetBIOS parameters" (page 232)
- "Transmission Control Protocol (TCP) parameters" (page 232)
- "User Datagram Protocol (UDP) parameters" (page 234)

You should read the description for a parameter before you change it using **inconfig**(ADMN) as described in "Using inconfig to change global TCP/IP parameters" (page 226). The default values of the parameters are configured to work efficiently in most situations.

NOTE Never edit the settings for these parameters in the file */etc/default/inet*; always use **inconfig** to change them.

Address Resolution Protocol (ARP) parameters

The following parameters control the behavior of the Address Resolution Protocol (ARP).

arpprintfs
Controls logging of warnings from the kernel ARP driver. These are displayed on the console. If set to 0 (the default), debugging information is not displayed.

arp_maxretries
Sets the maximum number of retries for the address resolution protocol (ARP) before it gives up. The default value is 5; the minimum and maximum configurable values are 1 and 128.

arpt_down
Sets the time to hold onto an incomplete ARP cache entry if ARP lookup fails. The default value is 20 seconds; the minimum and maximum configurable values are 1 and 600 seconds.

arpt_keep
Sets the time to keep a valid entry in the ARP cache. The default value is 1200 seconds; the minimum and maximum configurable values are 1 and 2400 seconds.

arpt_prune
Sets the interval between scanning the ARP table for stale entries. The default value is 300 seconds; the minimum and maximum configurable values are 1 and 1800 seconds.

The number of ARP units is controlled by the value of the defined constant **ARP_UNITS**.

Asynchronous half-duplex (ASYH) line connection parameters

The following parameter controls the behavior of asynchronous half-duplex (ASYH) line connections used by PPP.

ahdlcmtu
Sets the maximum transmission unit (MTU) for an asynchronous PPP link. This is normally set on a per-system basis in the */etc/ppphosts* file — if not defined there, this value is used.

The default value of **ahdlcmtu** is 296 bytes; the minimum and maximum configurable values are 128 and 2048 bytes.

Internet Control Message Protocol (ICMP) parameters

The following parameters control the behavior of the Internet Control Message Protocol (ICMP).

icmp_answermask
If set to 1, the system will respond to ICMP subnet mask request messages. This variable must be set to 1 to support diskless workstations. The default value is 0, do not respond, as specified in RFC 1122.

icmpprintfs
Controls logging of warnings from the kernel ICMP driver. These are displayed on the console. If set to 0 (the default), debugging information is not displayed.

Internet Group Management Protocol (IGMP) parameters

The following parameter controls the behavior of the Internet Group Management Protocol (IGMP).

igmpprintfs
: Controls logging of warnings from the kernel IGMP driver. These are displayed on the console. If set to 0 (the default), debugging information is not displayed.

Configuring the in-kernel network terminal (IKNT) driver

The number of IKNT driver units is determined by the number of pseudo-ttys configured on the system. Use **mkdev ptty** to tune the number of pseudo-ttys.

Internet Protocol (IP) parameters

The following parameters control the behavior of the Internet Protocol (IP). The number of interfaces supported by IP is dynamic and does not need tuning.

NOTE The value of the parameters **in_fullsize, in_recvspace**, and **in_sendspace** affect the systemwide interface defaults. Their values may be overridden on a per-interface basis by **ifconfig**(ADMN). This allows you to mix fast and slow network hardware on the same system with optimal performance parameters defined for each interface.

in_fullsize
: Controls the systemwide default TCP behavior for attempting to negotiate the use of full-sized segments. If set to 1 (the default), TCP attempts to use a segment size equal to the interface MTU minus the size of the TCP/IP headers. If set to 0, TCP rounds the segment size down to the nearest power of 2.

in_loglimit
: Controls how many bytes of the error packet to display when debugging. Note that the appropriate ***xxx*printfs** parameter (such as **tcpprintfs**) must be set to a non-zero value to enable logging. The default value is 64. The minimum and maximum configurable values are 1 and 255.

in_recvspace
Sets the systemwide default size of the TCP/IP receive window in bytes. The default value is 4096 bytes. The minimum and maximum configurable values are 2048 and 65535 bytes.

in_sendspace
Sets the systemwide default size of the TCP/IP send window in bytes. This should be at least as large as the loopback MTU. The default value is 8192 bytes. The minimum and maximum configurable values are 2048 and 65535 bytes.

ip_checkbroadaddr
Controls whether IP validates broadcast addresses. If set to 1 (the default as specified in RFC 1122), IP discards non-broadcast packets sent to a link-level broadcast address. In the unlikely event that a data-link driver does not support this, packets may be discarded erroneously. If the **netstat -sp ip** command shows that many packets cannot be forwarded, set this parameter to 0 to turn off checking.

ip_dirbroadcast
If set to 1 (the default), allows receipt of broadcast packets only if they match one of the broadcast addresses configured for the interface upon which the packet was received. If set to 0, allows receipt of broadcast packets that match any configured broadcast address.

ip_perform_pmtu
IP performs Path MTU (PMTU) discovery as specified in RFC 1191 if set to 1 (the default). This causes IP to send packets with the "do not fragment" bit set so that routers will generate "Fragmentation Required" messages. If this causes interoperability problems, a value of 0 disables PMTU.

If you disable PMTU, you should also set **tcp_offer_big_mss** (described in "Transmission Control Protocol (TCP) parameters" (page 232)) to 0.

ip_pmtu_decrease_age
Controls how many seconds IP will wait (while performing PMTU) after decreasing an MTU estimate before it starts raising it. The default value is 600 seconds. The maximum configurable value is 32667. If set to `0xffffffff`, the estimate is never raised; this is useful if there is only one path out of your local network and its MTU is known to be constant.

ip_pmtu_increase_age
Sets the number of seconds between increasing the MTU estimate for a destination once it starts to increase. The default value is 120 seconds. The minimum and maximum configurable values are 0 and 600 seconds.

ip_settos
If set to 1 (the default), IP sets type-of service TOS information (as specified in RFS 1122) in packets that it sends down to the data-link layer. Set this to 0 if your network card link-level driver cannot handle this.

ip_subnetsarelocal

The default value of 1 specifies that other subnets of the network are to be considered as local — that is, TCP assumes them to be connected via high-MSS paths and adjusts its idea of the MSS to be negotiated. Otherwise, TCP uses the default MSS specified by **tcp_mssdflt** (described in "Transmission Control Protocol (TCP) parameters" (page 232)) — this is typically 512 bytes in accordance with RFC 793 and 1122. By default, the parameter **tcp_offer_big_mss** is non-zero so that Path MTU discovery will provide the maximum benefit. If the value of **tcp_offer_big_mss** is zero, the value of **ip_subnetsarelocal** is not checked. This allows for good local performance even when PMTU discovery is not used.

The message "ICMP Host Unreachable" is generated for local subnet routing failures. When this value is set to 0, the packet size is set to 576 bytes, as specified in RFC 1122.

The default value of 1 enables this feature; if set to 0, it is disabled.

ip_ttl

Sets the time to live (TTL) of an IP packet as a number of hops. This value is used by all kernel drivers that need it (including TCP). The default value is 64 as recommended by RFC 1340. The minimum and maximum configurable values are 1 and 255.

ipforwarding

ipsendredirects

If you want to use your machine as a gateway, set both these parameters to 1.

ipforwarding controls whether the system will forward packets sent to it which are destined for another system (that is, act as a router). The default value is 0 (off) as defined by RFC 1122. A system acting as a host will still forward source-routed datagrams unless **ipnonlocalsrcroute** is set to 0.

ipsendredirects controls whether IP will redirect hosts when forwarding a packet out of the same interface on which it was received. This should be set to 1 if **ipforwarding** is set to 1.

The **Network Configuration Manager** configures these values when additional drivers are added. This feature usually makes it unnecessary to change **ipforwarding** and **ipsendredirects** with **inconfig**.

ipnonlocalsrcroute
Controls whether source-routed datagrams will be forwarded if they are not destined for the local system. On hosts, the default value is 0 (off). If your machine is acting as a router (**ipforwarding** is set to 1), the **Network Configuration Manager** sets its value to 1. Set its value back to 0 if you are concerned that this may open a security hole.

ipprintfs
Controls logging of warnings from the kernel IP driver. These are displayed on the console. If set to 0 (the default), debugging information is not displayed.

Message block control logging (MBCL) parameters

The following parameter controls the behavior of message block control logging (MBCL).

mbclprintfs
Controls logging of warnings from the kernel MBCL driver which converts STREAMS messages (`mblock`) to character lists (`clist`). The warnings are displayed on the console. If set to 0 (the default), debugging information is not displayed.

NetBIOS parameters

The following parameters control the behavior of NetBIOS.

nb_sendkeepalives
Turns NetBIOS level keepalives on or off. When turned on, NetBIOS keepalives are sent periodically on dormant NetBIOS connections. NetBIOS keepalives are independent of TCP/IP keepalives, and are useful for systems that do not use TCP/IP keepalives. This parameter is set to 0 (turned off) by default. Set it to 1 to enable NetBIOS keepalives.

nbprintfs
Controls logging of warnings from the kernel NetBIOS driver as specified in RFC 1001/2. The warnings are displayed on the console. If set to 0 (the default), debugging information is not displayed.

Transmission Control Protocol (TCP) parameters

The following parameters control the behavior of the Transmission Control Protocol (TCP). You can increase the number of TCP units beyond the default number (256) using the **Network Configuration Manager** for the appropriate **sco_tcp** chain.

tcp_initial_timeout
Sets the TCP/IP retransmit time for an initial SYN segment. The default value is 180 seconds as defined by RFC 1122. The minimum and maximum configurable values are 1 and 7200 seconds.

tcp_keepidle

Sets the idle time before TCP/IP keepalives are sent (if enabled). The default value is 7200 seconds. The minimum and maximum configurable values are 300 and 86400 seconds.

tcp_keepintvl

Sets the TCP/IP keepalive interval between keepalive packets once they start being sent. The default value is 75 seconds. The minimum and maximum configurable values are 1 and 43200 seconds.

tcp_mss_sw_threshold

Defines the small window threshold for interface MTUs. If the MTU of an interface is small enough to force TCP to use an MSS smaller than this threshold, then TCP will use the receive window size specified by **tcp_small_recvspace**. This is an optimization to avoid buffering too much data on low-speed links such as SLIP and PPP. The default value is 1024 bytes. The minimum and maximum configurable values are 512 and 4096 bytes.

tcp_mssdflt

Sets the default TCP segment size to use on interfaces for which no MSS and Path MTU information is available. The default and minimum value is 512 bytes. The maximum configurable values is 32768. You should keep the value of this parameter small if possible.

tcp_nkeep

Sets the number of TCP/IP keepalives that will be sent before giving up. The default value is 8. The minimum and maximum configurable values are 1 and 256.

tcp_offer_big_mss

In order to get the maximum benefit out of Path MTU (PMTU) discovery, TCP normally offers an MSS that is derived from the local interface MTU (after subtracting the packet header sizes). This allows the remote system to send the biggest segments that the network can handle. Set this parameter to 0 for systems that cannot handle this, or that do not implement PMTU discovery. This causes TCP to offer a smaller MTU for non-local connections (see **ip_subnetsarelocal** in "Internet Protocol (IP) parameters" (page 229)). The default value of 1 (offer it) allows maximum benefit to be gained from PMTU discovery; a value of 0 disables this.

tcp_small_recvspace
Sets the receive window size to use on interfaces that require small windows (see also **tcp_mss_sw_threshold**). MTU is less than **tcp_mss_sw_threshold**. The default value is 4096 bytes. The minimum and maximum configurable values are 1024 and 16384 bytes.

tcp_urgbehavior
Controls how TCP interprets the urgent pointer. If set to 0, it interprets it in RFC 1122 mode; if set to 1 (the default), it interprets it in BSD mode.

tcpalldebug
If non-zero, captures trace information for all connections. The default value is 0 which causes TCP to trace only those connections that set the SO_DEBUG option. This information can be retrieved using the **trpt**(ADMN) command, or displayed on the console if **tcpconsdebug** is set.

tcpconsdebug
Directs TCP/IP connection trace output to the console if set to 1 (see also **tcpalldebug**). The default value is 0.

tcpprintfs
Controls logging of warnings from the kernel TCP driver. These are displayed on the console. If set to 0 (the default), debugging information is not displayed.

User Datagram Protocol (UDP) parameters

The following parameter controls the behavior of the User Datagram Protocol (UDP).

udpprintfs
Controls logging of warnings from the kernel UDP driver. These are displayed on the console. If set to 0 (the default), debugging information is not displayed.

Appendix D

Quick system tuning reference

Table D-1, "Diagnosing performance problems" (this page) summarizes the symptoms and possible solutions for some important performance problems. Note that the measured values represent averages over time. Suggested critical values may not be suitable for all systems. For example, you may be able to tolerate a system that is paging out if this is not impacting the performance of the rest of the system seriously.

Table D-1 Diagnosing performance problems

Insufficient CPU power at high load	Possible solutions
[mp]sar -q shows runq-sz > 2 **[mp]sar -u** shows %idle < 20% on multiuser system **[mp]sar -u** shows %idle < 5% on dedicated database server Additionally for SMP: **mpsar -q** shows %runocc > 90% **cpusar -u** shows %idle < 20% on any CPU of multiuser system **cpusar -u** shows %idle < 5% on any CPU of dedicated database server	Measures that can be taken include: • check that the system is not swapping or paging out excessively • reschedule jobs to run at other times • tune applications to use less CPU power • replace applications with ones needing less CPU power • replace non-intelligent serial cards with intelligent ones • upgrade the system to use faster CPU(s) • upgrade to a multiprocessor system • add more CPUs to a multiprocessor system • purchase an additional system to share the load

Excessive paging out or swapping	Possible solutions
[mp]sar -p shows rclm/s >> 0 **[mp]sar -q** shows %swpocc > 20% **[mp]sar -w** shows swpot/s > 1 **swap -l** shows free < 50% of blocks	Increase free memory until swapping does not occur by: • reducing number of buffers (watch out for reduced cache hit rates) • running fewer large applications locally • moving users to another machine • adding RAM

Poor disk performance	Possible solutions
[mp]sar -u shows %wio > 15% **[mp]sar -d** shows avque >> 1 and %busy > 80%	Increase disk performance by: • using HTFS filesystem(s) • using striping across several disks to balance load • keeping filesystems < 90% full • reorganizing directories • keeping directories small • distributing different types of activity to different disks • adding more disks • using faster disks, controllers, and host adapters • improving buffer cache performance • improving namei cache performance • reducing filesystem fragmentation

Poor buffer cache performance	Possible solutions
[mp]sar -b shows %rcache < 90% and %wcache < 65%	Improve buffer cache performance by: • increasing number of buffers • increasing number of buffer hash queues per buffer

Poor namei cache performance	Possible solutions
[mp]sar -n shows %Hhit < 65% or %Dhit < 65%	Increase namei cache hit rate by: • tuning namei cache parameters for each filesystem type • make each pathname component less than or equal to 14 characters

Fragmented filesystem	Possible solutions
df -v shows blocks %used > 90%	Reduce the number of disk blocks used by: • using DTFS filesystem(s) • removing unwanted files regularly • archiving and removing, or compressing infrequently used files • mounting commonly used resources across the network using NFS • adding disk(s) Reduce fragmentation by: • archiving and removing the files, and rebuilding the filesystem

Kernel tables too small	Possible solutions
error messages displayed on console **[mp]sar -v** shows ov > 0 (overflows)	Allow table sizes to grow dynamically; for example, set MAX_PROC to 0 for the process table

The desirable attributes of systems with many logged-in users and database server systems differ in some respects. Use the following tables to check that you have not overlooked anything:

- Table D-2, "Attributes of a well-tuned multiuser system" (page 238)
- Table D-3, "Attributes of a well-tuned dedicated database server system" (page 239)

Note that the performance values suggested in these tables may not be suitable for all systems. The appropriate values depend greatly on the mix of applications that is running and the likely demands placed on the system.

To record system activity to a file for later analysis, use the **-o** option of **sar**(ADM) on a single processor system, and of **mpsar**(ADM) on a multiprocessor system. Take the measurements over a period of at least an hour with a sampling interval sufficiently small to capture the level of detail which you are interested in. Record the system's activity at varying levels of loading so that you can identify when bottlenecks are appearing.

Table D-2 Attributes of a well-tuned multiuser system

CPU performance	Explanation
[**cpu**]**sar -u** shows %idle > 20%	Some idle time on each CPU at high load
[**mp**]**sar -q** shows runq-sz < 2	Few processes waiting to run
mpsar -q shows %runocc < 90% (SMP only)	Run queue is not continually occupied

See Chapter 3, "Tuning CPU resources" (page 21).

Memory performance	Explanation
[**mp**]**sar -p** shows rclm/s ≈ 0	Little or no swapping or paging out activity
[**mp**]**sar -w** shows swpot/s ≈ 0	Little or no activity on the swap device(s)
[**mp**]**sar -q** shows swpq-sz ≈ 0 and %swpocc ≈ 0%	No swapped-out runnable processes
[**mp**]**sar -r** shows freemem >> GPGSHI and freeswp ≈ constant	Ample free memory and swap space

See Chapter 4, "Tuning memory resources" (page 41).

Disk I/O performance	Explanation
[**cpu**]**sar -u** shows %wio < 15%	Little time spent waiting for I/O to complete
[**mp**]**sar -b** shows %rcache > 90% and %wcache > 65%	Good hit rate for reading and writing to the buffer cache
[**mp**]**sar -d** shows avque ≈ 1	Low average number of disk requests queued
[**mp**]**sar -n** shows %Hhit > 65% or %Dhit > 65%	Good hit rate for namei cache

See Chapter 5, "Tuning I/O resources" (page 71).

Table D-3 Attributes of a well-tuned dedicated database server system

CPU performance	Explanation
[**cpu**]**sar -u** shows %idle > 5%	Some idle time on each CPU at high load
[**mp**]**sar -q** shows runq-sz < 2	Few processes waiting to run
mpsar -q shows %runocc < 90% (SMP only)	Run queue is not continually occupied

See your database documentation and Chapter 3, "Tuning CPU resources" (page 21).

Memory performance	Explanation
[**mp**]**sar -p** shows rclm/s ≈ 0	Little or no swapping or paging out activity
[**mp**]**sar -w** shows swpot/s ≈ 0	Little or no activity on the swap device(s)
[**mp**]**sar -q** shows swpq-sz ≈ 0 and %swpocc ≈ 0%	No swapped-out runnable processes
[**mp**]**sar -r** shows freemem ≈ GPGSHI and freeswp ≈ constant	Little excess free memory; allow the database to use any excess memory by increasing its internal work area.

See your database documentation and Chapter 4, "Tuning memory resources" (page 41).

Disk I/O performance	Explanation
[**cpu**]**sar -u** shows %wio < 15%	Little time spent waiting for I/O to complete
[**mp**]**sar -d** shows avque ≈ 1	Low average number of disk requests queued

See your database documentation and Chapter 5, "Tuning I/O resources" (page 71).

Bibliography

The following books provide more information about topics outlined in this guide. This list is provided for reference only; it is not comprehensive and The Santa Cruz Operation, Inc. does not guarantee the accuracy of these publications. The implementation of the UNIX system, networking and performance analysis software described in these books may differ in some details from that of the current SCO OpenServer software.

Several references are also included on the subject of algorithmics which has direct relevance to programmers who wish to improve the performance of applications programs.

Ammeraal, Leendert. *Programs and Data Structures in C, Second Edition.* New York, NY: Wiley, 1992. A practical introduction to the implementation and manipulation of data structures using the ANSI C programming language.

Bach, Maurice J. *The Design of the UNIX Operating System.* Englewood Cliffs, NJ: Prentice Hall, 1986. A technical discussion of the internals of the UNIX System V Operating System, written shortly before the release of UNIX System V Release 3.

Deitel, Harvey M. *An Introduction to Operating Systems, Second Edition.* Reading, MS: Addison-Wesley, 1990. Discusses general performance issues for operating systems.

Harel, David. *Algorithmics: The Spirit of Computing, Second Edition.* Reading, MS: Addison-Wesley, 1992. A very readable introduction to the subject of algorithmics.

Hunt, Craig. *TCP/IP Network Administration.* Sebastopol, CA: O'Reilly and Associates, 1993. Contains information about the configuration of IP packet routing and name service.

Knuth, Donald E. *The Art of Computer Programming, Volume I: Fundamental Algorithms.* Reading, MS: Addison-Wesley, 1968. The first volume of the classic three-volume series on the subject of computer programming.

Knuth, Donald E. *The Art of Computer Programming, Volume II: Seminumerical Algorithms.* Reading, MS: Addison-Wesley, 1969.

Knuth, Donald E. *The Art of Computer Programming, Volume III: Sorting and Searching.* Reading, MS: Addison-Wesley, 1973.

Loukides, Mike. *System Performance Tuning*. Sebastopol, CA: O'Reilly and Associates, 1991. Includes many excellent tips for getting the best performance out of UNIX systems.

Mansfield, Niall. *The Joy of X*. Reading, MS: Addison-Wesley, 1993. Contains useful information about performance issues for the X Window System.

Messmer, Hans-Peter. *The Indispensable PC Hardware Book*. Reading, MS: Addison-Wesley, 1994. Provides comprehensive information about system hardware issues.

Miscovitch, Gina and David Simons. *The SCO Performance Tuning Handbook*. Englewood Cliffs, NJ: Prentice Hall, 1994. Written by two senior kernel engineers at SCO, this book describes performance tuning for SCO® UNIX® Release 3.2 Version 4.2, SCO MPX™ 3.0, SCO Open Desktop 3.0, and SCO Open Server™ 3.0 systems.

Press, William H., Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing, Second Edition*. Cambridge University Press, 1994. Includes many numerical algorithms for scientific and engineering applications.

Stern, Hal. *Managing NFS and NIS*. Sebastopol, CA: O'Reilly and Associates, 1991. Contains a detailed chapter on performance analysis and tuning as well as useful references on IP packet routing and NFS benchmarks.

Glossary of performance terminology

This section contains definitions of the key terms used throughout this book in discussing the performance of computer systems.

AIO
See asynchronous I/O.

asymmetric multiprocessing
A multiprocessor system is asymmetric when processors are not equally able to perform all tasks. For example, only the base processor is able to control I/O. Most machines acknowledged to be symmetric may still have some asymmetric features present such as only being able to boot using the base processor.

asynchronous I/O
Provides non-blocking I/O access through a raw device interface.

bandwidth
The maximum I/O throughput of a system.

base processor
The first CPU in a multiprocessor system. The system normally boots using this CPU. Also called the default processor, it cannot be deactivated.

bdflush
The system name for the buffer flushing daemon.

benchmark
Software run on a computer system to measure its performance under specific operating conditions.

block device interface
Provides access to block-structured peripheral devices (such as hard disks) which allow data to be read and written in fixed-sized blocks.

blocking I/O
Forces a process to wait for the I/O operation to complete. Also known as synchronous I/O.

bottleneck
Occurs when demand for a particular resource is beyond the capacity of that resource and this adversely affects other resources. For example, a system has a disk bottleneck if it is unable to use all of its CPU power because processes are blocked waiting for disk access.

bss
Another name for data which was not initialized when a program was compiled. The name is an acronym for *block started by symbol.*

buffer

A temporary data storage area used to allow for the different capabilities (speed, addressing limits, or transfer size) of two communicating computer subsystems.

buffer cache

Stores the most-recently accessed blocks on block devices. This avoids having to re-read the blocks from the physical device.

buffer flushing daemon

Writes the contents of dirty buffers from the buffer cache to disk.

cache memory

High-speed, low-access time memory placed between a CPU and main memory in order to enhance performance. See also level-one (L1) cache and level-two (L2) cache.

checkpointing

One of the functions of the **htepi_daemon**; marking a filesystem state as clean after it flushes changed metadata to disk.

child process

A new process created when a parent process calls the **fork**(S) system call.

clean

The state of a system buffer or memory page that has not had its contents altered.

client-server model

A method of implementing application programs and operating system services which divides them into one of more client programs whose requests for service are satisfied by one or more server programs. The client-server model is suitable for implementing applications in a networked computer environment.

Examples of application of the client-server model are:

- page serving to diskless clients
- file serving using NFS and NUCFS
- Domain Name Service (DNS)
- the X Window System
- many relational database management systems (RDBMSs)

clock interrupt

See clock tick.

clock tick

An interrupt received at regular intervals from the programmable interrupt timer. This interrupt is used to invoke kernel activities that must be performed on a regular basis.

contention

Occurs when several CPUs or processes need to access the same resource at the same time.

context

The set of CPU register values and other data, including the u-area, that describe the state of a process.

context switch

Occurs when the scheduler replaces one process executing on a CPU with another.

copy-on-write page

A memory page that is shared by several processes until one tries to write to it. When this happens, the process is given its own private copy of the page.

COW page

See copy-on-write page.

CPU

Abbreviation of central processing unit. One or more CPUs give a computer the ability to execute software such as operating systems and application programs. Modern systems may use several auxiliary processors to reduce the load on the CPU(s).

CPU bound

A system in which there is insufficient CPU power to keep the number of runnable processes on the run queue low. This results in poor interactive response by applications.

daemon

A process that performs a service on behalf of the kernel. Since daemons spend most of their time sleeping, they usually do not consume much CPU power.

device driver

Performs I/O with a peripheral device on behalf of the operating system kernel. Most device drivers must be linked into the kernel before they can be used.

dirty

The state of a system buffer or memory page that has had its contents altered.

distributed interrupts

Interrupts from devices that can be serviced by any CPU in a multiprocessor system.

event

In the X Window System, an event is the notification that the X server sends an X client to tell it about changes such as keystrokes, mouse movement, or the moving or resizing of windows.

executing

Describes machine instructions belonging to a program or the kernel being interpreted by a CPU.

fragmentation

The propensity of the component disk blocks of a file or memory segments of a kernel data structure to become separated from each other. The greater the fragmentation, the more work has to be performed to retrieve the data.

free list

A chain of unallocated data structures which are available for use.

garbage collection

The process of compacting data structures to retrieve unused memory.

htepi_daemon

A kernel daemon that handles filesystem metadata. It can also perform optional transaction intent logging and checkpointing on behalf of the HTFS filesystem.

idle

The operating system is idle if no processes are ready-to-run or are sleeping while waiting for block I/O to complete.

idle waiting for I/O

The operating system is idle waiting for I/O if processes that would otherwise be runnable are sleeping while waiting for I/O to a block device to complete.

in-core

Describes something that is internal to the operating system kernel.

in-core inode

An entry in the kernel table describing the status of a filesystem inode that is being accessed by processes.

inode

Abbreviation of Index Node. An inode is a data structure that represents a file within a traditional UNIX filesystem. It consists of a file's metadata and the numbers of the blocks that can be used to access the file's data.

interrupt

A notification from a hardware device about an event that is external to the CPU. Interrupts may be generated for events such as the completion of a transfer of data to or from disk, or a key being pressed.

interrupt bound

A system which is unable to handle all the interrupts that are arriving.

interrupt latency

The time that the kernel takes to handle an interrupt.

interrupt overrun

Occurs when too many interrupts arrive while the kernel is trying to handle a previous interrupt.

I/O

Abbreviation of input/output. The transfer of data to and from peripheral devices such as hard disks, tape drives, the keyboard, and the screen.

I/O bound

A system in which the peripheral devices cannot transfer data as fast as requested.

job

One or more processes grouped together but issued as a single command. For example, a job can be a shell script containing several commands or a series of commands issued on the command line connected by a pipeline.

kernel

The name for the operating system's central set of intrinsic services. These services provide the interface between user processes and the system's hardware allowing access to virtual memory, I/O from and to peripheral devices, and sharing resources between the user processes running on the system.

kernel mode

See system mode.

kernel parameter

A constant defined in the file */etc/conf/cf.d/mtune* (see *mtune*(F)) that controls the configuration of the kernel.

level-one (L1) cache

Cache memory that is implemented on the CPU itself.

level-two (L2) cache

Cache memory that is implemented externally to the CPU.

load average

The utilization of the CPU measured as the average number of processes on the run queue over a certain period of time.

logging

See transaction intent logging.

marry driver

A pseudo-device driver that allows a regular file within a filesystem to be accessed as a block device, and, hence, as a swap area.

memory bound

A system which is short of physical memory, and in which pages of physical memory, but not their contents, must be shared by different processes. This is achieved by paging out, and swapping in cases of extreme shortage of physical memory.

memory leak

An application program has a memory leak if its size is constantly growing in virtual memory. This may happen if the program is continually requesting more memory without re-using memory allocated to data structures that are no longer in use. A program with a memory leak can eventually make the whole system memory bound, at which time it may start paging out or swapping.

metadata

The data that an inode stores concerning file attributes and directory entries.

multiprocessor system

A computer system with more than one CPU.

multithreaded program

A program is multithreaded if it can be accessed simultaneously by different CPUs. Multithreaded device drivers can run on any CPU in a multiprocessor system. The kernel is multithreaded to allow equal access by all CPUs to its tables and the scheduler. Only one copy of the kernel resides in memory.

namei cache

A kernel data structure that stores the most-commonly accessed translations of filesystem pathname components to inode number. The namei cache improves I/O performance by reducing the need to retrieve such information from disk.

nice value

A weighting factor in the range 0 to 39 that influences how great a share of CPU time a process will receive. A high value means that a process will run on the CPU less often.

non-blocking I/O

Allows a process to continue executing without waiting for an I/O operation to complete. Also known as asynchronous I/O.

operating system

The software that manages access to a computer system's hardware resources.

overhead

The load that an operating system incurs while sharing resources between user processes and performing its internal accounting.

page

A fixed-size (4KB) block of memory.

page fault

A hardware event that occurs when a process tries to access an address in virtual memory that does not have a location in physical memory associated with it. In response, the system tries to load the appropriate data into a newly assigned physical page.

page stealing daemon

The daemon responsible for releasing pages of memory for use by other processes. Also known as vhand.

paging in

Reading pages of program text and pre-initialized data from the filesystems, or stack and data pages from swap.

paging out

Releasing pages of physical memory for use by making temporary copies of the contents of dirty pages to swap space. Clean pages of program text and pre-initialized data are not copied to swap space because they can be paged in from the filesystems.

parent process

A process that executes a **fork**(S) system call to create a new child process. The child process usually executes an **exec**(S) system call to invoke a new program in its place.

physical memory

Storage implemented using RAM chips.

preemption

A process that was running on a CPU is replaced by a higher priority process.

priority

A value that the scheduler calculates to determine which process(es) should next run on the CPUs. A process' priority is calculated from its nice value and its recent CPU usage.

process

A single instance of a program in execution. This can be a login shell or an operating system command, but not a built-in shell command. If a command is built into the shell a separate process is not created on its invocation; the built-in command is issued within the context of the shell process.

process table

A data structure inside the kernel that stores information about all the processes that are present on a system.

protocol

A set of rules and procedures used to establish and maintain communication between hardware or software subsystems.

protocol stack

Allows two high-level systems to communicate by passing messages through a low-level physical interface.

pseudo-device driver

A device driver that allows software to behave as though it is a physical device. Examples are ramdisks and pseudo-ttys.

pseudo-tty

A pseudo-terminal is a device driver that allows one process to communicate with another as though it were a physical terminal. Pseudo-ttys are used to interface to programs that expect to receive non-blocking input and to send terminal control characters.

queue

An ordered list of entities.

race condition

The condition which occurs when several processes or CPUs are trying to write to the same memory or disk locations at the same time. The data that is eventually stored depends on the order that the writes occur. A synchronization mechanism must be used to enforce the desired order in which the writes are to take place.

RAID array

Abbreviation of redundant array of inexpensive disks. Used to implement high performance and/or high integrity disk storage.

ramdisk

A portion of physical memory configured to look like a physical disk but capable of fast access times. Data written to a ramdisk is lost when the operating system is shut down. Ramdisks are, therefore, only suitable for implementing temporary filesystems.

raw device interface

Provides access to block-structured peripheral devices which bypasses the block device interface and allows variable-sized transfers of data. The raw interface also allows control of a peripheral using the **ioctl**(S) system call. This allows, for example, for low-level operations such as formatting a disk or rewinding a tape.

region

A region groups a process' pages by their function. A process has at least three regions for its data, stack, and text.

resource

Can be divided into software and hardware resources. Software resources may be specific to applications, or they may be kernel data structures such as the process table, open file, and in-core inode tables, buffer and namei caches, multiphysical buffers, and character lists. Hardware resources are a computer's physical subsystems. The three main subsystems are CPU, memory and I/O. The memory subsystem can be divided into two resources — physical memory (or main memory) and swap space (or secondary memory). The I/O subsystem comprises one or more resources of similar or different types — hard and floppy disk drives, tape drives, CD-ROMs, graphics displays and network devices.

ready-to-run process

A process that has all the system resources that it needs in order to be able to run on a CPU.

response time

The time taken between issuing a command and receiving some feedback from the system. This is not to be confused with turnaround time which is a measure of how long a particular task takes from invocation to completion.

run queue

The list of ready-to-run processes maintained by the kernel.

runnable process

See ready-to-run process.

scaling

A computer system's ability to increase its processing capacity as CPUs are added. If the processing capacity increases in direct proportion to the number of CPUs, a system is said to exhibit 100% scaling. In practice, a system's ability to scale is limited by contention between the CPUs for resources and depends on the mix of applications being run.

sched

The system name for the swapper daemon.

scheduler

The part of the kernel that chooses which process(es) to run on the CPUs.

single threaded program

A program is single threaded if it can only run on one CPU at a time. Single threaded devices drivers can only run on the base processor in a multiprocessor system.

sleeping on I/O

See waiting for I/O.

spin lock

A method of synchronizing processes on a multiprocessor system. A process waiting for a resource which is currently in use (locked) by a process running on a different CPU repeatedly executes a short section of kernel code (spins) until the lock is released.

stack

A list of temporary data used by a program to handle function calls.

strd

The system name for the STREAMS daemon.

stream head

The level of the STREAMS I/O interface with which a user process communicates.

STREAMS I/O

A mechanism for implementing a layered interface between applications running in user space and a device driver. Most often used to implement network protocol stacks.

STREAMS daemon
The daemon used by the STREAMS I/O subsystem to manage STREAMS memory.

swap area
A piece of swap space implemented as a disk division or as a block device married to a regular file in a filesystem.

swap space
A collection of swap areas used to store the contents of stack and data memory pages temporarily while they are used by other processes.

swapper daemon
Part of the kernel that reclaims physical pages of memory for use by copying whole regions of processes to swap space.

swapping
The action take by the swapper daemon when the system is extremely short of physical memory needed for use by processes. Swapping can place a heavy load on the CPU and disk I/O subsystems.

symmetric multiprocessing
A multiprocessor system is symmetric when any processor can perform any function. This ensures an even load distribution because no processor depends on another. Each process is executed by a single processor.

system mode
The state of a CPU when the kernel needs to ensure that it has privileged access to its data and physical devices. Also known as kernel mode.

text
Executable machine instructions (code) that a CPU can interpret and act on.

throughput
The amount of work (measured in number of jobs completed, disk requests handled, and so on) that a system processes in a specified time.

time slice
The maximum amount of time for which a process can run without being preempted.

transaction intent logging
One of the functions of the **htepi_daemon**; writing the intention to change filesystem metadata to a log file on disk.

u-area
Abbreviation of user area and also known as a u-block. A data structure possessed by every process. The u-area contains private data about the process that only the kernel may access.

user mode
The state of a CPU when it is executing the code of a user program that accesses its own data space in memory.

vhand
The system name for the page stealing daemon.

virtual disk
A disk composed of pieces of several physical disks.

virtual memory
A method of expanding the amount of available memory by combining physical memory (RAM) with cheaper and slower storage such as a swap area on a hard disk.

waiting for I/O
A process goes to sleep if it has to wait for an I/O operation to complete.

X client
An applications program that communicates with an X server to request that it display information on a screen or to receive input events from the keyboard or a pointing device such as a mouse. The client may be running on the same computer as the server (local), or it may be connected via a network (remote).

X server
The software that controls the screen, keyboard and pointing device under the X Window System.

X terminal
A display device that is able to run X server software. All of an X terminal's clients must run on remote machines.

X Window System
A windowing system based on the client-server model.

zombie process
An entry in the process table corresponding to a process that no longer exists. The entry will only be removed if its parent process invokes a **wait**(S) system call. A zombie process does not consume any system resources apart from its slot in the process table. However, you should beware of runaway processes that generate many zombies. These will cause the system to become short of memory as the process table grows to accommodate them.

1 May 1995

AU20004P001