ULTRIX

Performance Management Guide

Order Number: AA-PKDVA-TE
December 1991

Product Version: ULTRIX Version 4.2A or higher

digital equipment corporation
Maynard, Massachusetts

Restricted Rights: Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013.

© Digital Equipment Corporation 1991
All rights reserved.

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by Digital or its affiliated companies.

The following are trademarks of Digital Equipment Corporation:

ALL-IN-1, Bookreader, CDA, DDIF, DDIS, DEC, DECnet, DECstation, DECsystem, DECUS, DECwindows, DTIF, MASSBUS, MicroVAX, Q-bus, ULTRIX, ULTRIX Mail Connection, ULTRIX Worksystem Software, UNIBUS, VAX, VAXstation, VMS, VT, XUI, and the DIGITAL logo.

Ethernet is a registered trademark of Xerox Corporation. Prestoserve is a trademark of Legato Systems, Inc.; the trademark and software are licensed to Digital Equipment Corporation by Legato Systems, Inc. Xenix, MS-DOS, and MS-OS/2 are trademarks of Microsoft Corporation. This manual is derived from MIT documentation, which contains the following permission notice: Permission to use, copy, modify, and distribute this documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appears in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of MIT or DIGITAL not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. MIT and DIGITAL make no representations about the suitability of the software described herein for any purpose. It is provided "as is," without express or implied warranty. Network File System and NFS are trademarks of Sun Microsystems, Inc. Open Software Foundation, OSF, OSF/1, OSF/Motif, and Motif are trademarks of the Open Software Foundation, Inc. UNIX is a registered trademark of UNIX System Laboratories, Inc. X Window System Version 11 and its derivatives (X, X11, and X Version 11) are trademarks of the Massachusetts Institute of Technology.

Contents

About This Manual

1 Review of ULTRIX Subsystem Resource Usage

1.1 File System . . . . . 1—1
1.1.1 Function of the File System . . . . . 1—1
1.1.2 Component Parts of a File System . . . . . 1—2
1.1.3 Disk Blocks . . . . . 1—3
1.1.4 Relationship of File Systems to Devices . . . . . 1—3
1.1.5 Relationship of Files to Other Files in a File System . . . . . 1—4
1.1.5.1 Root Directory . . . . . 1—4
1.1.5.2 Primary Directories Off of the Root . . . . . 1—5
1.1.6 Associating a Device with a File System . . . . . 1—5
1.1.7 Relationship Between the File System and Performance . . . . 1—6
1.1.7.1 Size of the Files . . . . . 1—6
1.1.7.2 Location of Files . . . . . 1—6
1.1.7.3 Factors Determined at Time of Configuration . . . . . 1—6
1.1.7.4 Organizing Files to Improve Performance . . . . . 1—7
1.2 Understanding Process Control and Scheduling . . . . . 1—7
1.2.1 Types of Processes . . . . . 1—7
1.2.2 Process Priorities and Privileged Processes . . . . . 1—8
1.2.3 Factors Used to Determine Process Priorities . . . . . 1—8
1.2.4 Scheduling a Process to Run During Off-Peak Hours . . . . . 1—8
1.2.5 Relationship Between Process Control and Performance . . . . 1—9
1.3 Understanding Memory Management . . . . . 1—9
1.3.1 Types of Memory Management . . . . . 1—9
1.3.2 Allocating Memory . . . . . 1—10
1.3.3 Buffer Cache . . . . . 1—10
1.3.4 Swap Area . . . . . 1—11
1.3.5 Changes in Memory Management that Can Affect Performance . . . . . 1—11
1.3.5.1 Size of the Buffer Cache . . . . . 1—11
1.3.5.2 Size and Location of the Swap Area . . . . . 1—11
1.3.5.3 System Management . . . . . 1—12
1.4 Understanding the I/O Subsystem . . . . . 1—12
1.4.1 Components of the I/O Subsystem . . . . . 1—12
1.4.1.1 Block Devices . . . . . 1—13
1.4.1.2 Character Devices . . . . . 1—13

1.4.2 Device Special Files . . . . . 1—13
1.4.2.1 Block Device Switch Table . . . . . 1—13
1.4.2.2 Character Device Switch Table . . . . . 1—13
1.4.3 Device Drivers . . . . . 1—13
1.4.3.1 Device Numbers . . . . . 1—14
1.4.4 Disk Drives . . . . . 1—14
1.4.5 Tape Drives . . . . . 1—15
1.4.6 Terminals . . . . . 1—15
1.4.6.1 Control Terminal . . . . . 1—15
1.4.6.2 Role of the Kernel . . . . . 1—15
1.4.6.3 Raw Mode Versus Cooked Mode . . . . . 1—16
1.4.7 Printers . . . . . 1—16
1.4.8 I/O Subsystem's Effect on Performance . . . . . 1—17
1.5 Understanding the Network . . . . . 1—17
1.5.1 Function of a Network . . . . . 1—17
1.5.2 Components of a Network . . . . . 1—17
1.5.2.1 Controllers . . . . . 1—17
1.5.2.2 Connectors . . . . . 1—17
1.5.3 Network Software . . . . . 1—18
1.5.3.1 TCP/IP . . . . . 1—18
1.5.3.2 DECnet . . . . . 1—18
1.5.3.3 NFS . . . . . 1—18
1.5.3.4 RFS . . . . . 1—19
1.5.4 Network Daemons . . . . . 1—19
1.5.5 Effect of the Network on Performance . . . . . 1—19
1.6 Understanding Interprocess Communications . . . . . 1—20
1.6.1 Role of the Scheduler . . . . . 1—20
1.6.2 Messages . . . . . 1—20
1.6.3 Shared Memory . . . . . 1—20
1.6.4 Semaphores . . . . . 1—21
1.6.5 The Effect of Interprocess Communications on Performance . 1—21

2 Tools for Monitoring Subsystem Resource Usage

2.1 Tools That Provide System Status . . . . . 2—2
2.1.1 The crash Utility . . . . . 2—2
2.1.2 The cpustat Command . . . . . 2—2
2.1.3 The iostat Command . . . . . 2—3
2.1.4 The netstat Command . . . . . 2—3
2.1.5 The nfsstsat Command . . . . . 2—5
2.1.6 The pstat Command . . . . . 2—6
2.1.7 The vmstat Command . . . . . 2—6
2.2 Tools That Monitor the File System . . . . . 2—8
2.2.1 Gathering Information About Disk Organization . . . . . 2—8
2.2.1.1 Using chpt . . . . . 2—9
2.2.1.2 Using df . . . . . 2—9
2.2.1.3 Using mount . . . . . 2—10
2.2.2 Creating, Checking, and Tuning a File System . . . . . 2—10
2.2.2.1 Using newfs and tunefs to Create and Evaluate a File System . . . . . 2—10
2.2.2.2 Using fsck to Evaluate a File System . . . . . 2—11
2.2.3 Disk Usage Space Allocation . . . . . 2—12
2.2.4 Swap Space Usage . . . . . 2—13
2.2.5 Exercising the Disk and File System . . . . . 2—13
2.2.5.1 Using dskx . . . . . 2—13
2.2.5.2 Using fsx . . . . . 2—14
2.2.6 Monitoring File System Activity . . . . . 2—14
2.3 Commands that Monitor Process Control and Scheduling . . . . . 2—14
2.3.1 Determining the System Users and Tasks They Are Running 2—14
2.3.2 Setting or Resetting the Priority of a Process . . . . . 2—15
2.3.2.1 Using nice . . . . . 2—15
2.3.2.2 Using renice . . . . . 2—16
2.3.3 Scheduling, Rescheduling, or Stopping a Process . . . . . 2—16
2.3.4 Monitoring Interrupts, Context Switches, and System Calls . 2—17
2.3.4.1 Using vmstat . . . . . 2—17
2.3.4.2 Monitoring Other Process Control Activity . . . . . 2—17
2.4 Tools for Monitoring Memory Management . . . . . 2—19
2.4.1 Memory Exercisers . . . . . 2—19
2.4.1.1 Using memx . . . . . 2—19
2.4.1.2 Using shmx . . . . . 2—20
2.4.2 Using vmstat to Monitor Memory Usage . . . . . 2—20

2.5 Tools for Monitoring the I/O Subsystem . . . . . 2—20
2.5.1 Determining Which Devices are Connected to the System . . 2—21
2.5.1.1 Using devstat . . . . . 2—21
2.5.1.2 Using pstat . . . . . 2—21
2.5.2 Exercising Terminals, Printers, and Magnetic Tape . . . . . 2—22
2.5.2.1 Using cmx . . . . . 2—22
2.5.2.2 Using lpx . . . . . 2—22
2.5.2.3 Using mtx . . . . . 2—23
2.6 Tools for Monitoring the Network . . . . . 2—23
2.6.1 Determining Network Usage . . . . . 2—23
2.6.1.1 Using rwho . . . . . 2—23
2.6.1.2 Using ruptime . . . . . 2—24
2.6.2 Determining the Status of the Network Interfaces . . . . . 2—24
2.6.2.1 Using ifconfig . . . . . 2—24
2.6.2.2 Using netstat . . . . . 2—25
2.6.3 Displaying Statistics for NFS . . . . . 2—25
2.6.4 Exercising the Network . . . . . 2—27
2.6.4.1 Using ping . . . . . 2—28
2.6.4.2 Using netx . . . . . 2—28
2.6.5 Monitoring Network Activity . . . . . 2—29
2.6.5.1 Tools For Monitoring Network Activity . . . . . 2—29
2.6.5.2 Using netstat . . . . . 2—30
2.7 Tools for Monitoring Interprocess Communications . . . . . 2—30
2.7.1 Using ipcs . . . . . 2—30
2.7.2 Exercising Shared Memory . . . . . 2—31
2.7.3 Monitoring IPC Activity . . . . . 2—31

3 Recognition and Diagnosis of Resource Constraints

3.1 Identifying File System Limitations . . . . . 3—1
3.1.1 Determining Whether the Disk Subsystem Needs Tuning . . . 3—1
3.1.2 Recognizing When the Disk Subsystem Is Disk-Bound . . . . . 3—2
3.1.3 Recognizing When the Disk Subsystem Is Swap-Bound . . . . . 3—3
3.2 Identifying Process Control and Scheduling Limitations . . . . . 3—5
3.2.1 Determining Whether the Process Control Subsystem Needs Tuning . . . . . 3—5
3.2.2 Computing the Load Average for the System Under Test . . . . 3—6
3.2.3 Recognizing a Well Balanced Load Over the Available Time . . 3—8
3.2.4 Recognizing a Shortage of Slots in the Process Table . . . . . 3—8
3.2.5 Recognizing Problems with Interrupts, Context Switches, or System Calls . . . . . 3—8

3.3 Identifying Memory Management Limitations . . . . . 3—10
3.3.1 Determining Whether the Memory Management Subsystem Needs Tuning . . . . . 3—12
3.3.2 Recognizing When Active Virtual Memory Is Too Large . . . . 3—12
3.3.3 Recognizing a Shortage of Physical Memory . . . . . 3—12
3.3.4 Buffer Cache Size . . . . . 3—13
3.4 Identifying I/O Subsystem Limitations . . . . . 3—14
3.4.1 Determining Whether the I/O Subsystem Needs Tuning . . . 3—15
3.4.2 Recommendation for Improving I/O Performance . . . . . 3—15
3.4.3 Measuring I/O Subsystem Throughput . . . . . 3—16
3.4.4 Identifying NFS Bottlenecks . . . . . 3—17
3.4.5 NFS Tuning . . . . . 3—17
3.5 Identifying Network Limitations . . . . . 3—18
3.5.1 Determining Whether the Network Needs Tuning . . . . . 3—18
3.5.2 Measuring the Network Subsystem Throughput Rate . . . . . 3—18
3.6 Identifying Interprocess Communication Limitations . . . . . 3—19
3.6.1 Determining Whether the Interprocess Communications Subsystem Needs Tuning . . . . . 3—19
3.6.2 Measuring the Interprocess Communications Throughput Rate . . . . . 3—20
3.7 Chapter Summary . . . . . 3—21

4 Tuning Subsystem Resource Usage

4.1 System Configuration File . . . . . 4—1
4.1.1 Global Definitions . . . . . 4—1
4.1.2 System Image Definitions . . . . . 4—3
4.1.3 Device Definitions . . . . . 4—3
4.1.4 Pseudodevice Definitions . . . . . 4—3
4.2 Tuning the File System . . . . . 4—4
4.2.1 Reorganizing the File System . . . . . 4—4
4.2.2 Changing the Size of Disk Partitions . . . . . 4—5
4.2.3 Adding a Second Swap Partition . . . . . 4—7
4.2.4 Changing the Size of the Buffer Cache . . . . . 4—8
4.3 Tuning Process Control and Scheduling . . . . . 4—10
4.3.1 Balancing the Workload . . . . . 4—10
4.3.2 Changes Involving Global Parameters . . . . . 4—10
4.3.3 Changing the Priority of a Process . . . . . 4—10
4.3.4 Setting the Sticky Bit for a Frequently Executed Process . . . 4—11

4.4 Tuning Memory Management . . . . . 4—12
4.4.1 Changes Involving maxuva, maxtsiz, maxdsiz, and maxssiz . 4—12
4.4.2 Changes Involving the Size of the Buffer Cache . . . . . 4—13
4.4.3 Changes Involving Physical Memory and physmem . . . . . 4—14
4.5 Tuning the I/O Subsystem . . . . . 4—14
4.5.1 Changes Involving Software . . . . . 4—15
4.5.2 Changes Involving Hardware . . . . . 4—15
4.6 Tuning the Network . . . . . 4—15
4.6.1 Changes Involving Global Parameters . . . . . 4—15
4.6.2 Changes Involving Software . . . . . 4—16
4.7 NFS Performance Problems . . . . . 4—17
4.7.1 Network Problems . . . . . 4—17
4.7.2 Client Problems . . . . . 4—18
4.7.3 Server Problems . . . . . 4—18
4.7.4 NFS Server Performance . . . . . 4—19
4.7.5 Prestoserve's Impact on NFS Server Performance . . . . . 4—20
4.7.6 Recommendation for Increasing NFS Performance . . . . . 4—20
4.8 Tuning Interprocess Communications . . . . . 4—21
4.8.1 Changes Involving Global Parameters . . . . . 4—21
4.8.2 Changes Involving Software . . . . . 4—22
4.9 Chapter Summary . . . . . 4—22

A DEVSTAT Source Code

B Testing System Calls, Messages, and Semaphores

C The AIM Benchmark Suite III

D The SPEC Benchmark

Index

Figures

1–1: Disk Block Organization . . . . . 1—2
1–2: Typical Root File System . . . . . 1—4
3–1: Normalized Benchmark Time Versus Response Time . . . . . 3—7

Tables

2–1: Commands for Monitoring the Subsystems . . . . . 2—1
2—2: Process Status Flags . . . . . 2—18
2—3: nfsstsat Output . . . . . 2—26
3–1: Process Timing Experiments . . . . . 3—7
3–2: Paging Parameter Defaults . . . . . 3—11
4–1: Default Partition Table . . . . . 4—4
4–2: Reorganizing the File System . . . . . 4—5
4–3: Partitions e, f, g, and h Redefined . . . . . 4—6
4–4: Further Reorganizing the File System . . . . . 4—6
4–5: File System Tests with Two Swap Partitions . . . . . 4—8
4–6: Test with Different Sized Buffer Caches . . . . . 4—9
4–7: Impact of Changing Process Priorities . . . . . 4—11
4–8: Impact of Setting the Sticky Bit . . . . . 4—12
4–9: Impact of Increasing the Size of the Buffer Cache . . . . . 4—13

Examples

1–1: Directory Listing Showing Device Numbers . . . . . 1—14
2–1: The chpt -q Command . . . . . 2—9
2–2: The newfs Command . . . . . 2—10
2–3: The ifconfig Command . . . . . 2—24
2–4: The ping -l Command . . . . . 2—28
2–5: The ipcs Command . . . . . 2—31
3–1: time and vmstat Output for bldfile Program . . . . . 3—9
3–2: time and vmstat Output for bldfile Program . . . . . 3—9
3–3: Output from vmstat when the System Is Idle . . . . . 3—10
3–4: vmstat Output when the System Could Benefit from More Memory . . . 3—13
3–5: Output from ps Showing Excessive Time for the Page Daemon . . . . . 3—13
3–6: Testing the Message IPC Subsystem . . . . . 3—20
3–7: Testing the Semaphore IPC Subsystem . . . . . 3—20
4–1: Commands that Change Partitions e, f, g, and h . . . . . 4—6
4–2: File System Mount Table . . . . . 4—7
4–3: Adding bufcache to the Configuration File . . . . . 4—8
4–4: Setting the Sticky Bit for a Command . . . . . 4—12
4–5: Network Configuration from rc.local . . . . . 4—16

About This Manual

The *ULTRIX Performance Management Guide* is a compilation of tuning information related to the ULTRIX operating system for system managers and engineers who need to tune their ULTRIX operating systems.

Tuning, in this context, means changing, adjusting, monitoring, and manipulating those parameters, subsystems, and resources that are defined and controlled by the operating system and that affect the overall performance of the computer system.

Performance is calculated using a metric such as response time, processes executed per second, or I/O throughput. The metric must be a quantitative, objective measurement making comparisons.

Specifically, this guide contains the following:

- An overview of the ULTRIX operating system
- A description of ULTRIX operating system tools that you can use to monitor and improve system performance
- Examples of performance problems
- Advice on how to improve system performance

Audience

The guide is intended for experienced managers and users of the ULTRIX operating system; however, it does provide basic information that can be useful to novice users.

This guide provides guidelines; it does not define rules for tuning the ULTRIX operating system.

Structure of This Guide

This guide consists of four chapters and four appendixes:

Chapter 1	Reviews the four major areas of the operating system where performance improvements can be found: — File system — Memory management and process control subsystem — I/O subsystem — Networking and interprocess communication subsystem This chapter is useful if you are familiar with operating systems, but not with the ULTRIX operating system.
Chapter 2	Summarizes information about tools that are available to monitor system performance, including Digital-supported utilities.
Chapter 3	Discusses how tools mentioned in Chapter 2 can be used to identify performance problems.
Chapter 4	Presents examples for tuning drawn from the problems identified in Chapter 3.
Appendix A	Provides an example of source code written for a utility that gathers information about devices connected to the system.
Appendix B	Provides source code for three different test programs used to illustrate system performance issues.
Appendix C	Provides a description of the AIM Benchmark Suite III, which is used as a sample workload in many examples in this guide.
Appendix D	Provides a description of the SPEC Benchmark, which is used as a sample workload in many examples in this guide.

Associated Documents

The following ULTRIX documentation is of interest to the general user and system administrator:

- System and Network Management Documentation Kit — seven volumes
- Software Development Tools Documentation Kit — five volumes
- General Information Documentation Kit — three volumes including an index
- Supplementary Documents Kit — three volumes

Conventions

The following typeface conventions are used in this manual:

`csh>`	The csh followed by a right angle bracket represents the C shell system prompt.
`#`	A number sign represents the superuser prompt.
`% cat`	Boldface type in interactive examples indicates typed user input.
`cat(1)`	A cross–reference to a reference page includes the appropriate section number in parentheses. For example, `cat(1)` indicates that you can find information on the `cat` command in Section 1 of the reference pages.
file	Italic (slanted) type indicates variable values, placeholders, and function argument names.

When the complete reference is not given in a footnote, bibliographic references begin with the author's last name followed by the last two digits, in brackets ([]), of the publication year of the document.

1 Review of ULTRIX Subsystem Resource Usage

The overall performance of a system depends on the performance of its CPU, memory, and I/O resources. Not only must each resource operate efficiently by itself, but it must also interact with other resources. For example, the CPU may appear to have a performance problem when it is actually waiting for disk I/O to be completed or memory to be allocated. A difficult aspect of performance management can be diagnosing the limiting resource.

This chapter discusses the four major areas of the operating system where you can look for performance improvements:

- File system
- Memory management and process control
- I/O subsystem
- Networking and interprocess communication subsystem

In each area, the observed behavior is related to the underlying software and hardware that is responsible. Specific features or aspects of each subsystem that is a candidate for tuning are explored.

1.1 File System

Users want satisfactory performance from any operating system. To understand ULTRIX, it is essential to understand the file system and its components. If you understand the way files are set up and optimal ways to locate and use files, you will benefit from faster access and more efficient computer use. Conversely, by knowing the most efficient ways of using ULTRIX, you can educate others, thereby increasing overall satisfaction and performance. Knowing how to use the file system is one of the most immediate ways to improve the performance of the ULTRIX operating system.

1.1.1 Function of the File System

The file system allows users to accomplish the following objectives:

- Store large quantities of data
- Retrieve data quickly and easily

In addition, a file system handles the following functions:

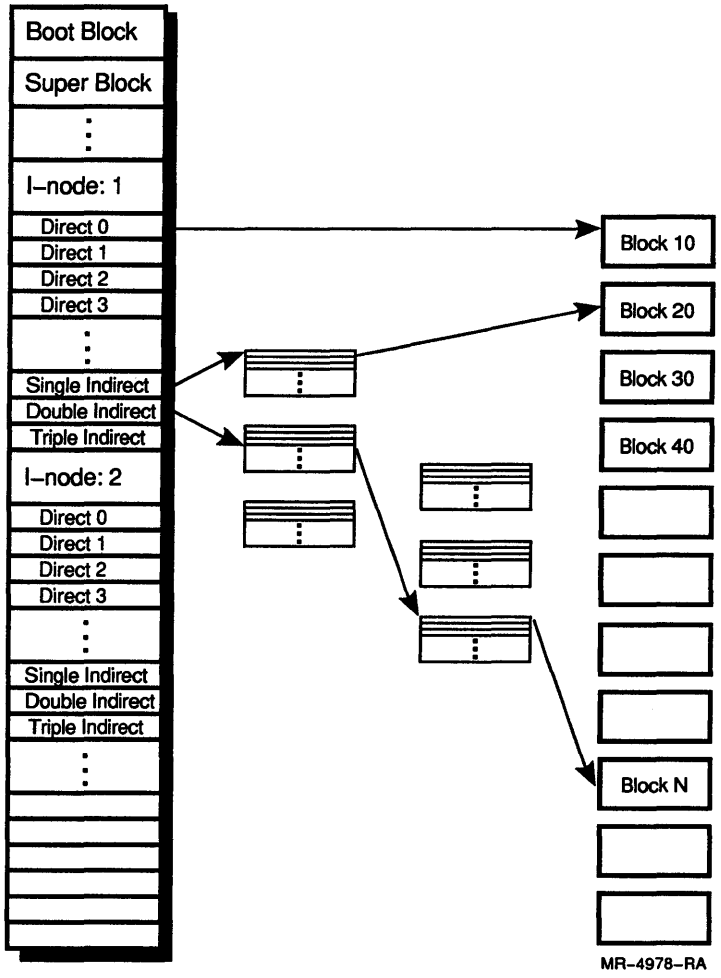
- Manage files
- Allocate file space
- Administer free space
- Control access to the files

1.1.2 Component Parts of a File System

Every file system consists of four components: a boot block, a super block, inodes, and data blocks.

In addition to blocks of user information, each file system has blocks that contain information about the system's organization. The file system depends on the organizational information contained in these blocks. Figure 1–1 shows the disk block organization.

Figure 1–1: Disk Block Organization



A block is comprised of contiguous bytes of information and contains data or file system information. The block size on early versions of UNIX was 512 bytes, but it was increased first to 1024 bytes (1K) and then to 2048 bytes (2K). The ULTRIX file system is based on Berkeley UNIX; its block size is user selectable. The default size is 8K bytes.

A file system's granularity is its block size. As files increase in size, they are assigned additional blocks. The use of large 8K blocks improves the throughput of the file system; however, for small files, 8K blocks are an inefficient use of space. Therefore, the ULTRIX file system allows an 8K

block to be divided into smaller pieces called frags. The default size frag is 1K bytes. File system data structures keep track of the disk at the frag level.

An inode (index node) describes the files and directories in the system. One inode exists for each directory and each file. An inode contains information about permissions, number of links, and block numbers of the data blocks comprising a file.

The super block contains critical information about the file system, such as the file system's name, its size in blocks, and number of blocks for inodes. It is located at block 16 of a disk partition. Backup super blocks are allocated at the time the file system is created by ULTRIX. One is always created at Block 32, and periodically after that. The number of backup super blocks created depends on the size and layout policy of the file system.

The boot block is reserved for the bootstrap program that boots (or initializes) the operating system. It is located at the beginning of a file system in the first sector.

1.1.3 Disk Blocks

ULTRIX is ideally suited to addressing small files, though it does allow files to become quite large. Small files are comprised of direct data blocks. Larger files are comprised of direct data blocks plus indirect blocks that contain pointers to more data blocks. Because ULTRIX uses larger block sizes, it is unlikely that more than a direct block or single indirect block is needed to address a single file.

The first 12 blocks of data are called direct blocks; they can handle a fixed number of bytes of information determined by the block size when the file system is created (4K bytes minimum).

If a file contains more than 12 direct blocks, the thirteenth block is called an indirect block. It contains pointers to additional data blocks for the file system. This block can also point to another block that contains pointers to additional data blocks or pointers.

1.1.4 Relationship of File Systems to Devices

A file system must exist on a device. In this context, a device is a physical piece of equipment (hardware). Examples of devices that can contain a file system are hard disks, floppy disks, and sometimes compact discs. Examples of devices that cannot contain a file system are terminals, printers, and communication lines.

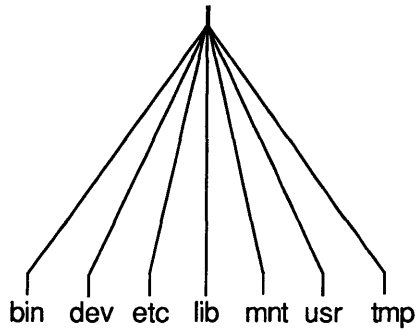
A file system cannot span physical disk drives or partitions on the same physical disk drive. It must begin and end on the same physical disk drive and, furthermore, must begin and end on one partition. Thus, the size of a file system is limited to the size of a partition on the target disk drive. Partitions can be adjusted up to the entire physical disk drive if necessary. See Section 1.1.6 for more information about the relationship between the ULTRIX file system and devices.

1.1.5 Relationship of Files to Other Files in a File System

The file system in ULTRIX is hierarchal. Directories contain files, which can include other directories (subdirectories). Subdirectories also can contain files or other directories. These directories and subdirectories relate to each other in predictable ways, depending on their location in the hierarchy.

The directory structure of ULTRIX is graphically depicted as an inverted tree structure. This image conveys the directory system with its roots, the trunk, the strong branches and smaller limbs, and finally the leaves. The file system has a parallel structure. Figure 1–2 shows the tree structure of the root directory.

Figure 1–2: Typical Root File System



1.1.5.1 Root Directory

The root directory (/) is where the directory structure begins. The root directory is the base from which all other directories are referenced.

Note that the term root is also given to the primary administrative person involved with the ULTRIX system, the superuser. The superuser assigns user names among other systemwide tasks and is often a valuable resource to users. In addition, the superuser maintains and ensures the smooth operation of the system.

1.1.5.2 Primary Directories Off of the Root

Each directory off the root directory is significant. The important representative directories are as follows:

- `bin` (binary)
  The `bin` directory contains the binary executable versions of ULTRIX system commands.

- `dev` (device)
  The contains device-special files that facilitate communication between the operating system and the hardware devices.

- `etc` (et cetera)
  The `etc` directory is a catchall directory. It usually contains miscellaneous system utilities (executable programs such as `passwd`) and data files (`termcap`, the terminal capabilities database).

- `mnt` (mount)
  The `mnt` directory is an empty directory. It is used primarily to temporarily mount file systems.

- `usr` (user)
  The `usr` directory contains several key subdirectories, including another `bin` (for additional executable commands), `dict`, `games`, `include`, `lib` (for library), and others.

- `tmp` (temporary)
  The `tmp` directory contains files that are used for short periods of time and then deleted.

1.1.6 Associating a Device with a File System

When you associate a device with a file system, you mount the file system.

A file system can be created on a diskette or hard disk; however, the rest of the ULTRIX system is unaware of its existence until it is mounted. This procedure associates the particular physical device with the file system and relates its existence to the kernel and other parts of the file system that may be located on different physical devices.

To mount a file system, do the following:

1. Create an empty directory entry if one does not already exist.
2. Mount the device onto that directory. For example:

```
csh> mount /dev/rz1g /usr/tools
```

When you disassociate a device from a file system, you unmount the device.

To unmount a device, use the `umount` command. For example:

```
csh> umount /dev/rz1g
```

Mounting and unmounting a device allows the operating system to perform some necessary housekeeping tasks. For example, under certain circumstances, the operating system does not write data to a disk when a program tells it to do so. Buffered data (the data not yet written to disk) must be written to the disk before the device is unmounted.

1.1.7 Relationship Between the File System and Performance

The effective use of the ULTRIX file system can significantly impact the speed at which you can access data. Key variables are the size of files, location of files, and the location of frequently accessed files.

1.1.7.1 Size of the Files

Smaller files are preferable to large files. With large files, many data blocks are used; some of those may be indirect or doubly indirect referenced. Also, as the file system ages, free blocks become randomly distributed over a disk. Thus, files may contain blocks that are randomly distributed (slower to access) rather than virtually contiguous (faster to access). The system must locate these data blocks each time the file is read.

1.1.7.2 Location of Files

The location of files affects file access time. Some locations are determined when the system administrator configures the system. Other locations are determined by the user. Correctly locating files, to the extent that the user has control over their location, is a way to improved performance.

1.1.7.3 Factors Determined at Time of Configuration

Many factors that affect file system performance are determined when the file system is configured (for example, the physical location on the disk of the file system).

Other factors, such as contiguous versus noncontiguous blocks, are determined when files are created and as they grow. When data blocks are contiguous (located in blocks that are physically close to each other), access time is less because the read-and-write head of the disk drive has less distance to travel.

You have little control over these factors, but knowing about them helps you to understand how the way you organize files can affect file system performance.

1.1.7.4 Organizing Files to Improve Performance

You can affect file system performance in several ways. For example, organizing files into directories affects file system performance:

- Locating files in one large directory
  This is the least desirable way to store data, since finding a file in a large directory takes longer. Ideally, a directory should not be larger than one block, although the penalty for searching a larger directory is not great because ULTRIX search algorithms are very efficient.

- Locating files in multiple subdirectories
  By using multiple subdirectory entries, organization is enhanced. If you create a file that increases the size of a directory by more than one file system block, then it is best to create a subdirectory.

1.2 Understanding Process Control and Scheduling

ULTRIX is known as a multitasking, multiuser, and timesharing operating system. Multitasking means that the system can perform several tasks at a time, or run several processes at a time, for any given user. A process is a running program with a specific job to do. Multiuser means that the system is capable of serving more than one user simultaneously. A timesharing system is one that can schedule tasks and users so both information and computer time can be shared.

Process control and scheduling is the way that ULTRIX handles all of these demands. The responsibilities of the process control and scheduling subsystem include:

- Handling requests from users
- Scheduling related and unrelated tasks
- Responding to simultaneous commands
- Maintaining files
- Coordinating requests for resources with user demands

Users make constant demands on the ULTRIX system. It is important to understand how the system assigns priorities to processes and what impact this has on the system's throughput. Users can also be aware of the processes underway and choose optimal times to run certain processes.

1.2.1 Types of Processes

The process that communicates with your terminal or workstation is called a foreground process. Most commands by users are foreground processes. For example, your shell runs as a foreground process. When you ask the shell to run a command, the shell translates your request into actions by the kernel and other programs. The shell runs that command as a process.

A background process is a process that is running but is neither connected directly to any terminal nor waiting for input from the shell. Many system utilities are background processes that are running all the time. The `getty` program, for example, listens constantly for any attempt to login on a line. If it detects a login attempt, it forks the login process. You also can put a process in the background and continue to work on another task while the background process executes.

1.2.2 Process Priorities and Privileged Processes

Process control and scheduling is necessary to set up priorities and schedule processes to run according to these priorities. The kernel runs with the highest priority; it is the part of the system that manages the resources of the computer system. It keeps track of random access memory, the disks, tapes, printers, communication lines, in short, all resources.

The kernel must respond to requests from executing programs and hardware interrupts from devices that need service. Depending on the priority of the interrupt, the kernel can suspend all other processes immediately to handle the interrupt, or it can wait until some other higher priority task is completed. User processes are assigned a lower priority than system services; however, these priorities are constantly changing, based on a number of factors.

Certain processes are also given privileged status and their execution takes precedence over all other processes. To manage memory appropriately, the swapper (the utility that controls the memory-management facilities) is always the highest priority task and can preempt other processes.

1.2.3 Factors Used to Determine Process Priorities

User processes are assigned a base priority when they begin. The base priority is adjusted by the kernel as the process runs to allow all user processes equal access to system resources. As a process executes, its priority decreases based on the amount of time it has been active. When the process is preempted by a higher priority task, it is put onto the run queue. It waits until its priority is higher than the task that is currently running.

Other factors cause a process to be temporarily suspended. When a process requires access to a system resource, it executes a system call. For example, a process that must read data from the disk drive executes a read system call. Because it takes a finite amount of time for the kernel to pass the request for data to the disk controller and for the disk to respond with the requested data, the process that requests the data is suspended until the request is completed. When the system call is completed, the process is marked as eligible to run and is restarted in turn. Process priority increases when waiting for I/O.

1.2.4 Scheduling a Process to Run During Off-Peak Hours

By using the `cron` or `at` commands, you or the system administrator can change the time when a task is run. System performance is improved if certain kinds of tasks are scheduled to run when the machine is lightly loaded. For example, if you schedule system backups when only a few users are on the machine, users during normal hours are not affected by the considerable degradation in response time caused by system backup.

The `cron` command runs a process as a batch job at a specific time. The `cron` command periodically examines the directory `/usr/lib/crontab` for files that contain directions on tasks to perform. Based on the instructions it finds in the file, `cron` schedules the command or set of commands to run at the specified times. Tasks that need to be performed on a regular basis (for example, nightly backups, large jobs, and other administrative tasks) should be executed through `cron`.

The `cron` command is a daemon (a process that is continually running in the background). You must receive specific authorization from the system administrator to use `cron`.

The `at` command runs a command at a specified time. The command to be run is supplied as an argument to the `at` command at the shell prompt. Usually, jobs executed through the `at` command are done only once and not on a regular basis.

1.2.5 Relationship Between Process Control and Performance

Performance can be improved through process scheduling at the time of configuration and through system administration.

One way to improve performance is during configuration by increasing or decreasing the number of processes that can run simultaneously.

You or the system administrator can change the priority with which a process runs by using the `nice` command. By default, `nice` reduces process priority by 10 percent when it is used to execute a command. You can reduce process priority by up to 20 percent by using an optional argument to `nice`. Only the system administrator can increase the priority of a process.

1.3 Understanding Memory Management

Memory management is the allocation of memory. If physical or main memory is insufficient for any process, the kernel moves processes between main memory and secondary memory (refers to back-end storage or page/swap storage) so all processes can execute. The subsystems of the kernel and hardware that work together to translate virtual to physical addresses make up the memory management subsystem.

1.3.1 Types of Memory Management

ULTRIX systems use two types of memory management: simple memory management, and memory management with swapping and paging. ULTRIX uses memory management with swapping and paging.

With simple memory management, the kernel allocates the necessary memory from physical memory to execute a process. Physical memory is divided into a set of equal sized blocks, called pages. Pages can vary in size from 4K to 512K bytes. Page tables exist that identify the location of these pages. For example, in one popular simple memory model[1], the system contains a set of memory-management register triples. A register triple is a set of three registers that contain the following information:

- The first register contains the address of the page table in physical memory.
- The second register contains the virtual address mapped via the triple.
- The third register contains control information, such as number of pages in the page table and page access permissions.

---

[1] Maurice J. Bach, The Design of the UNIX Operating System, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1986, pp. 155-158.

When the kernel prepares a process for execution, it loads these registers with the corresponding data stored in the pregion (per process region table) tables. A pregion table includes information to determine where the contents related to the process are located in physical memory.[2]

In memory management with swapping and paging, the kernel is responsible for making certain that sufficient primary memory exists for a process. It might be necessary at times to write processes to a secondary memory device to provide more space in primary memory. The secondary memory device is called a swap device and is on a configurable disk.

By understanding how memory is stored and allocated, you can plan your workload so that the machine is not running too many processes simultaneously. Through monitoring the CPU, using the `vmstat` command for example, you can learn whether paging and swapping are occurring and to what extent. Both swapping and paging are CPU intensive and are considered common elements in poor system performance.

1.3.2 Allocating Memory

ULTRIX allocates a number of blocks of memory to a program when it first starts. As the program executes, it can request additional memory. The kernel allocates additional memory up to a predefined maximum that also is determined when the kernel is built.

When a program is run, the memory management subsystem sets up two separate areas, the text area (or region) and the data area. The program instructions are assigned to the text area; the data related to the process is located in the data area. By keeping these two sections separate, both protection and sharing are possible.

The kernel tries to use memory as efficiently as possible. It allows users executing the same program to share the text segment, the section of memory in which the program's instructions are stored. This is possible through the use of virtual addresses. The compiler generates addresses for virtual address space, and the memory management subsystem translates these virtual addresses into address locations in physical memory. Thus, several programs can execute the same virtual addresses but use different physical addresses.

1.3.3 Buffer Cache

The buffer cache is the part of random access memory (RAM) used for disk blocks that contain data from recently used or frequently used data. A buffer is memory in which information is temporarily stored and cache is a hiding place. Thus, the buffer cache functions between the kernel and the memory as a quickly accessible source of data and instructions.

The buffer cache decreases the frequency of disk access. When the kernel tries to read data from the disk, it first reads from the buffer cache. If the data is not there, it reads the data from the disk and caches it for potential future use. Similarly, pages are written to the buffer cache first until the buffer cache is full. Only then are the oldest pages written to the disk swap area.

The size of the buffer cache is determined when you configure the system. The default size of the buffer cache is 10 percent of physical memory. By increasing the size of the buffer cache, the system may perform fewer reads

[2] Ibid., p. 155.

and writes from the disk and therefore runs more quickly. However, as more memory is allocated to the buffer cache, less memory is available for program execution. This reduces the system's capacity, and paging or swapping or both will increase when the system is loaded. As a result, fewer users can run programs, and fewer processes can run simultaneously.

1.3.4 Swap Area

The swap area, (also known as swap space) is the section of the swap device allocated in groups of contiguous blocks. The swap device is a block device in a configurable section of a disk. The kernel's allocation of swap space depends on the process scheduling. An in-core table, called a map, maintains a directory of free space for the swap area. As the kernel allocates and frees up resources, it updates this map. This enables the kernel to have current information about free resources.

1.3.5 Changes in Memory Management that Can Affect Performance

You can improve performance by the way you configure the kernel and through ongoing system management. The size of the buffer cache and the size and location of the swap partition are two variables that you can set when you configure the kernel. You can add memory, change process priorities, and set the sticky bit for executable files. These are examples of system management that affect system performance.

1.3.5.1 Size of the Buffer Cache

The default size of the buffer cache is usually 10 percent of physical memory. The increase or decrease in the buffer cache size affects both the speed of the system and the amount of swapping and paging. The buffer cache is best tuned for each application. For example, a file server application benefits from 50 percent of physical memory allocated to the buffer cache, but it does not benefit from a large number of swap segments. By reducing the size of the buffer cache, you can give some memory back to the system to use for programs.

1.3.5.2 Size and Location of the Swap Area

The ULTRIX operating system allows multiple swap devices. The kernel interleaves swap requests; that is, the kernel attempts to evenly distribute the swap requests among all the swap devices. A parameter in the configuration table determines the size of the swap area. The number and location of swap devices is also specified in the configuration file. Place the swap partition on the fastest disk. The swap areas are best spread between the controllers and disks.

1.3.5.3 System Management

Once you configure the system, the opportunities to effect higher performance in memory management are reduced. The purchase of additional memory is the most obvious option. Using the `iostat` and `vmstat` commands, you can determine the extent of paging and swapping. If extensive paging is occurring, the system might require more memory.

These monitoring programs and utilities can measure the efficiency of the buffer cache by its hit rate. A 98 percent hit rate is possible with a machine that has sufficient memory. As the hit rate improves, the number of blocks transferred to and from the disks will be reduced. It is important to monitor the buffer cache before you decide to purchase additional memory. Administrators can create and remove swap devices by reconfiguring the kernel and rebooting the system.

A revaluation of the priorities assigned to various processes can also affect memory management. Managing processes that require a large amount of CPU time, have exclusive access to devices that are in short supply, or consume a large quantity of a system's scarcest resources, can affect system performance as well. You can improve the system's efficiency by doing the following:

- Lowering the priority of memory or disk intensive applications
- Educating users to avoid running a large number of processes simultaneously
- Informing users of the memory requirements of various processes and programs
- Reducing the number of users working simultaneously

The system administrator can set the sticky bit for frequently used executable files. The sticky bit is a permissions flag that tells the kernel to retain the text segment of a program. If the text segment remains in memory, it does not have to be reloaded when another process executes it. If the system does not have to reload the program, the program will be executed much more quickly. Even if the kernel swapped the file to the swap device, it is still faster to load the text from the swap device than to load it from the file system.

1.4 Understanding the I/O Subsystem

The I/O (Input and Output) subsystem involves the hardware that performs all reading and writing operations on the ULTRIX system. This refers to all peripheral equipment: terminals, disks, tape drives, printers, the network and communication lines.

You have direct contact with the pieces of physical equipment that are interfaced by the I/O subsystem. The more you understand about the relationships and communications between the hardware, the operating system, and the applications, the more effectively you can use the entire system.

1.4.1 Components of the I/O Subsystem

Each system has its own configuration of hardware, but ULTRIX addresses these devices in particular ways. Two categories of I/O devices exist: block devices and character devices.

1.4.1.1 Block Devices

Block devices are random access storage devices. The system can access particular blocks, not necessarily sequentially. Data is passed to block devices as complete blocks, buffered by the operating system until a whole transfer is ready. Data devices can also be accessed in a character device for unbuffered operations.

1.4.1.2 Character Devices

Character devices include all devices that are not block devices and that cannot be addressed by block. Data is passed to character devices sequentially in a stream of characters. Some examples of character devices are terminals, line printers, and tape drives.

1.4.2 Device Special Files

Each device has a name that looks like any other file name and that is accessed like a file. However, the device type is stored in the device file within the inode; it indicates whether the device is addressed as a block device or as a character device. Two device special files might exist for one physical device, with one file a block-special file and the other a character special file. These files contain pertinent information about communication and operation of the device.

1.4.2.1 Block Device Switch Table

The block device interface in the kernel is the block device switch table. This table includes entry points for device open, device close, and strategy procedures as well as others. The kernel uses the strategy procedures to transmit data between the buffer cache and the device. The strategy procedures can also schedule I/O jobs for a device.

1.4.2.2 Character Device Switch Table

The character device interface is called the character device switch table. It includes entry points for device open, device close, read, write, and `ioctl` procedures. The `ioctl` (I/O control) procedures deal with the flow of control information between devices and processes.

1.4.3 Device Drivers

Device drivers are kernel modules that control the devices. They enable peripheral devices to serve the entire system. A system might include one terminal driver for all terminals, one disk driver for all disks, and one tape driver for all tapes. Even though the specific devices might be different, the driver distinguishes between them. A buffering mechanism interacts with the block I/O device driver to initiate the transfer of data to and from the kernel. The file system interacts directly with the character I/O device driver without a buffering mechanism.

1.4.3.1 Device Numbers

In the inode of the file for each device are two numbers for identification: a major device number and a minor device number. The device number is a coded combination of these numbers. The major device number identifies the device type, such as a disk, terminal, or tape drive. This number serves as an index to the appropriate switch table. The minor device number indicates the unit number of the device and is an argument that is passed to the device driver.

Example 1–1 shows a partial directory listing of the `/dev` directory. The fourth and fifth fields, separated by commas, are the major and minor device number for the device special files shown. The last column contains the device name. For example, `rz0` is a fixed disk and the letters `a` through `h` refer to the partition. The major device number for the disk is 19 and the minor device numbers range from 0 to 7.

Example 1–1: Directory Listing Showing Device Numbers

```
csh> ls -l /dev
total 24
crw-r--r-- 1 root 3,          1 Nov 15 1988 kmem
crw-r--r-- 1 root 3,          0 Apr 24 12:58 mem
crw-rw-rw- 1 root 3,          2 Jul 25 10:13 null
crw-rw-rw- 1 root 21,         0 Jul 25 10:18 ptyp0
crw-r--r-- 1 root 21,         1 Jul 19 17:17 ptyp1
crw-r--r-- 1 root 21,         2 Jul 16 13:56 ptyp2
crw-rw-rw- 1 root 21,         3 Jul 19 11:00 ptyp3
crw-r--r-- 1 root 21,         4 Jul 18 12:06 ptyp4
brw------- 1 root 19,         0 Apr 25 09:50 rz0a
brw------- 1 root 19,         1 May 31 09:24 rz0b
brw------- 1 root 19,         2 Apr 24 09:58 rz0c
brw------- 1 root 19,         3 Apr 24 09:58 rz0d
brw------- 1 root 19,         4 Apr 24 09:58 rz0e
brw------- 1 root 19,         5 Apr 24 09:58 rz0f
brw------- 1 root 19,         6 Apr 24 09:58 rz0g
brw------- 1 root 19,         7 Apr 24 09:58 rz0h
```

1.4.4 Disk Drives

A disk drive is the physical hardware containing the hard disk used for data storage. The disks are partitioned (divided) into sections during the initial configuration (or initialization) of the system. Historically, disks used with ULTRIX systems are configured into sections that contain individual file systems to make the disk more manageable. Contiguous tracks and blocks of the disk are assigned to the file system located in that section, facilitating copying and data access.

The size of a disk partition depends on the disk type. System utilities configure an appropriately sized file system to reside on the selected disk partitions. Partitions may overlap, although the file systems in each partition must be configured not to overlap. To facilitate system backups, configure a partition so it includes the entire disk.

Disks can be addressed as either block or character devices, but most applications address the disk as a block device. The kernel initiates a process

to open the files to gain access to the block device through the block device switch table. The major device number is the index into the block device switch table where pointers to the various device routines are located.

1.4.5 Tape Drives

A tape drive is a character device that interfaces with the user through the character device table in the kernel. The minor number is an argument to the driver; it tells the driver to leave the tape positioned to the end of the data set when it is finished reading or writing, to rewind when the write or read is finished, or to perform another function.

Tape drives also can be addressed as block devices, although access is much slower. As a block device, blocks are located on the tape sequentially; thus, to access a random block, the device driver must know the current location on the tape and move the tape forward or backward to find the appropriate block.

1.4.6 Terminals

Terminals are character devices with which users have the most ongoing interaction. To establish a connection between the terminal and the system, the kernel runs the `getty` routine and the `login` and `init` commands. When the system detects an active terminal line, the kernel performs the following procedure:

1. The `init` command invokes the `getty` routine which runs continuously, alternating between the sleep and run states, on that line until it detects an attempt to log in
2. When `getty` detects a login attempt, it execs (replaces itself with) the `login` process
3. The `login` command prompts users to identify themselves with their user name and password and checks the validity of this information.
4. If the information is correct, it executes the user's shell, and the user can begin working.

1.4.6.1 Control Terminal

The control terminal is the terminal on which a user logs in. It controls processes that the user initiates from that terminal, although all processes run by a user have access to the terminal.

1.4.6.2 Role of the Kernel

In supporting terminals, the kernel uses character blocks (cblocks) and character lists (clists). Cblocks are buffers that hold the characters coming from and going to the terminal. Clists maintain pointers and linkage information to Cblocks. Specifically, the kernel associates three Clists with each terminal:

- Raw data from the keyboard
- Processed (cooked) data
- Output

The Clists serve as a buffer mechanism and allow manipulation of data, one character at a time, in groups of Cblocks.

The transmission of data to and from terminals is controlled by terminal drivers. Due to the terminal's important role as the user's interface with the system, the terminal drivers contain line discipline modules that interpret input and output. These line disciplines manipulate data on clists.

1.4.6.3 Raw Mode Versus Cooked Mode

When a terminal is in raw mode, data is transmitted exactly as the user types it without any conversion or processing. Many programs that process input from the user one character at a time must use raw mode. This allows a program to recognize and to act on a single character that is pressed on the keyboard. However, when a terminal is in raw mode, and keyboard input is not being processed by a program, the terminal appears to the user to be hung; that is, it does not respond to anything typed by the user.

In contrast, when the terminal is in cooked mode, data has gone through the line discipline process and special characters, such as carriage returns, erase, and kill, have been converted. In cooked mode, data is echoed back to the terminal as it is typed.

These modes are usually set by the application or by the system utility `stty`. The raw mode is particularly important for screen-oriented applications (such as the screen editor `vi`) that contain many commands that do not end with a carriage return.

1.4.7 Printers

Printers are character devices that are controlled by a daemon process. A daemon process handles system-wide functions, such as execution of time-dependent activities and line printer spooling (queueing). A daemon process always runs, waiting for jobs to be queued to it. When a user sends output to the printer, a temporary file is written to the spooler directory.

The `lpd` line printer daemon is a print spool handler. Normally, the program is invoked at boot time from the `/etc/rc` file. The daemon works with several system programs and files to coordinate and synchronize printer activity. The ULTRIX operating system provides this program. Although users cannot modify the file, they can specify spooling, logging, and locking activities. You must have superuser privileges to access this program.

When a print job is submitted using the `lpr` command, the printer daemon schedules jobs and notifies printers that have jobs waiting.

When signaled for inputs, the printer daemon checks the spooler directory, `/usr/spool/lpd`, for the existence of a lock file. If the lock file exists, `lpd` knows that another job is currently printing. If a lock file is not present, `lpd` creates one to reserve access to the printer for a particular print job.

1.4.8 I/O Subsystem's Effect on Performance

The I/O subsystem is very system dependent. The crucial decisions are made when you purchase the hardware. Most tuning is accomplished when an application is written. However, you can effect some change in performance during system configuration and through system administration.

The system administrator determines the way the disk is partitioned and the assignment of file systems to the partitions. The software engineer can improve system performance by judicious use of system I/O buffers. Understanding the way the kernel buffers I/O to devices helps in this effort.

The system administrator can increase or decrease the priority of the printer scheduler and other processes to improve performance.

1.5 Understanding the Network

The network is an I/O subsystem made up of a group of devices, workstations, minicomputers, and/or mainframe computers communicating through wires or cables for the purpose of moving information from place to place. It enables users to communicate with other users and devices without regard to their actual physical location. ULTRIX users may also be able to connect file systems from one computer to the file system of another with the Network File System (NFS), which is built on top of the Transmission Control Protocol/Internet Protocol (TCP/IP) network. Thus, users are no longer limited to the file systems physically located on the machine where they logged on.

1.5.1 Function of a Network

A network provides a means to move data from one device to another. This data might be no more complicated than electronic mail. You can copy files containing printable data (for example, word processor files), or binary data from a local computer to a remote computer, with the same ease as files copied from one directory to another on the local computer. With remote login, users can login to a remote computer on which they have an account and access programs and data as if they were at a terminal connected to their own host computer.

1.5.2 Components of a Network

A network consists of two essential component parts: the hardware implementation and the software that runs the network. The hardware consists of controllers and connectors.

1.5.2.1 Controllers

The controller sends and receives packets of data over the network. Controllers are specialized and are designed to work with a particular type of computer (bus architecture). For example, controllers designed to work with a Digital workstation will not work with a Sun or Hewlett-Packard workstation, or an IBM-PC, and vice versa.

1.5.2.2 Connectors

The cables or wires connecting different computers (or nodes) on a network can be twisted-pair, as with telephone wires, thick or thin Ethernet cable, or optical fiber. The type of controller determines the type of connector.

1.5.3 Network Software

Two relevant network software implementations exist for ULTRIX: TCP/IP and DECnet. Their names refer to the protocol used to send information from one network node to another, as well as to the software written to implement these protocols, which are the rules and formats that conduct communications on a network. Protocols govern the way messages are packaged, addressed, and routed; master/slave relationships among network nodes; polling; the exchange of control information; and the hierarchy.

1.5.3.1 TCP/IP

TCP/IP (Transmission Control Protocol/Internet Protocol) was developed by the U.S.Defense Department, Defense Advanced Research Projects Agency (DARPA). Its name comes from its two main standards: Transmission Control and Internet. Because TCP/IP is a collection of protocols, rather than a particular software program, the software that provides its services has been implemented for many different hardware platforms and operating systems. The TCP/IP protocols are used as building blocks on which other products or applications are built. As a result, it is a widely accepted standard in the UNIX world.

1.5.3.2 DECnet

DECnet is Digital Equipment Corporation's implementation of a network protocol common to its operating systems. The advantage of DECnet over TCP/IP in a Digital environment is that Digital controls its development and distribution and is able to optimize the implementation for its own hardware and systems software. However, from the user's point of view, DECnet and TCP/IP accomplish the same tasks. Because both can coexist on the same physical network, both can be present and used where their application is warranted.

1.5.3.3 NFS

The Network File System (NFS) is a product that utilizes TCP/IP, built originally in a Berkeley UNIX environment. It is a proprietary product developed by Sun Microsystems; however, it has now been ported and licensed on many other UNIX implementations, including ULTRIX. NFS allows users to mount remote file systems in their own local directories, thereby giving the appearance of an extension of their local file system. The machine that offers file systems for other machines to access is called the server or file server; the machines that access these file systems by remotely mounting them are called clients.

NFS, however, is not a network extension of UNIX and does not adhere to UNIX semantics. It does not support all UNIX file system operations, does not guarantee atomic operations, and cannot obtain access to remote devices. It operates independently of the machine and operating system and can be used on non-UNIX machines as well as those with UNIX.

The User Datagram Protocol (UDP) is commonly used in the Network File System. UDP is the internet standard protocol that allows an application program on one host to send a datagram to an application program on another host. UDP provides an unreliable, connectionless delivery service using IP to transport messages among hosts.

UDP is similar to TCP and it provides a mechanism for user applications to communicate with IP. UDP differs from TCP in that it is a simple protocol that is entirely dependent upon IP's best effort to provide reliability. UDP does not guarantee delivery, occasionally generates duplicate data packets, and may send data in the incorrect order. However, layers above UDP can create reliable services using UDP.

1.5.3.4 RFS

RFS (Remote File System) is a product developed by AT&T that is similar to NFS, but that can be used only with UNIX System V operating system. It ensures that all network transactions follow UNIX I/O semantics. RFS uses an I/O facility called streams to connect to a network protocol, such as TCP/IP or Ethernet. It consists of client and server machines living within a domain. The consistency in maintaining UNIX semantics with RFS contributes to high traffic, loss of performance, and bottlenecks.

1.5.4 Network Daemons

Both the host (client) and remote (server) machines start network daemon processes running when they are booted. Machines that can be reached from the network are listed in a data file with their network addresses. Each local machine knows its own name and network address. As data is sent out over the network, the address and routing information are filled in by the sending network daemon. Network daemons on receiving machines decode the address to determine for whom the message is intended. If the message is intended for the receiving machine, it decodes the message and processes it; otherwise, it does nothing.

1.5.5 Effect of the Network on Performance

The network's configuration is constrained by the hardware used to implement it. When the system is configured, the system administrator specifies the controllers and network packages in the configuration file, that is, DECnet, NFS, and TCP/IP.

Most tuning opportunities that relate to the network involve identifying bottlenecks. For example, if one of the links in the network times out waiting for an acknowledgment of a transmitted message, a second transmission of the message results. When the acknowledgment for the first message is received, it will be interpreted as the acknowledgment for the second message sent. This results in an error in the system's estimate of the round-trip delay. This problem is referred to as the Cypress syndrome.[3] The system tries to adjust its transmission rate based on this incorrect transmission delay and continues to transmit and retransmit packets as the acknowledgments are delayed.

---

[3] Douglas Comer, Internetworking with TCP/IP: Principles, Protocols, and Architecture, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1988, p. 294

1.6 Understanding Interprocess Communications

Interprocess communication (IPC) occurs when mechanisms allow processes to exchange data and synchronize execution. Some examples of IPC are messages, shared memory, semaphores, pipes, signals, process tracing, and processes communicating with other processes over a network. IPC is a functional interrelationship of several ULTRIX subsystems. Elements are found in process scheduling and networking.

1.6.1 Role of the Scheduler

The scheduler decides when the CPU can act upon a process. The scheduler uses messages, semaphores, or shared memory to determine if a process is ready to run, is already running, or should be blocked.

To illustrate the scheduler's function, sometimes a process needs exclusive access to a system resource (such as writing to a tape drive or sending output to a printer). The system call to the device indicates to the scheduler whether a user's program can or cannot gain exclusive access to the requisite resource. If the process has exclusive access, the scheduler frees the resource by blocking any process currently using it. If the process cannot have exclusive access, the scheduler blocks the process until the resource becomes available.

Messages, shared memory, and semaphores are components of the interprocess communication package provided by the UNIX System V operating system, as well as by the ULTRIX operating system.

1.6.2 Messages

Messages allow processes to send formatted data packets to other processes. They are a construct that you cannot access directly but that are implemented at the program level to allow processes to communicate with each other. Buffers used by the message system are maintained in the kernel's address space. The standard ULTRIX configuration file contains no parameters that affect messages.

1.6.3 Shared Memory

Shared memory enables processes to access blocks of memory (data segments) outside their normal address space and to share these data segments with other processes. For example, they can be used as a fast way to pass data back and forth between processes accessing a common database.

In the DECwindows environment, shared memory is used by processes running in different windows to communicate with the server.

The ULTRIX configuration file contains five tunable parameters that affect shared memory. For example, the size of shared memory that can be accessed, and the number of shared memory segments that can be attached, per process, can be changed when the kernel is configured.

1.6.4 Semaphores

A semaphore is a data structure that allows processes to synchronize execution, obtain exclusive access to a device, or coordinate some other activity between separate processes by doing a set of operations atomically on a set of event indicators. A semaphore, like messages and shared memory, is a construct that is not directly accessible to you; however, it is used by the software developer when a program is written.

A semaphore contains a table in which entries describe all instances of its use. Each entry has a numeric key by which it is referenced, a data structure that maintains status information, a permissions structure, the state of the semaphore, and other information.

No parameters exist with which to tune semaphore operations in ULTRIX.

1.6.5 The Effect of Interprocess Communications on Performance

The maximum number of shared memory segments and the maximum size of a shared memory segment are parameters that are set in the configuration file during system configuration. The ULTRIX operating system documents no other tunable parameters that affect messages and semaphores.

2 Tools for Monitoring Subsystem Resource Usage

One of the most important assets that a system manager brings to a performance evaluation is an understanding of the normal workload and behavior of the system. Each system manager must understand the system's workload sufficiently to be able to recognize normal and abnormal system behavior, to predict the effects of changes in application, operation or system usage, and to recognize typical throughput rates.

If you are a system manager, it is recommended that you spend some time using the tools available for examining and monitoring your system. Over time, you will learn the typical page fault rate for your system, the typical CPU and memory usage, and so on. You will see how certain applications or the number of users affect these values. As you continue to monitor your system, you will recognize the range of values that is acceptable for your system and to identify trends that indicate some part of your system is nearing capacity.

Table 2–1 lists the commands available on ULTRIX and the subsystems that they monitor.

The following utilities are available as layered products:

- `xnfsstsat` (Digital Network Tools/Management Station for ULTRIX)
- `dxpresto` and `presto` (Prestoserve)

This chapter describes most of these commands and provides examples of their output. The public domain utilities `top` and `snooper` are described, though not in detail.

Table 2–1: Commands for Monitoring the Subsystems

Utility	System Status	File System	Process Control & Scheduling	Memory Management	I/O	Network	Interprocess Communications
at			X				
bg			X				
chpt		X					
cpustat	X						
crash	X						
df	X	X					
devstat					X		
du		X					
fg			X				
fsck		X					
ifconfig						X	
iostat	X	X					
ipcs							X
jobs			X				
kill			X				
mount		X					
netstat	X					X	

Table 2–1: (Continued) Commands for Monitoring the Subsystems

Utility	System Status	File System	Process Control & Scheduling	Memory Management	I/O	Network	Interprocess Communications
newfs		X					
nfsstat	X					X	
nice			X				
ping						X	
ps			X				
pstat	X	X	X		X		
quot		X					
renice			X				
ruptime						X	
rwho						X	
stop			X				
suspend			X				
tunefs		X					
uptime			X				
vmstat	X	X	X	X			
w			X				
who			X				

2.1 Tools That Provide System Status

The commands described in the following sections help you to determine the status of the components of your system, such as your system's core image (also known as the kernel); the current state of your CPU, I/O throughput, and memory; and the state of the network to which your system is attached.

2.1.1 The `crash` Utility

The `crash` utility is an interactive program that lets you examine the core image of the operating system. This utility has facilities that interpret and format the various control structures in the system and certain miscellaneous functions that are useful when examining a dump.

2.1.2 The `cpustat` Command

The `cpustat` command provides a snapshot of how well the CPU is utilizing its time. For example, the `cpustat` command reports the percentage of time a CPU is in user or system mode, the currently running process, and the CPU's state. By default, the `cpustat` command reports a summary of the statistics since the system has been booted and the state of each CPU. You can use this information to determine if the CPU is contributing to a performance problem.

The following example shows the output of the `cpustat` command:

```
csh> cpustat
cpu  us%  ni%  sy%  id%    csw    sys    trap   intr   ipi  ttyin  ttyout
 0  33.4  0.0 66.6  0.0    55k    56k     3      1k     0      0    514
cpu      state      ipi-mask   proc   pid
 0        BR                     Y   1908
```

2.1.3 The `iostat` Command

The `iostat` (Input/Output Statistics) command reports on disk and terminal activity. It displays the following information:

- Number of characters sent to and received from all terminals
- Number of blocks written or read per second
- Number of transfers per second
- Number of 512 KB blocks transferred per second
- Percentages of CPU time used in the following:

  a. User mode for normal user processes
  b. User mode running `nice` (lower priority) processes
  c. System mode
  d. Idling (or in the idle loop)

To provide this information for each disk, the system counts the number of seeks, data transfer completions, and the number of words. It also counts the number of input and output characters for all terminals collectively. Every sixtieth of a second, it monitors each disk to see whether it is active. With the transfer rates for each device and the gathered data, the system determines the average seek times for each device.

When you run the `iostat` command without arguments as in the following example, the statistics reported are averages since the system was last booted. An optional argument indicating a sample time interval tells `iostat` to sample continuously.

The following shows the output of the `iostat` command:

```
csh> iostat
     rz0       rz2       rz4       rz5         cpu
 bps tps   bps tps   bps tps   bps tps   us ni sy id
   2   0     4   1     0   0     2   0    2  0  2 96
```

2.1.4 The `netstat` Command

The `netstat` command (Network Status) shows the contents of the network-related data structures symbolically. Three different output formats are available, depending on the options that are to display the data. The first format (chosen with the `-a` option) displays all active sockets for each protocol. The second format (chosen with a combination of the `-h`, `-i`, `-m`, `-n`, `-r`, and `-s` options) displays the contents of other data structures (as chosen with one or more of the options). The third form (chosen with the `-n` option and a time interval) continuously displays network addresses numerically.

With no options specified, `netstat` displays the current state of all active sockets using any protocols listed in `/etc/protocols`. It also shows local and foreign (remote) addresses, send and receive queue sizes (in bytes), the

protocol, and internal state of the protocol for all active sockets. The following shows the output of the `netstat` command with no options specified:

```
csh> netstat
Active Internet connections
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
tcp        0      0  ronyon.login           ronyon.1018            ESTABLISHED
tcp        0      0  ronyon.1018            ronyon.login           ESTABLISHED
tcp        0      0  ronyon.6000            ronyon.1207            ESTABLISHED
tcp        0      0  ronyon.1207            ronyon.6000            ESTABLISHED
 .
 .
 .
tcp        0      0  ronyon.800             ronyon.819             CLOSE_WAIT
tcp        0      0  ronyon.800             ronyon.821             ESTABLISHED
tcp        0      0  ronyon.821             ronyon.800             ESTABLISHED
udp        0      0  ronyon.elcsd           *.*
udp        0      0  localhost.ntp          *.*
udp        0      0  ronyon.ntp             *.*
Active UNIX domain sockets
Type   Recv-Q Send-Q    Inode     Conn     Refs  Nextref Addr
stream      0      0 80222164        0        0        0 /tmp/.X11-unix/X0
stream      0      0        0 c2013b80        0        0
stream      0      0        0 c2013c80        0        0
stream      0      0        0 c20135c0        0        0
stream      0      0        0 c2013b00        0        0
.
.
.
```

The address formats are either in the form *host.port* or *network.port*. The form *host.port* is used when the host address is known and is displayed symbolically from the database `/etc/hosts`. The form *network.port* is used when the socket's address specifies a network but not a specific host address. The symbolic representations come from the database `/etc/networks`.

The interface option provides cumulative statistics about the packets transferred, errors, and collisions. When an interval is specified, `netstat` provides a running count of statistics related to the network interfaces.

The `-i` option displays statistics on each active or open network interface. Outgoing packet errors (`Oerrs`) indicate that the local host may have a problem. Incoming errors (`Ierrs`) indicate a potential problem with the

network connected to the interface. The following example shows a normal display (not many `Ierrs` or `Oerrs`) from the `netstat -i` command:

```
csh> netstat -i
Name  Mtu  Network     Address            Ipkts  Ierrs     Opkts Oerrs     Coll
qe0   1500 DECnet      RTR              8324125      0   8347463     2   237706
qe0   1500 16.31.16    RTR.xxx.corp.co  8324125      0   8347463     2   237706
dmv0  1284 16.10.16    xx2nnn.pa.corp. 10746232      0   9569078    95       62
dmv1  1284 16.10.16    xx2yyy.pa.corp.c 4880317      0   4570916     0        0
dmv2  1284 16.10.16    RTR2zso          1368208      0   1729512     0       31
lo0   1536 loop        localhost         909234      0    909234     0        0
idl0  549  ptopnet     128.45.10.131          0      0        15     0        0
idl1* 549  none        none                   0      0         0     0        0
idl2  549  none        none                   0      0         0     0        0
idl3  549  none        none                   0      0         0     0        0
idl4  549  128.45.110 128.45.110.111         0      0        15     0        0
```

2.1.5 The `nfsstsat` Command

The `nfsstsat` command (Network File System Statistics) provides statistical data about the Network File System (NFS) and the Remote Procedure Call (RPC) interfaces in the kernel. If you do not specify any options, `nfsstsat` displays the information as if all options were specified, except `-z`. After booting the system, the statistics are reinitialized to zero.

You can view data about the client and server, NFS and RPC, and core and kernel images. The following example shows the output from the `nfsstsat` command. NFS was not active on the system on which this command was run, so all the data fields shown in the example are zero.

```
csh> /etc/nfsstat
Server rpc:
calls      badcalls   nullrecv   badlen     xdrcall
261        0          0          0          0

Server nfs:
calls      badcalls
261        0
null       getattr    setattr    root       lookup     readlink   read
39 14%     99  37%    0  0% %    0  0%      23  8%     0  0%      8  3%
wrcache    write      create     remove     rename     link       symlink
0  0%      0  0%      0  0%      0  0%      0  0%      0  0%      0  0%
mkdir      rmdir      readdir    fsstat
0  0%      0  0%      4  1%      87 33%

Client rpc:
calls      badcalls   retrans    badxid     timeout    wait       newcred
7664       3          9          0          12         0          0

Client nfs:
calls      badcalls   nclget     nclsleep
7622       0          7664       0
null       getattr    setattr    root       lookup     readlink   read
0  0%      3779 49%   213  2%    0  0%      1961 25%   0  0%      774 10%
wrcache    write      create     remove     rename     link       symlink
0  0%      282  3%    59  0%     297  3%    0  0%      107  1%    0  0%
mkdir      rmdir      readdir    fsstat
0  0%      0  0%      121  1%    29  0%
```

2.1.6 The pstat Command

The pstat command (Print System Statistics) provides information about the contents of specific system tables. If you want to examine the system tables of a system other than the running kernel and the tables in /dev/kmem (for example, the kernel and core file from a crash dump), you must use the -k option when specifying a namelist and corefile.

The following options print system tables or information about swap space usage:

-p Prints the process table for active processes

-s Prints information about swap space usage

-t Prints the table for terminals

-T Displays the number of used and total slots in several system tables

The -T option is useful for checking to see how full the system tables have become when the system is heavily loaded.

You can control the sizes of some of these tables. If the number of used slots is almost the same as total slots when the system is heavily loaded, increase the limit and rebuild the kernel. To do this, you must use the system configuration parameters maxusers and maxuprc. See Chapter 3 for more information on these parameters.

Output from pstat -s is discussed in Section 2.2.4.

2.1.7 The vmstat Command

The vmstat (Virtual Memory Statistics) command monitors paging, swapping, and memory activity, including the state of each processor (in multiple processor systems), virtual memory, paging and swapping activity and the utilization of the CPU. The vmstat command tells you how much memory is available, whether the system is paging, and how CPU time is allocated.

The vmstat command provides the following data:

- Number of runnable, blocked, and short-sleep processes
- Virtual memory subsystem statistics
- Disk activity (not only paging)
- Page fault averages
- Interrupts per second
- System calls per second
- Context switches per second
- Percentages of user time, system time, and idle time

The `vmstat` command is the most informative monitoring tool available among the standard tools distributed with ULTRIX. The following example shows the output of the `vmstat` command:

```
csh> vmstat
procs      faults    cpu      memory              page            disk
r b w   in  sy  cs us sy id  avm  fre  re at  pi  po  fr  de  sr s0 s1
0 0 0 289  74  23  3  2 96  26k 5440   0  0   0   0   0   0   0  0  0
```

The following table describes the fields shown in the previous example:

Field	Description	
`procs`	Shows the number of processes in various states:	
	`r`	In the run queue
	`b`	Blocked for resources (paging and I/O)
	`w`	Runnable or short sleeper, but swapped
`faults`	Shows the trap and interrupt rate averages:	
	`in`	(non clock) Device interrupts per second
	`sy`	System calls per second
	`cs`	CPU context switch rate (switches/second)
`cpu`	Shows the percentage breakdown of CPU usage:	
	`us`	User time for normal and low priority processes
	`sy`	System time
	`id`	CPU idle
`memory`	Shows information about current usage of virtual and real memory:	
	`avm`	Active virtual pages
	`fre`	Size of the free list
`page`	Shows information about page faults and paging activity:	
	`re`	Page reclaims
	`at`	Pages attached
	`pi`	Pages paged in
	`po`	Pages paged out
	`fr`	Pages freed per second
	`de`	Anticipated short term memory shortfall
	`sr`	Pages scanned by clock algorithm per second

Field	Description	
disk	Shows the number of disk operations per second:	
	s0, s1...sn	Paging/swapping disk sector transfers per second

2.2 Tools That Monitor the File System

A number of useful tools exist with which you or the system administrator can monitor or improve file system performance. Some of the tools show information about how users utilize the file system. This information is useful to system administrators who want to change user behavior to improve system performance.

Other tools run on a live file system and show current usage information directly relevant to tuning file system parameters. You can use this information to tune the file system.

2.2.1 Gathering Information About Disk Organization

Before you begin to tune the file system, it is necessary to understand its configuration; that is, it is necessary to know where you are before you can decide where you want to go. Using the `chpt`, `df`, and `mount` commands together can give you a clear picture of the configuration of the disk.

Using the `chpt`, `df`, and `mount` commands, you can accomplish the following:

- Learn the geometry of the disks attached to the system
- Determine which partitions on the disk are in use
- Determine the size of file systems on those partitions
- Determine the amount of space in the file systems currently in use
- Determine the amount of free space available for new files or for the expansion of existing files
- Determine where in the directory hierarchy the various file systems are mounted

This information is especially useful if you want to modify the file system to improve performance. The user must also know the swap and dump devices.

The following sections describe these utilities, their output, and how to use the resulting information to understand the physical layout of the disks on which the file system resides.

For more information, see `chpt`(8), `mount`(8), and `df`(1) in the *ULTRIX Reference Pages*.

2.2.1.1 Using chpt

The chpt (change partition table) command reads the partition table from the superblock of the a- or c- partition, if it exists, or the default partition table from the device driver, if it does not. You can run the chpt command only if you have superuser privileges. This precautionary measure is necessary because it is possible to change the partition table in the superblock, thereby destroying the file systems on the disk. However, with the argument -q, chpt runs without modifying the partition table and simply prints its contents to standard output (stdout). The following example shows an example of the command and its output. Note that you must use either the a or c partition of the raw device with this command.

Example 2–1: The chpt -q Command

```
# chpt -q /dev/rrz0a

/dev/rrz0a
Current partition table:
partition     bottom        top        size     overlap
    a              0      32767       32768     c
    b          32768     163839      131072     c
    c              0    1299173     1299174     a,b,d,e,f,g,h
    d         163840     456369      292530     c,g
    e         456370     748899      292530     c,g,h
    f         748900    1299173      550274     c,h
    g         163840     731505      567666     c,d,e
    h         731506    1299173      567668     c,e,f
```

In Example 2–1, the column labeled partition is typical of disks manufactured by Digital; that is, all have seven partitions (some may not be used, such as here, in the d–partition). The columns labeled bottom and top refer to the beginning and ending sector numbers. The column labeled size shows the size of the partition in sectors. The column labeled overlap shows the partitions that share some of the same sectors.

2.2.1.2 Using df

The df command displays information about mounted file systems. The df command lists the partition (corresponding to the output from chpt -q) and displays the total size of the partition (in kilobytes), the kilobytes used, kilobytes free, percent used, and the mount point of the partition all are displayed to standard output. The following example shows the df command and its output:

```
csh> df
Filesystem  Total   kbytes   kbytes %
node        kbytes  used     free   used   Mounted on
/dev/rd1a     7407   5089     1578  76%    /
/dev/rd1g   122598  88306    22033  80%    /usr
/dev/rd0g    41639  26848    10628  72%    /usr/tools
```

2.2.1.3 Using `mount`

You can use the `mount` command to display the currently mounted file systems. The command (with no arguments) shows the disk partitions with file systems mounted and their mount points in the root file system. Any user can run the command, but superuser privileges are required to mount a file system. The following example shows the `mount`command and its output:

```
csh> mount
/dev/rz0a on / type ufs
/dev/rz2h on /usr type ufs
/dev/rz0d on /var type ufs
/dev/rz2g on /usr/users type ufs
rag110:/usr/projects/tools on /usr/tools type nfs (rw,hard,bg,intr)
```

2.2.2 Creating, Checking, and Tuning a File System

As part of improving file system performance, you might have to create a new file system on one of the disk partitions as shown in Example 2–1. In rare instances, you might even have to redefine a partition's starting sector and size before creating a new file system on it. Generally, it is not a good idea to change partition sizes from those found in the device driver's default partition table; it makes the file system (disk) hard to maintain. However, with some small disks, such as the RD53 used in some of the examples in this manual, or disks with a strange geometry, the benefits of changing partition sizes might outweigh the liabilities.

In addition, you might need to check the integrity of a file system, although the system normally does this automatically when it is booted. You might need to change some of the tunable parameters associated with a mounted file system. The commands `newfs`, `chpt`, `fsck`, and `tunefs` accomplish these tasks.

For more information, see `newfs`(8), `chpt`(8), `fsck`(8), and `tunefs`(8) in the *ULTRIX Reference Pages*.

2.2.2.1 Using `newfs` and `tunefs` to Create and Evaluate a File System

The `newfs` command is an ULTRIX interface to the UNIX utility `mkfs`. It takes as arguments the raw partition (character special device file) on which the file system is to be created and the disk type. The following example shows the`newfs` command and its output:

Example 2–2: The `newfs` Command

```
# newfs /dev/rrd0g RD53
Warning: 16 sector(s) in last cylinder unallocated
/dev/rrd0b: 33440sectors in 246 cylinders of 8 tracks,
17 sectors 17.1Mb in 16 cylgroups (16 c/g, 1.11Mb/g,
384 i/g) super-block backups (for fsck -b#)at:
32, 2232, 4432, 6632, 8832, 11032, 13232, 15432,17440,
19640, 21840, 24040, 26240, 28440, 30640, 32840,
```

The `newfs` command prints information about the created file system, including its size in sectors, cylinder groups, and megabytes. A backup

superblock is always located at block number 32, but the number and location of other backup superblocks depend on the size of the disk.

The `newfs` command reads necessary information about the disk's geometry from the file `/etc/disktab`. This information is passed on to `mkfs`, which creates the new file system.

You can set a number of parameters governing the file system's configuration, overriding the defaults normally passed to `mkfs`. You can change a limited number of these parameters later using `tunefs` , if you need to tune the file system. For example, you can adjust `rotdelay` and `maxcontig=1` with `tunefs`.

As a general rule, leave `maxcontig` equal to 1, and fix `rotdelay`. The disk controller used on the system affects the settings for best performance.

Use `dumpfs` to view the setting of `rotdelay`. The default for RISC computers is 0, and the default for VAX computers is 4. The default setting is correct for all VAX and RISC SCSI systems. The non-SCSI RISC optimal settings is 4 for RA drives. Note that ULTRIX Version 4.0 uses 0 and Version 4.1 and higher use 4. For DSSI drives, the default is 0 for RF71s and 4 for RF31s.

2.2.2.2 Using `fsck` to Evaluate a File System

The `fsck` command checks and repairs a file system. Normally, the system executes this command when it is first booted as part of the normal boot sequence. It performs a number of maintenance chores including the checking of blocks claimed by inodes and the free list, link counts, directory size and format, total free blocks, and free inodes. If it detects allocated but unreferenced files, it places them in the lost+found directory, if it exists. The following example shows the `fsck`command and its output:

For more information, see `fsck`(8) in the *ULTRIX Reference Pages*.

```
csh> fsck /dev/rz1f
** /dev/rz1f
** Last Mounted on /usr/users3
** Phase 1 -Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Cyl groups
3 files, 10 used, 263413 free (21 frags,32924 blocks, 0.0% fragmentation)
```

2.2.3 Disk Usage Space Allocation

One of the factors that affects the performance of the file system is the percentage of the total number of blocks in the file system that are in use. When the file system is created, a free-space threshold is specified. Usually it is 10 percent of the total number of kilobytes in the file system. It is possible to change this percentage, using the `-m` argument with the `newfs` command, at the time the file system is created, or later using the `-m` option with the `tunefs` command. However, setting the minimum free-space threshold to zero decreases throughput by a factor of as much as three. The percentage of kilobytes used, reported by the `df` command, refers to the total capacity of the file system, less the free-space threshold.

Two commands, `du` and `quot` , are useful when determining how the file system is currently used. The `du` command displays the number of blocks in each directory of the file system. Higher-level directories in the hierarchy show the total number of blocks occupied in all files and subdirectories below them. The `quot` command summarizes file system ownership for names listed in the password file and displays total blocks owned by file system and user name. The following examples show the `du` and `quot` commands and their output:

```
csh> du /usr/tools/aim
290  /usr/tools/aim/bin
6    /usr/tools/aim/h
58   /usr/tools/aim/src/utils
366  /usr/tools/aim/src
1029 /usr/tools/aim/suite2
25   /usr/tools/aim/suite2.data
1    /usr/tools/aim/tmp
1945 /usr/tools/aim

csh> quot /dev/rdla
/dev/rdla:
31470  bin
10892  root
1958   uucp
818    sysinfo
428    lp
4      network
```

By using `du` and `quot`, system administrators can determine the most populated directories and identify the users whose files consume the largest number of disk blocks. This information can help formulate a strategy to spread disk activity out across multiple spindles.

In addition, system administrators can use `quot` in conjunction with `quotaon` and `quotaoff` to establish and monitor disk quotas. The objective of these efforts is to improve disk performance. Note that the argument to `du` is the directory where the tabulation is to start, whereas the argument to `quot` is the device-special file for the partition on which the file system resides.

For more information, see `du`(1) and `quot`(8) in the *ULTRIX Reference Pages*.

2.2.4 Swap Space Usage

Configuring the system with enough swap space is primarily an administrative chore. Monitoring swap space usage with the `pstat -s` command shows how the system is using its existing swap space. The following example shows the `pstat` command with the `-s` option and its output:

```
csh> pstat -s
33440k swap configured
         10917k used (2981k text, 0k smem)

         22521k free, 2720k wasted, 2k missing
avail: 9*2048k1*1024k 2*512k 5*256k 2*128k 4*64k 4*32k 3*16k 73*1k
```

The `pstat -s` command shows the number of 1 KB pages used, free, wasted, and missing. The missing field includes space allocated for storing arguments to a currently executing process. It also shows space allocated to alternate swap partitions when the kernel was configured if the partition does not appear in the `/etc/fstab` file. The wasted field indicates the degree of fragmentation with a larger value indicating more fragmentation. In addition, the available field shows the number of free contiguous blocks.

You can determine how much swap space is used or free on a percentage basis by dividing the number in the used field by the swap configured field. In the previous example, this is 10917k÷33440k, or 32.7 percent used. By looking at the available field and noting the size of the available blocks, you can determine the degree of fragmentation of the swap space. With a greater number of large contiguous blocks available, the swap partition is less fragmented.

2.2.5 Exercising the Disk and File System

ULTRIX provides several system exercisers you can use to identify disk and file system problems. The exercisers `dskx` and `fsx` are relevant to the file system. All of the exercisers help identify hardware problems and their respective subsystems. Make sure that performance problems are not hardware related before analyzing each subsystem further.

Each exerciser can be run for a particular length of time (command line option) or until killed by a signal. The exercisers collect output in a file specified by the user; by default, it creates the file in the `/usr/field` directory. You must have superuser privileges to run the system exercisers.

For more information, see `dskx`(8), `fsx`(8), and other system exercisers in the *ULTRIX Reference Pages*.

2.2.5.1 Using `dskx`

The `dskx` command is the disk drive exerciser. It performs random seeks and reads and writes to an entire disk or to specified partitions on the disk. Writes are potentially destructive; if `dskx` detects a valid file system on the disk, it prompts you twice for confirmation before overwriting it.

2.2.5.2 Using `fsx`

The `fsx` command is the file system exerciser. It works by forking a specified number of processes determined by a command line option. The default number is 20. Each process creates, opens, reads and writes files of random data.

2.2.6 Monitoring File System Activity

The commands mentioned in the previous sections give an overall picture of how the file system is set up, who is using it, or how the physical components are functioning. However, this information does not reflect the system while it runs a normal workload. The following sections discuss two ULTRIX commands, `iostat` and `vmstat`, that you can use to understand how the system is working under a normal load. These commands allow you to periodically sample various system tables in the running kernel to provide statistics about the file system's performance.

You can use the `iostat` command to monitor file system activity. A small number of bytes transferred per seek indicates a highly fragmented file system. Such a file system can benefit from reorganization. In addition, the file system is fragmented if the bytes transferred per second divided by the transfers per second (tps) is much less than the disk's transfer rate.

The output from the `vmstat` command contains information related to paging and swapping. The fields `re`, `at`, and `sr` (page reclaims, ages attached, and pages scanned by clock algorithm). The fields `pi` and `po` (pages paged in and pages paged out) relate to activity on the swap partitions.

For more information, see `iostat`(1) and `vmstat`(1) in the *ULTRIX Reference Pages*.

2.3 Commands that Monitor Process Control and Scheduling

The commands that you or the system administrator can use to monitor or improve process control and scheduling are limited. Some of the commands (`finger`, `w`, `who`, `uptime`) show information about who is using the system and what they are doing.

Two of the commands (`ps` and `pstat`) show current process control and scheduling information for all processes, systems services, and users. Unfortunately, neither of these commands runs in a continuous sampling mode similar to `iostat` and `vmstat`. Information provided by these commands is useful for system management rather than system tuning.

2.3.1 Determining the System Users and Tasks They Are Running

To determine who is using the system, you can use several utilities:

- The `who` command prints out the login name, terminal name, and login time for each current user.
- The `finger` command displays information about each user currently connected and a selectable amount of additional information.
- The `uptime` and `w` commands allow you to see who is logged in and what they are doing. The first line of the output gives the uptime for the system (how long since the last boot) and the system's load average.

The following example shows the output from the `who` and `finger` commands:

```
csh> who
jim  tty01 Jun 4 08:45
chris  tty02 Jun 4 09:06
csh> finger
Login     Name      TTY Idle    When      Office
jim   Jim Sumrall    01          Mon 08:45
chris Chris Farrow   02        6 Mon 09:06
```

The following example shows the output of the `w` command. The `uptime` command shows the same information as that in the first line of the output from the `w` command. The last field (the `what` field) shows the command being executed by the user.

```
csh> w
1:42pm up 3 days, 33 mins, 3 users, load average: 6.19, 2.42, 1.18
User  tty    login@  idle   JCPU  PCPU   what
root  ttyp1  1:13pm     3 278:25  4:36   ../cc1 jump.i -quiet -O -o jump.
root  ttyp2  1:58pm     2 288:57 94:01   sh -c exec mul_double/usr/tools
root  ttyp0  1:12pm 71:44     24    16   /usr/bin/dxwm
```

For more information on these commands, see the *ULTRIX Reference Pages*.

2.3.2 Setting or Resetting the Priority of a Process

When ULTRIX executes commands, the scheduler decides to run a particular process based on a number of factors, one of which is the process priority. All processes have a base priority that is determined when they begin to execute.

Some processes, such as the `swapper`, `pagedaemon`, `init`, and other system level services, have high base priorities, whereas user processes start with a lower base priority. This priority is called a process' nice value. A high priority translates into a low `nice` value. Changing process priorities with `nice` or `renice` can improve system performance, but this option falls into the category of system management rather than system tuning.

For more information, see `nice`(1) and `renice`(8) in the *ULTRIX Reference Pages*.

2.3.2.1 Using `nice`

When you enter a command, you can increase its `nice` value, that is, you can lower its priority. The `nice` command varies if you are using the C-shell or the Bourne shell. The semantics of the command in C-shell refers to its `nice` value, while in Bourne shell, it refers to its priority. For example, the argument (10) to the `nice` command changes the priority by 10 percent. You can lower the priority of your own process by a `nice` value of as much as 20 (the upper limit). You must have superuser privileges to increase the priority of a process.

2.3.2.2 Using `renice`

ULTRIX also allows you to reset the priority of a running process using the `renice` command. The format of the command and its semantics are the same as that for the C-shell `nice` command. Unlike the `nice` command, the `renice` command resides in the `/etc` directory.

You can change the priorities of your own processes only by increasing the `nice` value. If you have superuser privileges, you can change the priority of any process and can increase or decrease the priority.

You can determine the process ID by using the `ps` command. For more information, see Section 2.3.4.2.

2.3.3 Scheduling, Rescheduling, or Stopping a Process

You can use several commands to schedule or reschedule a process. The `at` command allows you to schedule a process to run at a later time. To use the `at` command, your user ID must appear in the file`/usr/lib/cron/at.allow` or it must not appear in the file `/usr/lib/cron/at.deny`. The system clock daemon, `cron`, runs the process at the time specified by the `at` command.

The `at` command uses a copy of a named file or standard input as input to `sh` or `csh`. The command `at` allows you to specify the time when you want the command to run. The following example shows using the `at` command to schedule the command file `arc` to run at 10:00 PM.:

```
csh> at 2000 arc
90.154.2000.15
```

The system breaks the output from the commands into four fields separated by periods. In order, the fields show the year, the day from the Julian calendar, the time at which the system runs the job, and the job's queue number.

The C-shell provides several built-in commands that allow some flexibility in process scheduling. These commands are `bg`, `fg` , `jobs`, `kill`, `stop`, and `suspend`. With respect to job control, the C-shell gives you greater control over processes and job scheduling than the Bourne shell.

The `bg` command places a process in the background. The `fg` command brings a process to the foreground. The `jobs` command displays active jobs. The `kill` command sends a signal to a job. The `stop` command and the `suspend` command stops or suspends a job, which you can restart with the `bg` or `fg` commands. You can suspend a process by pressing Ctrl/Z.

The `kill` command (on `/bin`) allows you to stop a currently running process. To kill a process, you must have the same user ID or process group ID, or have superuser privileges.

The `kill` command allows you to send a signal to a running process. For more information about the signals you can send, see `sigvec`(2) in the *ULTRIX Reference Pages*. If the executing process is designed to catch signals, it takes whatever action is prescribed on receipt of the quit, terminate, or interrupt signals. Other signals usually are not used; they are reserved for system-specific exception conditions.

If the objective is to terminate the process, whether it is catching signals or not, you can send the signal –9; it is guaranteed to stop the process. Without

superuser privileges, you can send a signal only to your own processes. With superuser privileges, you can send a signal to any process. A system administrator might kill a process that appears to be endlessly looping, or one that is consuming an inordinate number of system resources, and could then reschedule the job at a time when the system was less heavily loaded.

For more information, see at(1), kill(1), and the built-in commands (bg(1), fg(1), jobs(1), kill(1), stop(1), and suspend(1)) of the C-shell in the *ULTRIX Reference Pages*.

2.3.4 Monitoring Interrupts, Context Switches, and System Calls

The commands mentioned in previous sections primarily dealt with process scheduling. Specifically, these commands relate to system management and might have some impact on system performance. The vmstat command can monitor the system as it is running and give some useful information relating to process control and scheduling.

2.3.4.1 Using vmstat

The information from the vmstat command relevant to process control and scheduling is found in the faults field. Section 2.1.7 provides a description of the of this command. The following example shows the first three fields of the vmstat command:

```
csh> vmstat
procs       faults    cpu
r b w   in   sy  cs  us sy id ...
1 0 0   75  101  21  23 25 52 ...
```

The relevant fields, in, sy and cs, by themselves have little meaning. You must compare these fields for a system when it is not heavily loaded and when it is processing several different workloads. If any of the three faults fields are large when the system is heavily loaded and when the id field under the cpu heading is also large (showing the percentage of time the CPU is idle), this indicates some problem. See Chapter 3 for more information about the type of problem and its remedy.

For more information, see vmstat(8) in the *ULTRIX Reference Pages*.

2.3.4.2 Monitoring Other Process Control Activity

The remaining tools you can use to monitor process control activity are ps and pstat. Both of these tools give a snapshot when you run them. The output from ps and pstat are similar. However, the quantity of information available from pstat is greater, as it covers more subsystems than process control and scheduling.

The ps command prints process status statistics. With no options specified, the command prints information for the user's processes only. With the -a option specified, the system prints all user process statistics. With the -x option, the system prints all process statistics, including those for processes such as printer and network daemons, system utilities, and other background tasks. With the -l option, the system displays all information collected by ps. The following example shows the edited output from the ps command with the -al options:

```
csh> ps -al
      F UID   PID PPID CP PRI NI ADDR  SZ RSS WCHAN STAT TT   TIME COMMAND
b008001   0   135 131   0   1  0 3f54 614  78 c7060 I    p0   0:08 /usr/bin/dx
b008001   0   136 131  26   1  0 512a 880 122 c7060 I    p1   4:03 /usr/bin/dx
b008001   0   137 136   3   3  0 2d8c  65  33 e08cc I    p1   0:12 - (csh)
b008001   0   607 131   2   1  0 46ca 842 151 c7060 S    p21 26:49 /usr/bin/dx
b008201   0   608 607   3  15  0 486a  63  44 fe400 I    p2   0:24 - (csh)
b000001   0 7778 608 101  50  0 6d5c 231 192       R    p2   0:05 s5make mult
b808201 205 6683 6682  5  15  0 255a  54  35 fe400 S    p3   0:04 -csh (csh)
b000001 205 7880 6683 76  44  0 1b78 254 140       R    p3   0:00 ps -al
b000001 205 7881 6683  5   1  0 53d2   6   3 5e358 S    p3   0:00 tee ps.out
```

You can obtain the process ID field (PID) (mentioned earlier relating to the `kill` command) from the default listing. The system prints the process priority (PRI) and `nice` value (NI), which also were discussed earlier. The state of the process is of interest. Table 2—2 shows the process status flag.

Table 2—2: Process Status Flags

Flag	Description
R	Running Process
T	Stopped processes
P	Processes in page wait
D	Processes in disk (or othershort-term) waits
S	Processes sleeping for less than 20 seconds
I	Idle processes (sleeping longer than 20 seconds)
W	Processes that are swapped out

The system prints other fields of interest for different options. The `-au` option displays user-oriented output. The following example shows the output from `ps` command with the `-au` options:

```
csh> ps -au
USER  PID %CPU  %MEM   SZ  RSS  TT  STAT  TIME COMMAND
root 8071 14.3   0.5   93   50  p2  R     0:10 multiuser
root 8057 13.5  16.6 2257 1848  p1  R     0:22 ../cc1 jump.i -quiet -O -o jum
root 8070 13.4   0.5   93   50  p2  R     0:11 multiuser
root 8069 13.4   0.5   93   50  p2  R     0:10 multiuser
root 8068 13.4   0.5   93   48  p2  R     0:10 multiuser
root 8067 13.2   0.5   93   48  p2  R     0:09 multiuser
root 607   1.1   2.3 2323  237  p2  S   130:46 /usr/bin/dxterm -ls
root 8061  0.1   0.3   93   28  p2  I     0:02 multiuser
root 8027  0.0   1.3  185  136  p1  I     0:08 make -f Makefile
root 8059  0.0   0.4   78   36  p2  I     0:00 sh5 aim.control
```

Two fields output by the `ps -au` command are `%CPU` and `%MEM`. The `%CPU` field shows the CPU utilization of the process. The `%MEM` field shows the percentage of real memory used by the process. These fields indicate processes that consume an inordinately large amount of either resource.

The sum of the CPU percentages can total more than 100 percent because the computation uses a decaying average over one minute. If many of the processes are young, the total could reach a maximum of 200 percent.

The `pstat` command prints out the contents of some kernel system tables. For the purposes of monitoring process control and scheduling, the most useful information that the `pstat` command displays is the number of used and free slots in several system tables. The following example shows the output from the `pstat` command with the `-T` option:

```
csh> /etc/pstat -T
182/ 609       files
431/ 444       gnodes
 68/ 288       processes
 38/ 38/  76   active/reclaimable/total texts
  2/1843       00k swap
```

You can configure the size of these tables. If the output from this command reveals that one of these system tables (such as the process table) is at or near capacity when the system is heavily loaded, reconfigure the kernel, placing larger limits on that table.

For more information, `ps`(1) and `pstat`(8) in the *ULTRIX Reference Pages.*

2.4 Tools for Monitoring Memory Management

Few real opportunities exist to tune the memory management subsystem, other than to add more memory. Even the system management policies do not significantly affect memory management. The only tool that monitors memory management on the running system is `vmstat`. In addition, the ULTRIX field service kit provides two memory exercisers.

2.4.1 Memory Exercisers

Previous sections mentioned system exercisers that ULTRIX provides for the file system. The system provides two more that test the memory subsystem, `memx` and `shmx`. These do not test memory management; they test the memory itself. However, you should know about all tools available so that you can use the appropriate one to test the system, eliminating the possibility of any other source of problems.

2.4.1.1 Using `memx`

The memory exerciser `memx` forks 20 (the default) processes that exercise memory by writing and reading a series of 1s and 0s to memory. The `memx` command has the following attributes:

- It tests all of memory by dividing available system memory by the number of processes to be run. The amount of swap space or system memory acts as the upper bound on the amount of memory that is tested; the smaller of the two is the maximum.
- It runs until terminated by the QUIT signal, or it runs for a specified period of time.
- It runs the shared memory exerciser unless that option is disabled.

2.4.1.2 Using `shmx`

The shared memory exerciser `shmx` forks a background process, and the two processes create, attach, and write and read to as many as six (the default) shared memory segments. You can specify the number of shared memory segments that you want to test on the command line when you start the exerciser. The `shmx` command runs until terminated by the QUIT signal, or it runs for a specified period of time. The size of shared memory segments tested is SMMAX (set in the configuration file) divided by the number of shared memory segments to be created.

For more information, see `memx`(8), `shmx`(8), and the other system exercisers in the *ULTRIX Reference Pages.*

2.4.2 Using `vmstat` to Monitor Memory Usage

Using the `vmstat` command, you can check principal memory features such as swap or paging status. From the output of the `vmstat` command, you can decide whether more memory would improve system performance. The following example shows the output from the `vmstat` command:

```
csh> vmstat
procs      faults    cpu       memory              page
r b w   in  sy  cs us sy id  avm  fre  re at  pi  po  fr  de  sr ...
4 1 0  404 193   7  9  8 83  10k  18k   0  0   0   0   0   0   0...
```

This section discusses the fields relevant to memory management.

To determine whether the system is swap bound, examine the fields `avm` (active virtual memory) and `fre` (free list). When the system is heavily loaded, `avm` should be less than 80 percent of real memory, and `fre` should be small. If `avm` is larger and `fre` is small, the system has to write pages to the swap partition, and system performance deteriorates.

If `avm` is too large, given the amount of real memory on the system, the `de` field (anticipated short term memory deficit) becomes nonzero, and the system begins to write pages to the swap partition as indicated by nonzero values in the `pi` and `po` fields. You can increase available real memory either by reducing the buffer cache or by adding more memory.

Positive values in the `fr` (pages freed per second) and `sr` (pages scanned per second) fields can indicate a real memory shortage. If the values in the `fr` and `sr` fields are nonzero, use the command `ps -x` to determine whether the page daemon process is using large amounts of time. If it is, you can improve system performance by adding more memory.

2.5 Tools for Monitoring the I/O Subsystem

ULTRIX provides a limited number of tools to monitor the I/O subsystem. Some of them (`finger`, `w`, and `who`) were described in Section 2.4.1 in relation to process control and scheduling. These commands allow you to monitor who is using the system and the terminal on which they are working. The `iostat` command shows the number of characters sent and received by all the terminal interfaces.

2.5.1 Determining Which Devices are Connected to the System

The kernel collects some information about devices that are configured and connected to the system. Information about their status is usually available. For a limited number of devices, performance statistics are also available. The `pstat` command prints information that relates to terminals attached to the system for each terminal line.

2.5.1.1 Using `devstat`

The `devstat` command gathers some of the information that the kernel maintains on attached I/O devices. In general, this information is built into the device driver for the device. The kernel is configured with this information when it is built. The strategy to obtain this information is as follows:

- To attempt to open the device-special file (if the device is not present, the open fails)
- To read information from the driver using the `stat` and `ioctl` functions

The following example shows the `devstat` command and its output:

```
csh> devstat /dev /console /dev/rmt0h
                                                    ULTRIX    DRIVER
                                          DEVICE    DEVICE    ERR CNT
DEVICE NAME        NAME     INTERFACE    MAJ MIN   PNMC #    SFT HRD
------------       ----     ---------    --- ---   --- ---   --- ---
/dev/console       VR260    VS_SLU       0   0      sm  0     0   0
/dev/rmt0h         TK50     VS_TAPE      46  8      st  0     0   0
```

The `devstat` command does not format its output; the output is piped to an `awk` script, which formats it. More information is available from the kernel, including information about the bus, controller, plug, and status mask, but it was not reproduced here.

See Appendix A for the source code for the `devstat` command.

2.5.1.2 Using `pstat`

The command `pstat -t` examines buffers in the kernel's data space for all the terminal devices configured with the system, regardless of whether a device is attached. By combining information from this command with information from the `w` command, you can determine which terminal lines are active, the processes generating the activity, and the number of characters the application is generating. The following example shows the output of the `pstat` command with the `-t` option:

```
csh> /etc/pstat -t
 # RAW CAN OUT    MODE     ADDR    DEL COL  STATE       PGRP DISC
4 dc lines
 0   0      c98 801c9250  0   0    OC               83
 1   0        0        0  0   0    OC                0
 2   0        0        0  0   0                      0
 3 602 48000003 801c94a8  0  13    OCBA              0

32 pty lines
 0   0 540500d8        0  0  11    OC             3048 ntty
 1           82        0  0  10    OC             3049
 2   0       82        0  0   0    OC             3082
 3   0 5c0500d8        0  0    0   OC             3225 ntty
```

The RAW, CAN, and OUT fields show the number of characters in the raw and canonicalized input queues and the output queue, respectively. The STATE field shows the status of the device, where the characters O, C, D, and W represent open, carrier on, open nodelay, and waiting for an open to complete. For a description of the other fields that the pstat(8) command displays, see the *ULTRIX Reference Pages*.

2.5.2 Exercising Terminals, Printers, and Magnetic Tape

Previous sections mentioned ULTRIX system exercisers in the file system. ULTRIX provides three more that test the I/O subsystem: cmx, lpx , and mtx. The exercisers provided for the I/O subsystem test terminals, printers, and tape drives.

The following sections briefly describe the cmx, lpx, and mtx commands. For more information about these commands and the other system exercisers, see the *ULTRIX Reference Pages*.

2.5.2.1 Using cmx

The cmx utility is the generic communications exerciser. It writes, reads and validates random data of random packet lengths on specified communication lines. The line-under test must have a loopback connector attached to it, either to the distribution panel or to the cable. Disable the line in the /etc/ttys file. Like the other exercisers, the cmx utility runs for a specified length of time or until it receives a QUIT signal.

2.5.2.2 Using lpx

The lpx utility is the line printer exerciser. It prints five pages of a rolling character pattern to the printer, sleeps for fifteen minutes, and then repeats. It continues to do this for a specified length of time or until it receives a QUIT signal. You must specify the device-special file for the printer under test and disable the printer queue for that printer while the exerciser is running.

2.5.2.3 Using `mtx`

The `mtx` utility is the generic magnetic tape exerciser. It writes, reads, and validates a random pattern of data to a specified tape device. You can test four different record lengths. The exerciser runs for a specified length of time, or until it receives a `QUIT` signal.

2.6 Tools for Monitoring the Network

ULTRIX provides a number of tools that monitor network activity. You can use some of these to determine who is using the network and what they are doing. You can use these tools for system management and user education. ULTRIX also provides tools that show what the network is doing at a lower level of abstraction, that is, at the level of message packets sent and received. Use this type of tool to diagnose network problems related to hardware and systems software. The tools described in the following sections apply only to the TCP/IP protocols and NFS.

2.6.1 Determining Network Usage

You can use two commands, `rwho` and `ruptime`, with `ps` to determine who is logged in anywhere on the local area network, the machine on which they are working, and the terminal controlling their session. Use the `ruptime` command to determine which nodes are live and the `rwho` command to determine who is logged on to each live node in the network. If you want to know what a particular user is doing (what processes they are executing), use the `ps` command on their local machine. You can run the `ps` command as a remote job (`rsh` or `rexec`), or you can log in to the remote machine (`rlogin` or `telnet`) and run the `ps` command in a local shell.

2.6.1.1 Using `rwho`

The `rwho` command displays the login name, the node name and login terminal (separated by a colon), and the date and time the user logged in. If the terminal is idle for a minute or more, the system prints the idle time as well. If a terminal is idle for an hour or more, `rwho` does not display an entry unless you specify the `-a` option. The following example the `rwho` command and its output:

```
csh> rwho
chris    Alpha:console    Jun 5 03:34
chris    Opus:ttyp0       Jun 7 09:04
jim      Opus:tty01       Jun 6 09:10
root     tinsel:ttyp1     Jun 1 13:13 :21
root     tinsel:ttyp2     Jun 1 13:58 :04
```

The `rwhod` daemon sends its information in broadcast packets, which generates a large amount of network traffic. On large networks, the extra traffic may be objectionable. Therefore, the `rwhod` daemon is disabled by default. To make use of the `rwhod` daemon for both the local and remote host, remove the comment symbols (#) in front of the lines specifying `rwhod` in the `/etc/rc` file.

2.6.1.2 Using `ruptime`

The `ruptime` command gives a status report in the same format as the `uptime` command (discussed in Section 2.3.1) for each machine in a local area network. Each machine in the network has the system status server, `rwhod`, running; it broadcasts its own status once a minute. All the machines in the network have this process running. It listens for and broadcasts incoming status messages and updates several status files in the `/usr/spool/rwho` directory. The following example shows the `ruptime` command and its output:

```
csh> ruptime
Alpha     up 2+21:37, 1 user,  load 0.32, 0.31, 0.31
Opus      up 2+23:54, 2 users, load 0.03, 0.00, 0.00
tinsel    up 5+20:08, 2 users, load 5.53, 5.44, 4.86
```

If the `rwhod` daemon is not running on a remote machine, the machine may incorrectly appear to be down when you use the `ruptime` command to determine its status. For more information, see `ruptime`(1) in the *ULTRIX Reference Pages*.

2.6.2 Determining the Status of the Network Interfaces

ULTRIX provides two tools, `ifconfig` and `netstat`, to gather information about the status of the interfaces. The `ifconfig` utility is usually run in the system's startup file `rc.local` to mark the network as up. Run interactively, it provides some basic information about the configured interfaces. The `netstat` command shows the network status. See Section 2.1.4 for a general description of the `netstat` command.

2.6.2.1 Using `ifconfig`

The `ifconfig` utility configures the network interfaces for the host machine, marks an interface as up or down, sets the network mask (determines how the network address is interpreted in a Class A or B network), and sets other parameters. You must run `ifconfig` when the system is booted; it is usually part of the system's `rc.local` startup file. When you run the command and specify only the network interface after the network is up, it displays some status information about that interface. Example 2–3 shows an example of the command and its output.

Example 2–3: The `ifconfig` Command

```
csh> /etc/ifconfig se0
se0: 128.10.1.3 netmask ffff0000 flags=0x443<DYNPROTO,RUNNING,BROADCAST,UP
broadcast: 128.10.255.255
```

2.6.2.2 Using `netstat`

The `netstat` command displays the contents of network-related data structures in the running kernel. Using any of the options `-A`, `-a`, or `-n` displays a list of all active sockets specifying the protocol being used and any currently active connections. With any of the options, the `netstat` command displays the contents of data structures according to the options chosen. If you specify no options, the `netstat` command displays the state of all active sockets. The following example shows the `netstat` command and its output:

```
csh> netstat
Active Internet connections
Proto Recv-Q Send-Q  Local Address Foreign Address (state)
tcp        0      0  tinsel.login  Alpha.1022     ESTABLISHED
tcp        0      0  tinsel.704     *.*           LISTEN
udp        0      0  tinsel.elcsd   *.*
Active UNIX domain sockets
Address  Type   Recv-Q Send-Q    Inode    Conn  Refs Nextref Addr
8123dc80 stream      0      0        0 8123d58    0       0
8123dd00 stream      0      0        0 8123d5c    0       0
8123b580 stream      0      0        0 8123d64    0       0
8123ff00 stream      0      0        0 8123d60    0       0
8123cc00 stream      0      0        0 8123d50    0       0
81235e80 stream      0      0        0 8123d54    0       0
81236700 stream      0      0        0 8123d6c    0       0
8123fd80 stream      0      0        0 8123d68    0       0
81238d00 dgram       0      0 80103fd8       0    0       0 /dev/elcscntlsckt
81238c80 stream      0      0 8010409c       0    0       0 /dev/printer
8123cc80 stream      0      0 80105238       0    0       0 /tmp/.X11-unix/X0
```

2.6.3 Displaying Statistics for NFS

The `nfsstsat` command displays information about the Network File System (NFS) and Remote Procedure Call (RPC) interfaces in the kernel. Options to the `nfsstsat` command enable to display the following:

- Client information
- Server information
- NFS information
- RPC information
- Information from either a live system or a crash dump

The following example shows the `nfsstat` command and its output:

```
# /etc/nfsstat
Server rpc:
calls    badcalls   nullrecv  badlen    xdrcall
12121452 0          0         0         0
Server nfs:
calls    badcalls
1312142  0
null     getattr    setattr   root      lookup     readlink read
0  0%    319612 24% 1220 0%   0 0%      795544 60% 5857 0%  163962 12%
wrcache  write      create    remove    rename     link     symlink
0  0%    7294 0%    165 0%    239 0%    75 0%      74 0%    0  0%
mkdir    rmdir      readdir   fsstat
0  0%    0  0%      17612 1%  334 0%

Client rpc:
calls    badcalls   retrans   badxid    timeout    wait     newcred
30156    40         256       0         296        0        0


Client nfs:
calls    badcalls   nclget    nclsleep
30143    40         30156     0
null     getattr    setattr   root      lookup     readlink read
0  0%    5833 19%   21 0%     0 0%      17630 58%  420 1%   3455 11%
wrcache  write      create    remove    rename     link     symlink
0  0%    475 1%     84 0%     10 0%     4  0%      0  0%    0  0%
mkdir    rmdir      readdir   fsstat
2  0%    0  0%      1423 4%   786 2%
```

The client structure is where clients keep track of an outstanding RPC call. Because there are six client structures, `nclsleep` is the number of times that there were six operations in progress when a seventh one arrived and had to wait until one of the client structures was freed.

Table 2—3 shows the `nfsstat` fields and their meanings.

Table 2—3: `nfsstsat` Output

Field	Meanings
`badcalls`	Number of badly formed calls
`badlen`	Number of RPC calls with too small a body
`badxid`	Number of times reply transaction ID did not match request transaction ID
`calls`	Number of calls begun or received
`create`	Number of times a new file was created
`fsstat`	Number of times file system attributes and statistic were retrieved
`getattr`	Number of file attributes that were retrieved

Table 2—3: (Continued) `nfsstsat` Output

Field	Meanings
`link`	Number of times a hard link was created
`lookup`	Number of times that a directory pathname was looked up
`mkdir`	Number of times a directory was created
`nclget`	Number of times a client structure was successfully acquired
`nclsleep`	Number of times all client structures were busy
`newcred`	Not currently used (contains a zero)
`null`	Number of null operations
`nullrec`	Number of empty RPC calls
`read`	Number of times data was read from a file
`readdir`	Number of times a directory was read
`readlink`	Number of times a symbolic link was read
`remove`	Number of times a file was removed
`rename`	Number of times a file was renamed
`retrans`	Number of times RPC calls were transmitted
`rmdir`	Number of times a directory was removed
`root`	Not currently used (contains a zero)
`setattr`	Number of file attributes that were stored
`symlink`	Number of times a symbolic link was created
`timeout`	Number of times a request was made but not answered
`wait`	Number of times client system had to sleep because client structure was busy
`wcache`	Not currently used (contains a zero)
`write`	Number of times data was written to a file
`xdrcall`	Number of RPC calls that failed to decode in XCR

2.6.4 Exercising the Network

ULTRIX provides a command and a utility, `ping` and `netx`, respectfully, to test the physical transmission accuracy of the network. The `ping` command is part of the standard TCP/IP implementation. The `netx` utility is another ULTRIX system exerciser.

2.6.4.1 Using `ping`

The `ping` command is part of the standard TCP/IP implementation; it uses the echo request and echo response feature of the Internet Control Message Protocol (ICMP). With no options other than the host name on the command line, `ping` reports that the host is alive or that there is no answer from the host.

On the command line you can specify several options: the `-l` option chooses long format. You can specify the size of the data packet (the default is 64 bytes), as well as the number of packets you want sent. The average round trip time of the packets is summarized at the conclusion of the test. This figure is important to identify any bottlenecks between the local and a remote host. Example 2–4 shows the output from the `ping` command with the `-l` option.

Example 2–4: The `ping -l` Command

```
csh> /etc/ping -l opus
PING Opus (128.10.1.1): 64 data bytes
72 bytes from 128.10.1.1: icmp_seq=0. time=10. ms
72 bytes from 128.10.1.1: icmp_seq=1. time=10. ms
72 bytes from 128.10.1.1: icmp_seq=2. time=10. ms
72 bytes from 128.10.1.1: icmp_seq=3. time=10. ms
72 bytes from 128.10.1.1: icmp_seq=4. time=10. ms
72 bytes from 128.10.1.1: icmp_seq=5. time=10. ms
72 bytes from 128.10.1.1: icmp_seq=6. time=10. ms
72 bytes from 128.10.1.1: icmp_seq=7. time=10. ms
72 bytes from 128.10.1.1: icmp_seq=8. time=10. ms
72 bytes from 128.10.1.1: icmp_seq=9. time=10. ms


----Opus PING Statistics---
10 packets transmitted, 10 packets received, 0% packet loss
round-trip (ms) min/avg/max = 10/10/10
```

2.6.4.2 Using `netx`

The `netx` utility is the TCP/IP network exerciser. It writes, reads, and validates random data using a stream socket connection to the `miscd` server in the TCP/IP internet domain. By default, `netx` utility uses the port number of the echo service in the `/etc/services` file. Also, you must turn on the echo service in the `/etc/inetd.conf` file. The `netx` utility runs for a specified length of time or until it receives a `QUIT` signal. For more information, see `netx`(8) in the *ULTRIX Reference Pages*.

2.6.5 Monitoring Network Activity

If you understand the normal ranges of performance in your network, you can better determine when the network is performing poorly due to a network problem. You can use the Network Control Program (NCP) to accumulate performance information. For TCP/IP networks, you can use DECmcc Management Station for ULTRIX (DECmcc) to gather performance information.

In understanding how network performance relates to network troubleshooting, the important concepts are thresholds and usage peaks. Threshold is the maximum value set for a parameter. Usage peak is the maximum level of activity the network can withstand before performance is adversely affected.

When thresholds are reached or when activity reaches a peak, transient and intermittent performance problems may begin to surface. Performance problems can be among the most difficult to isolate, primarily because they are often transient and intermittent.

For example, when traffic and queuing requests increase, bandwidth thresholds can be reached and performance problems can occur.

Maintaining historical performance data is essential for troubleshooting transient and intermittent problems. For example, if you have a transient problem, you can compare the historical performance data to the current performance data, and evaluate the difference to help isolate the source of the problem. Historical performance data is also useful in trend analysis for network growth and network planning.

2.6.5.1 Tools For Monitoring Network Activity

ULTRIX provides the `netstat` command to monitor network traffic on a continuous basis. Previous sections describe this command; however, this section deals with its relevance to monitoring network activity at specified intervals. See the following section for more information on using the `netstat` command.

The Network Control Program (NCP), bundled with the DECnet software, runs on the ULTRIX operating system and allows you to configure and control DECnet networks. Using NCP, you can monitor network resources, test network components by manipulating the configuration database, and display error counter information.

DECmcc is a network management tool for both IP and DECnet networks. DECmcc collects, formats, and monitors network data from simple network management protocol (SNMP)-based agents and DECnet nodes. With DECmcc, you can do the following:

- Display the data graphically and generate hardcopy reports
- Display information using a DECwindows graphic map, which signals changes on network by changing the colors of network entities displayed
- Observe trends and prevent problems

For more information about the DECmcc, see the *DECmcc Management Station for ULTRIX User's Manual.*

2.6.5.2 Using `netstat`

When you start the `netstat` command with a sample interval, it continuously displays information regarding packet traffic on the configured network interfaces until you terminate it with the `QUIT` signal. These statistics include the number of packets coming in to and going out from the system, the number of transmission errors, and the number of packet collisions. The following example shows the output from the `netstat` command with a sample interval of 5 seconds:

```
csh> netstat 5
  input    (se0)    output       input     (Total)    output
packets errs packets errs colls packets errs packets errs colls
23299       0   31597    0      0   25043    0   33341    0      0
    3       0       1    0      0       3    0       1    0      0
    1       0       1    0      0       1    0       1    0      0
    2       0       1    0      0       2    0       1    0      0
    1       0       2    0      0       1    0       2    0      0
    1       0       2    0      0       1    0       2    0      0
    1       0       1    0      0       1    0       1    0      0
    1       0       1    0      0       1    0       1    0      0
```

2.7 Tools for Monitoring Interprocess Communications

You can monitor three of the four interprocess communications (IPC) resources of ULTRIX using the `ipcs` command. The `ipcs` command displays information about shared memory, messages, and semaphores. You cannot access information on sockets using this command.

No equivalent to the continuous monitoring of I/O, memory, or network interfaces is available for interprocess communications. Nevertheless, you can access a considerable amount of information about interprocess communications.

2.7.1 Using `ipcs`

To determine which interprocess communications features are in use, enter the `ipcs` command without any arguments. If one or more of the features are not in use, the command prints a message that states that the particular feature is not currently defined.

The `ipcs` command has several options that governs reported statistics. The `-a` option prints information for all the options. In addition, the options `-m`, `-q`, and `-s` display information for shared memory segments, message queues, or semaphores only. You can use them with the other options.

Example 2–5 shows the output from the `ipcs` command.

Example 2–5: The ipcs Command

```
csh> ipcs
IPC status from /dev/kmem as of Wed Oct  2 14:40:00 1991
Message Queues:
T      ID      KEY         MODE         OWNER    GROUP
q       0 0x0000000d -Rrw-rw-rw-       snow     snow

Shared Memory:
T      ID      KEY         MODE         OWNER    GROUP
m       0 0x0000000b --rw-rw-rw-       snow     snow
m       1 0x00000014 --rw-rw-rw-       snow     snow
m       2 0x0000000a --rw-rw-rw-       snow     snow
m       3 0x0000000c --rw-rw-rw-       snow     snow

Semaphores:
T      ID      KEY         MODE         OWNER    GROUP
s       0 0x0000000f --ra-ra-ra-       snow     snow
```

For more information, see ipcs(1) in the *ULTRIX Reference Pages*.

2.7.2 Exercising Shared Memory

The shared memory exerciser, shmx, forks a background process and the two processes create, attach, write to, and read as many as six (the default) shared memory segments. You can specify the number of shared memory segments you want to test when you start the exerciser. The shmx utility runs until terminated by the QUIT signal or for a specified period of time as determined by an option on the command line. The size of shared memory segments tested is SMMAX (set in the configuration file) divided by the number of shared memory segments to be created.

2.7.3 Monitoring IPC Activity

ULTRIX provides no utilities to monitor IPC resources continuously, nor are there any unsupported or public domain tools. If you want to monitor IPC activity on a continuous basis, you can write a utility to do that. Each of the IPC resources maintains a status structure within the kernel's data space. A program that knows the resource's key can access that structure. Thus, you can write a utility that accepts the resource's key as an input parameter, runs at periodic intervals sampling the IPC resource's status structure, and prints that information.

3 Recognition and Diagnosis of Resource Constraints

This chapter discusses how you can use the tools described in Chapter 2 to identify performance problems typically encountered on a workstation.

By testing a subsystem, you can determine whether it needs tuning. This chapter provides suggestions on how to determine whether a subsystem is performing as well as expected, and gives examples of potential problems. For more information about specific steps to improve system performance, see Chapter 4.

3.1 Identifying File System Limitations

Most performance problems manifest themselves in I/O hardware saturation. You can reduce the I/O load on the hardware by reducing or stopping the paging and swapping, by running fewer applications, or by performing more efficient I/O.

In many ways, the file system is the easiest of the subsystems to tune. More tools are available to identify potential problems within the file system, and you can address more of those problems without relinking the kernel.

ULTRIX disk layout has block and fragment sizes to optimize performance versus space utilization (usually 8 KB/1 KB). Block size is the largest (normal) I/O request to disks. An I/O request may perform read-ahead operations for sequential file access.

ULTRIX uses a buffer cache to avoid disk I/O. Buffer cache statistics can be viewed with the `crash bufstats` command. Normally, successful cache operations should be above 90 percent. Buffer cache size is adjustable; by default, it is 10 percent of physical memory. Changing the buffer cache size involves a kernel rebuild and reboot.

ULTRIX attempts to do asynchronous writes when possible. It must write file system meta data synchronously. Prestoserve can improve synchronous write performance.

3.1.1 Determining Whether the Disk Subsystem Needs Tuning

Determining that the disk or file system needs tuning requires experience and judgment. Usually, if the answer to any of the following questions is yes, you can improve file system performance by tuning it:

- Is the CPU idle field of `iostat` or `vmstat` nonzero even when the system is heavily loaded?
- Is the disk load unbalanced? That is, are the `bps` and `tps` fields of the `iostat` command's output large for one of the disks, relative to the other disks?
- Is the average seek time reported by `iostat` significantly different than the average seek time for the device as listed by the manufacturer in product literature?

3.1.2 Recognizing When the Disk Subsystem Is Disk-Bound

Limitations in the disk or file system lead to less than optimal performance. Problems with the disk or file system can be categorized into two areas:

- Those related to the file system and the disk partition on which it resides
- Those related to the swap partition

If the performance problem is traced to the file system and the partition on which the file system resides, the problem is generally classified as one in which the disk subsystem is disk-bound.

A disk subsystem is disk-bound when one or more its disk drives experience a large amount of traffic and a significant proportion of CPU time is idle when the system is heavily loaded.

The following example shows the output from the `iostat` command when a system is disk-bound:

```
csh> iostat 5
  tty          rd0          rd1          cpu
tin tout    bps tps    bps tps    us ni sy id
  0    2      0   0      2   1     1  0  5 93
  0   57      0   0    258  61    28  0 72  0
  0   57      0   0    266  63    43  0 57  0
  0   47      0   0    266  63    48  0 52  0
  0   41      0   0    256  60    43  0 54  4
  0   42      0   0    222  55    11  0 46 43
  0   60      3   1    238  57    14  0 61 26
  0   60      0   0    250  60    14  0 55 31
  0   54      0   0    252  60    18  0 57 25
  0   59      2   1    207  52    14  0 42 44
  0   61      1   0    254  60    15  0 57 28
```

The first line of output from the `iostat` command shows statistics for the system from the time it was booted to the present. Subsequent lines show statistics from samples taken at 5 second intervals. The system is heavily loaded, as evidenced from the three lines following the first line; where the `cpu id` field shows there is 0 percent CPU idle time. The next several lines show a rise in the CPU idle time, ranging from 4 percent to 44 percent, and significant activity on one of the disks.

The source of the problem in this example is the disk–load balance. The disk `rd1` shows 50-60 transfers per second and 200-270 blocks read or written per second. The second disk `rd0` shows almost no activity.

To confirm that the problem is a poorly balanced disk load, you can examine the output from `vmstat`. The three procs fields (`r,b,w`) in the output from `vmstat` indicate the number of processes running (`r`), blocked for a resource (`b`), or runnable but swapped (`w`).

If poor disk performance were the result of fragmentation, the disk load would be more evenly balanced. The system would still be heavily loaded; however, some CPU idle time would still exist. Using only the output from `iostat` to determine that poor disk performance is due to excessive head movement is, at best, a guess. The previous example shows output from the `iostat` command for a disk in need of reorganization.

The CPU idle time shown in the following example ranges from 4 percent to more than 40 percent. The `iostat` and `vmstat` commands were not run simultaneously, so the statistics in each of the figures cover slightly different time periods. The important statistics to note, however, are the `bps` and `tps` fields. These fields show that the disk load is more evenly balanced.

```
csh> iostat 5
  tty       rd0      rd1        cpu
tin tout  bps tps  bps tps  us ni sy id
  0    2    1   1    2   1   1  0  5 93
  0   41   88  21  128  30  43  0 54  4
  0   42   80  19  101  25  11  0 46 43
  0   60   82  20  113  27  14  0 61 26
  0   60   91  22  125  30  14  0 55 31
  0   54  101  24  126  30  18  0 57 25
  0   59   84  20   88  22  14  0 42 44
  0   61   80  19  127  30  15  0 57 28
```

Output from the `iostat` command shows some CPU idle time. This indicates that the system is disk-bound.

3.1.3 Recognizing When the Disk Subsystem Is Swap-Bound

The output from the `iostat` and `vmstat` commands might indicate that the system is not disk-bound, but these tools can indicate another problem with the disk subsystem. In this case, you can expect to see the following:

- A fairly evenly balanced disk load
- The average seek time for one or more of the disks might or might not be large relative to the manufacturer's specification
- The system might be paging or swapping
- CPU idle time occurs even though the system is being heavily used

These conditions can indicate the system is swap-bound.

A system is swap-bound when active virtual memory (`avm`) is large relative to available real memory, the free list is small, and swap space is less than two times `avm`.

When the system is swap-bound, the operating system sends messages indicating that it has killed processes because it has run out of swap space as follows:

```
pid 492 was killed in xalloc: no swap space
```

If you suspect that the system is swap-bound, the `pstat` command gives a quick indication of how the system is using its swap space. The following example shows that only 4.5 percent or 753 KB of the configured swap space is free:

```
csh> pstat -s
16720k swap configured
          15965k used (3424k text, 2k smem)
          753k free, 4063k wasted, 2k missing
avail: 2*128k 1*64k9*32k 4*16k 81*1k
```

If you suspect the system is swap-bound, you must determine the source of the problem. Two likely causes to investigate are as follows:

- A heavily loaded system running very large processes
- A system configured with too little swap space

To determine whether some large processes are forcing the system to exhaust its swap space, examine the output from the `ps` command. In the following example, the fields of interest are `SZ` and `RSS`, which list the virtual size and real memory used by the process, respectively, and `%CPU`, which indicates the percentage CPU utilization of the process.

```
csh> ps -uef
USER  PID  %CPU  %MEM   SZ  RSS  TT STAT  TIME COMMAND
spec 1888  27.1   6.4  418  330  p1    R  0:02 ccom /tmp/ctm018864 /tmp/ctm01
spec 1738  15.9   4.4  317  221  p1    R  0:39 espresso -t bca.in DISPLAY=:0.
spec 1709  14.7   2.4  161  116  p1    R  0:41 xlisp li-input.lspDISPLAY=:0.
spec 1724  14.3   4.6  269  236  p1    R  0:40 doducDISPLAY=:0.0 EXINIT=set
root  139   4.1   9.5 2454  477  p1    S  3:54 /usr/bin/dxterm -ls HOME=/ SHE
spec 1886   1.0   0.5   22   20  p1    S  0:00 cc -O -c y.tab.c DISPLAY=:0.0
root  140   0.5   1.8  149   85  p1    S  0:13 - HOME=/ SHELL=/bin/csh TERM=v
spec 1598   0.0   1.7  143   81  p1    I  0:02 make FC f77 validate DISPLAY=:
spec 1737   0.0   0.3   13    7  p1    I  0:00 /bin/time espresso -t bca.in D
spec 1721   0.0   0.3   13    7  p1    I  0:00 /bin/time doduc DISPLAY=:0.0 E
spec 1634   0.0   1.5  141   71  p1    I  0:00 make -f M.vax validateDISPLAY
spec 1704   0.0   0.3   13    7  p1    I  0:00 /bin/time xlispli-input.lsp D
spec 1622   0.0   1.4  139   65  p1    I  0:00 make -fM.vax validate DISPLAY
spec 1593   0.0   1.4  139   65  p1    I  0:00 make -f M.vax validate DISPLAY
spec 1581   0.0   1.5  139   68  p1    I  0:00 make -f M.vax validate DISPLAY
spec 1597   0.0   0.3   29    8  p1    I  0:00 sh -ce make "FC=f77" validate
```

The `STAT` field indicates processes that are swapped: `S` for processes sleeping less than 20 seconds, `I` for processes idle for more than 20 seconds, and `W` for processes which are swapped out. In the previous example, none of the processes use an inordinately large amount of memory, none use an unusually large amount of CPU time, and none are swapped. Thus, the source of the swapping problem is elsewhere.

The second possible cause of a swap-bound system is easy to check. The output from `pstat` in the previous example shows the total swap space for the system; output from the `vmstat` command in the following example shows active virtual memory and free memory. You also should know the amount of physical memory with which the system is configured.

```
csh> vmstat 5
procs      faults    cpu      memory          page              disk
r b w in    sy cs us sy id avm fre    re at pi   po fr de   sr s0 s1
3 0 0 126   66 12 89 11  0 14k 1163   0  0  0    0  0  0    0  0  0
4 1 0 119   77 40 81 19  0 16k 1060   9  0 62   17 14  0   37  0 24
4 0 0  91   60 35 82 18  0 16k  899  15  0 44   30 36  0   87  0 18
5 0 0  88   72 40 78 22  0 16k  870  29  0 47   86 43  0 102  0 24
5 0 0 133  111 28 78 22  0 16k  922  32  0 21  111 52  0 101  0 17
4 0 0 146   98 23 79 21  0 16k  984  14  0  9   35 16  0   33  1 10
4 0 0 134   82 19 88 12  0 16k  993  14  0  7   29 19  0   32  0  5
4 0 0 118   38 12 93  7  0 15k 1239   5  0  1    8  5  0    9  0  1
3 0 0  78   39  9 95  5  0 15k 1450   2  0  0    1  0  0    2  0  0
3 0 0  68   78 12 80 20  0 15k 1524   3  0  0    0  0  0    0  1  5
```

Three conditions can exist if a system is configured with too little swap space:

- The active virtual memory is large relative to physical memory
- Little free space exists
- Swap space is not at least two times `avm`

Such a system is, by definition, swap-bound.

3.2 Identifying Process Control and Scheduling Limitations

Diagnosing problems in the process control subsystem is much more difficult than diagnosing problems in the file subsystem. Part of the difficulty is that no clear metric exists with which to determine how the system is performing; there are, however, several possible measures of performance:

- The absolute number of processes the system is capable of running. However, this measure does not give a good picture of how rapidly the system can handle these processes.
- The number of processes completed per second, but this measure is sensitive to which processes are run.
- The response time to a command typed at a user's terminal when the system is heavily loaded.

All of these measures depend on system administrators' and users' perceptions of the responsiveness of the system.

3.2.1 Determining Whether the Process Control Subsystem Needs Tuning

As discussed in Chapter 2, most opportunities to tune the process-control subsystem require access to kernel source code. Without this access, tuning the process control subsystem is accomplished only through system management.

Deciding whether the process control subsystem needs tuning is a matter of judgment. It is necessary for you to do some homework before evaluating the system's performance. You must have a baseline performance measure when the system is lightly loaded against which to measure the system's performance when it is heavily loaded.

The following general rule is based on a load average computed as the average number of processes completed per second: Maximize the load

average constrained by some acceptable benchmark that simulates a specified number of users and computes the processes completed per second.

The AIM Benchmark Suite III is used in some of the examples in this chapter because it was available and the test environment was not characterized by any other specific workload. In your own operating environment, you can use your own workload to evaluate your system's performance.

3.2.2 Computing the Load Average for the System Under Test

Compute the load average for the system and compensate for response time by timing a typical command a user might execute. You must use a stopwatch, because the time of interest is the time from which the command is issued to the time at which the system prompt is received again. The built-in C-shell command time and the system utility `/bin/time` are inappropriate, because they calculate only the time it takes to run the command. The steps are as follows:

1. Measure response time by entering the `date` command followed by the `ls` command.

```
csh> date; ls -l
Thu Oct 10 15:41:05 EDT 1991
130
1 jim 100000 Jun 21 10:58 big.file
1 jim 9216 Jun 21 09:50 bldfile
1 jim 433 Jun 21 09:50 bldfile.c
1 jim 6144 Jun 18 10:20 msgtest
1 jim 714 Jun 18 10:19 msgtest.c
1 jim 16 Jun 21 15:36 time.out
```

2. Enter the `date` command when the prompt appears again.

```
csh> date
Thu Oct 10 15:41:07 EDT 1991
```

   Some overhead is involved when you invoke the `date` command, but it is small compared with the time it takes to invoke the `ls` command on a fairly large directory.

3. Compute response times for the same commands when the AIM Benchmark Suite III is simulating 5, 10, 15, or 20, or more users.
4. Normalize the total time taken to run the benchmark for each of the simulated number of users.

   Normalization is accomplished by dividing the total time taken to run the benchmark for each group of simulated users by the time taken by the largest group of simulated users to complete the benchmark. This transforms the time to complete the benchmark into the range 0 to 1.

5. Normalize the response time for each group of simulated users in the same way.
6. Chart the resulting normalized load average time and response time on a graph as follows:
   - Place the number of simulated users as the vertical axis.

- Place the normalized load average time on the horizontal axis increasing from left to right.
- Place the normalized response time on the horizontal axis increasing from right to left.

The point where the two lines cross indicates the maximum number of users constrained by an acceptable response time.

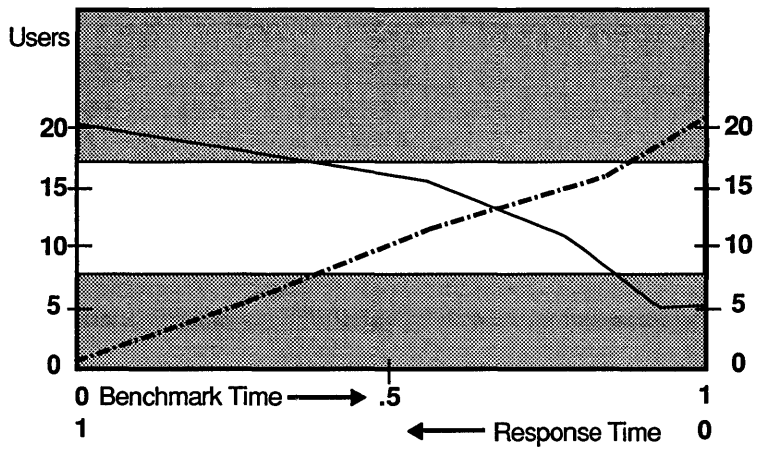
Table 3–1 shows the results of these steps, and Figure 3–1 shows the results when they are graphed. Columns 2 and 3 in Table 3–1 show the processes per second and total time to complete the benchmark computed by the AIM Benchmark Suite III for 5, 10, 15, and 20 simulated users. Column 4 shows the normalized total time computed by dividing the value in column 3 by 3481. Column 5 shows response time for the corresponding user level in column 1, and column 6 shows its normalized value.

Table 3–1: Process Timing Experiments

Users	Processes per Second	Real Time	Normalized Time	Response Time	Normalized Response Time
5	.053	975.2	.280	2	.143
10	.028	1890.4	.543	3	.214
15	.020	2704.2	.777	6	.429
20	.015	3481.0	1.0	14	1.0

Given the constraint placed on response time, Figure 3–1 shows the maximum number of users to be between 10 and 15 users.

Figure 3–1: Normalized Benchmark Time Versus Response Time



3.2.3 Recognizing a Well Balanced Load Over the Available Time

You can calculate the maximum acceptable load as described in Section 3.2.2, but this calculation addresses only the instantaneous load. Another method that maximizes the load a system is capable of handling is to balance the load over a longer period of time. The command `uptime` displays the 1, 5, and 15 minute load averages for the computer. Even these are short-term measures, if you are concerned with the amount of work that users do during the course of a day. Thus, you can further improve the instantaneous load shown in Figure 3–1 by rescheduling jobs that would otherwise increase the load average over the optimal value displayed there; however, this is a system management issue.

3.2.4 Recognizing a Shortage of Slots in the Process Table

The process table is a table internal to the kernel that is set up when the system is configured. The running system keeps information about active processes in this table.

Two parameters in the configuration file govern its size: `maxusers` and `maxuprc`. The `maxusers` parameter is unrelated to the user license seen when the system is booted. A typical workstation, for example, has a two–user license; however, the value of `maxusers` can be 16. The number of processes that an individual user is allowed to run simultaneously is governed by `maxuprc`. The size of the process table is determined by `maxusers` times `maxuprc`, plus some overhead.

You may encounter two separate problems that you can trace to the size of the process table. If you exhaust the number of processes that you are allowed, the system displays a message that indicates there are no more processes, and your job will not run.

A more serious problem exists if the process table is full. If the total number of processes that active users run plus system utilities and other system–related processes that utilize slots equals the number of available slots in the process table, the system cannot run any more processes. In this instance, the system displays a message that indicates that the process table is full. To increase the size of the process table, you can increase the `maxusers` or `maxuprc` parameters. For more information about this topic, see Chapter 4.

3.2.5 Recognizing Problems with Interrupts, Context Switches, or System Calls

You can achieve performance improvements in the process control subsystem through careful software development. Again, to understand the normal state of the system, you must monitor the system when it is idle, under a light load, or when a suspected inefficient program is running. Poorly written software can impose a significant load. Some obvious symptoms are inordinately large numbers of interrupts, context switches, or system calls.

A program called `bldfile` writes a 100 KB file 100 times; source code for the program appears in Appendix B. Execution of the program is timed with the C-shell built-in `time` command. The first version writes the file using 1000 byte buffers with the write system call. Example 3–1 shows output from the `time` and `vmstat` commands.

Example 3–1: `time` and `vmstat` Output for `bldfile` Program

```
csh> vmstat 5 &
 procs      faults    cpu      memory              page            disk
 r b w   in   sy cs us sy id avm   fre  re at pi  po fr de  sr s0 s1
 0 0 0 277   55 14  1  1 98 19k 8436   0  0  0   0  0  0   0  0  0
 0 0 0  74   20  3  2  1 97 20k 8280   0  0  0   0  0  0   0  0  0
 0 0 0  70   10  2  2  0 98 19k 8116   0  0  0   0  0  0   0  0  0
 0 0 0  68   13  1  2  0 98 17k 8060   0  0  0   0  0  0   0  0  0
csh> time bldfile.1 big.file
 1 0 0 129   86  9  6 15 79 20k 8040   0  0  0   0  0  0   0  0  0
 0 1 0 164  117 13  2 24 74 20k 8016   0  0  0   0  0  0   0  0  0
 0 1 0 164  110 13  2 23 75 20k 8008   0  0  0   0  0  0   0  0  0
 0 1 0 164  114 13  2 23 74 19k 8008   0  0  0   0  0  0   0  0  0
 0 1 0 165  120 13  2 24 74 17k 8008   0  0  0   0  0  0   0  0  0
0.0u 5.1s 0:24 20% 10+20k 0+1204io 0pf+0w
 1 0 0 105   55  6  4  9 87 17k 8008   0  0  0   0  0  0   0  0  0
 0 0 0  70   14  1  2  1 97 18k 8024   0  0  0   0  0  0   0  0  0
 0 0 0  68   16  1  2  1 97 18k 8024   0  0  0   0  0  0   0  0  0
 0 0 0  66    8  0  2  0 97 23k 8024   0  0  0   0  0  0   0  0  0
```

The second version substitutes the `fwrite` subroutine for the write system call. The `fwrite` subroutine allows the system to buffer all disk accesses in the most efficient manner. Example 3–2 shows output from `time` and `vmstat`.

Example 3–2: `time` and `vmstat` Output for `bldfile` Program

```
csh> vmstat 5 &
 procs      faults    cpu      memory              page            disk
 r b w   in   sy cs us sy id avm   fre  re at pi  po fr de  sr s0 s1
 0 0 0 277   55 14  1  1 98 18k 8456   0  0  0   0  0  0   0  0  0
 0 0 0 104   28  6  2  1 97 20k 8284   0  0  0   0  0  0   0  0  0
 0 0 0  89   17  1  2  0 98 23k 8116   0  0  0   0  0  0   0  0  0
 0 0 0  91   22  2  2  0 97 23k 8060   0  0  0   0  0  0   0  0  0
csh> time bldfile.2 big.file
 0 0 0 173   47  8  6 10 84 24k 8040   0  0  0   0  0  0   0  0  0
 0 1 0 211   37  5  3 14 83 23k 8004   0  0  0   0  0  0   0  0  0
 0 1 0 223   36  8  3 15 81 18k 7988   0  0  0   0  0  0   0  0  0
 0 1 0 218   39  6  3 15 82 18k 7988   0  0  0   0  0  0   0  0  0
 0 1 0 228   39  9  3 16 81 18k 7988   0  0  0   0  0  0   0  0  0
 0 1 0 214   34  5  3 14 83 17k 7988   0  0  0   0  0  0   0  0  0
 0 1 0 226   38  9  3 16 82 21k 7988   0  0  0   0  0  0   0  0  0
0.3u 1.9s 0:33 6% 20+61k 0+1510io 0pf+0w
 4 0 0 101   26  5  6  1 93 21k 7996   0  0  0   0  0  0   0  0  0
 0 0 0  89   15  1  2  1 97 21k 8020   0  0  0   0  0  0   0  0  0
 0 0 0  89   22  1  2  1 97 15k 8024   0  0  0   0  0  0   0  0  0
 0 0 0  89   12  1  2  0 97 17k 8024   0  0  0   0  0  0   0  0  0
```

Example 3–3 shows additional output from `vmstat` when the system is idle. Note that the number of context switches (`cs`) is approximately the same as those shown in Example 3–2 and Example 3–3, and the number of system calls is not much greater in Example 3–2 than it is in Example 3–3. Both of these fields are much larger in Example 3–1. The only field in Example 3–1 that is smaller than the same field in Example 3–2 is the number of interrupts. In general, system calls and context switches require a lot of

system resources because they cause the kernel to transfer data back and forth between kernel and user space.

Example 3–3: Output from `vmstat` when the System Is Idle

```
csh> vmstat 5
 procs      faults    cpu      memory                 page              disk
 r b w  in  sy cs us sy id avm fre   re at pi  po fr de  sr s0 s1
 0 0 0 277  55 14  1  1 98 26k 8468   0  0  0   0  0  0   0  0  0
 0 0 0 142  36  3  3  1 96 25k 8248   0  0  0   0  0  0   0  1  3
 0 0 0 135  26  2  2  0 97 26k 8104   0  0  0   0  0  0   0  0  2
 0 0 0 134  26  2  2  0 97 26k 8056   0  0  0   0  0  0   0  0  0
 0 0 0 130  16  1  2  0 98 17k 8040   0  0  0   0  0  0   0  0  0
```

3.3 Identifying Memory Management Limitations

To improve the performance of the memory management subsystem, you must be able to recognize when the system has too little physical memory or when the buffer cache is too large or too small. To recognize memory management limitations, you must be familiar with how ULTRIX uses memory in running processes and the relationship between total virtual memory, active virtual memory, real memory, and the swap partition. For information about these concepts, see Chapter 1.

Excessive paging and swapping can cause memory bottlenecks. Memory bottlenecks can become I/O bottlenecks because of the operation of the virtual memory system. CPU and memory subsystems are getting faster at a greater rate than disk subsystems; however, the decrease of paging and swapping is more dramatic on faster CPU systems.

Applications have virtual size. Generally, virtual size is not required for execution. The size required for execution is the resident set size (RSS). Virtual and resident sizes of applications can be viewed with the `ps` command. Look for `SZ` and `RSS` headings. Active virtual memory is the total of the virtual sizes for processes running in the last 20 seconds. Active real memory is the total of the resident set sizes. You can use the `vmstat` command to view virtual memory, active virtual memory, real memory, and active real memory.

The system handles application–execution changes to the resident set size by paging. Paging in adds memory to the application. Usually, this is application code or data added to the resident set size for the application. This is done by application demand (execution). Paging out removes memory from the application. This may be application code or data, removed from resident set size. This is done automatically by the system when free memory is scarce. Some paging is good and removes unnecessary memory from the resident set size. Excessive paging is bad for performance.

The system pages according to a global page replacement algorithm. When free memory is less than `lotsfree`, paging begins. When free memory is less than `minfree`, swapping occurs. When free memory is less than `desfree` for a significant amount of time (30 seconds), desperation swapping begins. The parameters `fastscan` and `slowscan` control the paging rate; the default is computed on `memsize`.

The `handspread` parameter is the amount of memory between two checks of memory pages. The first check clears a page's reference bit. The second

check, which occurs after the amount of time the system needs to scan the handspread, looks to see if the reference bit has been set. If the bit has not been set, then the page becomes eligible for paging. Changing the handspread changes the distance between checks which affects the amount of time it takes for a memory page to become eligible for paging. Systems that continuously run large applications may benefit by increasing the handspread parameter because the entire application is more apt to remain in memory.

Table 3–2 shows the parameter defaults stored as pages. Defaults for parameters may not be optimal for systems with greater than 8MB.

Table 3–2: Paging Parameter Defaults

Parameter	VAX	RISC	Maximum Limit
PAGESIZE	512 bytes	4096 bytes	None, fixed amount
LOTSFREE	512 KB	512 KB	$\frac{1}{4}$ of physical memory
DESFREE	200 KB	256 KB	$\frac{1}{8}$ of physical memory
MINFREE	64 KB	96 KB	$\frac{1}{2}$ of DESFREE
HANDSPREAD	2 MB	8 MB	size of user memory

You can change the default values for each of the parameters listed in Table 3–2 by editing your system's configuration file to include the parameter and its new value and rebuilding the kernel, as described in the *Guide to Configuration File Maintenance*.

Swapping occurs for severe memory shortages and for expansion of the process page table. The system attempts to select long sleeping processes (greater than 20 seconds) to be swapped. Generally, swapping is worse than paging. It causes system performance to vary wildly. Swapping may be good for very long sleeping processes.

Paging and swapping can be viewed with vmstat -s. Take the following actions to reduce excessive paging and swapping:

- Reduce memory demands on the system by running fewer applications, by adjusting virtual memory parameters to make better use of memory, and by swapping and paging across multiple devices
- Add more memory for use by the virtual memory system by reducing the buffer cache (not below 10 percent of physical memory) and adding physical memory to the system.

Swap space is preallocated for processes when they begin execution. This is required in case they need to swap and page or both. Usually, when there is not enough swap space configured in the system, the system displays a message that says "not enough core" or "malloc failed." The /etc/pstat -s command shows swap space that is configured and virtual address space that is reserved. If the two are close, or if you are seeing errors, more swap space is needed.

3.3.1 Determining Whether the Memory Management Subsystem Needs Tuning

The primary tool used to evaluate the memory management subsystem is the `vmstat` command, with which you look at the fields for active virtual memory (`avm`) and free memory (`fre`).

The memory management subsystem needs to be tuned when the free list is small and active virtual memory is more than 80 percent of available real memory when the system is heavily loaded.

3.3.2 Recognizing When Active Virtual Memory Is Too Large

Active virtual memory is probably too large when it exceeds 80 percent of the available real memory. Look at the short-term memory deficit field (`de` field from the `vmstat` command) to determine if your system is performing excessive swapping. If the field contains a number greater than 0, then your system's active virtual memory is too large, resulting in a serious degradation of performance. Also, check the paging activity fields (`pi` and `po` from the `vmstat` command) for a large amount of activity. A large value (especially in the `po` field) can indicate the beginning of memory problems. The following example shows the output from the `vmstat` command when active virtual memory is too large.

```
csh> vmstat 5
procs      faults    cpu      memory              page            disk
r b w    in   sy cs us sy id avm  fre    re at pi  po  fr de  sr s0 s1
4 0 0   126   66 12 89 11  0 37k 1080   21  0  0   0   1  0  76  1  0
4 0 0    62   56 16 81 19  0 34k  685   31  0  0   5   4  0 138  2  5
3 1 0    78   50 28 73 50  0 35k  274   33  0  1 103  17  0 157 15 12
4 0 0    84   54 19 70 30  0 35k  308   56  0  0 225  79  0 153  9 19
3 1 0    76   54 17 78 22  0 36k  338   68  0  1 147 116  0 145  7  8
5 0 0    72   54 18 75 25  0 36k  451   71  0  5  94 107  0 131  3 11
4 0 0    76   51 27 73 51  0 36k  572   58  9 44  64  66  0  72  9  9
4 0 0    71   42 22 88 42  0 33k 2749   22  2 30  16  16  0  17  6  4
4 0 1    71   44 23 84 16  0 35k 2583   19 11 52   4   4 34   4 12  2
5 0 0    76   38 32 85 15  0 36k 2129   14  4 72   0   0  0   0 14  7
4 0 0    64   34 15 97  3  0 36k 1923    5  0 23   0   0  0   0  0  0
```

3.3.3 Recognizing a Shortage of Physical Memory

Active virtual memory can be large relative to physical memory but not so large that it causes excessive paging, a short-term memory deficit, or both. In this case, a shortage of physical memory may exist. If you suspect a shortage of physical memory, look at the fields that report the number of pages scanned per second (`sr` in `vmstat`) and the number of pages freed per second (`fr` in `vmstat`). If the values in these fields seem large, check the output from the `ps` command.

If the page daemon process, `pagedaemon`, shows a large amount of time in the time field, system performance may be improved by adding more memory or by reducing the size of the buffer cache, if possible. Examples 3–4 and 3–5 show output from `vmstat`and `ps` for a system that might benefit from added physical memory.

Example 3–4: `vmstat` Output when the System Could Benefit from More Memory

```
csh> vmstat 5
procs      faults    cpu      memory              page            disk
r b w   in  sy cs us sy id avm fre   re at pi  po  fr de  sr s0 s1
4 0 0   63  34 11 95  5  0 34k 1740  14  0  4   2   0  0   2  2  1
5 1 0   79  50 37 74 26  0 35k 1596  30  0 68   0   7  0  30  2 21
4 0 0   68  46 22 91 46  0 35k 1160  27  1 26   5  21  0  65  2  1
4 0 0   66  51 19 83 17  0 35k  978  54 23 19  12  24  0  98  5  1
4 0 0   64  51 18 87 13  0 39k  652  69  7 12  12  27  0 124  4  2
4 0 0   73  41 17 80 20  0 39k  518  69  0  4  52  55  0 127  8  8
4 0 0   71  41 16 84 16  0 38k  615  65  0  2  90  62  0 116  4  9
4 0 0   68  36 17 91  9  0 38k  776  75  0  1  75  37  0 108  2  6
```

Example 3–5: Output from `ps` Showing Excessive Time for the Page Daemon

```
csh> ps -efax
  PID  TT STAT TIME COMMAND
    0   ? D    0:15 swapper
    1   ? IW   0:10 init
    2   ? D   22:16 pagedaemon
  142  p1 S N 35:06 /usr/bin/dxterm -ls HOME=/usr/users/jim SHELL=/bin/csh TERM
  144  p1 S    0:07 - HOME=/usr/users/jim SHELL=/bin/csh TERM=vt300 USER=jim PA
 2548  p1 R    0:01 ps -efax HOME=/usr/users/jim SHELL=/bin/csh TERM=vt300 USER
 2549  p1 S    0:00 tee ps -efax.out HOME=/usr/users/jim SHELL=/bin/csh TERM=vt300
```

3.3.4 Buffer Cache Size

The configuration file parameter, `bufcache`, allows a specified percentage of physical memory to be set aside by the file system for use by the file system buffer cache. The percentage must be 10 or greater, but less than 100.

By default, buffer cache occupies 10 percent of main memory. Increasing the buffer cache size means that more file system data is stored in memory. While a large buffer cache may make a benchmark test run faster, there are trade-offs. The ULTRIX operating system uses a static buffer cache allocation methodology. Main memory that is allocated at boot time for the file system buffer cache cannot be used for user program text or data. Therefore, actual performance depends on the application.

For example, to set the cache buffer size to 25 percent of memory, add the following line to your system's configuration file located in the directory `/sys/conf/mips` for RISC processors or `/sys/conf/vax` for VAX processors:

```
bufcache    25
```

After editing the configuration file, you need to rebuild your kernel. See the *Guide to Configuration File Maintenance* for more information on the configuration file and its options and for instructions on rebuilding your kernel.

Optimal values for `bufcache` differ among large timesharing systems, mid-range file servers, and workstations. However, do not alter `bufcache` if you have a workstation with 8 megabytes of memory. For workstations with 16 megabytes of memory limit the value to 30 or less. If you specify a value greater than 30, your system's file system performance may suffer because of excessive paging and swapping.

For file servers, increasing the buffer cache can improve performance. Note that if you make the buffer cache too large, the resulting system may be less efficient in processing the requests to it from multiple users. To help you determine the optimal value, use the results from the `bufstats` command of the `crash` utility. This command can provide useful data on cache hit or miss ratios. See `crash`(8) in the *ULTRIX Reference Pages* for more information on `bufstats`.

You can experiment with different sized buffer caches, increasing the size of the buffer cache until the number of blocks transferred to and from the disks is minimized for a particular workload. If the buffer cache becomes too large, you will notice an increase in the paging activity and perhaps even a short-term memory deficit. When this happens, the system has become swap-bound. Reduce the size of the buffer cache until paging activity returns to normal.

By default, the buffer cache of the file system block is located in a memory space not cached by the processor data cache. You can alter this so that the buffer cache is located in a cached memory space by editing the `param.c` file. The `param.c` file is located in the `/sys/conf/mips` for RISC machines or `/sys/conf/vax` for VAX machines. You must add the following to the `param.c` file:

```
int cache_bufcache = 1;
```

After editing the `param.c` file, you need to rebuild the kernel. See the *Guide to Configuration File Maintenance* for information on rebuilding the kernel.

Depending on your application mix, this change may improve file system throughput. Preliminary work shows that DECSystem 5500s and DECSystem 5100s with Prestoserve perform significantly better under heavy NFS loads with the buffer cache located in cached memory. Other benchmarks have shown a slight negative impact of setting this flag.

3.4 Identifying I/O Subsystem Limitations

Although, access to kernel source code is required to improve the efficiency of kernel algorithms that handle I/O processing, some improvement may be possible by replacing hardware. Carefully monitor the I/O hardware to determine whether hardware limitations are responsible for system performance bottlenecks. If such bottlenecks exist, replace the hardware that is responsible.

3.4.1 Determining Whether the I/O Subsystem Needs Tuning

You may feel that the I/O subsystem needs tuning because you or other users perceive system response time to be sluggish. Such perceptions may be the best indicators that a system needs tuning. If users' perceive that the system is slow, then it is slow.

The I/O subsystem consists of terminals, printers, tape drives, and so on, so an objective measure of the throughput rate must be constructed. If the throughput rate of these peripherals differs widely from the throughput rate on another similar workstation, the I/O subsystem may need tuning.

3.4.2 Recommendation for Improving I/O Performance

Results of I/O performance tests show that users can tune simple ULTRIX kernel parameters to significantly improve I/O.[1]

The tuning parameters available to you are as follows:

- Change the buffer cache size in the configuration file `/sys/conf/`*`machine`*`/`*`SYSTEMNAME`*, as described in Section 3.3.4.
- Change the write-scheduling policy in the `/sys/`*`SYSTEMNAME`*`/param.c` file.

  By default, the ULTRIX operating system returns write requests immediately. If the last byte of a block is written, then the dirty block is asynchronously sent to disk. When this happens, the block becomes unavailable until the disk write is complete. While this scheduling method is beneficial in a time–sharing environment, it hinders some benchmark tests that read data immediately after writing it.

  To set the ULTRIX system so that data can be read as soon as it is written and writes to disk are delayed as long as possible, make the following change in the `param.c` file located in the directory `/sys/conf/mips` for RISC processors or `/sys/conf/vax` for VAX processors:

  ```
  int delay_wbuffers = 1;
  ```

  After editing the `param.c` file, you need to rebuild the kernel. See the *Guide to Configuration File Maintenance* for instructions on rebuilding your system's kernel.

- Change the update frequency `/etc/update`.

  By default, the `update` daemon synchronizes dirty blocks to disk every 30 seconds. You can alter this time interval in two ways. The first way is to add a value to the `/etc/update` command in the file `/etc/rc`. For example, to adjust the `update` time interval from 30 seconds to 2 minutes, edit the file as follows:

  ```
  /etc/update 120;  echo -n 'update' > /dev/console
  ```

  The second way is to kill the `update` daemon process and restart it with the new value.

---

[1] Alexandre Bronstein, Memo on ULTRIX Disk/File-System Performance on DS3100/DS5000 (Palo Alto, CA.: Digital Equipment Corporation, 1990) x-x.

If you have a big cache and an application that often writes over the same blocks of a file, consider increasing the time interval for `update`. Note, the update frequency is the easiest parameter to experiment with because kernel recompilation is not required.

The first two changes require rebuilding the kernel, as described in the *Guide to Configuration File Maintenance*.

Before you set these parameters, you must realize that there are the following trade-offs:

- Speed versus safety

  The main factor that affects safety (that is, preventing loss of data) is the update (syncing) frequency. In between the synchronization of memory and disk (syncing), changes intended to be written on the disk only reside in memory. If the power fails before the disk is updated, those changes are lost. The default updating the disk is 30 seconds. Choosing the optimal value involves many factors, some of which are purely technical, such as the hardware configuration or expected disk traffic; others could be environmental, such as the application.

  The write-scheduling strategy affects safety. The default strategy increases the level of safety by ensuring that most blocks of data written to disk before they are read back into memory for access by an application.

- Speed of one application versus the speed of other applications

  In the process of tuning the system to accelerate the performance of an application, you may slow down other applications. For example, if you increase the `update` interval and change the write–schedule policy, an application such as a benchmark may process faster. However, when memory needs to write its contents to the disk, the system may seem slow temporarily because of the amount of processing required to write the contents of memory to disk.

Another important tuning parameter is disk organization; that is, you can achieve significant improvements by spreading the I/O traffic across as many physical disks as possible.

Some caution is in order; unless you understand the value of modifying these parameters and can detect a performance improvement after doing so, you should use their default values. If you understand the application being run, know what the machine is capable of doing, and understand how the parameters are used, your system's performance can improve by modifying these parameters.

3.4.3 Measuring I/O Subsystem Throughput

As an example of the type of metric you can construct to measure I/O subsystem performance, consider the following example for the throughput of a tape drive:[2]

$$\text{throughput} = \frac{\text{speed} * \text{blocksize}}{(2 * \text{gapsize}) + \frac{\text{blocksize}}{\text{density}}}$$

[2] Richard W. Stevens, *UNIX Network Programming*. Englewood Cliffs, NJ: Prentice–Hall, Inc., 1990

For a tape drive that has a density of 6250 bits per inch (bpi), a tape speed of 125 inches per second (ips), and a block size of 32,768 bytes, its optimal throughput is 678,000 bytes per second. You can copy files of known size to your tape drive and measure the actual throughput rate. If you compare the actual rate with the optimal rate, you can determine how well the tape drive is performing.

3.4.4 Identifying NFS Bottlenecks

NFS bottlenecks can occur in the following areas:

- CPU
- File system
- Network subsystem
- Disk subsystem

You can improve NFS performance by doing the following:

- Using Prestoserve, if your system supports it
- Increasing the buffer cache
- Adding more disks
- Reducing client paging and swapping or both
- Adjusting the number of `biod` or `nfsd` daemons

NFS write operations from a client must be done synchronously to a stable storage device (normally a disk). A single client write can require up to 3 writes on the server: an `inode` update (indirect blocks), a directory update, and a data write. Unless there are network problems, the server's NFS performance is bounded by the time needed to access a stable storage device. Prestoserve circumvents this limitation by using a nonvolatile cache as a stable storage device. As a result, write performance increases by 600 percent. Overall NFS performance increases by 100 percent.

3.4.5 NFS Tuning

You can improve NFS performance in the following ways:

- Adjust the number of `nfsd` and `biod` daemons.

  The `nfsd` daemons are used on the server to service client requests. The number of daemons executing is equal to the number of parallel operations. You can adjust the number of daemons to find maximum performance. The `biod` daemons are used on the client to provide asynchronous operation. The number of `biod` daemons executing is equal to the number of outstanding I/O requests for the system. The `nfsd` and `biod` daemons are started in `/etc/rc.local` at system startup.

- Turn off kernel's checking and generating UDP checksums by setting `UDPCKSUM` equal to 0 by using the `kvar`(8) command or the `dbx`(1) utility for RISC systems or the `adb`(1) utility for VAX systems. For more information on these commands and utilities, see the *ULTRIX Reference Pages*.

  Note, however, that this is only recommended for benchmarks because of possible data corruption. The `nfsstsat -s` command can show the number of corrupt packets detected (`xdrcall`). This command can also indicate hardware problems.

3.5 Identifying Network Limitations

Because performance is governed largely by the hardware that makes up the network, identifying network limitations involves testing network connections to establish a throughput rate. You can compare these values with theoretical maximum or optimal values to determine how well the network is performing.

3.5.1 Determining Whether the Network Needs Tuning

To determine whether the network needs tuning, you must understand about the theoretical maximum throughput of an Ethernet network.

Richard Stevens, in his study *UNIX Network Programming*, presents a good summary of experimental data about network performance. Stevens notes that the theoretical maximum throughput rate for a thin Ethernet cable, using TCP/IP, and accounting for all network overhead, is 1.2 MB per second. Experiments using a variety of machines from IBM-PCs to Cray super computers have generated transmission results ranging from a low of 144,000 bytes per second to a high of 890,000 bytes per second.

Network performance can be stated as a percentage of the theoretical maximum; that is, 144,000 bytes per second represents 12 percent of the maximum and 890,000 bytes per second represents 74 percent of the maximum. If you compute the throughput rate for your network, and it falls somewhere within this 12-74 percent range, you will have to determine the minimum acceptable level.

3.5.2 Measuring the Network Subsystem Throughput Rate

One way to measure your network's throughput rate is to send a file of known size from one node on the network to another and to compute the time needed from start of transmission to completion. To avoid overhead on the receiving node's end, copy the file to `/dev/null` rather than to the file system. The following example shows a 1 MB file that is transmitted to two different nodes in a small LAN. The time to complete the transmission is computed by `/bin/time` in order to get seconds and tenths of seconds in real-time.

```
csh> /bin/time rcp huge.file Opus:/dev/null
9.4 real                0.1user          4.6 sys
csh> /bin/time rcp huge.file Alpha:/dev/null
14.7 real                0.1user          2.7 sys
csh> /bin/time rcp Opus:/tmp/huge.file /dev/null
12.7 real               0.1 user          4.3 sys
csh> /bin/time rcp Alpha:/tmp/huge.file /dev/null
14.7 real               0.1 user          2.7 sys
```

The transmission rate computed in the previous example averages only 93,204 bytes per second, 7.8 percent of the theoretical maximum. The three nodes in this example are running three different versions of UNIX. The node `Tinsel`, the originator of the remote copy, is running ULTRIX. The other two nodes, `Alpha` and `Opus`, are running different versions of UNIX, and consequently, each node is running a different implementation of TCP/IP. Both `Alpha` and `Opus` are PCs; `Alpha`'s clock speed is 16 MHz and `Opus`' is 20 MHz. Thus, the difference in transmission rates between the two (about 2 seconds) is

probably attributable to the different clock speeds. The overall slowness of the transmissions also is likely due to the slowness of the receiving nodes.

One source of problems in communications over the network is transmission errors resulting in the retransmission of packets. To determine whether problems are being encountered during transmissions, you can run `netstat` to collect information about packets transmitted and received, and about collisions and errors. The following example shows the output from the `netstat` command.

```
csh> netstat 5
   input    (se0)     output           input   (Total)     output
packets  errs packets  errs colls packets  errs  packets errs colls
      0     0       0     0     0       0     0        0    0     0
      2     0       4     0     0       2     0        4    0     0
    413     0     222     0     0     413     0      222    0     0
    409     0     223     0     0     409     0      223    0     0
    225     0     124     0     0     225     0      124    0     0
      0     0       0     0     0       0     0        0    0     0
      1     0       0     0     0       1     0        0    0     0
```

3.6 Identifying Interprocess Communication Limitations

The ULTRIX (or UNIX) operating systems provide a variety of facilities to communicate between processes and to coordinate resource usage of those processes. These facilities include shared memory and semaphores (in UNIX System V), and sockets (in BSD UNIX and ULTRIX).

The usefulness of these facilities depends upon the number available, and on their throughput.

3.6.1 Determining Whether the Interprocess Communications Subsystem Needs Tuning

One set of limitations in the interprocess communication (IPC) subsystem is obvious; the other is less so. Limitations in the number of any component will be announced by the operating system. If, for example, one of your processes tries to create a semaphore set and all semaphores allowed by the system are in use, your process will fail and the system will display a message that indicates there is no space left on device.

If the number of pipes, sockets, messages or other IPC resources is within the system–wide resource limit, the other possible limitation is the throughput rate of the resource. Thus, a measure by which you can recognize limits in throughput rate of the IPC subsystem is needed. To identify whether an IPC subsystem is in need of tuning, you can maximize throughput as measured by bytes per second for pipes, messages, or sockets, or maximize the number of semaphore operations per second.

Deciding whether the IPC subsystem needs tuning consists of two steps:

- Determine whether system-wide resource limits are being exceeded
- Determine whether the throughput rate is acceptable

3.6.2 Measuring the Interprocess Communications Throughput Rate

You can construct a measure of the IPC subsystem's throughput rate with which to compare performance under a variety of scenarios, from a lightly loaded system to a heavily loaded one. If you find a marked difference in performance under these scenarios, you can begin to look for the source of these differences. However, if the source of the differences appears to be in the kernel itself, very little can be done to improve interprocess communications subsystem performance without access to the kernel source code.

Example 3–6 and Example 3–7 show output from two simple programs that test interprocess communications. Appendix B contains source code for these two programs. In the first example, the program sets up a message queue with a buffer size of 128 bytes and transmits the message to itself 20,000 times. In the second example, a semaphore set is created; the value of each member is set to 1; and the value of the semaphore is retrieved by itself 20,000 times. Running these simple programs and timing them enables you to compute a transmission rate of bytes per second in the first example, and semaphore operations per second in the second example.

Example 3–6: Testing the Message IPC Subsystem

```
csh>  /bin/time msgtest
     26.3 real          1.5 user      24.0 sys
```

Example 3–7: Testing the Semaphore IPC Subsystem

```
csh>  /bin/time semtest
     18.9 real          2.3 user      15.8 sys
```

Example 3–6 shows that the message transmission rate is 97 KB per second (20000*128/26.3), but only 760 messages are transmitted per second (20000/26.3). Example 3–7 shows the number of semaphore operations per second is just over 1,000 (20000/18.9). Normally, a semaphore might be used in conjunction with shared memory where data is shared between two processes. Thus the faster signaling rate of semaphores versus messages is slightly illusory.

By themselves, these numbers have little meaning. However, you can make these calculations for a machine when it is under a variety of loads to determine whether some circumstances exist in which it performs less well than others. Also, you can compare performance between workstations to determine whether one workstation performs better in some situations than in others.

3.7 Chapter Summary

The objective in this chapter has been to examine situations in which performance of one of the ULTRIX subsystems might be less than optimal and how to identify those situations. In each instance, examples have been used to illustrate problems that you might encounter and to show the commands you can use to generate data with which to understand the problem.

Several suggestions are offered to assess the performance of the system being tested. In most cases, deciding whether a system needs tuning is a matter of judgment that will be developed over time. Use the suggestions presented here as the starting point to identify suspected problems. Tuning solutions are presented in Chapter 4.

4 Tuning Subsystem Resource Usage

This chapter presents some concrete examples of particular tuning techniques. In particular, it discusses the contents of the configuration file, reviews each of the major subsystems, and provides suggestions on how to improve performance for some selected conditions.

The tuning methodology requires that you take the following steps:

1. Run a benchmark on the workstation to establish how it performs under a known load
2. Modify global parameters in the configuration file, or you can modify other factors that affect the system's performance
3. Rerun the benchmark so that you can observe changes in system performance

It is important to learn the recommended methodology so as to understand performance issues. Troubleshooting and making performance improvement are skills that require insight, an understanding of the system, and an ability to set up tests that highlight performance problems. It is an exercise in experimentation: make a change, observe the impact on performance, and try something else if the desired objective is not achieved.

4.1 System Configuration File

The system configuration file contains two primary elements:

- Instructions used by the utility `/etc/config` to build
- Dependencies used to compile and link a new kernel

You can build a new kernel automatically by using the utility `/etc/doconfig`, or manually by using `/etc/config` and a series of other steps. For more information about the procedure used to build a new kernel, see the *Guide to Configuration File Management.*

4.1.1 Global Definitions

You can locate global definitions in the first part of the configuration file. Global definitions convey information to the kernel building process in the following ways:

- Flags to include some optional code segments
- Indicators of physical features of the machine or its environment
- Values of parameters that can be set optionally at compile time

Examples of parameters used as flags are machine and cpu. The machine parameter defines the hardware architecture of the workstation, the parameters vax or mips, for example. The cpu parameter describes the processor class or processor name.

Parameters that describe the physical features of the machine or its environment are as follows:

- ident
  The ident parameter must have the same name as the host name which the system uses during system installation. The host name is used by procedures residing in the kernel, for example, mail routing and network access.

- processors
  The value of processors defines the number of CPUs making up the system.

- scs_sysid
  The value of scs_sysid identifies each host uniquely on the CI star cluster to the SCS subsystem.

- timezone
  Setting the timezone allows the system clock to display the correct time (standard time and daylight savings time) for your location.

The remaining global definitions in the configuration file are used to compute sizes of various internal kernel tables or to set minimum and maximum sizes of several memory allocation parameters. These parameters are maxusers, maxuprc, maxuva, bufcache, swapfrag, maxtsiz, maxdsiz, maxssiz, physmem, smmin, smmax, smseg, smbrk, and smsmat. The parameters maxdsiz and maxssiz in ULTRIX Version 4.0 replace the parameters dmmin and dmmax in earlier versions.

The maxusers parameter is used in several computations, but most importantly it is used to compute the size of the system process table. The maxuprc parameter controls the maximum number of processes one user can run simultaneously. An internal algorithm using the values of maxusers and maxuprc determines the size of the system process table. The size of the system process table controls the maximum number of processes that can be run at any one time.

The parameters maxuva, maxtsiz, maxdsiz, maxssiz, smmin, smmax, smseg, smbrk, and smsmat are all concerned with different aspects of the manner in which memory is allocated. The maxuva and maxtsiz parameters set system–wide virtual address space for users and the size of the largest text segment, respectively. The maxdsiz parameter defines the largest data segment, in megabytes, allowed by the system; the default value is 32 MB. The maxssiz parameter defines the largest stack segment, in megabytes, allowed by the system; the default is 32 MB. The parameters beginning with the sm prefix control various aspects of shared memory.

The physmem parameter sets the amount of the system's physical memory. The system uses this when the kernel is built to determine the size of system page table. Set the physmem parameter value equal to the actual amount of physical memory. Setting physmem larger or smaller than the size of physical memory decreases the amount of memory available to applications.

The parameter bufcache is used to change the size of the buffer cache when the kernel is built. The default value of 10 percent is used if bufcache does not appear in the configuration file. The parameter swapfrag satisfies requests for additional swap space using the value supplied; the default is 64. Thus, a process requesting additional swap space will be allocated 64 512-byte blocks.

The last set of parameters appearing in the global definitions are options flags. By specifying these flags, different optional sections of code are linked with the kernel when built. These options include the parameters INET and DECNET for internet communications protocol and the DECnet layered product, respectively. If the kernel being built does not implement some of these options, you can delete their options flags in the configuration file. The text size of the resulting kernel is smaller.

4.1.2 System Image Definitions

The system image definition section of the configuration file consists of lines beginning with the keyword config. A separate kernel is generated for each config line. Normally, only one config line exists in the configuration file. The following keywords control the generation of the system image:

- root specifies the disk partition on which the root file system is found
- swap identifies the disk partition or partitions which are used for the swapping and paging area
- dumps specifies the disk partition on which crash dumps are stored

4.1.3 Device Definitions

The device definitions section of the configuration file contains a line describing each hardware component connected to the system or hardware that can be connected to the system in the future. That is, the device specified in the configuration file does not have to be present. The keywords that define the device types are adapter, master, controller, device, disk, and tape. The format for fields in each of the device definitions is dependent on the device type.

4.1.4 Pseudodevice Definitions

The pseudodevice definitions section of the configuration file is the link between the device definitions section, which specifies the hardware controllers, and the equipment connected to those controllers. The pseudo-device keyword is followed by a name that specifies the identifier for the device driver. The device driver code is compiled (if necessary) and linked with the kernel to give access to that device. The doconfig utility also uses this section to create the device-special files needed by the kernel to interface with the hardware attached to the system.

4.2 Tuning the File System

Chapter 3 examined several situations in which the file system appeared to need tuning. You can take four specific actions to address the problems identified there:

- Reorganize the file system or move it to a different disk partition
- Change the disk partitions
- Add swap partitions
- Change the size of the buffer cache

The following sections examine examples of each of these actions.

4.2.1 Reorganizing the File System

As an example of different levels of performance achieved by manipulating the file system, consider the following experiment. A test was conducted with a VAXstation 2000 with one RD54 disk and one RD53 disk. A standard installation was followed so that the root file system resides on the RD54's a-partition, the b-partition is used for paging and swapping, and the g-partition is used for system software and the various supported and unsupported packages that can be installed. This configuration leaves less than 20 MB free on the RD54's g-partition, so the RD53 is used exclusively for the user file system.

Table 4–1 shows the default partition table for the RD53. The a-partition is reserved for a backup `root` file system. This leaves sectors 15884 through 138671 available for user file systems. Two partitions have no mutual overlap, partitions b and g; the first demonstration uses these partitions.

Table 4–1: Default Partition Table

Partition	Bottom	Top	Size	Overlap
a	0	15883	15884	c,d,e
b	15884	49323	33440	c,e,h
c	0	138671	138672	a,b,d,e,f,g,h
d	0	0	0	a,c,e
e	0	50713	50714	a,b,c,d,g,h
f	50714	138671	87958	c,g,h
g	49324	138671	89348	c,e,f,h
h	15884	138671	122788	b,c,e,f,g

The b-partition is relatively close to the beginning of the disk, whereas the g-partition encompasses the physical center of it. The first test demonstrates the impact of locating a file system with a large amount of disk traffic close to the physical center of the disk. As mentioned in Chapter 3, when disk statistics indicate that the read/write heads spend excessive time moving around the disk to locate information (the number of milliseconds per seek is large, relative to the manufacturer's advertised average milliseconds per seek), moving frequently accessed files closer to the physical center of the disk results in reduced seek times.

The first set of test results, shown in Table 4–2, demonstrates performance measures in a simulated multiuser environment derived from AIM Technologies AIM Benchmark Suite III tests, Version 1.4 dated 12 October 1987 (the version number and date are from the source code file `multiuser.c`).[1] The benchmark allows you to specify a mix of subsystem tests to model your own environment. The first set of tests are intended to reflect an environment in which a large amount of file system activity exists. Thus, the benchmark is run with weightings of 50 percent disk access, and 10 percent each of the `ram`, `float`, `pipe`, `logic`, and `math` subsystem tests for 10 simulated users.

Table 4–2 shows the results of Test 1.

Table 4–2: Reorganizing the File System

Test	Processes per Second	System Time	Sys+User Time	Real Time
1	0.028458	1076.2	1757.0	1861.0
2	0.028199	1080.5	1773.2	1891.7

All the benchmark software and all files created by the benchmark reside in the `g`-partition. In Test 2, all the files created by the benchmark are created in the `b`-partition. The files created are all 1 MB; 10 simulated users are active, in addition to the normal system activity. Locating test activity in the `g`-partition results in a small improvement over that achieved when disk test activity occurs in the `b`-partition. The number of processes per second is 1 percent higher in Test 1, while the real time to complete the test is almost 30 seconds less, or 1.6 percent smaller. The file system in both tests was unpopulated except for the test software and the files created. Had the file system been more populated, a larger improvement would have been observed.

4.2.2 Changing the Size of Disk Partitions

If large disks, or a large number of disks, are available, it is possible to spread disk activity out over multiple disks or to locate file systems with a large amount of disk activity on partitions close to the physical center of the disk. If this luxury is not present, you can consider reorganizing a disk by changing the size of the disk partitions. Changing partition sizes is not recommended except in extreme situations, because it makes maintaining the disk more difficult. However, in some circumstances, no other choice is available.

Table 4–3 shows an example in which the `e`, `f`, `g`, and `h`-partitions have been redefined. The intent is to break the existing `g`-partition into two smaller partitions — one where the benchmark software resides on the outermost

[1] Several modifications were made to the software to improve the validity of results bearing on the tests being run. They include the following: (1) Statistics are averages computed over all simulated users (rather than the results of the first simulated user only); (2) File names, used to create and test reading or writing of files, have been modified to guarantee that unique file names are used; (3) An option was added to increase the size of files that are created from 250 KB to 1 or 2 MB in order to more fully test the handling of large files; (4) the algorithm used to select the random order of the test to be run was changed from one that uses samples with replacement to one that uses samples without replacement in order to guarantee that the desired workload and the sample workload are statistically equivalent.

part of the disk, and a smaller inner partition closer to the center of the disk where frequent file activity takes place.

Table 4–3: Partitions e, f, g, and h Redefined

Partition	Bottom	Top	Size	Overlap
a	0	15883	15884	c,d,g,h
b	15884	49323	33440	c
c	0	138671	138672	a,b,d,e,f,g,h
d	0	0	0	a,c,g,h
e	79924	138671	58748	c
f	49324	79923	30600	c
g	0	0	0	a,c,d,h
h	0	0	0	a,c,d,g

Example 4–1 shows the commands that repartition the disk.

Example 4–1: Commands that Change Partitions e, f, g, and h

```
csh> chpt -pe 79924 58748 /dev/rd0a
csh> chpt -pf 49324 30600 /dev/rd0a
csh> chpt -pg 0 0 /dev/rd0a
csh> chpt -ph 0 0 /dev/rd0a
csh> chpt -q /dev/rd0a
```

Note

Changing partitions that contain file systems requires that you back up and remake the file system once reconfiguration is complete.

The same benchmark tests reported in Table 4–2 were rerun; however, now files were created, written, and read in the `f`-partition, as defined in Table 4–3. Table 4–4 shows the results of that test run, where the results of Tests 1 and 2 are repeated for reference. Note that the number of processes completed per second is larger by 1.4 percent than they were in Test 1, and larger by 2.5 percent than they were in Test 2.

Table 4–4: Further Reorganizing the File System

Test	Processes per Second	System Time	Sys+User Time	Real Time
1	0.028458	1076.2	1757.0	1861.0
2	0.028199	1080.5	1773.2	1891.7
3	0.028913	1062.8	1729.3	1840.9

Similarly, the times required to complete the tests are less. Real Time in Test 3 is 1840.9 seconds (1.1 percent less than it was in Test 1 and 2.7 percent less than it was in Test 2).

4.2.3 Adding a Second Swap Partition

The rationale for adding a second swap partition is based on the concept that if you spread disk activity over several disks on a heavily loaded system, you can improve performance. Performance would improve because the kernel is not forced to wait until the first disk is ready for the next chunk of data. Instead it can transfer data to the second disk while the first disk is still busy handling the first chunk of data. This presupposes that the system is busy enough for a delay in access to the first swap partition to create a bottleneck for the processor.

To add another swap partition to your system, you must perform two tasks:

- First, in the configuration file, you must change the system image definition to include the second swap partition.

  The following shows the old swap configuration file line:

  ```
  config vmunix root on rd1a swap on rd1b dumps on rd1b
  ```

  The following shows the new line with the secondary swap partition added:

  ```
  config vmunix root on rd1a swap on rd1b and rd0b dumps on rd1b
  ```

- Second, include a line in the `/etc/fstab` file to use the second swap partition. When the system is booted, it reads the `fstab` file to determine which partitions to mount and where to mount them in the file system. An example of the `fstab` file is shown in Example 4–2, before the command to mount the second swap partition is inserted, and after it is inserted.

Example 4–2: File System Mount Table

```
OLD:

/dev/rd1a:/:rw:1:1:ufs::
/dev/rd1g:/usr:rw:1:2:ufs::
/dev/rd0g:/usr/tools:rw:1:3:ufs::

NEW:

/dev/rd1a:/:rw:1:1:ufs::
/dev/rd1g:/usr:rw:1:2:ufs::
/dev/rd0b::sw:0:0:ufs::
/dev/rd0g:/usr/tools:rw:1:3:ufs::
```

After adding a second swap partition, Tests 1 and 3 were run again. Table 4–5 summarizes the results of those tests. Test 1(a) is the same as Test 1 in Tables 4–2 and 4–4. Test 3(a) is the same test as Test 3 in Table 4–4. A small improvement is evident in Test 1. The number of processes per second increases, and the real time to complete the test decreases. The improvement in Test 3 is mixed. Processes per second falls slightly, but real time to complete the tests also falls. The files being created in Test 3 are close to the physical center of the disk, so some improvement over Test 1 has been realized already. These tests indicate that some marginal improvement is still possible, but that adding a second swap partition does not gain much.

Table 4–5: File System Tests with Two Swap Partitions

Test	Processes per Second	System Time	Sys+User Time	Real Time
1(a)	0.028458	1076.2	1757.0	1861.0
2(b)	0.028566	1077.5	1750.3	1860.2
3(a)	0.028913	1062.8	1729.3	1840.9
3(b)	0.028852	1063.5	1733.0	1838.5

4.2.4 Changing the Size of the Buffer Cache

The final set of tests indicates the impact of changing the size of the buffer cache. Because the operating system tries to delay writing disk blocks to disk (a slow operation) until necessary, a large buffer cache can translate directly to less frequent disk accesses. If a file is small, relative to the size of the buffer cache, the operating system may never have to write the file to disk until it is closed. A lower limit to the frequency with which the operating system accesses the disk is determined by its write-scheduling policy.

Usually a sync is done every 30 seconds. In ULTRIX Version 4.0 and later, this interval can be tuned by running `/etc/update` *n*, where *n* can be as large as 600 (10 minutes). You can force more frequent accesses in the workload by increasing file sizes that are read or written.

You must edit the configuration file to increase the size of the buffer cache to 20 percent of total physical memory. The standard configuration file has no entry for the buffer cache, so you must add it. In the global parameters section of the file, add a `bufcache` line as depicted in Example 4–3. Rebuild the kernel and reboot the system to activate the new, larger size buffer cache.

Example 4–3: Adding bufcache to the Configuration File

```
ident TINSEL
timezone 5dst
physmem 16
bufcache 20
maxusers 16
```

Table 4–6 summarizes tests done with three different-sized buffer caches and for three different file sizes. The size of the buffer cache is increased

from 10 percent to 20 percent and 30 percent of memory. Three different file sizes are used, 250 KB, 1 MB, and 2 MB. The number of simulated users is reduced from 10 to 5 because creating files as large as 2 MB leaves insufficient disk space to run the tests.

Table 4–6: Test with Different Sized Buffer Caches

Test	Processes per Second	System Time	Sys+User Time	Real Time
250 KB Files				
4(a)	0.0565	534.1	884.9	921.7
4(b)	0.0563	532.3	887.5	919.7
4(c)	0.0566	531.6	884.1	921.5
1 MB Files				
5(a)	0.0443	623.8	1128.3	1200.6
5(b)	0.0446	622.0	1121.4	1187.2
5(c)	0.0454	621.1	1101.1	1156.6
2 MB Files				
6(a)	0.0325	743.4	1536.6	1651.3
6(b)	0.0332	744.7	1507.3	1630.4
6(c)	0.0333	741.1	1503.7	1644.4

In the first set of tests, with file sizes of 250 KB being created and five processes running, the size of the buffer cache needed could be as large as 2.5 MB. The buffer cache as configured is only 1.6 MB (10 percent of 16 MB real memory). However, it is unlikely that all five processes access their 250 KB files at the same time, and thus it is unlikely that the system will exceed buffer cache capacity. However, with 1 and 2 MB files it is much more likely that the system will exceed the size of the buffer cache, unless it is increased to 20 percent or 30 percent of memory.

Tests 4(a), 5(a), and 6(a) in Table 4–6 show results when the size of the buffer cache is 10 percent of total physical memory. Tests 4(b), 5(b), and 6(b) represent tests results when the buffer cache is increased to 20 percent of system memory. Finally, Tests 4(c), 5(c), and 6(c) show the test results when the buffer cache is increased to 30 percent of system memory.

The results of these tests show that if you increase the size of the buffer cache, system performance improves slightly for any of the test file sizes. The number of processes completed per second is larger in every case for a buffer cache of 30 percent of system memory as compared to a buffer cache of 10 percent or 20 percent. These may not seem like significant improvements, but on a heavily loaded system, the increased level of performance may be even larger. The best recommendation is to experiment and to see what works best in your situation.

4.3 Tuning Process Control and Scheduling

To improve the performance of the process control and scheduling subsystem without access to kernel source code, you must rely primarily on system management. You can do this in several ways:

- Balance the workload of the system over time.
- Change two global parameters that govern the size of the process table and the number of processes one user can run.
- Change the priority of a process.
- Set the sticky bit for a process that is executed often by many users.

4.3.1 Balancing the Workload

By moving less critical jobs to periods of the day when the system is lightly loaded, you can improve throughput for users who run during those peak demand periods.

You can use the `at`(1) and `cron`(1) commands to execute programs at a specified time. For more information about these commands, see the *ULTRIX Reference Pages*.

4.3.2 Changes Involving Global Parameters

Global parameters that indirectly affect the process control and scheduling subsystem are `maxusers` and `maxuprc`. The size of the process table is calculated from the value of `maxusers` and `maxuprc`. Have the value for the `maxusers` parameter approximate the number of users and processes logged on to the system at any one time. If `maxusers` is too large, the kernel becomes unnecessarily large, which results in less available memory; if it is too small, not all users and processes are able to log on.

The `maxuprc` parameter also limits the number of processes an individual user may have running simultaneously. Setting a limit may prevent an user from monopolizing or swamping the system with too many processes. However, setting the number of processes too low can result in a task not being completed.

4.3.3 Changing the Priority of a Process

To demonstrate the effect of changing relative priorities of simulated users, the same set of tests used in Section 4.2 are used again. As before, the workload comprised of 50 percent from file system activity and 50 percent from activity of other tests. This set of tests used 2 MB file sizes for files being created, written, and read.

After a baseline was established the tests were rerun with another benchmark running, the SPEC 001.gcc1.35 benchmark. This benchmark was chosen because it takes approximately the same amount of time to complete as the AIM Benchmark Suite III (30 minutes clock time), and it heavily loads the system itself. With both benchmarks running, system idle time falls from at most 10 percent to 0 percent. Paging activity increases, but no swapping occurs.

Finally, both benchmarks were rerun; however, now the SPEC benchmark ran with a `nice` value of `+10`. That is, the priority of the SPEC benchmark is reduced by 10 percent. Tests involve comparing the running times and

processes completed per second for the AIM Benchmark Suite III under each of the three scenarios. The following example shows the command to start the SPEC benchmark with a reduced priority. The `nice +10` command tells the system to increase the `nice` value for the command that follows it, and the `make` command starts the SPEC benchmark.

```
csh> nice +10 make IDENT=vax VERSION=1.0 001.gcc1.35
```

Table 4–7 shows the results of the three tests. Test 1 shows the other system activity beyond normal background tasks. Tests 2 and 3 are the results for the AIM Benchmark Suite III when the SPEC benchmark is run at normal priority and then after its priority has been reduced by 10 percent. The system slows down when the SPEC benchmark is added. However, when the process priority of the SPEC benchmark is reduced by 10 percent, process statistics for the AIM Benchmark Suite III return to the level achieved when the system was under no additional load.

Table 4–7: Impact of Changing Process Priorities

Test	Processes per Second	System Time	Sys+User Time	Real Time
1	0.028667	781.9	1745.4	1904.4
2	0.025841	771.4	1937.8	2079.5
3	0.028730	764.0	1741.7	1879.7

4.3.4 Setting the Sticky Bit for a Frequently Executed Process

Another method by which you can improve a system's performance is to set the sticky bit for processes executed often by many users. When the AIM Benchmark Suite III runs, it forks 31 different subsystem exercisers serially in a random order determined by an internal algorithm. Each of these processes is run a number of times, depending on the weighting applied to it. The weighting is dependent on the workload mix that is tested. Thus, even when it simulates only five users, it is possible that some of the subsystem exercisers will run 50 or 100 times during the 30 minutes needed to complete the test.

Example 4–4 shows the command to set the sticky bit. The subsystem exerciser `disk_cp` is one of the 31 subsystem exercisers mentioned. The individual numbers in the change mode command `chmod` are as follows:

1 Set the sticky bit
7 Set read, write, and execute privileges for the owner
5 Set read and execute privileges for group
5 Set read and execute privileges for others

Example 4–4: Setting the Sticky Bit for a Command

```
csh> chmod 1755 disk_cp
```

Table 4–8 shows the results of the tests with the sticky bit set. Tests 1 and 2 are the same tests listed in Table 4–7 (while set for all of the subsystem exercisers). Note that the system's performance is improved with the sticky bit set; however, improvement is not as great when the priority of the SPEC benchmark was reduced by 10 percent.

Table 4–8: Impact of Setting the Sticky Bit

Test	Processes per Second	System Time	Sys+User Time	Real Time
1	0.028667	781.9	1745.4	1904.4
2	0.025841	771.4	1937.8	2079.5
4	0.026834	764.8	1863.3	2018.6

4.4 Tuning Memory Management

Like the process control and scheduling subsystem, you must have access to kernel source code to tune the memory management subsystem. However, four global parameters in the configuration file have a direct impact on how system memory is used. In addition, the `bufcache` global parameter affects memory management, because as the size of the buffer cache increases, less memory is available for executing programs. The following sections consider the impact on system performance when you change each of these global parameters.

4.4.1 Changes Involving `maxuva`, `maxtsiz`, `maxdsiz`, and `maxssiz`

Four global parameters, `maxuva`, `maxtsiz`, `maxdsiz`, and `maxssiz`, control the way in which active virtual memory is allocated to running processes. The parameters `maxuva` and `maxtsiz` set system–wide maximums for memory allocated to virtual address space and the largest text segment for a user's program. Unless a system is being so heavily used that it exhausts one of these constraints, no reason exists to change either.

The parameters `maxdsiz` and `maxssiz` control the way memory is allocated for individual users' processes. The value of `maxdsiz` determines the maximum size of the data segment that one of your processes can allocate. The parameter `maxssiz` sets the maximum stack size that one of your processes can address. These parameters replace `dmmin` and `dmmax` in versions of ULTRIX earlier than Version 4.0.

You can use the `limit` built-in command of the C-shell to display the current settings of the data, the stack region size, and the stack size. The following example shows output from the C-shell built-in command `limit`:

```
csh> limit
cputime   unlimited
filesize  unlimited
datasize  83936 kbytes
stacksize 512 kbytes
coredumpsize unlimited
memoryuse unlimited
```

For some programs using a large amount of temporary data storage, a stack size of 512 KB can be insufficient. The following example shows how individual users can adjust the stack size by increasing it with the `limit` command. However, the largest data–plus–stack region is shown by `datasize` and an individual user cannot increase it. Users can decrease their data region size. If you need a larger data–plus–stack region, you must relink the kernel using a larger value for `maxdsiz`.

```
csh> limit stacksize 4096
csh> limit
cputime   unlimited
filesize  unlimited
datasize  83936 kbytes
stacksize 4096 kbytes
coredumpsize unlimited
memoryuse unlimited
```

4.4.2 Changes Involving the Size of the Buffer Cache

For a system with a predictable workload, an optimal size buffer cache exists. You can find the optimal size of the buffer cache only by experimentation. However, remember that as more memory is used for the buffer cache, less is available to run programs. Thus, system performance improves for a while as the size of the buffer cache increases; however, it declines again as the size of the buffer cache increases beyond the optimal size, because less memory is available for the executing programs.

Table 4–9 shows the results of an experiment to find the optimal size for the buffer cache. Using the AIM Benchmark Suite III simulating 15 users and simultaneously running five of the SPEC benchmarks, performance statistics for the simulated users are collected.

The results indicate a noticeable improvement in execution times and processes completed per second as the size of the buffer cache increases to 20 percent of system memory. After that, system performance begins to fall fairly steadily as the buffer cache is increased to 70 percent of system memory (the largest allowed by system configuration). However, not until the buffer cache increases to 60 percent does system performance fall below its level when the buffer cache was 10 percent of system memory.

Table 4–9: Impact of Increasing the Size of the Buffer Cache

Test	Processes per Second	System Time	Sys+User Time	Real Time
10%	0.018396	1414.2	2718.0	2892.0
20%	0.020190	1403.8	2476.5	2639.6

Table 4–9: (Continued) Impact of Increasing the Size of the Buffer Cache

Test	Processes per Second	System Time	Sys+User Time	Real Time
30%	0.018814	1407.9	2657.5	2842.5
40%	0.018806	1404.6	2658.7	2824.0
50%	0.018744	1406.3	2667.5	2852.3
60%	0.017298	1404.9	2890.4	3099.3
70%	0.016884	1417.5	2961.4	3161.1

4.4.3 Changes Involving Physical Memory and `physmem`

As a final method to improve performance of the memory management subsystem, you can add more memory. Before you embark on this more expensive means to improve system performance, be sure that you explore less costly methods:

- Adjust the size of the buffer cache so that active virtual memory is no more than 80 percent of real memory
- Adjust users' data segment size to a size sufficient for normal user processes

If neither of these adjustments frees enough memory to improve system performance, additional memory may be the only answer.

Note that the global parameter `physmem` does affect system memory utilization. That is, if you change its value, it can effect on the amount of memory the system has or recognizes. It does affect the size of the page table when the kernel is built. Unless testing is being done, set its value to be equal to the total amount of real memory attached to the system for the following reasons:

- If you set `physmem` smaller than the total amount of physical memory on the system, the system does not use a portion of physical memory that equals the difference between two values.
- Setting `physmem` larger results in an unnecessarily large system page table that requires more memory.

4.5 Tuning the I/O Subsystem

Tuning the I/O subsystem requires access to kernel source code. As suggested in Chapter 3, I/O subsystem performance can appear less than it should be, and some objective measures of performance can be developed. However, beyond monitoring software that runs on the system and paying attention to the hardware purchased (terminals, tape drives, and so on), the primary means to improve I/O subsystem performance is system management.

4.5.1 Changes Involving Software

Programs that interface with the I/O subsystem should do so in the most efficient manner possible. In general, reads and writes should avoid direct dependence on the system call interface, unless dictated by sound design reasons, and should use buffered I/O routines.

4.5.2 Changes Involving Hardware

If an evaluation indicates that software is not the source of performance problems in the I/O subsystem, look at the I/O hardware. Monitor throughput using the `iostat` command (described in Chapter 2), or some variation on the technique described in Chapter 3 to measure the throughput for a tape drive. You must obtain a statistic that is comparable to the metric that the manufacturer uses to describe throughput of the I/O hardware unit in question. If the measure is significantly different from the one in the manufacturer's literature, contact the manufacturer's hardware technical support personnel. However, if the hardware unit is performing as expected, you might consider replacing it with a faster model.

4.6 Tuning the Network

You can improve network performance in the following ways:

- Monitoring traffic on the network
- Calculating throughput for test message packets between different nodes on the network
- Isolating links where the throughput is less than acceptable

You need access to kernel source code to change the way network communications are handled on individual machines in software. Thus, tuning is primarily an exercise in the following:

- Evaluating hardware links
- Monitoring software that communicates over the network
- Balancing loads on machines connected on the network

Chapter 2 describes a number of utilities you can use to test the network.

4.6.1 Changes Involving Global Parameters

None of the global parameters in the configuration file have any connection with the network subsystem. In the options portion of the configuration file, be sure that the flags `INET`, `DECnet`, or both are included if the system requires either the Internet Communication Protocols or the DECnet layered product. The `LAT` flag allows you to access the machine from a local area terminal server. Remember that including these flags causes the code to be linked, which implements these features when the kernel is built. If you do not include these flags, the related service is not available.

4.6.2 Changes Involving Software

Although you cannot change the way programs communicate over the network without access to source code, you can attribute some problems to improper configuration of the network. TCP/IP configures itself from a number of files stored on the `/etc` directory when the network is started. Be sure the following files are configured correctly:

- The `hosts` file contains names of all of the other known nodes on the network, as well as any aliases by which those nodes are known.
- The `hosts.equiv` file lists trusted hosts, nodes on which users have reciprocal privileges. Note that for this feature to work, users must have the same login name and user ID on all systems. In addition, individual users can have a `.rhosts` file, which lists other nodes that have reciprocal login privileges for that user account.
- The `networks` file lists the host network's own address and network addresses of other networks on the internet to which the host's network has access.
- The `protocols` and `services` files are present, but are not changed under normal circumstances. Protocol names and numbers listed in the `protocols` file are in the `services` file, which contains four fields listing the following: the name of the service, its service number, the name of the protocol used by the service, and aliases by which the Network Information Center (NIC) specifies service names and numbers.

Next, verify that the network was started correctly. Have the `rc.local` file contain the two lines that configure the network; they look something like the lines in Example 4–5. The first line sets up the local loopback; the second sets the local host's broadcast address by which other machines on the network are able to find it.

Example 4–5: Network Configuration from `rc.local`

```
/etc/ifconfig lo0 localhost
/etc/ifconfig se0 Tinsel broadcast128.10.255.255 netmask 255.255.0.0
```

The `rc.local` file starts various network daemons.

Be sure that the `inetd` and `rwhod` daemons are running. The `inetd` daemon is the internet super server; it listens on multiple ports for incoming connection requests. The `rwhod` daemon is the server that maintains the database used by `rwho` and `ruptime` programs.

Depending on the particular system's setup, other daemons that may be running are as follows:

- `routed` (the network routing daemon)
- `sendmail` (the network mail routing daemon)
- `rwalld` (the network message server that notifies all network machines when its local host is going down)
- `biod` (the NFS asynchronous block I/O daemon (multiple copies) if NFS is brought up on your system)

After you verify that the network is configured correctly and was started correctly, check for protocol problems. To do this, you must isolate any problems in a packet trace. When you start the network daemons before a connection is established on a socket, turn on packet tracing with the `-d` option of the `inetd` daemon. The system traces all traffic and internal actions, writing the information to a circular trace buffer. You can examine this buffer by running the `trpt` utility.

The system's buffering requirement can be adjusted by changing the socket buffer size for TCP and UDP. TCP's buffer size parameters are `tcp_sendspace` and `tcp_recvspace`. The defaults are 4 KB. Most gain is measured going to 8 KB (40 percent). UDP's buffering size parameters are `upd_sendspace` and `udp_recvspace`. The default is 9 KB, which can be increased up to 60 KB.

4.7 NFS Performance Problems

NFS performance problems can be broken down into three basic areas: client, network, and server problems. The following sections describe each of these areas and show why the server and, in particular, the server's I/O subsystem usually are the primary causes of poor NFS performance.

4.7.1 Network Problems

The network used to communicate between the client and server does not normally cause a performance problem. There are, however, two conditions to look for: network delays and high retransmission rates. If the Ethernet is overutilized, clients experience long delays waiting for a free slot to send requests. Ethernet utilization over 50 percent often indicates excessive network delay.

Network topology often contributes to excessive delay. If clients are located across many gateways from the servers that they often use, their requests experience long delays. You may be able to solve the problem by restructuring the network topology to distribute the load more evenly.

Excessive retransmissions can cause poor performance because the client waits for the server to respond before it retransmits a request. The causes of excessive retransmissions include the following: overloaded servers that drop packets due to insufficient buffering, inadequate Ethernet transceivers that cause packets to be dropped under busy conditions, and physical network errors, such as those caused by a noisy coaxial cable.

You can use the `nfsstat -c` command to measure the NFS retransmission rate on client machines. The average NFS response time to a client request under a low to medium load is approximately 30 milliseconds. Most clients retransmit a request after approximately 1 second. If a 10 percent reduction in performance is acceptable, then a 3 millisecond increase in response time is an acceptable limit. This reduction gives an acceptable NFS retransmission rate of 0.3 percent. The calculation is as follows:

$$\frac{.003 sec./\text{request}}{1.0 sec./\text{retransmission}} = 0.003 \text{ retransmission/request}$$

Because the worst case NFS request (8 KB read requests) requires seven packets (one request and six fragmented replies), the error rate of the network must be less than 0.04 percent. The calculation is as follows:

$$\frac{0.3\%}{7} = 0.04\%$$

The calculation shows the overall acceptable error rate for both the client and the server, so the acceptable error rate measured at either machine is half of this rate (0.02 percent).

You can use the `nfsstat -i` command to measure the network error rate. If this rate is unacceptably high, determine whether an individual machine is generating an excessive number of errors. If the problem appears to be pervasive, analyze the cabling technology that is being used. For example, if you have difficulties with noisy nonstandard coaxial cable, you could switch to a twisted-pair Ethernet.

For more information about the `nfsstat`(8) command, see the *ULTRIX Reference Pages*.

4.7.2 Client Problems

Adding disks or memory to a client can improve performance in the following ways:

- By improving access time
- By reducing the overall load on the server and network

A client can avoid NFS performance problems for files that are not shared (such as root, swap, and temporary files) by using local disks for these files. For diskless clients, increased memory can make greatly improve performance by allowing the client to swap and page less often. By adding local resources, you can reduce the demands on the server and the network.

Although, it is easy to improve client performance by adding memory or disk, there are no simple rules to determine whether the improvements are cost effective. If local disks are used to hold valuable data, administrative activities, such as backing up the disks and sharing their data, can become a problem. However, adding resources to the server is often more cost effective.

4.7.3 Server Problems

The server's CPU can be a problem, but in an environment with a reasonably powerful server (a machine with more than 2 MIPS), the server is probably not a problem. A powerful server usually can keep up with the rate of NFS requests that a single Ethernet can deliver. A less powerful server (a machine with less than 2 MIPS) cannot keep up with the NFS requests, and even moderate loads can overwhelm the server's CPU.

On most NFS servers, the limiting factor is the speed of the disk. Most high-speed disks can sustain from 30 to 40 disk operations per second. Most of the time spent waiting for a disk operation occurs during head seeks or rotational delay. If you use a faster disk or disk controller, and if you spread the load over multiple disks, you can obtain a small improvement in I/O performance. However, the best way to improve I/O performance is to reduce the number of disk operations.

To alleviate performance problems, concentrate your resources on the server. If you have already added memory to your server to increase the size of the buffer cache and the server is still too slow, you can obtain another server and split the load between the two servers. However, not only does this solution have a large direct cost, but there is a significant administrative cost associated with supporting an additional server.

If your system supports it, Prestoserve (discussed in Chapter 3 and in Section 4.7.5) is an alternative solution that can increase the performance of the NFS server without an additional server and its added administrative cost.

4.7.4 NFS Server Performance

ULTRIX uses a buffer cache in memory to avoid disk operations whenever possible. This memory is effective in reducing the client waiting time for relatively slow disk I/O. It also makes disk I/O more efficient by allowing the staging and scheduling of disk operations.

You can obtain a gain in performance by allowing the disk device driver to schedule several requests at a time to take advantage of the position of the disk arm. The total amount of disk I/O is reduced, because repeat requests may be found in the cache. If NFS read activity is high, then adding more memory to your server can improve server performance because the size of the buffer cache is a percentage of the size of memory.

Performance problems at the server make the ULTRIX buffer cache inefficient when serving remote write requests. NFS uses a simple stateless protocol, which requires that each client request be complete and self-contained and that the server completely process each request before sending an acknowledgment back to the client. If the server crashes or if an acknowledgment is lost, the client will retransmit its request to the server. Because of this, the server cannot acknowledge the client's request until data is safely written to nonvolatile storage. Thus, the client knows exactly how much modified data has been safely stored by the server. Consequently, the server cannot cache modified data in volatile storage because the data may be lost if the server crashes.

The ULTRIX buffer cache cannot be used to improve performance with NFS requests that modify data. If the server cached modified data in volatile memory without writing this data to nonvolatile storage before acknowledging the request, a server crash would make that acknowledgment false. The users of the client may assume that its data is safely stored, but if a crash occurs, the data may be lost. Because a single server stores data for many clients, many clients can be affected. However, if modifications are always synchronously written to disk, data will not be lost, and you can easily recover from server crashes.

In addition to write operations, other NFS operations cannot be cached. Operations that modify data, such as file creation, file removal, and attribute modification, significantly add to the amount of file system data that the server must write synchronously to disk before responding. For example, when the client creates a new file, the server may have to update the data and file definition blocks for the directory that contains the file. To ensure file system integrity in the local case, these operations are also written synchronously to disk.

If NFS operations are synchronously committed to disk, a server can survive system failures because data integrity in ensured. However, performance is degraded because these operations take place at disk speeds and not at the memory speeds available to cachable operations. In addition, because these operations are processed serially, there is no opportunity to optimize the scheduling of the disk arm. Modifications to the cache are written synchronously to disk, so there is no opportunity to decrease write-disk traffic.

Unless your server is only supplying read-only access to files, some NFS operations must be synchronously committed to disk. Because disks are much slower than memory, this is a large burden.

4.7.5 Prestoserve's Impact on NFS Server Performance

Prestoserve's performance impact on any particular server can vary widely as a result of the demands placed on the NFS server by its client systems. Heavily loaded NFS servers, or those with a high percentage (more than 10 percent) of NFS writes, will benefit the most from Prestoserve. Conversely, when an NFS server is lightly loaded or performing a low percentage (less than 4 percent) of NFS writes, Prestoserve may have no noticeable impact on performance.

In addition to increased response time, Prestoserve provides for greater NFS server throughput because Prestoserve uses the server's disk more efficiently, and the disk throughput is usually the biggest NFS concern. For example, in many cases, Prestoserve allows you to double the number of diskless clients that a single NFS server can support if it has the necessary disk capacity and a sufficient amount of main memory. Prestoserve's improvement to an NFS server is most noticeable when the server is busy.

4.7.6 Recommendation for Increasing NFS Performance

The results of several NFS client/server performance tests prove that the performance of the Digital servers is improved dramatically by modifying the size of the buffer cache.

In most cases, results show that increasing the buffer cache improves performance; however, if the buffer cache is increased too much, performance can degrade. The optimum value for the buffer cache varies, depending on the type of load placed on the server and also the server under test.

You can use the `cache_bufcache` parameter in `param.c` to map the file system buffer cache into CPU cache. Good results have been found with NFS file servers.

You can increase the performance of Digital servers by adjusting the NFS daemons and the size of the buffer cache. It is important to realize that the optimum setting of the NFS daemons or the buffer file cache is dependent on the server load and the server.

A characterization study using the SECG Standard ULTRIX NFS Workload examined the way a server's performance was affected by the number of NFS daemons executing on that server.

Performing a standard ULTRIX installation procedure on a server results in four NFS daemons executing on the server. Changing the number of NFS

daemons varies the number of paths through the NFS server code, which affects the number of NFS requests that can be simultaneously processed.

Three ratios of server NFS daemons to a number of supported clients were tested: 1:1, 1:2, and 3:4. The 3:4 ratio can be used to form an ULTRIX NFS server tuning rule: Set the number of NFS daemons executing on the server equal to 75 percent of the number of clients supported by the server.

This can be achieved by using the following procedure:

1. Search the site-specific startup file `/etc/rc.local` for the following line:

```
/etc/nfsd 4; echo -n 'nfsd'
```

2. Change the value 4 to a value that observes the tuning rule.
3. Reboot the server.

This is a generic 80 percent/20 percent tuning rule. That is, 80 percent of a server's performance improvement will come from applying the 3:4 rule, and 20 percent will come from normal ULTRIX system management techniques applied to the server. Because the 3:4 ratio is probably workload dependent, it does provide customers with a starting point for tuning their NFS servers.

4.8 Tuning Interprocess Communications

To improve the performance of the interprocess communications subsystem, it is necessary to have access to kernel source code. Several global parameters in the configuration file control the use of shared memory, but otherwise tuning consists primarily of monitoring program development and program usage, and undertaking user education.

4.8.1 Changes Involving Global Parameters

The configuration file contains five global parameters that control the following:

- The minimum size of shared memory segment (`smmin`)
- The maximum size of shared memory segments (`smmin` and `smmax`)
- The maximum number of shared memory segments per process that can be created (`smseg`)
- The address offset at which a shared memory segment will be attached (`smbrk`)
- The largest address at which a shared memory segment can be attached (`smsmat`)

No parameters that affect messages, semaphores, or sockets are available.

The default maximum size of a shared memory segment is 128 KB. The `smbrk` parameter is important because it defines the offset from the end of the user's private data space at which a shared memory segment will be attached. Once shared memory is attached, the user's private data space cannot grow beyond the starting address of the shared segment

4.8.2 Changes Involving Software

Chapter 3 presented an example that showed a way to measure different throughput rates for two different methods of interprocess communications, one involving messages and the other involving semaphores. Although the example allows you to measure how rapidly a process is able to communicate with another, there is really nothing you can do to improve performance without access to the source code for programs that are communicating. Thus, performance improvements really hinge on software developers understanding the environments in which their software will run and designing the software accordingly. Communicating crucial needs to software developers is the most effective input from a user that will influence a workstation's performance

4.9 Chapter Summary

The objective of this chapter has been to give some examples where particular performance problems have been identified and corrected. Recognizing the problem generally requires setting up a benchmark or some other performance standard against which to compare the system before undertaking any changes and after making changes. In identifying system problems, it is critical that you are methodical in your approach to solve the problem and that you understand the tests used to verify the problem. This careful approach improves the chances of finding the problem and fixing it. Each subsystem is a little different, but this basic methodology will work with all of them.

Tuning your system can improve its performance. ULTRIX cannot be default tuned for all environments. In general, avoid excessive paging and swapping and I/O hardware saturation by using caching techniques and spreading I/O over multiple devices.

A DEVSTAT Source Code

The following program, `devstat.c`, gathers information on attached I/O devices. Specifically, it displays the device name, the device type, the interface the device uses, the device major and minor numbers, the device mnemonic, and the hard and soft error count. Other information, such as bus and controller information, can be obtained with modifications to the program.

```
/*
**   DEVSTAT:   @(#)devstat.c
**              Get information about generic device.
*/
#include <stdio.h>
#include <sys/param.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/file.h>
#include <sys/ioctl.h>
#include <sys/devio.h>
#include <sys/termio.h>

extern int errno;
extern int sys_nerr;
extern char*sys_errlist[];

#define  FALSE    0
#define  TRUE     1

#define  GENERIC  1
#define  TTY      2
#define  ERROR   -1
typedef union {
  struct devget DEVData;
  struct termcb TTYData;
 }DataType;
```

```
main(argc, argv)
int argc;
char *argv[];
{
 register int i, max;
 DataType: Data;
 struct devget *pDD;
 struct termcb *pTD;
 struct stat   *pDB, DevBuffer;
 int fd;
 short MajDev, MinDev, DevType;
 long DevMode;
 char DevName[MAXPATHLEN];
 pTD = &Data.TTYData;
 pDB = &DevBuffer;
 for (i =1; i < (max = argc); i++, errno = 0) {
 memset(&Data, 0,sizeof(Data));
 strcpy(DevName,argv[i]);
 if(stat(DevName,&DevBuffer) == -1) {
    terror(2); continue;
}
DevType = pDB->st_rdev;
MajDev  = pDB->st_rdev/256;            /* major/minor of fs */
if((MinDev  = pDB->st_rdev-(MajDev*256)) == -1)
continue;
DevMode = pDB->st_mode;
if ((fd = open(DevName, O_NDELAY))== -1) {
  terror(3);
  continue;
}
```

```
switch (devctl(fd, DevName, &Data)) {
  case GENERIC:
    iprintf(DevName, MajDev, MinDev, &Data);
  break;
  case TTY:
    iprintf(DevName, MajDev, MinDev, &Data);
  break;
  default:
    terror(5);
  }
  if(fd != -1)
  close (fd);
  }
}

iprintf(DevName, MajDev, MinDev, pGD)
char DevName[];
short MajDev, MinDev;
struct devget *pGD;
{
  printf("%-19.19s", DevName);        /* Raw device name           */
  printf("%hd  ",MajDev);             /* Major device number       */
  printf("%hd  ",MinDev);             /* Minor device number       */
  printf("%hd  ",pGD->category);      /* Category                  */
  printf("%hd  ",pGD->bus);           /* Bus                       */
  printf("%s   " ,pGD->interface);    /* Interface (string)        */
  printf("%s   ",pGD->device);        /* Device (string)           */
  printf("%hd  ",pGD->bus_num);       /* Bus number                */
  printf("%hd  ",pGD->ctlr_num);      /* Controller number         */
  printf("%hd  ",pGD->rctlr_num);     /* Remote controller number  */
  printf("%hd  ",pGD->slave_num);     /* Plug or line number       */
  printf("%s   " ,pGD->dev_name);     /* Ultrix device pneumonic   */
  printf("%hd  ",pGD->unit_num);      /* Ultrix device unit number */
  printf("%ld  ",pGD->soft_count);    /* Driver soft error count   */
  printf("%ld  ",pGD->hard_count);    /* Driver hard error count   */
  printf("%ld  ",pGD->stat);          /* Generic status mask       */
  printf("%ld\n",pGD->category_stat); /* Category specific mask    */
}
```

```
/*
**   If this is a tty we want to use a standard 'ioctl' call; however,
**   for generic devices (disk drives, tape drives, etc.) we will use
**   a the generic call to 'ioctl' to get the information about the
**   device ... if it exists, that is.
*/
devctl(fd, DevName, pGD)
int fd;
char DevName[];
char *pGD;
{
  if(strncmp(&DevName[5], "tty", 3) == 0) {     /* Is it a tty?  */
    if (ioctl(fd, TIOCSINUSE) == -1)            /* Is it in use? */
      terror( 6 );
    if (ioctl(fd, LDGETT, pGD) != -1)
      return( TTY );
  }
  else {                                        /* Not atty!     */
    if (ioctl(fd, DEVIOCGET, pGD) != -1)
    return( GENERIC );
   }
  return( ERROR );
}
```

B Testing System Calls, Messages, and Semaphores

This appendix provides the source code for test programs used in this book. The following programs are described:

- `bldfile.c`
- `msgtest.c`
- `semtest.c`

The first program, `bldfile.c`, tests system calls by writing a 100 KB file 100 times. You can build this program in the following ways:

1. To compile `bldfile.1`, issue the following command line:

   ```
   csh> cc -DSLOW bldfile.c -o bldfile.1
   ```

   The program, `bldfile.1`, writes the file using 1000 byte buffers with the `write` system call.

2. To compile `bldfile.2`, issue the following command line:

   ```
   csh> cc bldfile.c -o bldfile.2
   ```

   The program, `bldfile.2`, writes the file utilizing the `fwrite` system call.

```
#include  <stdio.h>
#ifdef SLOW
#include  <sys/file.h>
#include  <limits.h>
#endif

#define   HUNDRED       100
#define   THOUSAND      1000
#define   PERMS         0666

main(argc,argv)
int argc;
char *argv[];
{
    register int i, n;
    char buffer [THOUSAND+1];
#ifdef SLOW
   int   fp,  j;
#else
   FILE  *fp;
#endif

    memset(buffer,'1',THOUSAND);
```

```
    for (n=0; n<HUNDRED; n++)  {
#ifdef SLOW
       if ((fp = open(argv[1], O_RDWR, PERMS)) == -1)
#else
       if ((fp = fopen(argv[1], "w")) == NULL)
#endif
           perror("open error");

       for (i=0; i<HUNDRED; i++)  {
#ifdef SLOW
           if (write(fp, buffer, THOUSAND) == -1)
#else
           if (fwrite(buffer, THOUSAND, 1, fp) == 0)
#endif
              perror("write error");
       }
#ifdef SLOW

       close(fp);
#else
       fclose(fp);
#endif
   }
   exit (0);
}
```

The second program, `msgtest.c`, enables you to measure a message transmission rate in bytes per second by setting up a message queue with a buffer size of 128 bytes and transmitting the message to itself 20,000 times.

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define  KEY  ((key_t) 54321)

#define  COUNT          20000
#define  BUFFSIZE       128
#define  PERMS          0666

main(argc,argv)
int argc;
char *argv[];
{
   register int i, msqid;
   struct {
        long m_type;
        char m_text[BUFFSIZE];
   } msgbuff;

   if ((msqid = msgget(KEY, PERMS | IPC_CREAT)) < 0)
      perror("msgsnd error");
   msgbuff.m_type = 1L;
```

```
   for (i=0; i<COUNT; i++) {
      if (msgsnd(msqid, &msgbuff, BUFFSIZE, 0) < 0)
         perror("msgsnd error");

      if (msgrcv(msqid, &msgbuff, BUFFSIZE, 0L, 0) != BUFFSIZE)
         perror("msgrcv error");
   }

   if (msgctl(msqid, IPC_RMID, (struct msqid_ds *)0) < 0)
      perror("IPC_RMID error");

   exit (0);
}
```

The third program, `semtest.c`, enables you to measure the semaphore operations per second by creating a semaphore set, setting the value of each member to 1, and retrieving the value of the semaphore 20,000 times.

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

#define  KEY   ((key_t) 54321)

#define  COUNT          20000
#define  NSEMS          10
#define  PERMS          0666

main(argc,argv)
int argc;
char *argv[];
{
   register int i, j, semid;
   struct {
        ushort  sem_num;        /* semaphore # */
        short   sem_op;         /* semaphore operation */
        short   sem_fig;        /* operation flags */
   } sembuff;

   if ((semid = semget(KEY, NSEMS, PERMS | IPC_CREAT)) < 0)
      perror("semget error");

   for (i=0; i<COUNT; i++) {
      if (semctl(semid, i % NSEMS, SETVAL, 1) < 0)
         perror("semctl error");
```

```
            sembuff.sem_op = GETVAL;
            sembuff.sem_num = i % NSEMS;
            sembuff.sem_flg = IPC_NOWAIT;
            if (semop(semid, &sembuff, 1) < 0)
               perror("semop error");
         }

         if (semctl(semid, 0, IPC_RMID, (struct semid_ds *)0) < 0)
            perror("IPC_RMID error");

         exit (0);
      }
```

C The AIM Benchmark Suite III

The AIM Benchmark Suite III is a synthetic, multiuser benchmark that runs on many UNIX systems. It is a proprietary product of AIM Technology, Inc., in Santa Clara, California. It was chosen to exercise the system for the examples because it simulates a multiuser workload and, at the same time, can be configured to model any environment; thus, it is much more flexible than a benchmark that simulates a single environment.

The benchmark runs a master program that controls the simulated users that are run. The master program forks (executes) a process for each simulated user, but forces them to wait until all can begin running simultaneously. These simulated users run a synthetic workload that is identical for all, although the order in which they execute individual components of the workload is randomly distributed.

The workload is comprised of 31 separate tests or subsystem exercisers. A representative test opens a file, writes 250 KBytes to the file, closes the file, and repeats. Other tests search directory paths, or add a sequence of numbers, or copy data from one area in memory to another, and so on. Each test is run enough times to generate a time interval of four or five seconds. The time to complete the workload when only one user is simulated is approximately two minutes and 30 seconds.

The benchmark reports the time used by the first of the simulated users to complete the workload, broken into system time, user time, clock time, and the number of processes completed per second. The number of processes completed per second is computed by dividing the number of processes run by the simulated user (fifty) by the cpu time to complete the workload. As the system becomes heavily loaded, the time to complete the workload increases, so the processes completed per second falls.

D The SPEC Benchmark

The SPEC Benchmark is used in several of this manual's examples in which the system load needed to be increased to the point that system memory and CPU cycles were used to capacity. Workloads that comprise the SPEC Benchmark are single-user, CPU intensive, natural workloads. Documentation provided with the SPEC Benchmark states the following:

> The SPEC benchmark suite consists of FORTRAN and C CPU intensive benchmarks that are intended to be meaningful samples of applications that perform fixed and floating point logic and computations in a technical computing environment. SPEC Benchmark Release 1.0 does not assess the ability of a system under test to handle disk, graphics, communication, or multi-user activity.

The workloads that make up the SPEC Benchmark are programs taken from real application and development environments. The setup and execution of the workloads is totally automated by the benchmark software using the standard UNIX `make` utility.

Index

A

AIM Benchmark Suite III, C—1
allocating disk space
  du command, 2—12
  quot command, 2—12
allocating swap space, 2—13
  pstat command, 2—13
at command, 2—16

B

background processes, 1—7
bg command, 2—16
bin directory, 1—5
block
  definition of, 1—3
boot block
  definition of, 1—3
buffer cache, 1—10, 3—1
  changing size of, 4—8, 4—13
  function of, 1—10
  recognizing shortage of, 3—13
  size of, 1—11

C

chpt command, 2—9
  changing disk partition size, 4—6
cmx utility, 2—22
cpustat command, 2—2
crash utility, 2—2

D

dev directory, 1—5
devstat command, 2—21
df command, 2—9
direct block
  definition of, 1—3
directory
  bin, 1—5
  dev, 1—5
  etc, 1—5
  primary, 1—5
  root, 1—4
  tmp, 1—5
  usr, 1—5
disk
  exercising with dskx, 2—13
  modifying partition size, 4—5
disk space allocation
  affect on performance, 2—12
dskx command, 2—13
du command, 2—12

E

etc directory, 1—5

F

fg command, 2—16
file system, 1—1
  adding a second swap partition, 4—7
  associating with a device, 1—5
  changing buffer cache size, 4—8
  changing disk partition size, 4—5
  checking, 2—10
  components, 1—2
    block, 1—2
    boot block, 1—3
    disk blocks, 1—3
    inode, 1—3
    super block, 1—3
  creating, 2—10
    newfs command, 2—10
  evaluating
    fsck command, 2—11
    tunefs command, 2—10
  exercising, 2—13

with fsx, 2—14
function of, 1—1
identifying a disk-bound state
using iostat command, 3—2
identifying a swap-bound state, 3—3
using ps command, 3—4
using pstat command, 3—3
using vmstat command, 3—4
identifying limitations, 3—1
identifying tuning needs, 3—1
improving performance of
configuration, 1—6
file location, 1—6
file size, 1—6
organizing files into directories, 1—7
monitoring activity, 2—14
mounting, 1—5
recognizing balanced loads, 3—8
recognizing problems
context switches, 3—10
interrupts, 3—8
process table shortages, 3—8
system calls, 3—9
relationship of blocks to devices, 1—3
relationship of files, 1—4
relationship to devices, 1—3
reorganizing, 4—4
tools for monitoring, 2—8
tuning, 2—10, 4—4
unmounting, 1—6
foreground processes, 1—7
fsck command, 2—11
fsx command, 2—14

G

gathering disk information, 2—8
chpt utility, 2—9
df command, 2—9
mount command, 2—10

I

I/O subsystem, 1—12
assessing tuning needs, 3—15
components of, 1—12
block device switch table, 1—13
block devices, 1—13
character device switch table, 1—13
character devices, 1—13
device special files, 1—13
determining I/O status, 2—21
device drivers, 1—13
disk drives, 1—14
effect on performance, 1—17
exercising, 2—22
using cmx, 2—22
using lpx, 2—22
using mtx, 2—23
identifying limitations, 3—14
identifying system devices, 2—21
improving file system performance
file location, 1—7
improving performance of, 3—15
major device number, 1—14
measuring throughput, 3—16
minor device number, 1—14
monitoring, 2—20
performance trade-offs, 3—16
printers, 1—16
tape drives, 1—15
terminals, 1—15
tuning
changing hardware, 4—15
changing software, 4—15
ifconfig utility, 2—24
indirect block
definition of, 1—3
inode
definition of, 1—3

interprocess communications, 1—20, 2—30
assessing tuning needs, 3—19
determining IPC status, 2—30
determining IPC usage, 2—30
effect on performance, 1—21
identifying limitations, 3—19
measuring throughput rate, 3—20
messages for, 1—20
monitoring activity, 2—30, 2—31
semaphores with, 1—21
shared memory with, 1—20
tuning, 4—21
changing global partitions, 4—21
changing software, 4—22
iostat command, 2—3
identifying a disk-bound state, 3—2
ipcs command, 2—30
options, 2—30

J

jobs command, 2—16

K

kill command, 2—16

L

lpx utility, 2—22

M

memory
allocating of by kernel, 1—10
buffer cache, 1—10
buffer cache size, 1—11
improving performance, 1—11
changing buffer cache size, 1—11
changing the swap area, 1—11
management techniques, 1—12
managing, 1—9
pregion tables, 1—10
recognizing shortage of, 3—12
swap area, 1—11
memory exercisers
memx, 2—20
shmx, 2—20
memory management, 1—9
assessing active virtual memory size, 3—12
assessing tuning needs, 3—12
identifying limitations, 3—10
memory exercisers, 2—19
monitoring
with vmstat command, 2—20
recognizing a small buffer cache, 3—13
recognizing shortage of physical memory, 3—12
tuning, 4—12
changing buffer cache size, 4—13
changing physical memory, 4—14
modifying global parameters, 4—12
memx exerciser, 2—20
mnt directory, 1—5
monitoring subsystem resource usage, 2—1
tools for, 2—1
cpustat command, 2—2
crash utility, 2—2
iostat command, 2—3
netstat command, 2—3
nfsstat command, 2—5
pstat command, 2—6
vmstat command, 2—6
mount command, 2—10
mtx utility, 2—23

N

netstat command, 2—3
  determining network status, 2—25
  measuring network throughput, 3—19
  monitoring network activity, 2—30
Network Control Program (NCP), 2—29
Network File System (NFS), 1—18
  client problems of, 4—18
  identifying bottlenecks, 3—17
  improving performance of, 3—17, 4—20
  network problems of, 4—17
  performance problems of, 4—17
  Prestoserve's impact on performance, 4—20
  server performance of, 4—19
  server problems of, 4—18
network, 1—17
  assessing tuning needs, 3—18
  components of, 1—17
    cabling, 1—17
    controllers, 1—17
    protocols, 1—18
    software, 1—18
  determining interface status, 2—24
    using ifconfig, 2—24
    using netstat, 2—25
  determining usage, 2—23
    using ruptime, 2—24
    using rwho, 2—23
  displaying NFS statistics using nfsstsat, 2—25
  effect on performance, 1—19
  exercising, 2—27
    using netx, 2—28
    using ping, 2—28
  function of, 1—17
  identifying limitations, 3—18
  measuring throughput rate, 3—18
    using netstat command, 3—19
  monitoring, 2—23, 2—29
    using netstat, 2—30
  NFS performance problems, 4—17
  tuning, 4—15
    changing global parameters, 4—15
    changing software, 4—16
network protocols, 1—18
  DECnet, 1—18
  NFS, 1—18
  RFS, 1—19
  TCP/IP, 1—18
netx utility, 2—28
newfs command, 2—10
nfsstat command, 2—5
nfsstsat command
  displaying statistics, 2—25
nice command
  changing process priority, 4—11

P

ping command, 2—28
Process control and scheduling, 1—7
process, 1—7
  background, 1—7
  determining priority, 1—8
  foreground, 1—7
  improving performance, 1—9
  rescheduling, 2—16
  resetting the priority with renice, 2—16
  scheduling, 2—16
  scheduling priority and privilege, 1—8
  setting stick bit, 4—11
  setting the priority, 2—15
  stopping, 2—16

process control and scheduling
assessing tuning needs, 3—5
changing global parameters, 4—10
changing process priority, 4—10
computing load average, 3—6
determining users and tasks, 2—15
finger command, 2—15
uptime command, 2—15
w command, 2—15
who command, 2—15
identifying limitations, 3—5
monitoring context switches, 2—17
monitoring interrupts, 2—17
monitoring of
ps command, 2—17
pstat command, 2—19
monitoring system calls, 2—17
setting the sticky bit, 4—11
tuning, 4—10
process table, 3—8
ps command
assessing physical memory, 3—12
identifying a swap-bound state, 3—4
monitoring process control activity, 2—17
pstat command, 2—6, 2—13
determining I/O status, 2—21
identifying a swap-bound state, 3—3
monitoring process control activity, 2—19

Q

quot command, 2—12

R

renice command, 2—16
root directory, 1—4
root directory (/), 1—4
ruptime command, 2—24
rwho command, 2—23

S

SPEC Benchmark, D—1
shared memory
exercising, 2—20, 2—31
shmx utility, 2—20, 2—31
stop command, 2—16
subsystem resource usage, 4—1
file system
adding secondary swap, 4—7
changing buffer cache size, 4—8
changing disk partition size, 4—5
reorganizing, 4—4
system configuration file, 4—1
tuning I/O subsystem, 4—14
tuning interprocess communications, 4—21
tuning memory management, 4—12
tuning the file system, 4—4
tuning the network, 4—15
NFS, 4—17
super block
definition of, 1—3
suspend command, 2—16
swap area, 1—11
system configuration file, 4—1
adding buffer cache entry, 4—8
device definitions in, 4—3
global definitions in, 4—2
pseudodevice definitions in, 4—3
system image definitions in, 4—3
system exercisers
cmx utility, 2—22
lpx utility, 2—22
mtx utility, 2—23
netx utility, 2—28

T

terminals
  control terminal, 1—15
  role of the kernel, 1—16
tmp directory, 1—5
tunefs command, 2—10

U

uptime command, 2—15
usr directory, 1—5

V

vmstat command, 2—6
  assessing active virtual memory size, 3—12
  assessing physical memory, 3—12
  identifying a swap-bound state, 3—4
  monitoring memory usage, 2—20

W

w command, 2—15
who command, 2—15

How to Order Additional Documentation

Technical Support

If you need help deciding which documentation best meets your needs, call 800-343-4040 before placing your electronic, telephone, or direct mail order.

Electronic Orders

To place an order at the Electronic Store, dial 800-234-1998 using a 1200- or 2400-baud modem from anywhere in the USA, Canada, or Puerto Rico. If you need assistance using the Electronic Store, call 800-DIGITAL (800-344-4825).

Telephone and Direct Mail Orders

Your Location	Call	Contact
Continental USA, Alaska, or Hawaii	800-DIGITAL	Digital Equipment Corporation P.O. Box CS2008 Nashua, New Hampshire 03061
Puerto Rico	809-754-7575	Local Digital Subsidiary
Canada	800-267-6215	Digital Equipment of Canada Attn: DECdirect Operations KAO2/2 P.O. Box 13000 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6
International	————	Local Digital subsidiary or approved distributor
Internal*	————	SSB Order Processing - WMO/E15 *or* Software Supply Business Digital Equipment Corporation Westminster, Massachusetts 01473

* For internal orders, you must submit an Internal Software Order Form (EN-01740-07).

Reader's Comments

ULTRIX
Performance Management Guide
AA-PKDVA-TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

Please rate this manual:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

What would you like to see more/less of? _____

_____

_____

What do you like best about this manual? _____

_____

_____

What do you like least about this manual? _____

_____

_____

Please list errors you have found in this manual:

Page	Description
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____

Additional comments or suggestions to improve this manual:

_____

_____

_____

_____

What version of the software described by this manual are you using? _____

Name/Title _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Email _____ Phone _____

Do Not Tear – Fold Here and Tape

digital™

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST-CLASS MAIL PERMIT NO. 33 MAYNARD MA

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
OPEN SOFTWARE PUBLICATIONS MANAGER
ZKO3-3/Y32
110 SPIT BROOK ROAD
NASHUA NH 03062-2698

Do Not Tear – Fold Here

Cut
Along
Dotted
Line

Reader's Comments

ULTRIX
Performance Management Guide
AA-PKDVA-TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

Please rate this manual:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

What would you like to see more/less of? _____

_____

_____

What do you like best about this manual? _____

_____

_____

What do you like least about this manual? _____

_____

_____

Please list errors you have found in this manual:

Page	Description
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____

Additional comments or suggestions to improve this manual:

_____

_____

_____

_____

What version of the software described by this manual are you using? _____

Name/Title _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Email _____ Phone _____

Do Not Tear – Fold Here and Tape

digital™

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST–CLASS MAIL PERMIT NO. 33 MAYNARD MA

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
OPEN SOFTWARE PUBLICATIONS MANAGER
ZKO3–3/Y32
110 SPIT BROOK ROAD
NASHUA NH 03062–2698

Do Not Tear – Fold Here

Cut
Along
Dotted
Line