

Software Installation Guide

REAL/IX® Operating System for 386/486 ISA Computers

AEG

Property of Logical Data Corporation

RECEIVED MAY 18 1995

214-870002-000

MODCOMP



Software Installation Guide

REAL/IX[®] Operating System for 386/486 ISA Computers

AEG

Property of Logical Data Corporation

RECEIVED MAY 18 1995

MODCOMP

PROPRIETARY NOTICE

THE INFORMATION AND DESIGNS DISCLOSED HEREIN WERE ORIGINATED BY AND ARE THE PROPERTY OF MODULAR COMPUTER SYSTEMS, INC. (MODCOMP). MODCOMP RESERVES ALL PATENT, PROPRIETARY DESIGN, MANUFACTURING, SOFTWARE PROPERTY, REPRODUCTION, USE, AND SALES RIGHTS THERETO, AND RIGHTS TO ANY ARTICLE DISCLOSED THEREIN. THIS INFORMATION IS MADE AVAILABLE UPON THE CONDITION THAT THE PROPRIETARY DESIGNS AND PRODUCTS DESCRIBED HEREIN WILL BE HELD IN ABSOLUTE CONFIDENCE AND MAY NOT BE DISCLOSED IN WHOLE OR IN PART TO OTHERS WITHOUT THE PRIOR WRITTEN PERMISSION OF MODULAR COMPUTER SYSTEMS, INC. THE FOREGOING DOES NOT APPLY TO VENDOR PROPRIETARY PRODUCTS.

SPECIFICATIONS REMAIN SUBJECT TO CHANGE IN ORDER TO ALLOW THE INTRODUCTION OF DESIGN IMPROVEMENTS.

FOR GOVERNMENT USE THE FOLLOWING SHALL APPLY:

RESTRICTED RIGHTS LEGEND

USE, DUPLICATION, OR DISCLOSURE BY THE GOVERNMENT IS SUBJECT TO RESTRICTIONS AS SET FORTH IN RIGHTS IN DATA CLAUSES DOE 952.227-75, DOD 52.227-7013, AND NASA 18-52.227-74 (AS THEY APPLY TO APPROPRIATE AGENCIES).

MODULAR COMPUTER SYSTEMS, INC.
1650 WEST McNAB ROAD
P.O. BOX 6099
FORT LAUDERDALE, FL 33340-6099

THIS MANUAL IS SUPPLIED WITHOUT REPRESENTATION OR WARRANTY OF ANY KIND. MODULAR COMPUTER SYSTEMS, INC. THEREFORE ASSUMES NO RESPONSIBILITY AND SHALL HAVE NO LIABILITY OF ANY KIND ARISING FROM THE SUPPLY OR USE OF THIS PUBLICATION OR ANY MATERIAL CONTAINED HEREIN.

THE PRODUCT DESCRIBED HEREIN IS BASED ON COPYRIGHTED SOFTWARE FROM VARIOUS COMPANIES INCLUDING MOTOROLA, INC. AND UNIX SYSTEM LABORATORIES, INC. THE SOFTWARE IS FURNISHED UNDER A LICENSE AGREEMENT AND MAY BE USED ONLY IN ACCORDANCE WITH THE TERMS OF SUCH AGREEMENT.

MANUAL HISTORY

Manual Order Number: 214-870002-000

Title: REAL/IX Operating System for 386/486 ISA Computers, Software Installation Guide

Revision Level	Date Issued	Description
000	09/93	Initial Issue.

Contents subject to change without notice.

Copyright © 1993, by Modular Computer Systems, Inc.
All Rights Reserved.
Printed in the United States of America.

Portions of this document are based on or reprinted from copyrighted documents
by permission of Motorola, Inc. and AT&T.

CLASSIC, CLASSIC REAL/IX, MAX, MODCOMP, MODMate, GLS, PACE/32, PACE/IX, PRIDE, REAL/IX, REAL/STAR, REAL/STONE, REAL/VU, Tri-Dimensional, Tri-D, TRI-SIM, and Modular Computer Systems are registered trademarks of Modular Computer Systems, Inc.

Development Server, PROTOP/IX, and REAL/MAX are trademarks of Modular Computer Systems, Inc.

The following are trademarks or registered trademarks of their respective companies: Macintosh of Apple Computer, Inc.; **Documenter's Workbench**, **TELETYPE** of AT&T; VCL, EDT+ of Boston Business Computing, Ltd.; Belden of Cooper Industries, Inc.; Centronics of Data Computer Corporation; DEC, DECnet, PDP, VAX, VMS of Digital Equipment Corporation; **EXOS**, **Excelan** of Excelan, Inc.; HP, Vectra of Hewlett-Packard Company; VRTX/68000 of Hunter & Ready, Inc.; **BITBUS**, **MULTIBUS**, Intel-(followed by any number) of Intel Corporation; **IBM**, **PC-DOS**, **PC/AT**, **PC/XT**, **PS/2** of International Business Machines Corporation; X Window System of Massachusetts Institute of Technology; **Microsoft**, **Windows**, **MS-DOS**, **Xenix** of Microsoft Corporation; **MODICON**, **Modbus**, **Modbus Plus** of Modicon, Inc.; **Motorola**, **System V/68**, **VME Delta Series**, **Delta Series**, **VMEexec**, **Business Assistant**, **OfficeLAN**, **VERSAbus**, **VERSAios**, **133Bug**, **147Bug**, **187Bug**, **HYPERmodule**, **VMEmodule**, **188Bug**, **SSID** of Motorola, Inc.; **OSF**, **OSF/Motif**, **Motif** of Open Software Foundation, Inc.; **ORACLE**, **SQL*Calc**, **SQL*Forms**, **SQL*Plus**, **SQL*Graph**, **SQL*Menu**, **SQL*Net**, **Pro*SQL**, **Pro*ORACLE**, **Pro*C**, **Pro*COBOL**, **Easy*SQL** of Oracle Corporation; **Q-ONE** of Quadretron Systems Inc.; **FORTRIX** of Rapitech Systems, Inc.; **PROBUG** of SBE, Inc.; **Sun**, **NFS**, **ONC** of Sun Microsystems, Inc.; **TEKTRONIX** of TEKTRONIX, Inc.; **TeleSoft**, **Telegen2** of TeleSoft; **TeleVideo** of TeleVideo Systems, Inc.; **UNIX** of UNIX System Laboratories, Inc.; **DIABLO**, **Ethernet**, **Xerox** of Xerox Corporation; **88open**, **BCS**, **OCS** of 88open Consortium, Ltd.,

PREFACE

Audience

This manual is designed for administrators performing initial setup or upgrading to a new release of the REAL/IX Operating System. Instructions are provided for loading the installation medium, setting up the operating system, and other administrative tasks that may be required to get the system up and running. Before beginning the procedures described here, you should have read the *Software Engineering Release Notes* and completed the hardware installation described in your computer hardware manuals.

Open Architecture Systems Defined

The term "open architecture system", in its simplest form, implies that a user may add a variety of vendors' components to a single system. This is possible when certain industry-accepted standards have been implemented in the system. MODCOMP open architecture systems are based on such software and hardware standards as the UNIX System V operating system; VMEbus, MULTIBUS, and SCSI bus interfaces; and CPUs built around standard microprocessors. By building on these standards, open architecture systems provide computer solutions that are portable and compatible.

The REAL/IX Operating System¹ allows applications to be ported easily between traditional UNIX systems and MODCOMP open architecture systems. Furthermore, by using industry standard I/O buses, MODCOMP open architecture systems ensure compatibility among a wide range of peripheral and I/O devices and the ability to expand as needs dictate. MODCOMP open architecture systems meet networking and communications needs with such industry standards as Ethernet and TCP/IP and have the flexibility to accommodate new standards as they are developed.

Related Publications

Refer to the following publications for additional information.

Books for All System Users

Concepts and Characteristics

Gives an overview of the internals of the REAL/IX Operating System and an introduction to the tools and facilities that are available.

Reference Manual, Sections 1, 1M, and 1R

Contains manual pages for user commands (Section 1), administrative commands (Section 1M), and realtime commands (Section 1R).

¹The REAL/IX Operating System, featuring realtime and multiprocessing capabilities, is the MODCOMP implementation of the UNIX System Laboratories UNIX System V operating system.

Reference Manual, Sections 2, 3, and 5

Contains manual pages for system calls (Section 2), library routines (Sections 3C, 3M, 3N, 3S, and 3X), and miscellaneous information (Section 5).

Reference Manual, Sections 4, 7, and 7A

Contains manual pages for system files (Section 4), special device files for standard devices (Section 7), and special device files for add-on packages (Section 7A).

User's Guide

Discusses basic user procedures including the login procedure and getting around the file system. Information is included about general user tools; for example, the **vi** and **ed** text editors, electronic mail, the shell programming language, and the Korn shell.

Using UUCP and Usenet

Introduces UUCP communications, describes how to transfer files and execute remote commands over UUCP, how to check on UUCP requests, and how to access the Usenet electronic bulletin board.

Books for System Administrators

Software Installation Guide

Gives instructions for installing the operating system (either for the first time or as an upgrade) and initially setting up the system.

System Administrator's Guide

Gives instructions and background information about administering the REAL/IX Operating System. Topics covered include ensuring system security; creating and maintaining user and group IDs; working with file systems (creating, repairing, backing up); setting up terminals and printers; using the **sysgen(1M)** utility to modify tunable parameters and to configure or deconfigure standard system devices; and setting up and using the Job Accounting System. Appendixes discuss the system files that control system operations and the file naming conventions for special device files.

Software Engineering Release Notes

Gives an overview of the new features in this release of the REAL/IX Operating System and provides usage notes for the system.

Managing UUCP and Usenet

Provides background information about UUCP for administrators and gives instructions for setting up a UUCP link, verifying that the link works, administering UUCP communications, and setting up and administering the Usenet access. This information is supplemented by the *System Administrator's Guide*, which includes information for administering UUCP over the TCP/IP protocol, and the *Software Installation Guide*.

Books for Programmers

Languages and Support Tools Guide

Provides tutorials for many of the special purpose languages and the programming support tools available on the REAL/IX Operating System.

Languages and Support Tools Guide Supplement

Contains information that is specific to the released system as it operates in the native microprocessor environment of your hardware platform.

Programmer's Guide

Gives an overview of the REAL/IX Operating System and realtime computing, describes the REAL/IX programming environment and the operating system interface, and provides programming examples for using the realtime extensions of the REAL/IX Operating System as well as the standard UNIX operating system features.

The C Programming Language, First Edition

Describes the traditional UNIX C language.

The C Programming Language, Second Edition

Describes the ANSI C language.

Books for Kernel Programmers

Driver Development Guide

Introduces the process of writing device drivers for the REAL/IX Operating System, including detailed information about porting and installing drivers.

Kernel Programming Guide

Gives background information about topics of interest to programmers writing device drivers and system calls. Topics discussed include how drivers and system calls execute and how various types of I/O operations are implemented.

Kernel Reference Manual

Contains reference pages for driver entry-point routines (Section D2X), kernel functions and macros (Section D3X), and kernel data structures (Section D4X) used for coding system calls and device drivers.

Industry Standard Publications

The REAL/IX Operating System complies with the industry standards listed below. These standards are commercially available and can be obtained from the following sources. While an effort was made to ensure that the ordering information was complete and up-to-date at time of printing, we cannot guarantee its accuracy.

System V Interface Definition (SVID)

AT&T Customer Information Center (CIC)

Customer Service Representative

P.O. Box 19901

Indianapolis, IN 46219

Phone: (800) 432-6600 (Inside U.S.A.)

(800) 255-1242 (Inside Canada)

(317) 352-8557 (Outside U.S.A. and Canada)

Documentation Conventions

The following table gives the textual conventions used in this book. Note that commands, library routines, system calls, kernel functions, driver entry points, files, and data structures are sometimes followed by a number enclosed in parentheses (for instance, "**cat**(1)"). This denotes the reference section in which they are located; Sections D2X, D3X, and D4X are in the *Kernel Reference Manual*; all others are in the *Reference Manual* volumes and available online through the **man**(1) command. Commands followed by empty parentheses (for instance, "**false**()") are available through the **man** command, but do not have their own manual page.





Style	Item	Example
bold	Shell commands	cat or cat(1)
bold	Library routines	printf or printf(3s)
bold	System call names	open or open(2)
bold	Kernel function names	copyin or copyin(D3X)
bold	Driver entry point names	strategy or strategy(D2X)
bold	Script names	MOUNTFSYS or S03MOUNTFSYS
<i>italics</i>	File names	<i>/etc/passwd</i>
monofont	Data structures	<i>user</i> or <i>user(D4X)</i>
bold	Data structure members	u_count or u.u_count
bold	Literal text in example	cat filename
<i>italics</i>	Variable text in example	<i>cat filename</i>
monofont	Code representations	<i>if size <= 0 return NULL;</i>
monofont	Screen representations	<i>Enter a number or q to quit: 2</i>
monobold	Operator input	
?	Single character wildcard	<i>/dev/tty??</i>
*	Multi-character wildcard	<i>/dev/*_ctl</i>
 WARNING!	<i>highlights information that, if not observed, could cause bodily harm.</i>	
 CAUTION	<i>highlights information that, if not observed, could cause the system or a procedure or practice to fail or could damage existing data on the system.</i>	
 NOTE	<i>highlights relevant information that does not require a caution or warning.</i>	
 HINT	<i>identifies material that is indirectly related to the subject matter being discussed. For instance, a procedure may specify one way of doing the task, and the HINT explains why it is done this way or suggests optional ways to accomplish the same task.</i>	



TABLE OF CONTENTS

	Page
Chapter 1 Introduction	
Chapter 2 Procedure for New Installation	
Before You Begin the Installation	2-1
Summary of Installation Steps	2-1
Getting Started	2-3
Phase 1: Configure the Hard Disk	2-4
Phase 2: Install Commands and Utilities	2-6
Phase 3: Install Update Software	2-6
Phase 4: Install Kernel Configuration Software	2-6
Chapter 3 Setting Up the Operating System	
Begin Setup Procedure	3-2
Verify the Installation	3-2
Set System Date	3-3
Assign Passwords	3-4
Define the Node Name	3-6
Set TERM Variable	3-7
Print Error Message List	3-7
Starting Accounting and Performance Statistics	3-8
Chapter 4 Starting the TCP/IP Network	
Chapter 5 Setting Up Terminals and Modems	
Activating getty Lines	5-2
Changing Default Baud Rate	5-2

LIST OF TABLES

	Page
2-1 Meaning of Prompts in Installation Script	2-2
2-2 Typing Conventions in Installation Script	2-2

Chapter 1

Introduction

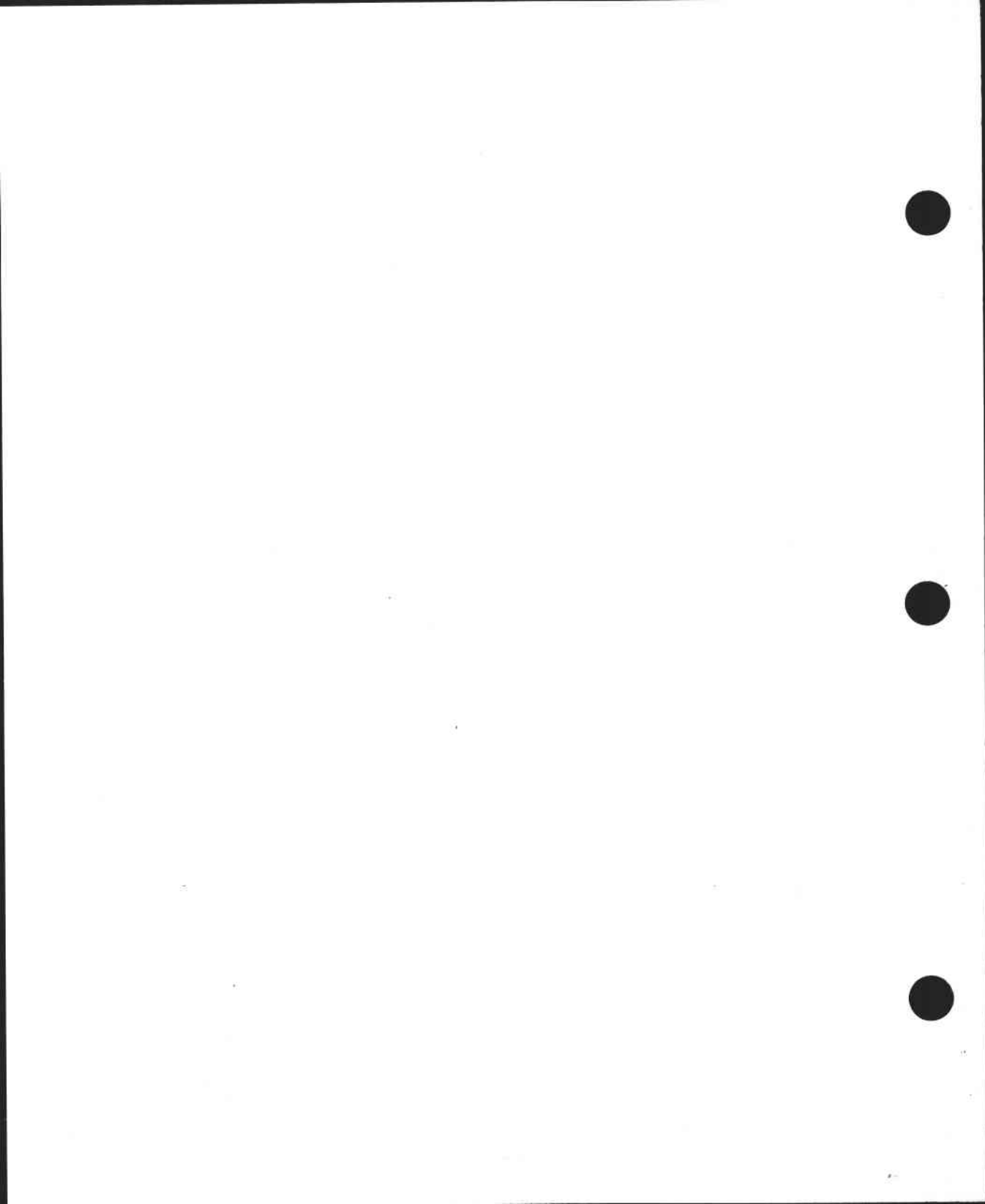
The *Software Installation Guide* gives instructions and guidelines for installing the REAL/IX® Operating System from the installation medium and setting up the operating system. Its intent is to enable you to get your operating system running as quickly as possible in a basic configuration that is adequate as a starting place for most environments.



In some cases, your computer may come with the operating system already loaded on the disk. In this case, you can skip the procedures in the installation chapter and begin to set up the environment as discussed in the chapter "Setting Up the Operating System."

This book does not tell you every way to do each task, nor does it go into the full range of options for many of these tasks. Pointers are provided to other books that give more details; once the system is set up, you should read other books in the documentation set and gradually customize the operating environment to one that is appropriate for your installation. The Preface lists all the books in the documentation set. The following books are closely related to this one:

- *Release Notes*
- *Concepts and Characteristics*
- *Sections 1, 1M, and 1R Reference Manual*
- *System Administrator's Guide*



Chapter 2

Procedure for New Installation

This chapter describes how to perform a first-time installation of the REAL/IX Operating System. It is an example of a typical installation and assumes that you are installing *only* REAL/IX on your computer's hard disk.

Before You Begin the Installation

Before you begin the installation, the hardware on your computer must be configured properly. The system manufacturer should provide a setup utility that allows you to tell your system what hardware is installed (CMOS configuration). Some systems have the setup utility resident in the ROM BIOS; others come with a setup diskette. Use your setup utility to correctly set up your system. Refer to your system documentation for information on how to use your setup utility.

The *REAL/IX Operating System Release Notes* document may contain supplemental installation information that was not available at the time this manual was printed. You should refer to the *Release Notes* before beginning the REAL/IX installation.



The REAL/IX installation procedure will overwrite existing data, files, and programs on your PC's hard drive. Be sure to make a backup of your entire system before beginning the REAL/IX installation.

Summary of Installation Steps

The REAL/IX Operating System for 386/486 ISA systems is delivered on a set of diskettes:

- ☐ Boot diskette
- ☐ Commands and Utilities diskettes
- ☐ Update diskette
- ☐ Kernel Configuration diskettes

During the installation you will perform the following steps:

- ☐ Boot the system from the installation (boot) diskette
- ☐ Create a REAL/IX partition on the hard drive
- ☐ Create the REAL/IX file systems and swap/paging area
- ☐ Install the software

A series of scripts that are fairly self-explanatory lead you through the installation procedure. The scripts use the symbolic prompts listed in Table 2-1.

Table 2-1. Meaning of Prompts in Installation Script

--->	An answer is expected; press <CR> to use the default choice. Defaults are shown as <i>[default]</i> at the end of each question or statement where an answer is expected.
***>	An action is in progress that does not require intervention.
ERROR	Is followed by text that describes the error. These messages usually mean you must start the installation over.
[text]	Is followed by a warning.

When responding to questions in the installation scripts, use the typing conventions listed in Table 2-2.

Table 2-2. Typing Conventions in Installation Script

^h	(CTRL-h) erase previous character
@	erase line (valid only if you have not pressed <CR> to enter the line)
DELETE key	interrupt the installation; CTRL-d will restart it
^ 	(CTRL-pipe) same as the DELETE key

As you go through the scripts, you will be prompted for input; the prompts are replicated in this book in monofont. The arrows (--->) preceding all the online prompts requiring input (see Table 2-1) have been omitted in this book for clarity. **Bold type** indicates answers you type in. Explanatory text that gives guidelines on how to answer the questions **precedes** each prompt.

Getting Started

To begin the installation:

- Insert the boot diskette into the floppy drive your computer boots from.
- If your computer is already powered on, reboot the system. If it is powered down, turn the power on.

After the computer goes through its memory configuration, you should then see a messages indicating that the REAL/IX Installation System is booting.

At this point you are warned that the installation procedure may overwrite and destroy existing data on the hard disk. If you have not done so already, make a full backup of your hard disk before continuing with the REAL/IX installation procedure.

The rest of the installation procedure is done in phases:

1. Configure the hard disk. This consists of the following steps:
 - Create partitions for REAL/IX and any other operating systems you may want to install on your computer's hard disk
 - Create a swap/paging area
 - Generate REAL/IX file systems
2. Load the software from the Commands and Utilities diskettes
3. Load the software from the Update diskette
4. Load the software from the Kernel Configuration diskettes. This step is optional; you only have to install this software if you want to reconfigure the kernel or use the REAL/IX networking services.

Phase 1: Configure the Hard Disk

In this phase you are first asked to create a partition for REAL/IX on the hard disk. At this time you can also create partitions for other operating systems that you may want to install on your computer's hard disk.

The default for the installation procedure is to create two partitions such that 90% of the disk is used by REAL/IX and 10% is reserved for DOS. You may not want to use the default configuration, depending on what other programs you may be installing on your computer. The following example shows how to create a partition where REAL/IX will use all (100%) of the hard disk. Your responses to the prompts are shown in **bold**.

Do you want to partition your hard disk as follows:

- 90% "UNIX System" -- lets you run UNIX System programs
- 10% "DOS (v 3.2 or later only)"

To do this please type "y". To partition your hard disk differently, type "n" and the fdisk program will let you select other partitions. n

Select one of the following:

1. Create a partition
2. Change Active (Boot from) partition
3. Delete a partition
4. Exit (Update disk configuration and exit)
5. Cancel (Exit without updating disk configuration)

Enter Selection: 1

Indicate the type of partition you want to create.
(1=UNIX system, 2=DOS only, 3=Other, x=Exit): 1

The UNIX System partition must use at least 9% of the hard disk.

Indicate the percentage (9-100) of the hard disk you want this partition to use (or enter "c" to specify in cylinders): 100

The next prompt asks you whether you want the REAL/IX partition to be the Active partition. The Active partition is the one the computer will boot from when you power on or reboot the system. Making REAL/IX the Active partition means that you will get the REAL/IX login prompt after you power on or reboot the system.

Do you want this to become the Active partition? If so, it will be activated each time you reset your computer or when you turn it on again. Please type y or n: y

The menu allowing you to create and delete partitions, change the Active partition, or update the disk configuration will then be displayed again. If you do not want to create another partition, enter 4 to update the disk configuration and exit.

Select one of the following:

1. Create a partition
2. Change Active (Boot from) partition
3. Delete a partition
4. Exit (Update disk configuration and exit)
5. Cancel (Exit without updating disk configuration)

Enter Selection: 4

The partitions file for the REAL/IX portion of the disk will now be generated.

After the partitions are created, the system will display some disk parameters (number of cylinders, tracks, and sectors per track) and ask you whether these parameters are correct.

A summary of the disk layout will then be displayed. By default the installation procedure creates *root*, *usr*, and *usr2* file systems and a swap/paging area.

The following seems like a reasonable partitioning of your UNIX System disk space:

- A root filesystem of xxx cylindres (xxx bytes)
- A user (/usr) filesystem of xxx cylinders (xxx bytes).
- An extra user filesystem (/usr2) of xxx cylinders (xxx bytes).
- A swap/paging area of xxx cylinders (xxx bytes)

The swap/paging area must be at least equal to or greater than the system memory size. To help you determine this size, you will be given a breakdown of statistics specific to your system configuration, including the number of cylinders per megabyte. For example, on a system with 16 megabytes of memory and 3 cylinders per megabyte, you could calculate the minimum size for the swap/paging area as $16 \times 3 = 48$. This could then be rounded up to 50.

Answer y if the disk layout that is displayed is acceptable.

Is this allocation acceptable to you (y/n)? y

The slice table entries for the file systems will then be built. After that, you will be asked to select the file system type (fast or standard UNIX); refer to the *System Administrator's Guide* for information about file system types. You will be prompted separately for each file system to be created.

Do you want to create a REAL/IX "fast" filesystem or a standard UNIX filesystem (<f>/s): f

You will then get some messages about each file system as it is created. After that is complete, remove the boot diskette from the floppy disk drive when prompted to do so.

The next phase of the installation is to load the software from the various release diskettes.

Phase 2: Install Commands and Utilities

During this phase you will insert each Commands and Utilities diskette as you are prompted. If your computer has both an A and B floppy drive, you will first be prompted for the drive you want to load the software from.

After each diskette is read, you will be prompted to remove the diskette, insert the next one, and press the enter key.

After the last Commands and Utilities diskette is finished loading, you will be prompted for a serial number. The serial number can be found on the label of the boot diskette.

Please enter the serial number found on the boot floppy.

This completes the installation of the REAL/IX Commands and Utilities.

Phase 3: Install Update Software

When prompted, insert the Update diskette into the floppy drive and press the enter key. The Update portion of the release will be loaded with no further intervention on your part.

Phase 4: Install Kernel Configuration Software

Installing the Kernel Configuration software is required only if you want to be able to reconfigure the kernel or use the networking services of the REAL/IX Operating System. After the Update software is installed, you will be prompted as to whether or not you want to install the Kernel Configuration software.

If you choose not to install this portion of the software, you will proceed to the prompt about entering a shell.

If you need the ability to reconfigure the REAL/IX kernel, the Kernel Configuration disk must be loaded.

Note: This is not a required part of the system. However, if you wish to use networking, you must answer yes.

Do you wish to load the Kernel Configuration set at this time (y/<n>)? y

The REAL/IX System Kernel Configuration set will now be installed.

When instructed, insert each Kernel Configuration diskette into the floppy drive and press the enter key. When the last diskette is finished loading, you will be prompted to reboot the system.

Do you wish to boot the system with a networking kernel (y/<n>)? y

The installation of your REAL/IX system is now complete.
REAL/IX will be activated when the system is rebooted.

To activate networking, the system must be setup, then tcpip must be activated using sysadm.

Before the system is shutdown, do you wish to enter a shell in order to make any other adjustments (y/<n>)? n

Do you wish an automatic reboot (y/n)? y

By answering y (yes) to the above prompt, the system will automatically reboot.

If you answered n (no) to the last prompt, you must manually reboot the system. Make sure that there are no diskettes in the floppy drive so that the system boots from the REAL/IX partition on the hard disk.

The release diskettes are not needed further in the system setup process. You might need them if, for some reason, you need to re-install the operating system some time in the future (a hard disk crash, for example). Store the release diskettes in a safe place in case they are needed again and proceed to the next chapter for instructions on setting up the operating system environment.



Chapter 3

Setting Up the Operating System

Now that the basic operating system is installed, you must do basic system setup. This involves the following steps:

1. Rebooting the system.
2. Running `setup(1)` to: verify the installation using `mkcomply(1M)`; set the date, time, and time zone if they are not already correct; set up system and administrative passwords; and define the system node name.
3. Following the instructions in later chapters of this book or the *System Administrator's Guide* to: create additional file systems and user IDs; set up terminal and modem devices; and start and administer the TCP/IP network and line printer devices, if you have not done so already.

At this point, reboot the system. You will see a series of startup messages, including messages telling you that the built-in file systems are being checked. This is followed by a display of the systems's full memory configuration. The devices displayed will vary depending on your configuration.



The amount of memory available is expressed in bytes. Real memory reflects the number of physical memory locations; available memory is the amount of memory available for user processes after the kernel is configured. By default, the system reserves a portion of the physical memory not allocated for the text and data of the kernel for the system buffer cache. The number and size of buffers can be modified through `sysgen` after the system is set up. The *System Administrator's Guide* has information about the default buffer cache size and selecting the number and size of buffers.

The `CFLAGS` line shows the `-D` option supplied when the operating system was compiled. Note that an alternate kernel is available for kernel debugging at sites that are doing kernel development work; see the *System Administrator's Guide* for information on booting the debug kernel.

As the system checks for damage to the built-in file systems, you should check for any error messages that indicate file system problems. `fsck(1M)` is the file checking program. It is described fully in the *System Administrator's Guide*.

At this point, the system checks the file systems listed in the `/etc/fstab` file. If you created additional file systems during the installation procedure, they will not be checked at this point, but should be checked after you run `setup`.

Some messages will be displayed describing the system processes that will be executed every time you boot the system.



*The console messages (and the processes associated with them) that print out when the system boots are controlled by a series of scripts executed by the **init(1M)** process according to information in the **/etc/inittab** file. These are described in the System Administrator's Guide.*

Begin Setup Procedure

Log in using the **setup** login. This automatically invokes the **setup** script.

Console Login: **setup**

Upon entering the **setup** login, you may get the message: "Warning: last login did not exist, creating it."

Verify the Installation

The **setup** script allows you to verify that all files were installed as they should have been. Verification is done through the **mkcomply(1M)** command.



***mkcomply** can be run after **setup** through the **sysadm** package. However, the **setup** procedure involves modifying some of the files that **mkcomply** checks. If you run it later, you must study the inconsistencies reported to determine if they reflect changes you made to the system.*

*To initiate **mkcomply**, execute **sysadm**, select **syssetup** from the system administration menu, select **verinstall** from the system setup menu, and follow the prompts.*

If the verification is successful, **setup** will continue. If the verification is not successful, you will be asked if you want to continue the **setup** procedure. You will probably want to continue if the differences found are due to local changes. However, if the differences are not due to local changes, or if differences are found on a new installation, contact Customer Support before proceeding.

Do you want to verify system installation? [y, n, q] **y**

Set System Date

Setting the date and time is done as part of your initial system setup. In most cases the date and time will therefore be correct. If it is not, you can follow the instructions here to set the correct date, time, and time zone.

Change the time zone? [y, n, ?, q] **y**

A list of available time zones will appear; enter the appropriate number for your time zone area and answer the following question.

Enter zone number:

Does your time zone use Daylight Savings Time during the year? [y, n, ?, q]

Verify that the current date and time are correct. The **setup** program stops and restarts the **cron** daemon if you change the time zone and/or time information. To restart **cron**, you must answer **y** or **n** to the following question. If you answer **q**, **cron** will not be restarted and the new time zone and/or time will not be implemented.



*In most cases, the date and time displayed in the **setup** menu will be accurate; if not, it should be reset before you create or modify any files. If the time displayed is not accurate, or if the system tells you to restore the time of day after the initial setup, the backup battery for the clock may be low or dead.*

*Time zone information is accessed with the **TZ** environment variable, which is defined and exported in the **/etc/TIMEZONE** file.*

*The system date and time can be reset using the **date** command. The system should always be in single-user mode when using the **date** command to avoid confusing the **cron** daemon and file access times. Otherwise you can reset the date and time while in multi-user mode through the **sysadm** package; the **cron** daemon will restart automatically.*

Change the date and time? [y, n, ?, q]

Assign Passwords

Next you assign passwords for a set of system and administrative logins. These passwords should be selected carefully because they provide privileged access to the system. See the *System Administrator's Guide* for more information on choosing passwords and other security guidelines. For now, if you have not made decisions about access, you might want to assign the same password to all system logins. Later, when you have decided who needs access to which system logins, you can assign other passwords. Once you assign a password, it can only be changed by the owner or by the superuser.

The system gives you the opportunity to create as many user logins at this point as you like. However, we suggest that you complete system setup and create any additional file systems you may need before creating user logins. Otherwise, all logins you create now will have home directories in the */usr* file system. Filling up the */usr* file system can slow system performance and perhaps cause some system processes to stop because of the lack of disk space; it also makes it more difficult to install new releases of the operating system or add-on packages.

You can create other logins later by running either `sysadm adduser` or editing the */etc/shadow* file directly. See the *System Administrator's Guide* for instructions.

Type `q` at the following prompt if you do not want to create user logins at this time.

Enter user's full name [?,q]:



All passwords must be 6 to 8 characters long; at least two characters must be alphabetic (letters), and one must be a number or a special character.

Select a password that is easily remembered, but not one that is easily guessed. Avoid proper names and words that can be found in a dictionary, as well as dates, employment data, company buzzwords, and common acronyms, or reverse versions of any of these.

The **root** login controls superuser permissions. It has no restrictions and overrides all other logins, protections, and permissions. It allows the user access to the entire operating system, and as such, should be carefully protected.

The other administrative and system logins grant the power of a normal user login over a set of files and, in some cases, call a specific program (rather than the shell) on login; see the *System Administrator's Guide* for more information on these logins.



*After completing the setup procedure, you may want to make some of these system and administrative passwords accessible only through root; in other words, no one can log into them directly, but the superuser can access them without being prompted for a password. To do this, edit the */etc/shadow* file and put the string **NONE** into the password field as described in the *System Administrator's Guide*. This cannot be done during setup.*

Do you want to give passwords to administrative logins? [y, n, ?, q] y

If you answer y, you will be prompted to assign a password to each login. When you assign new passwords, you are asked to re-enter the password for verification.

Do you want to give the 'xxx' login a password? [y, n, ?, q] y

New password:

Re-enter new password:

This sequence will then be repeated for the system logins.

Do you want to give passwords to system logins? [y, n, ?, q]

Define the Node Name

Here you are asked to provide your machine node name. The node name identifies your machine to other machines on the network. You should never arbitrarily change this name. Other systems on the TCP/IP and UUCP network use the node name as part of the communication protocol. If the node name is arbitrarily changed, network calls to your system will fail.



*The node name of your machine is used whenever it communicates with other systems, and is displayed before the **login:** prompt except when the `/etc/issue` file contains a message.*

*This name is actually a tunable kernel parameter that can later be changed with **sysgen**. The node name can also be changed with **uname -S**, however, this does not change the **sysgen** definition of the node name. See the System Administrator's Guide for information on changing the node name in **sysgen**.*

It is very important that each system have a unique identifier. On the REAL/IX Operating System, it can be 1 to 8 characters long; we suggest that you keep it between 3 and 6 characters because other operating systems and application programs with which you may be networked may require it. Type in your new node name; the node name change will take place when **init level 2** is initiated.

```
This machine is currently called "realix".  
Do you want to change it? [y, n, ?, q]  y
```

```
What name do you want to give it? [q]
```

This completes your initial machine setup. At this point you will be logged off the system; the console will give you a login prompt.

Set TERM Variable

Before you can run screen-based commands such as the screen editor (**vi**), your environment has to define the terminal device you are using. The released system expects you to be using an AT386-M (monochrome) terminal. If you are using a different terminal, such as a color terminal, you need to change and export the value of TERM.

To view the current setting of TERM, type **echo \$TERM** at your shell. If this is not correct, you can change it at the shell by typing the following two lines (which sets \$TERM for an AT386 color terminal):

```
TERM=AT386
export TERM
```

You can set this up in your *.profile* file¹ so that it executes whenever you log into the system. If you log in from different types of terminals, you can use a shell script in the *.profile* that queries you as to the terminal you are using and sets up TERM appropriately. A list of available terminal types can be found in */etc/ttytype*. Also see *Concepts and Characteristics* for information on determining the correct TERM setting for various terminal types.

Print Error Message List

Kernel error messages are described in the */stand/MESSAGES* file. Making these messages available online enables you to search for a character string in the file for information. However, you may need to reference this list when the system is unavailable. To solve this problem, you should print a hardcopy of this listing now, and put it in your System Log for future reference. The following command prints this list to the printer configured to use the Line Printer Administration Utilities:

```
lp /stand/MESSAGES
```

Refer to the *System Administrator's Guide* for information on administering the Line Printer Administration Utilities.



Every system should have a System Log in which you record all problems and changes with the system. See the *System Administrator's Guide* for information on setting up and maintaining the System Log.

¹The *.profile* file controls the individual's working environment on the REAL/IX Operating System. See the *System Administrator's Guide* for more information.

Starting Accounting and Performance Statistics

The REAL/IX Operating System supports standard UNIX® System V job accounting and performance statistics. The system is released with these turned off; you may want to turn them on at this point. Machines dedicated to specific applications may not want to devote the system resources to these tasks, but the statistics they gather are useful for general purpose and development environments.

The routine tasks associated with job accounting and performance statistics are controlled by files in the `/usr/spool/cron/crontabs` directory:

- ❑ The `sys` file executes the `sa1` and `sa2` shell scripts. These scripts call the `sadc(1M)` program to gather performance statistics that can be viewed with the `sar(1/1M)` command. Statistics are gathered every 20 minutes between 8:00 AM and 5:00 PM Monday through Friday, and once an hour at other times.
- ❑ The `adm` file executes the standard reporting programs for job accounting. The accounting statistics are gathered automatically after `/etc/init.d/acct` is executed (when the system goes to multi-user state). (Note that if accounting is turned on, the accounting file can grow quite large.)

The simplest way to turn these facilities on is to remove the `#` signs (used to "comment out" the command) at the beginning of the `cron` lines; to change the times these processes run, modify the `cron` specifications in the appropriate lines. After modifying the `cron` files, either reboot the system or run `crontab(1)` to activate the files. The *System Administrator's Guide* discusses how these facilities work; after reading this information, you may want to customize their use.

Chapter 4

Starting the TCP/IP Network

To configure and start the TCP/IP¹ network and enable it on subsequent reboots, use **sysadm(1M)**. **sysadm** is a system administration menu interface that will take you step-by-step through the setup process and will allow you to:

- ☐ Enter your system address and the addresses of all the systems with whom you want to communicate.
- ☐ Download and start the TCP/IP network controller.
- ☐ Set up the network listener for UUCP.

There are several ways to invoke **sysadm**. The easiest way is to login as **root** (you will need to know the **root** password if one is assigned) and type **sysadm setuptcpip**. Or you may login as **sysadm** (you need to know the **sysadm** password if one was assigned) and go through each menu:

1. The first menu that is displayed is the System Administration menu. Choose the Package Management menu (**packagemgmt**).
2. From this menu choose the TCP/IP Management menu (**tcpip**).
3. Finally you must choose the Setup TCP/IP Network menu (**setuptcpip**).

The **setuptcpip** menu will take you through all the steps and prompt you for input. If you are not sure how to answer any prompt, type **?** for help. All the prompts in the **setuptcpip** script requiring user input follow. For more information on Internet addresses, see the *System Administrator's Guide*.

```
Would you like to add any host addresses to the
tcpip hostname database? [y, n, q] y
```

```
Enter the HOST NAME to be added to the tcpip
hostname database [?, q]:
```

```
Enter the HOST ADDRESS to be added to the tcpip
hostname database [?, q]:
```

¹ TCP/IP refers to the suite of Internet protocols, including TCP, UDP, IP, and ICMP. See Concepts and Characteristics for more information.

If you answer y to the following question, the superuser on the host can log in as **root** on your machine without a password or copy files to and from your machine as if he or she were the superuser. This is used to extend superuser privileges to select machines.

Do you want the superuser on *<host name>* to have superuser permission on your system for copying files and logging in? [y, n, ?, q]

If you answer y to the next question, all users other than the superuser on the host can log into your machine without a password. They may also copy files to and from your machine as if they were logged in on your machine. This is used to extend user privileges to select machines.

Do you want users other than the superuser on *<host name>* to have equal permissions on your system for copying files and logging in? [y, n, ?, q]

If there is no entry for another host in the host name database, a message to that affect will be printed, and the **setuptcpip** script will exit. There must be an address for the host that is defined to continue.

Do you want to add another host? [y, n, q] **n**

If an error message appears after answering y to the following question, contact Customer Support.

Would you like to start the TCP/IP network and enable it for subsequent reboots? [y, n, ?] **y**

Do you want to setup the tcpip network listener for use with UUCP and enable it for subsequent reboots? [y, n, ?] **y**

Chapter 5

Setting Up Terminals and Modems

Terminals, modems, and serial printers are identified to the REAL/IX Operating System as "TTY devices." This refers to the internal `tty` structure that is used to control device characteristics (see `termio(7)`). Each TTY device has a special device file created in the `/dev` directory.

PCs typically have two serial ports and one parallel port. The special device files for these ports are named `/dev/com1` and `/dev/com2` (serial ports) and `/dev/lp` (parallel port).

The released system includes configuration information for two serial ports, a console terminal, and seven virtual terminals (`/dev/vt01` through `/dev/vt07`). To activate the ports for these devices:

- ❑ activate the **getty** lines for the ports in the `/etc/inittab` file
- ❑ if necessary, change the default line descriptions for the **getty** lines to a different baud rate.

Modems are set up exactly like terminals, using the same **getty** lines, special device files, and so forth. Note, however, that the port is sometimes jumpered differently for a modem than for a terminal; see the device's hardware documentation for instructions.

Activating getty Lines

The `/etc/inittab` file includes **getty** lines for various devices.. As released, all these lines are turned off, and look like the following:

```
00:23:off:/etc/getty tty01 9600
01:23:off:/etc/getty tty02 9600
02:23:off:/etc/getty tty03 9600
03:23:off:/etc/getty tty04 9600
```

See `inittab(4)` or the *System Administrator's Guide* for a detailed explanation of the fields in `inittab` lines. To activate terminal lines, change the "off" action in the third field to "respawn" for each port.

For the new settings to be read by the operating system, execute `init q`.

Changing Default Baud Rate

The **getty** lines also define the initial terminal settings for the terminal (in other words, the settings each terminal has when the **login** program runs, before any **stty** settings in the user's `.profile` file are applied). These are defined by the baud rate (9600 in the listing above), which is used as an index into the `/etc/gettydefs` file. The 9600 is appropriate for 9600 baud terminals that are hard-wired to the system.

You may want to change the "9600" to some other value in the following cases:

- ❑ If you have terminals that are attached through some sort of data switch or phone lines, use the H version of the baud line (9600H for a data switch that uses 9600 baud, 1200H for phone lines). This ensures that the user is logged off if they break the connection (HUPCL, or hang-up-on-close).
- ❑ If most of your terminals run at some speed other than 9600, use that rate here. If you connect to the system with a terminal running at a different speed than the default for that line, you go through a hunt sequence. The last field in the `/etc/gettydefs` file determines the next baud rate that is sought: 9600 next hunts to 2400, which hunts to 1200, and so forth. If **getty** lines do not match the terminal settings, you can eventually get logged on, but it can be frustrating.

Eventually you may want to add additional `gettydefs` lines that are called in `inittab`, but that should not be necessary at this point.



MODCOMP, founded in 1970,
is a worldwide supplier of
real-time systems, products,
and services. MODCOMP is an
AEG company.

Corporate Headquarters:
Modular Computer Systems, Inc.
1650 West McNab Road
P.O. Box 6099
Ft. Lauderdale, FL 33309-1088
Tel: (305) 974-1380
Twx: 510-956-9414

International Headquarters:
Modular Computer Services, Inc.
The Business Centre
Molly Millars Lane
Wokingham, Berkshire
RG11 2JQ, UK
Tel: 0734-786808
Fax: 0734-776399

Americas Headquarters:
Modular Computer Systems, Inc.
1650 West McNab Road
P.O. Box 6099
Ft. Lauderdale, FL 33309-1088
Tel: (305) 974-1380
Twx: 510-956-9414

MODCOMP sales and service
offices are located throughout
the world.

Copyright © 1993, Modular Computer Systems, Inc.
MODCOMP and Modular Computer Systems, Inc. are registered trademarks of Modular Computer Systems, Inc.

System Administrator's Guide

AEG

Property of Logical Data Corporation

REAL/IX® Operating System **for 386/486 ISA Computers**

RECEIVED MAY 18 1995

232-870002-000

MODCOMP

System Administrator's Guide

REAL/IX[®] Operating System for 386/486 ISA Computers

AEG

Property of Logical Data Corporation

RECEIVED MAY 1 8 1995

MODCOMP

PROPRIETARY NOTICE

THE INFORMATION AND DESIGNS DISCLOSED HEREIN WERE ORIGINATED BY AND ARE THE PROPERTY OF MODULAR COMPUTER SYSTEMS, INC. (MODCOMP). MODCOMP RESERVES ALL PATENT, PROPRIETARY DESIGN, MANUFACTURING, SOFTWARE PROPERTY, REPRODUCTION, USE, AND SALES RIGHTS THERETO, AND RIGHTS TO ANY ARTICLE DISCLOSED THEREIN. THIS INFORMATION IS MADE AVAILABLE UPON THE CONDITION THAT THE PROPRIETARY DESIGNS AND PRODUCTS DESCRIBED HEREIN WILL BE HELD IN ABSOLUTE CONFIDENCE AND MAY NOT BE DISCLOSED IN WHOLE OR IN PART TO OTHERS WITHOUT THE PRIOR WRITTEN PERMISSION OF MODULAR COMPUTER SYSTEMS, INC. THE FOREGOING DOES NOT APPLY TO VENDOR PROPRIETARY PRODUCTS.

SPECIFICATIONS REMAIN SUBJECT TO CHANGE IN ORDER TO ALLOW THE INTRODUCTION OF DESIGN IMPROVEMENTS.

FOR GOVERNMENT USE THE FOLLOWING SHALL APPLY:

RESTRICTED RIGHTS LEGEND

USE, DUPLICATION, OR DISCLOSURE BY THE GOVERNMENT IS SUBJECT TO RESTRICTIONS AS SET FORTH IN RIGHTS IN DATA CLAUSES DOE 952.227-75, DOD 52.227-7013, AND NASA 18-52.227-74 (AS THEY APPLY TO APPROPRIATE AGENCIES).

MODULAR COMPUTER SYSTEMS, INC.
1650 WEST McNAB ROAD
P.O. BOX 6099
FORT LAUDERDALE, FL 33340-6099

THIS MANUAL IS SUPPLIED WITHOUT REPRESENTATION OR WARRANTY OF ANY KIND. MODULAR COMPUTER SYSTEMS, INC. THEREFORE ASSUMES NO RESPONSIBILITY AND SHALL HAVE NO LIABILITY OF ANY KIND ARISING FROM THE SUPPLY OR USE OF THIS PUBLICATION OR ANY MATERIAL CONTAINED HEREIN.

THE PRODUCT DESCRIBED HEREIN IS BASED ON COPYRIGHTED SOFTWARE FROM VARIOUS COMPANIES INCLUDING MOTOROLA, INC., AND UNIX SYSTEM LABORATORIES, INC. THE SOFTWARE IS FURNISHED UNDER A LICENSE AGREEMENT AND MAY BE USED ONLY IN ACCORDANCE WITH THE TERMS OF SUCH AGREEMENT.

MANUAL HISTORY

Manual Order Number: 232-870002

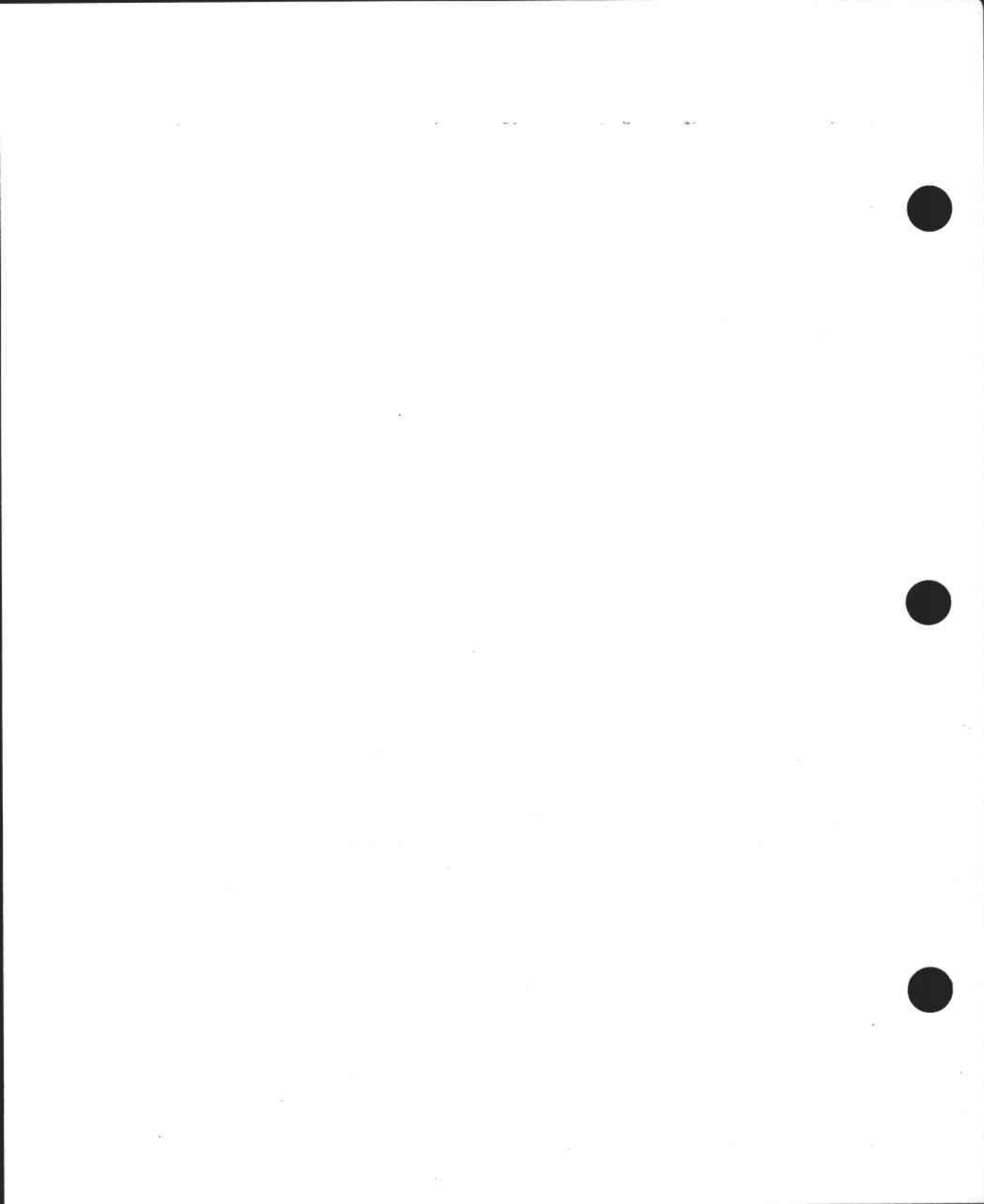
Title: REAL/IX Operating System for 386/486 ISA Computers, System Administrator's Guide

Revision Level	Date Issued	Description
000	09/93	Initial Issue.

Contents subject to change without notice.

Copyright © 1993, by Modular Computer Systems, Inc.
All Rights Reserved.
Printed in the United States of America.

Portions of this document are based on or reprinted from copyrighted documents
by permission of Motorola, Inc. and AT&T.



PREFACE

This manual gives background information and instructions for administering the REAL/IX Operating System. It should be used in conjunction with the reference manuals listed below. See the *Software Installation Guide* for instructions on installing and setting up the operating system.

Open Architecture Systems Defined

The term "open architecture system", in its simplest form, implies that a user may add a variety of vendors' components to a single system. This is possible when certain industry-accepted standards have been implemented in the system. MODCOMP open architecture systems are based on such software and hardware standards as the UNIX System V operating system; VMEbus, MULTIBUS, and SCSI bus interfaces; and CPUs built around standard microprocessors. By building on these standards, open architecture systems provide computer solutions that are portable and compatible.

The REAL/IX Operating System¹ allows applications to be ported easily between traditional UNIX systems and MODCOMP open architecture systems. Furthermore, by using industry standard I/O buses, MODCOMP open architecture systems ensure compatibility among a wide range of peripheral and I/O devices and the ability to expand as needs dictate. MODCOMP open architecture systems meet networking and communications needs with such industry standards as Ethernet and TCP/IP and have the flexibility to accommodate new standards as they are developed.

Related Publications

Refer to the following publications for additional information.

Books for All System Users

Concepts and Characteristics

Gives an overview of the internals of the REAL/IX Operating System and an introduction to the tools and facilities that are available.

Reference Manual, Sections 1, 1M, and 1R

Contains manual pages for user commands (Section 1), administrative commands (Section 1M), and realtime commands (Section 1R).

Reference Manual, Sections 2, 3, and 5

Contains manual pages for system calls (Section 2), library routines (Sections 3C, 3M, 3N, 3S, and 3X), and miscellaneous information (Section 5).

¹The REAL/IX Operating System, featuring realtime and multiprocessing capabilities, is the MODCOMP implementation of the UNIX System Laboratories UNIX System V operating system.

Reference Manual, Sections 4, 7, and 7A

Contains manual pages for system files (Section 4), special device files for standard devices (Section 7), and special device files for add-on packages (Section 7A).

User's Guide

Discusses basic user procedures including the login procedure and getting around the file system. Information is included about general user tools; for example, the *vi* and *ed* text editors, electronic mail, the shell programming language, and the Korn shell.

Using UUCP and Usenet

Introduces UUCP communications, describes how to transfer files and execute remote commands over UUCP, how to check on UUCP requests, and how to access the Usenet electronic bulletin board.

Books for System Administrators

Software Installation Guide

Gives instructions for installing the operating system (either for the first time or as an upgrade) and initially setting up the system.

System Administrator's Guide

Gives instructions and background information about administering the REAL/IX Operating System. Topics covered include ensuring system security; creating and maintaining user and group IDs; working with file systems (creating, repairing, backing up); setting up terminals and printers; using the *sysgen(1M)* utility to modify tunable parameters and to configure or deconfigure standard system devices; and setting up and using the Job Accounting System. Appendixes discuss the system files that control system operations and the file naming conventions for special device files.

Software Engineering Release Notes

Gives an overview of the new features in this release of the REAL/IX Operating System and provides usage notes for the system.

Managing UUCP and Usenet

Provides background information about UUCP for administrators and gives instructions for setting up a UUCP link, verifying that the link works, administering UUCP communications, and setting up and administering the Usenet access. This information is supplemented by the *System Administrator's Guide*, which includes information for administering UUCP over the TCP/IP protocol, and the *Software Installation Guide*.

Books for Programmers

Languages and Support Tools Guide

Provides tutorials for many of the special purpose languages and the programming support tools available on the REAL/IX Operating System.

Languages and Support Tools Guide Supplement

Contains information that is specific to the released system as it operates in the native microprocessor environment of your hardware platform.

Programmer's Guide

Gives an overview of the REAL/IX Operating System and realtime computing, describes the REAL/IX programming environment and the operating system interface, and provides programming examples for using the realtime extensions of the REAL/IX Operating System as well as the standard UNIX operating system features.

The C Programming Language, First Edition

Describes the traditional UNIX C language.

The C Programming Language, Second Edition

Describes the ANSI C language.

Books for Kernel Programmers

Driver Development Guide

Introduces the process of writing device drivers for the REAL/IX Operating System, including detailed information about porting and installing drivers.

Kernel Programming Guide

Gives background information about topics of interest to programmers writing device drivers and system calls. Topics discussed include how drivers and system calls execute and how various types of I/O operations are implemented.

Kernel Reference Manual

Contains reference pages for driver entry-point routines (Section D2X), kernel functions and macros (Section D3X), and kernel data structures (Section D4X) used for coding system calls and device drivers.

Industry Standard Publications

The REAL/IX Operating System complies with the industry standards listed below. These standards are commercially available and can be obtained from the following sources. While an effort was made to ensure that the ordering information was complete and up-to-date at time of printing, we cannot guarantee its accuracy.

System V Interface Definition (SVID)

AT&T Customer Information Center (CIC)

Customer Service Representative

P.O. Box 19901

Indianapolis, IN 46219

Phone: (800) 432-6600 (Inside U.S.A.)

(800) 255-1242 (Inside Canada)

(317) 352-8557 (Outside U.S.A. and Canada)

Documentation Conventions

The following table gives the textual conventions used in this book. Note that commands, library routines, system calls, kernel functions, driver entry points, files, and data structures are sometimes followed by a number enclosed in parentheses (for instance, "**cat**(1)"). This denotes the reference section in which they are located; Sections D2X, D3X, and D4X are in the *Kernel Reference Manual*; all others are in the *Reference Manual* volumes and available online through the **man**(1) command. Commands followed by empty parentheses (for instance, "**false**()") are available through the **man** command, but do not have their own manual page.





Style	Item	Example
bold	Shell commands	cat or cat(1)
bold	Library routines	printf or printf(3s)
bold	System call names	open or open(2)
bold	Kernel function names	copyin or copyin(D3X)
bold	Driver entry point names	strategy or strategy(D2X)
bold	Script names	MOUNTFSYS or S03MOUNTFSYS
<i>italics</i>	File names	<i>/etc/passwd</i>
monofont	Data structures	<code>user</code> or <code>user(D4X)</code>
bold	Data structure members	u_count or u.u_count
bold	Literal text in example	cat filename
<i>italics</i>	Variable text in example	
monofont	Code representations	<code>if size <= 0 return NULL;</code>
monofont	Screen representations	<code>Enter a number or q to quit: 2</code>
monobold	Operator input	
?	Single character wildcard	<code>/dev/tty??</code>
*	Multi-character wildcard	<code>/dev/*_ctl</code>
 WARNING!	<i>highlights information that, if not observed, could cause bodily harm</i>	
 CAUTION	<i>highlights information that, if not observed, could cause the system or a procedure or practice to fail or could damage existing data on the system</i>	
 NOTE	<i>highlights relevant information that does not require a caution or warning</i>	
 HINT	<i>identifies material that is indirectly related to the subject matter being discussed. For instance, a procedure may specify one way of doing the task, and the HINT would explain why it is done this way or optional ways of accomplishing the same task.</i>	



TABLE OF CONTENTS

	Page
Chapter 1 Introduction	
Related Documentation	1-1
Administrative Privileges	1-2
Customizing the Environment	1-3
REAL/IX Administrative Features	1-4
General Features	1-4
Realtime Features	1-5
Chapter 2 System Operations	
Maintaining a System Log	2-1
Startup and Shutdown	2-3
init and shutdown Commands	2-3
Transitions between Machine States	2-4
System Startup	2-5
Booting the Debug Kernel	2-6
System Panics	2-7
Autodump and Autoboot	2-7
Scheduling Downtime	2-9
System Security	2-9
Important Security Guidelines	2-10
setuid and setgid	2-10
Check setuid Bits Owned by root	2-11
Check setuid Bits in the root File System	2-11
Check setuid Bits in Other File Systems	2-12
The login Program	2-12
Shadow Password File	2-13
Installing /etc/shadow	2-13
Backing out /etc/shadow	2-13
Chapter 3 User Management	
Creating and Modifying User IDs	3-2
/etc/passwd	3-2
Password Aging	3-3
Adding Users	3-5
Special Administrative and System Logins	3-8
Adding Realtime Users	3-9
Changing User IDs	3-10
Deleting User IDs	3-10
Listing User IDs	3-11
Locking Unused Logins	3-11

Chapter 3 User Management [continued]

Creating and Modifying Group IDs	3-12
/etc/group	3-12
Adding and Changing Groups	3-13
Deleting Groups	3-14
Listing Groups	3-14
Establishing the User Environment	3-15
Environment Variables	3-16
Character Display Options (stty)	3-17
umask	3-17
Default Shell and Restricted Shell	3-18
User Communications Services	3-19
Pre-Login Message	3-19
Login Message (Message of the Day)	3-19
Broadcast to All Users	3-20
mail and mailx	3-20
news	3-20

Chapter 4 Creating File Systems

Disk and File System Overview	4-3
File Systems and Special Device Files	4-3
About the F5 File System Architecture	4-3
About Logical Blocks and the Buffer Cache	4-4
Formatting the Disk	4-6
Using mkfs to Create a File System	4-6
Labeling the File System	4-8
Mounting the File System	4-9
Unmounting File Systems	4-11
Creating a lost+found Directory	4-11
Moving Users to a File System	4-12
Expanding and Reconfiguring File Systems	4-13
Monitoring Disk Usage	4-14
Monitoring File System Space	4-14
Cleaning Up Log Files	4-15
Identifying Large and Inactive Files	4-16
Reorganizing the File System	4-17

Chapter 5 Maintaining File Systems

Backup and Restore Procedures	5-1
Scheduling Backups	5-2
General Guidelines	5-2
Summary of Backup and Restore Commands	5-3

Chapter 5 Maintaining File Systems [continued]

Backing Up an Entire File System	5-5
volcopy	5-5
cpio and sysadm backup	5-6
tar	5-7
dcopy	5-8
dd	5-8
Running Incremental Backups	5-9
cpio and sysadm backup	5-9
finc(1M)	5-11
frec(1M)	5-12
Backing Up Specific Directory Trees and Files	5-13
cpio	5-13
sysadm store	5-13
Using fsck to Check and Repair the File System	5-14
Running fsck	5-14
/etc/checklist	5-16
Using fsdb to Repair Damaged File Systems	5-17
Using fsdb to View a File System	5-19
Starting the fsdb Session	5-19
Viewing the Structure of an inode	5-20
Viewing the Data for an inode	5-22
Using fsdb to View the Super Block	5-24
Example of Repairing a File System with fsdb	5-28

Chapter 6 Performance Management

Creating an Efficient Environment	6-3
Hardware Configuration	6-3
Shifting Workload	6-3
File System Structure	6-4
Choosing the Size and Number of Buffers	6-4
Disk I/O Balancing	6-4
Setting Text Bit (Sticky Bit)	6-5
Arranging the Multiple Swap Areas	6-5
Reducing Memory Usage	6-6
Maintaining System Efficiency	6-8
File System Organization	6-8
Cleaning Up Log Files	6-9
Directory Organization	6-9
Free Blocks and Inodes	6-9
Solving Performance Problems	6-10
Check for Excessive Paging	6-11
Disk or Swapping Bottleneck	6-13

Chapter 6 Performance Management [continued]

Improving System Usage Patterns	6-14
Efficient Application Programs	6-14
Measuring Performance	6-16
Kernel Profiler	6-17
ps	6-19
sar	6-20
sar -a	6-21
sar -b	6-22
sar -B	6-23
sar -c	6-24
sar -C	6-25
sar -d	6-26
sar -E	6-27
sar -I	6-28
sar -m	6-29
sar -p	6-30
sar -q	6-31
sar -r	6-32
sar -T	6-33
sar -u	6-34
sar -v	6-35
sar -w	6-36
sar -y	6-37
sar -A	6-38
systat	6-38
timex	6-39
icount(1M)	6-40
bfree(1M)	6-42
Tunable Parameters	6-43
Modifying Tunable Parameters	6-43
Memory Usage Parameters	6-44
System Tables for All Processes	6-46
Resources for User Programs	6-49
Configuring the Buffer Cache	6-50
Hash Buckets	6-52
Tunable Parameters for Multi-Block Transfers	6-53
Tunable Parameters for Disk Updates	6-55
Tunable Parameters for Physical I/O	6-56
AUTODUMP and SYSRESET Parameters	6-56
Paging Parameters	6-58
Process Priority Parameters	6-59
RESIZESWAP Parameter	6-61
NODE Parameter	6-61

Chapter 6 Performance Management [continued]

Other Kernel Parameters	6-62
STREAMS Parameters	6-63
Interprocess Communications Parameters	6-66
Binary Semaphore Parameters	6-66
Message Parameters	6-67
Semaphore Parameters	6-68
Shared Memory Parameters	6-69
Tunable Parameters for Realtime Facilities	6-71
Tunable Parameters for Asynchronous I/O	6-71
Tunable Parameters for Connected Interrupts	6-72
Tunable Parameters for Common Event Mechanism	6-72
Tunable Parameters for Timer Subsystem	6-73
Tunable Parameters for Disjoint I/O	6-76

Chapter 7 TTY Management

Login Cycle	7-1
/etc/getty Lines in the /etc/inittab File	7-3
Lines in the /etc/gettydefs File	7-5
Using sysadm to Set Up TTYs	7-7
sysadm lineset	7-7
sysadm mklineset	7-8
sysadm modtty	7-9
Controlling Block-Mode Display Processing	7-11

Chapter 8 Line Printer Administration

Administrative Commands	8-2
lpadm(1M)	8-2
lpsched(1M)	8-4
lpshut(1M)	8-5
lpmove(1M)	8-5
accept(1M)	8-6
reject(1M)	8-6
Setting Up the Printer	8-7
Printer Interface Programs	8-7
Model Interface Programs	8-8
Writing Interface Programs	8-8
Sample Interface Programs	8-9
Files and Directories	8-11
/usr/spool/lp/fifos	8-11
/usr/spool/lp/default	8-11
/usr/spool/lp/logs	8-11

Chapter 8 Line Printer Administration [continued]

/usr/spool/lp/system/pstatus	8-11
/usr/spool/lp/admins/lp	8-12
/usr/spool/lp/model	8-12
/usr/spool/lp/requests and /usr/spool/lp/temp	8-12
/usr/spool/lp/SCHEDLOCK	8-12
Cleaning Out Log Files	8-13

Chapter 9 sysgen

Running sysgen	9-1
Notes for Using sysgen	9-2
A Walk Through the Screens	9-3
Rebuilding the Operating System	9-4
Rebooting the Operating System	9-4
Creating New Configurations	9-5
Enabling and Disabling Collections and Devices	9-6
Enabling and Disabling Collections	9-6
Enabling and Disabling Devices	9-8
Modifying Tunable Parameters	9-9

Chapter 10 Job Accounting

Overview of Job Accounting	10-1
Gathering Raw Statistics	10-2
Daily Summaries	10-3
Monthly Summaries	10-3
Setting Up Job Accounting	10-3
Job Accounting Scripts	10-5
ckpacct	10-5
dodisk	10-5
runacct	10-6
monacct	10-6
Job Accounting Data Files	10-7
Files in /usr/adm Directory	10-7
Files in nite Directory	10-8
Files in sum Directory	10-9
Files in fiscal Directory	10-9
Troubleshooting the Accounting System	10-9
Fixing wtmp Errors	10-10
Fixing tacct Errors	10-10
Interpreting Job Accounting Reports	10-11
Daily Report	10-11
Daily Usage Report	10-12

Chapter 10 Job Accounting [continued]

Daily Command Summary	10-13
---------------------------------	-------

Chapter 11 Administering Internet Protocols

Addressing	11-1
Internet Addresses	11-1
Host Names	11-3
Ethernet Addresses	11-4
Mapping an Internet Address to Its Link Level Address	11-4
The Network Database	11-5
The /etc/hosts File	11-5
The /etc/hosts.equiv File	11-7
The .rhosts File	11-8
The /etc/services File	11-9
The /etc/protocols File	11-11
Interhost Access Control	11-12
telnet	11-12
File Transfer Protocol (ftp)	11-12
The r-series Commands	11-12
The Internet Super-Server	11-13
Establishing Connections	11-14
Making ftp Connections	11-15
Making remsh Connections	11-15
Making rcp Connections	11-16
Making telnet and rlogin Connections	11-17
Observing Network Processes	11-19
UUCP Over TCP/IP	11-20
The sysadm setuptcpip Command	11-20
The Network Listener	11-20
Setting Up the Systems File	11-21
The Devices, Dialers, and Devconfig Files	11-22

Appendix A Special Device Files	A-1
Structure of Special Device Files	A-2
Major and Minor Numbers	A-2
Block and Character Devices	A-3
/dev Subdirectories	A-3
Creating and Linking Special Device Files	A-4
Special Files for Memory	A-5
Special Device Files for Terminals	A-5
Console Special Files	A-6
System Special Device Files	A-6

	Page
Appendix B Administrative Directories and Files	B-1
Released File Systems	B-2
/etc/TIMEZONE	B-3
TZ Environment Variable	B-3
Editing the /etc/TIMEZONE File	B-3
Files that Control cron	B-4
Files in the crontabs Directory	B-5
The atjobs Directory	B-7
The queuedefs File	B-7
crontabs/sys	B-8
Controlling Use of cron	B-8
/usr/lib/cron/log	B-9
/usr/adm/sulog	B-10
Appendix C Controlling Run State Transitions	C-1
/etc/init.d Directory	C-2
/etc/inittab	C-4
Booting the System	C-6
/etc/rc2 Script and /etc/rc2.d Directory	C-6
/etc/rc0	C-8
/etc/rc0.d Directory	C-8
/etc/shutdown	C-8
Appendix D fsck Internals and Error Messages	D-1
File System Components	D-1
The Super Block	D-1
File System Size and Inode List Size	D-1
Free Block List (S5 Architecture Only)	D-2
Free Block Count	D-2
Free Inode Count	D-2
The Free Block Bitmap (F5 Architecture Only)	D-3
The Inode List	D-3
Format and Type	D-3
Link Count	D-4
Duplicate Blocks	D-4
Bad Block Numbers	D-4
Inode Size	D-5
Extent List (F5 Architecture Only)	D-5
Indirect Blocks	D-5
Directory Data Blocks	D-5
Directory Unallocated	D-6
Bad Inode Number	D-6
Incorrect "." and ".." Entries	D-6
Disconnected Directories	D-6
Regular Data Blocks	D-6

Appendix D fsck Internals and Error Messages [continued]

fsck Phases and Error Messages D-7

 General Error Messages D-8

 Initialization Phase D-9

 Phase 1: Check Blocks and Sizes D-9

 Phase 1B: Rescan for More Duplicate Blocks D-11

 Phase 2: Check Path Names D-11

 Phase 3: Check Connectivity D-13

 Phase 4: Check Reference Counts D-14

 Phase 5: Check Free List or Free Block Bitmap D-16

 Phase 6: Salvage Free List D-18

 Cleanup Phase D-18

Index Index-1

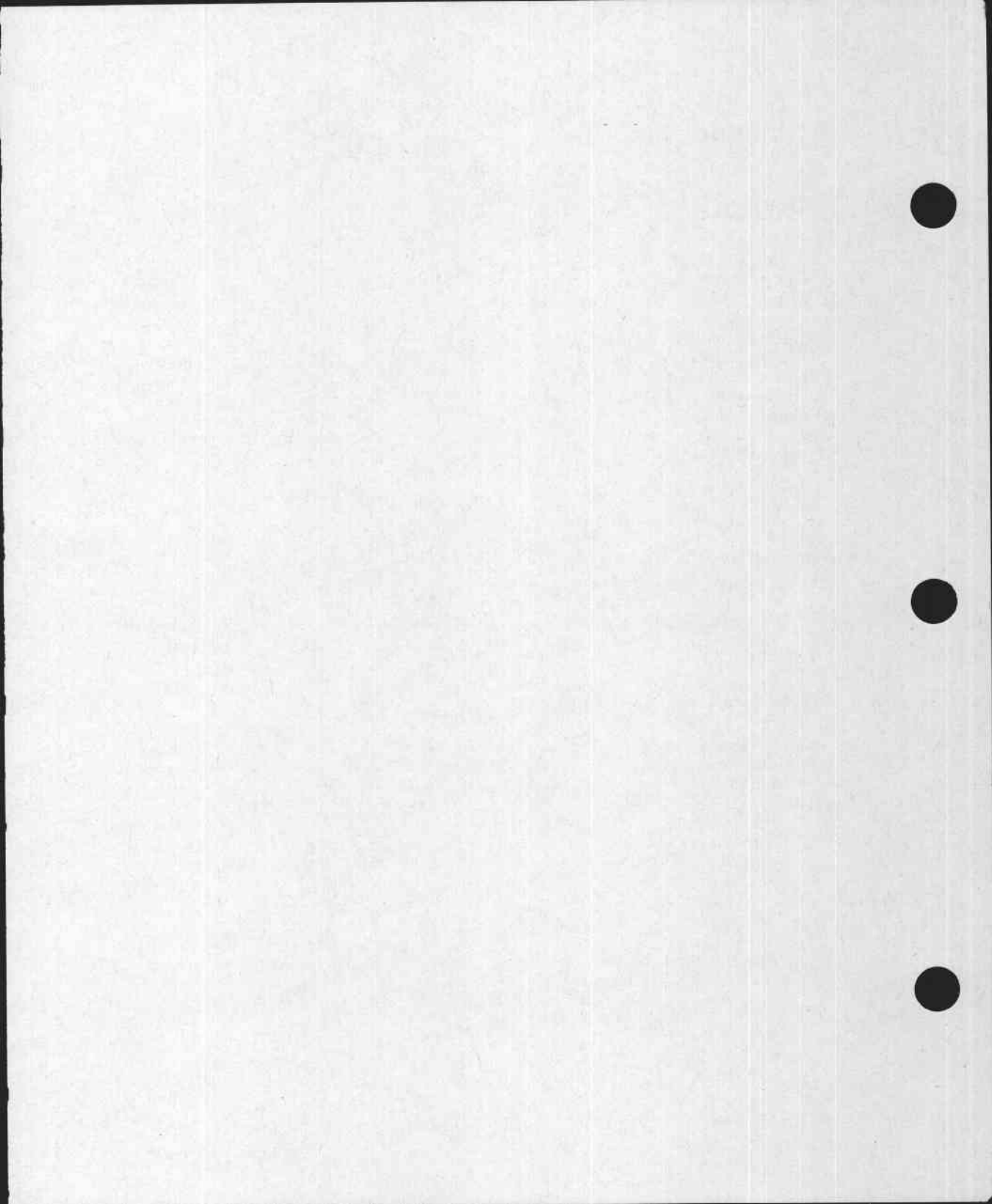
LIST OF FIGURES

	Page
4-1 Logical Blocks in a 29168 Block File System	4-4
5-1 Basic File System Information Displayed when fsdb is Invoked	5-19
5-2 Display of inode 2	5-20
5-3 Graphic Representation of Extent List	5-21
6-1 Profiler Statistics	6-17
7-1 The login Cycle	7-2
11-1 Sample /etc/hosts Entries	11-6
11-2 Sample /etc/services Entries	11-10
11-3 Sample /etc/protocols Entries	11-11
11-4 Inetd Spawned Server Process to Take Over Connection Request	11-14
11-5 ftp Connection	11-15
11-6 remsh Connection	11-15
11-7 rcp Connection	11-16
11-8 Server Side of a telnet or rlogin Connection	11-17
11-9 User Side of an rlogin Connection	11-18
11-10 Observing Network Processes with ps -eal	11-19
A-1 ls -l Listings for Regular and Special Files	A-2

LIST OF TABLES

	Page
2-1 Transitions between Operating System States	2-4
2-2 Operating System Transitions	2-5
3-1 Code for Password Aging	3-3
3-2 System Logins Defined During Setup	3-8
5-1 fsdb Command Summary	5-18
5-2 Flag Values for Preallocated Files	5-21
6-1 Reducing Memory Usage through Kernel Parameters	6-7
6-2 Overview of Performance Problems	6-11
6-3 sar Options	6-20
6-4 Memory Usage Parameters	6-45
6-5 Buffer Cache Parameters	6-50
6-6 Multi-Block Transfer Parameters	6-53
6-7 Disk Update Parameters	6-55
6-8 Physical I/O Parameters	6-56
6-9 Paging Parameters	6-58
6-10 STREAMS Parameters - General	6-63
6-11 Parameters to Control STREAMS Data Blocks and Buffers	6-65
7-1 terminfo Suffixes	7-11
8-1 User Commands for the LP Spooling System	8-1
8-2 Administrative Commands for the LP Spooling System	8-2
9-1 sysgen Typing Conventions	9-2
9-2 Updating Files	9-4
9-3 Collections that Can be Disabled	9-7
9-4 Collections that Cannot be Disabled	9-7
9-5 System-Specific Collections	9-8
10-1 Raw Accounting Data	10-2
10-2 Legal Holidays in the United States	10-4
10-3 Files in the /usr/adm Directory	10-7
10-4 Files in /usr/adm/acct/nite Directory	10-8
10-5 Files in /usr/adm/acct/sum Directory	10-9
10-6 Files in /usr/adm/acct/fiscal Directory	10-9
11-1 Network Classes	11-3
11-2 Pseudo-TTY Master and Slave Entry Points	11-17
A-1 Permissions for Terminal Devices	A-6
A-2 System Special Device Files Changed by Administrators	A-7
A-3 System Special Device Files Never Changed by Administrators	A-8
B-1 Directories in / and /usr	B-2
B-2 Time Specifications for crontabs	B-5
B-3 Permissions to Use cron Facility	B-8
C-1 Controlling Run States	C-1
C-2 Contents of init.d Directory	C-3

1. Introduction



Chapter 1

Introduction

This guide provides instructions for administering and operating the REAL/IX® Operating System. System administration and operation are closely related tasks that are essential to running an effective REAL/IX installation. System operation typically involves daily tasks such as performing system startup and shutdown procedures, backing up and restoring files and file systems, and ensuring that file systems are well organized. System administration is the art of keeping the system running smoothly by monitoring system usage and resource availability, identifying hardware and software problems and arranging appropriate corrective actions, creating and closing user accounts and user groups, and maintaining system security. In this guide, the term *system administration* refers to both administrative and operation tasks. This chapter introduces the administrative tasks with specific information on:

- related documentation that provides essential administrative information
- a discussion of administrative privileges
- an overview of administrative tasks and pointers to where they are discussed
- a discussion of ways the administrator can customize the environment
- a summary of the major REAL/IX administrative features that are different from UNIX® System V

Related Documentation

Good system administration requires an understanding of all facets of the hardware and software environment, and so the administrator is likely to reference all the documents listed in the Preface to this book. A few of these books are intimately tied to the administrative tasks and so are listed below:

Sections 1, 1M, and 1R Reference Manual provides reference material on all system commands. It is arranged into three sections: Section 1 is commands that can be executed by any system user; Section 1M is administrative commands, most of which can be executed only by the superuser; and Section 1R is commands that can be executed by the superuser or any realtime user.

Sections 4, 7, and 7A Reference Manual provides reference information on system files that the administrator must maintain (Section 4), special device files used to access standard system devices (Section 7), and special device files for add-on packages (Section 7A).

Concepts and Characteristics provides background information on how the operating system and its special features work. Administrators need to be familiar with this information to properly use the operating system features and to select appropriate values for tunable parameters.

Software Installation Guide gives instructions for initially installing or upgrading, and setting up the operating system.

Programmer's Guide includes information on writing applications that meet the performance specifications.

Driver Development Guide and *Kernel Programming Guide* include information on writing drivers and system calls for optimal performance and testing the performance of drivers.

Administrative Privileges

Many of the tasks and commands discussed in this book require special administrative privileges. These are sometimes referred to as *root* privileges or superuser privileges. The password required to use these privileges is associated with the *root* line in the */etc/passwd* file that lists all users on the system.

Because the superuser privileges bypass all security checks on the system, you should restrict knowledge of this password to only those trusted users who need to have it. Chapter 3 discusses how to set up groups. Establishing groups can help minimize the number of tasks that must be done as superuser.

We recommend that even users who are entitled to superuser privileges do not log in with such privileges except when doing tasks that require them. We all make mistakes; when an individual makes a mistake as a superuser, the results can be disastrous. When logged in as the superuser, be extremely careful about walking away from the active terminal. In addition, because anyone who logs in as *root* has superuser privileges as well as a certain amount of anonymity, you have greater accountability if people log in using their own login and then access the superuser privilege through the *su(1M)* command. The system maintains a log (*/usr/adm/sulog*) of superuser activities; this is discussed in Appendix B.

Note that some commands (such as the screen editor, *vi(1)*) do not understand superuser privileges. If you edit a file that only has read privileges, you will not be able to write the changed file with **ZZ** or **w**; you can, however, write the changed file by using the **w!** editor command.

Customizing the Environment

Throughout this book, instructions are given for various administrative tasks as if you were going to do them manually. However, most administrators write a number of shell scripts that automate these tasks, both to save themselves tedium and to ensure that tasks are done consistently for the site no matter who performs them. Examples of procedures for which you might create shell scripts are:

- ☐ running backups and deleting or truncating log files in the file system after the backup completes
- ☐ checking free blocks and inodes in the file systems and notifying the administrator by mail when they fall below a certain threshold
- ☐ creating user and group IDs
- ☐ checking system logs and error reports
- ☐ cleaning out system files that grow
- ☐ printing out daily reports
- ☐ starting and stopping application processes

These scripts can be run using any of the following methods:

- ☐ manually on an as-needed basis by anyone with appropriate permissions; use `at(1)` to schedule the run for a future time (such as during off-hours).
- ☐ at regularly scheduled time intervals through the `crontab(1)` command
- ☐ whenever the system enters or leaves multi-user state by installing the script in the `/etc/rc2.d` directory
- ☐ whenever the system is shutdown or rebooted by installing the script in the `/etc/rc0.d` directory
- ☐ whenever the system enters single-user state by adding a command line to the `/etc/inittab` file

In addition, you can customize the configuration of the kernel to provide the appropriate allocation of resources for your environment by modifying the `sysgen(1M)` parameters. See Chapter 6 for guidelines on determining the appropriate configuration.

REAL/IX Administrative Features

Administrators who have worked on UNIX operating systems before will find most tasks similar to the other system, with allowances for small differences between the various flavors of the operating system. The REAL/IX Operating System is based on AT&T's UNIX System V. The following sections summarize the administrative differences and enhancements you may encounter, with pointers to places in this and other books where you can find details on each feature. The features are listed in two groups: general administrative tasks and administrative tasks associated with the realtime features included in the REAL/IX Operating System.

General Features

The REAL/IX Operating System includes a number of general features to streamline administrative tasks and improve system performance. These are summarized below:

- ❑ **alternate file system architecture:** In addition to the standard UNIX System V file system architecture (S5), the REAL/IX Operating System includes a fast file system architecture (F5) that solves a number of the performance problems associated with the file system architecture. Some of these improvements are specifically for realtime applications, but the fast file architecture is more efficient for many general applications as well. Both file system architectures support logical block sizes ranging from 512 bytes to 128 Kbytes. Chapter 4 discusses how to create file systems using the F5 file system architecture.
- ❑ **variable buffer sizes:** To further enhance the performance of file access, the system buffer cache can be configured to have a mix of buffer sizes. Chapter 6 discusses how to implement this.
- ❑ **menu-driven configuration specifications:** The REAL/IX Operating System includes the `sysgen(1M)` program to simplify the task of configuring the system. Rather than edit precisely formatted files to modify tunable parameters or specify the devices configured in the system, you can use these screens to provide the configuration information. `sysgen` then updates all requisite files. See Chapter 9 for information on running `sysgen`.

Realtime Features

Most realtime features are implemented in the application programs and associated drivers. *Concepts and Characteristics* discusses how these features work; the *Programmer's Guide* discusses how to use these features in application programs. The following list summarizes the functionality available to the administrator for these features.

- ❑ **realtime privileges:** In addition to the regular and superuser privileges, a number of commands and system calls can be executed by users with realtime privileges. Chapter 3 explains how to grant realtime privileges to users with the `setrtusers(1M)` command and populating the `setrtusers` script in the `/etc/rc2.d` directory.
- ❑ **control of realtime processes:** Realtime processes execute at priorities 0 through 127; the operating system never modifies the priority unless the program or the administrator requests it. The administrator can set the priority of an executing realtime process and can issue a command to resume execution of a realtime process that has suspended itself. To be sure that you will be able to execute these commands, you should keep one terminal logged on at a realtime priority higher than any realtime application programs. See the *Programmer's Guide* for information on how to use the administrative commands that control realtime processes.
- ❑ **preallocation of file space:** The fast file system architecture (F5) discussed in the previous section allows you to preallocate file space for critical files used by realtime processes. This can be done from within the program code or with administrative commands.

2. System Operations

Chapter 2

System Operations

This chapter discusses a number of general administrative tasks, including:

- ❑ maintaining a system log
- ❑ system startup, shutdown, and transitions between run states
- ❑ system security concerns

Maintaining a System Log

To maintain a sane computing environment, you must keep a system log in the computer room for tracking system performance. The log can be as simple or as complicated as you deem necessary. For example, some sites keep a single log by the console in which they record all system problems, maintenance, and backup activities; other sites keep separate books or a single book divided into sections for each of these topics.



The /usr/adm/putbuf file is an on-line system log that can be viewed any time you need to check error messages that have been output to the console. This file also contains all the configuration information and error messages that were printed to the console during the software installation and setup procedures.

A well-designed, well-maintained log book should include:

- ❑ All system availability data
- ❑ A history of system problems and hardware maintenance
- ❑ A history of application-specific system problems
- ❑ A history of backups so you will know what backups are available when you need to restore files or file systems
- ❑ A history of equipment and configuration changes
- ❑ The storage location of all dump files

The following information should be recorded for each problem:

- ☐ The exact date and time the problem occurred
- ☐ A description of the system configuration at the time, especially if it is different from the regular configuration
- ☐ A complete description of the problem and any corrective action taken
- ☐ The name of the person making the entry and the names of other staff members involved

In addition to the system log, you should have a ring binder available in which to save relevant printouts and service receipts.

Startup and Shutdown

This section discusses how you power the computer up and down and move between single-user and multi-user modes. The REAL/IX Operating System allows you to control much of what happens as you transition between the various machine run levels by modifying the files and directories discussed in Appendix C.

init and shutdown Commands

In the following sections, you will find references to both the **init**(1M) and **shutdown**(1M) commands. You should understand what both commands do and decide which is most appropriate for your use.

- **init** is the fastest way to change from one system state to another. It executes any processes associated with that state in the */etc/inittab* file, which may call files in an associated directory (*/etc/rc0.d* for **init 0**, for instance). When using **init**, you must be sure to broadcast a warning to users with the **wall**(1) command, see that processes have been terminated appropriately, and flush the buffer cache to disk with the **sync**(1M) command.
- **shutdown** is a safer way to change from multi-user state to single-user state. **shutdown** runs the */etc/shutdown* script, which provides some nice safety features and can be customized for your environment:
 - A message is broadcast to all current users warning them that the system is about to be shutdown.
 - 60 seconds later (you can override this time either by modifying the *shutdown* script or by using the **-g** option on the command line to specify a different grace period, in seconds) the system asks "Do you want to continue?" If you respond **n**, the system does not shutdown; if you respond **y** (or use the **-y** option on the command line), **shutdown** calls **init** for the state you specified.
 - You must specify an **init** state to **shutdown** when using the **-i** option. When **shutdown** is used with no options, the default is **init S**.

You can edit the *shutdown* script to explicitly terminate application processes before calling **init**. Alternatively, you can add scripts to the *rc0.d* directory to terminate application processes, as discussed in Appendix C.

Transitions between Machine States

Transitions between machine states are controlled by administrative commands. Exactly what happens in response to each command is affected by the contents of the system files discussed in Appendix C. Note the following:

- ❑ Any time you are taking the system out of multi-user mode, notify your users first and give them adequate time to stop processes and close files.
- ❑ Always use either the **init(1M)** or **shutdown(1M)** command to change from a machine state where the operating system is running. Resetting or powering down the computer while the operating system is running will leave the file system in a state such that it should be checked using the file system check/repair utility **fsck(1M)**.
- ❑ When the system detects an error and panics, several subsequent actions take place that are controlled by **sysgen** options. See page 2-7 for more information about system panics.
- ❑ As the system is booted it mounts the file systems, starting with the **root** file system (when entering single-user state) and followed by each of the file systems in **/etc/fstab** (when going to multi-user state). If a file system does not look mountable, it is checked and repaired using **fsck(1M)**. See **mountall(1M)** for details.

Table 2-1 summarizes the transitions between operating system states.

Table 2-1. Transitions between Operating System States

State	Command	Meaning
Single-user	init 1	Only console is active (no user terminals). Only root file system is mounted.
	init s or S	Only console is active. All file systems that were mounted before the init command was issued remain mounted.
Multi-user	exit	When you boot the system and stop in single-user mode, or when you are in single-user state, use the exit command to go to multi-user state.
	init 2	All user terminals turned on in /etc/inittab are active. All file systems are mounted. Processes started by scripts in the /etc/rc2.d directory are started, including cron , job accounting, error logging, and UUCP.

Transitions between machine states where the operating system is running and states where it is not running are affected by system files and commands. Table 2-2 summarizes the transitions between operating system and non-operating system states.

Table 2–2. Operating System Transitions

Transition	OS Stops	Reboot OS	Comments
init 0	yes	no	Operating system is not running (can be powered down)
init 6	yes	yes	Operating system is automatically rebooted
panic	yes	no	Sysgen tunable parameter SYSRESET = 0: system will enter kernel debugger
	yes	no	SYSRESET = 1 and parameter AUTOBOOT = 0
	yes	yes	SYSRESET = 1 and parameter AUTOBOOT = 1

The sysgen tunable parameter AUTOBOOT may be changed at runtime using the **uadmin** system call.

System Startup

When the system is coming up, you will get a prompt asking if you want to enter a run level other than that specified by **initdefault** in */etc/inittab*. The system will wait for your response for 5 seconds¹.



If the **TIMEOUT** value is set to zero, the **init** process will not pause to enable you to specify a run level different from the default provided by **initdefault** in */etc/inittab*.

- If you do nothing, the **init(1M)** process will use the run level specified by **initdefault**. Typically, the system boots to multi-user state and displays a "Console Login:" prompt.
- If you press the RETURN key within 5 seconds, or if there is no **initdefault** specified in */etc/inittab*, the **init** process will prompt you to enter a run level. This option is available for you to decide if you should enter the single-user state or continue (by default) the boot process. If you choose the single-user state, you would receive the following prompt:

```
Spawning shell [ ^D to continue booting ] ... Password:
```

Type the system password to go to single-user state. To go on to multi-user state, type **exit(2)**. Note that the system comes up in single-user state with the following restrictions:

¹The number of seconds that the **init** process pauses is configured as a comment (**# TIMEOUT = n**) through the process field of the **initdefault** entry in */etc/inittab*. If **TIMEOUT** is not specified, the system will default to a pause of 5 seconds before continuing the boot process.

- Only the *root* file system is mounted. Other file systems can be mounted with either the **mount(1M)** or **mountall(1M)** command.
- System daemons, such as **cron** and **errlog**, are not running.



If you have set the **initdefault** line in the */etc/inittab* file to 1, the system will boot to single-user state rather than multi-user state. You can then use **init 2** or type **exit** to go to multi-user state.

*Running the system with **initdefault** set to 1 is not advised except in special circumstances, because a system panic will leave the system unavailable to users until an operator intervenes manually.*

Booting the Debug Kernel

The REAL/IX Operating System is released with an alternate bootable kernel image that was built with options useful for kernel debugging activities. As a space-saving measure, the alternate kernel is not located in the *root* directory on the release media. To boot the debug kernel:

1. Copy the standard */realix* file to another name, such as */realix.std*. The */realix* that you copy in this step should be a known good bootable kernel image.

```
cp /realix /realix.std
```

2. Copy the debug kernel to the *root* directory, retaining the ".db" filename extension that identifies the debug kernel:

```
cp /usr/src/uts/realix/cf/realix.db /realix.db
```

3. Shut the system down:

```
init 0 or shutdown
```

4. Reset the system and go to the operator's interface.
5. Change the appropriate boot parameters to */realix.db* and reboot the system.
6. Type CTRL-d when you get the system debug prompt.

Upon reboot, */realix.db* is automatically linked to */realix*, allowing commands such as **ps(1)** and **sadc(1M)** to work with the debug kernel. (As a rule, */realix* is always linked to the most recently booted copy of the operating system contained in */* or */stand*.) Successive reboots done specifying */realix* as the boot kernel actually use */realix.db* because of this automatic linking. At first, this seems convenient, being able to boot from */realix* where it is really a link to the debug kernel. This is bad habit to get into, however, in a development environment where there may be several different debug kernels with different owners. It's easy to lose track of the real name of both the debug kernel you want to be working with and that of the known-good kernel. For that reason, we

recommend that you always boot manually using the full name of the debug kernel (*/realix.db* in our example here).

The alternate kernel will use the default **sysgen(1M)** parameters. You can run **sysgen** to create a new debug kernel, but as released, this will change the standard **sysgen** database. You can create an alternate **sysgen** configuration by following the instructions in Chapter 9.

To return to the regular, non-debug kernel, boot manually from the */realix.std* file you created in step 1 above by modifying the appropriate boot parameter.

Once you do this, */realix* is linked to */realix.std* and successive reboots done specifying */realix* as the boot kernel actually use */realix.std*.

System Panics

The first action of the **panic** routine is to ensure that any queued console messages are printed. Next it checks to see at what interrupt level the error was detected and, if possible, attempts to **sync(1M)** the disks in order to prevent data loss.

If the system was built with the debug option, the debugger is entered automatically at this point. If you exit from the debugger back to the **panic** routine, it will continue processing as described below.

Optionally, an automatic dump of a copy of the kernel memory to the system dump device is performed. The *swap* device is used as the *dump* device in order to economize on disk usage. Usually the system is set up so that when it is successfully rebooted, the dump is saved from the dump device to a file within a *dumps* directory. See **sysdump(1M)** and **savedump(1M)**, and the discussion about the **autodump** tunable parameter in Chapter 6.

The system supports an automatic reboot feature. The system only reboots after the first system panic that occurs after a successful reboot. This function is controlled by the settings of the **AUTODUMP** and **SYSRESET sysgen(1M)** parameters.

Autodump and Autoboot

If your system is being used as a development machine, it is generally best not to automatically dump and reboot in the event of a system panic for the following reasons:

- ❑ The problem may be due to some fault in an attached hardware device. The system dump will not preserve the state of hardware devices, and the system reset prior to reboot will reset the hardware state, making diagnosis more difficult.
- ❑ The failing system may be under test, and any reboot should use a different, trusted, version of the kernel.

In contrast, when a system is used in a production environment, automatic dump and reboot are generally desirable because:

- ☐ Often it is necessary to restore the service being provided by the machine as soon as possible.
- ☐ The system may be running unattended.
- ☐ The system should be stable, meaning that the same kernel should be used for the reboot.

The exception to this is when a recurrent fault develops in such a way that the system cannot successfully reboot. If the system is stuck in a fail-reboot-fail cycle, then it can be more difficult to break into the cycle to investigate the underlying cause of failure. To allow for this, the autodump and autoreboot features can be enabled only after the system has successfully reached a certain stage in the boot process. By default, the system autodumps and autoreloads only if it has successfully reached multi-user state.

This is implemented by a call in **sysrealix** that sets the "good boot" flag and is normally called from the **noteOKboot** script in *rc2.d*.

The **SYSRESET** and **AUTODUMP** **sysgen(1M)** parameters take three possible values:

- 0 No action
- 1 No action if before "good boot" flag set, otherwise do reset/dump
- 2 Unconditionally do reset/dump

Scheduling Downtime

Some administrative tasks require that the system be made unavailable to users for short periods. This involves two system states:

- ❑ **system powerdown:** necessary for some preventive and corrective maintenance and hardware upgrades. In addition, some administrators choose to powerdown the system during storms if the AC power source is unstable to avoid system problems that could be caused by a power failure.
- ❑ **single-user state:** necessary or desirable when performing tasks such as running backups, restoring files, checking and reorganizing file systems, modifying the */etc/passwd* file. Single-user state may also be advisable when testing and installing drivers and some application programs.

When it is necessary to shutdown the system, the administrator must minimize the frustrations of the user community. The following suggestions can make shutdowns more bearable for your users:

- ❑ Try to schedule downtime for off-hours such as evenings and weekends. Try to schedule backups at a regular time so your users can plan for them.
- ❑ Schedule downtime as far ahead as possible and use the message-of-the-day or *news(5)* facilities to notify users of the times the system will be unavailable. Refer to page 3-19 for more information about user communication services.
- ❑ Before taking the system to single-user state, dismounting a file system, or doing anything else that could affect logged-in users, check who is logged in with the *who(1)* command. Use the */etc/wall* command to inform any logged-in users that the system needs to be shut down. Give them a chance to close any files and to log off before you take the system down.

Note that the *shutdown(1M)* command automatically sends a message to each logged-in user and allows you to select a time delay before the system shuts down.

System Security

Computer security has become a major concern in recent years. While the only way to guarantee security is to administer a system with one or two trusted users who have sole access to hard-wired terminals capable of communicating with no other systems, there are less drastic steps you can take to enhance the security of the system. The following sections summarize these issues and give hints on some things you can do to watch for unauthorized access.

Important Security Guidelines

No system is totally secure or tamperproof. Take the following steps to help ensure your system's security.

- ❑ Remember that anyone with physical access to a machine, especially to a small computer, can literally walk off with it. Monitor building and computer-room access.
- ❑ Never leave a logged in terminal unattended, especially if you are logged in as *root*. Log off the system when you are away from your terminal.
- ❑ Assign only the necessary *owner*, *group*, and *other* access permissions to directories and files. See *Concepts and Characteristics* for a discussion of how these permissions are assigned.
- ❑ Assign passwords to all logins and change the passwords regularly. Do not pick obvious passwords. Six to eight character nonsense strings using letters and numbers are recommended over standard names. Remove or block unnecessary logins. Chapter 3 discusses how passwords are assigned, as well as explaining how to use the password aging feature to prevent users from keeping one password forever.
- ❑ Systems with dial up ports are never really secure. Do not keep secret information on a system with dial up ports.
- ❑ The more people who know a given login and password, the less secure the system. The **su** command is dangerous because several users may know the *root* password. Check the file */usr/adm/sulog* to monitor use of the **su** command. If several users need to access the same files, it is better to set up group permissions as discussed in Chapter 3.
- ❑ Do not give *others* write permissions for login directories, *.profile* files, and files in */bin*, */usr/bin*, */sbin*, and */etc*.
- ❑ Avoid using *."* (current directory) in the *PATH* of *.profile* files.¹ If you do use *."*, it should be the last element in the *PATH*. This is especially true for the *.profile* file in the *root* directory.

setuid and setgid

Use the set-user identification (**setuid**) and set-group identification (**setgid**) bits very carefully to maintain security. These bits are set through the **chmod** command and can be specified for any executable file. When a user runs an executable file with either of these bits set, the system gives the user the same permissions as the owner of that file.

¹Having *."* in *PATH*, especially before the standard directories for executable files, makes you more susceptible to "Trojan horses," where a program that breaks security is planted in your directory. An example of this is a shell program called *ls* that ships a copy of all files to another user (on another system, if UUCP is running), or deletes all files. The script may delete itself after running, making it very hard to catch.

System security can be compromised if a user copies another program onto a file with `-rwxrwxrwx` permissions. For example, if the `su` command gives write access permission for *other*, then anyone can copy the shell onto it and get a password-free version of `su`. The following sections provide a few examples of command lines that identify the files with a `setuid` bit on.

For more information about the `setuid` and `setgid` bits, see `chmod(1)` and `chmod(2)`.

Check setuid Bits Owned by root

The following sample command line lists all set-UID programs owned by *root*. The results are mailed to *root*. All mounted paths are checked by this command starting at */*. Any surprises in *root*'s mail should be investigated.

```
# find / -user root -Perm -4000 -exec ls -l {} \; | mail root
you have mail
# mail
From root Mon Aug 27 07:20 EDT 1984

-r-sr-xr-x 1 root bin 38836 Aug 10 16:16 /usr/bin/at
-r-sr-xr-x 1 root bin 19812 Aug 10 16:16 /usr/bin/crontab
-r-sr-xr-x 1 root bin 27748 Aug 10 16:16 /usr/bin/shl
---s--x--x 1 root sys 46040 Aug 10 15:18 /usr/bin/ct
-r-sr-xr-x 1 root sys 12092 Aug 11 01:29 /usr/lib/mv_dir
-r-sr-sr-x 1 root bin 33208 Aug 10 15:55 /usr/lib/.lpadmin
-r-sr-sr-x 1 root bin 38696 Aug 10 15:55 /usr/lib/lpsched
---s--x--- 1 root rar 45376 Aug 18 15:11 /usr/rar/bin/sh
-r-sr-xr-x 1 root sys 11416 Aug 11 01:26 /bin/mkdir
-r-sr-xr-x 1 root sys 11804 Aug 11 01:26 /bin/rmdir
-r-sr-xr-x 1 root bin 12524 Aug 11 01:27 /bin/df
-rwsr-xr-x 1 root sys 21780 Aug 11 01:27 /bin/newgrp
-r-sr-sr-x 1 root sys 23000 Aug 11 01:27 /bin/passwd
-r-sr-xr-x 1 root sys 23824 Aug 11 01:27 /bin/su
? d
#
```

In this example, an unauthorized user (***rar***) made a personal copy of `/bin/sh`, and did a `setuid` to *root*. This means that ***rar*** can execute `/usr/rar/bin/sh` and become the superuser.

Check setuid Bits in the root File System

The following sample command line reports all files with a set-UID for the *root* file system. You can use the `ncheck` command, by itself, on a mounted or unmounted file system. The normal output of the `ncheck`'s command includes special files. Here, the `grep` command is used to remove device files from the output. In the example below, the filtering to remove the device files applies only to the root file system (`/dev/root`). The output of the modified `ncheck` is used as an argument to the `ls` command. The use of the `ls` command is possible only with the file system mounted.

```
# ls -l `ncheck -s /dev/root` | cut -f2 | grep -v dev`
-r-sr-xr-x 1 root bin 12524 Aug 11 01:27 /bin/df
-rwxr-sr-x 1 root sys 32272 Aug 10 15:53 /bin/ipcs
:
-r-sr-xr-x 1 root sys 11804 Aug 11 01:26 /bin/rmdir
-r-sr-xr-x 1 root sys 23824 Aug 11 01:27 /bin/su
-r-xr-sr-x 1 bin sys 21212 Aug 10 16:08 /etc/whodo
#
```

This example shows a normal output.

Check setuid Bits in Other File Systems

The **ncheck(1M)** command can also be used to check other file systems. For example, the following is run against the */usr* file system looking for files with a **setuid** bit. Note that, while the full path names for the files listed begin with */dev/usr*, the **ncheck** output does not include that string (since it was specified on the command line).

```
# ncheck -s /dev/usr | cut -f2 | grep -v dev
/bin/at
/bin/crontab
/bin/shl
/bin/sadp
:
/lib/uucp/uuxqt
/rar/bin/sh
#
```

In this example, the */usr/rar/bin/sh* should be investigated for reasons discussed above.

The login Program

In the */etc/default/login* file, parameters can be set up for the **login** program. Different security levels for **login** can be chosen with two of these parameters (**PASSREQ** and **CONSOLE**).

- PASSREQ** YES means that a password is required to login. NO means that a password is not required to login.
- CONSOLE** If the *devicename* of the console is given (for example, */dev/console*), the superuser can login only on the console. If **CONSOLE** is set to **NULL**, the superuser can login on any terminal.

Shadow Password File

To protect encrypted user passwords, an optional security feature allows a system administrator to move all password and password aging information from the publicly readable password file (*/etc/passwd*) to an access-restricted file called the shadow password file (*/etc/shadow*). This file contains one entry per login; each entry consists of a single line with five colon-separated fields:

1. **username**: The login name (ID) by which the user is known to the system.
2. **password**: A 13-character encrypted password for the user in the first field and a *lock* string to indicate that the login is not accessible (or no string to show that there is no password for the login).
3. **lastchanged**: The number of days between January 1, 1970, and the date that the password was last modified.
4. **min**: The minimum number of days required between password changes.
5. **max**: The maximum number of days the password is valid.

Installing */etc/shadow*

Initial conversion of a system's single password file (*/etc/passwd*) to the new scheme using two files (*/etc/passwd* and */etc/shadow*) is done by running the **pwconv(1M)** command, which creates */etc/shadow* with information from */etc/passwd*. The command populates */etc/shadow* with the user's login name, password, and password aging information.

Further updates of these password files should be done by using the **passmgmt(1M)** and **passwd(1)** commands. **passwd** updates the password and password aging information in the appropriate password file. **passmgmt** is used to add or change all other information in the password files(s). **sysadm chgpaswd** can also be used to update the password file(s).

pwconv may be run more than once. If for example, the two files ever become inconsistent (i.e., one of the files was manually changed), they can be made consistent by running **pwconv**.



*If password aging information does not exist in */etc/passwd* for a given user, none is added to */etc/shadow*, but the "last changed" information is updated. This occurs only when **pwconv** creates */etc/shadow* or adds an entry not previously in the shadow password file.*

Backing out */etc/shadow*

Certain applications may not work with the new security password file changes. A possible indication of this problem is that you may not be able to log in. If you are running such an application, you may have to "back out" the shadow password changes so that the application will

run. The **pwunconv**(1M) command (*/usr/bin/pwunconv*) accomplishes the reverse of the **pwconv** command. It converts a system from the two-password file scheme back to the one-password file scheme. **pwunconv** can be run to solve compatibility problems caused by the introduction of the shadow password file.

To "unconvert" your two-password scheme, type:

```
pwunconv
```

This will perform the following tasks:

- ❑ If a login ID has entries in both */etc/passwd* and */etc/shadow*, **pwunconv** will:
 - Transfer the password portion in */etc/shadow* to the corresponding entry in */etc/passwd*.
 - If the password aging information is present (*max* is greater than or equal to 0), aging information in */etc/shadow* will be translated and transferred into the corresponding entry in */etc/passwd*. Note that because there are different formats for storing aging information (days in */etc/shadow* versus weeks in */etc/passwd*), **pwunconv** will convert days to weeks.
 - If the password aging information is not present (*max* is less than 0), but the *lastchanged* field in */etc/shadow* contains a zero (**login** will force the password to expire), the aging field in the */etc/passwd* entry will contain a "." which forces the user to change the password at the next login.
- ❑ If a login ID has an entry in */etc/passwd*, but not in */etc/shadow*, the entry will not be affected.
- ❑ If a login ID has an entry in */etc/shadow*, but not in */etc/passwd*, the entry will not be added to */etc/passwd*.

When converting from days (in */etc/shadow*) to weeks (in */etc/passwd*), integer addition and division is used to round up the aging fields, thus reducing the impact of division truncation. For example, one day is converted to one week; eight days becomes two weeks.

3. User Management

Chapter 3

User Management

The operating system uses the User ID (UID) and Group ID (GID) to control access to the system, files, and executable programs. The administrator is responsible for maintaining the two files that control users and groups:

- */etc/passwd* has a one-line record for every user on the system, as well as lines for special system logins (such as **root**, **sys**, and **bin**).
- */etc/group* has a one-line record for every group defined on the system.

These are ASCII files, owned by root, with read-only permissions for all user categories (owner, group, and other). The superuser can edit either of these files using an editor, or they can be updated using **sysadm**. The menus under **sysadm(1M)** are useful for new administrators, but some features (such as password aging) can only be implemented by editing the file directly.

Proper maintenance of these files is a major factor in maintaining system security. Chapters 1 and 2 include a discussion on various security issues and decisions; you should consider those issues when creating and modifying UID and GID entries.

This chapter discusses the format of the *passwd* and *group* files and gives instructions for creating, modifying, and deleting entries by editing the files directly or using **sysadm**.

In addition to maintaining the *passwd* and *group* files, user management involves helping users establish their individual environments with their *.profile* files, and maintaining the */etc/profile* file (which executes before individuals' *.profile* files at login) to meet the needs of the installation. The chapter ends with a brief discussion of the system facilities provided to communicate with system users.

Creating and Modifying User IDs

Each user on the system must have a unique login name, user ID (UID), and associated password. The login name could be a person's name, and is used by other users (and sometimes by the system) to identify the user; the UID is a unique number that is used by the system to identify the user.

The following sections discuss the format of records in the */etc/passwd* file that controls user IDs and gives instructions for adding and modifying entries in that file.

/etc/passwd

/etc/passwd is an ASCII file owned by *root* that has one record for every user on your system. This file also has entries for certain administrative logins and commands. An entry in the *passwd* file consists of a single line with seven colon-separated fields.

```
abc:g0Q3xsv05bWuM,9/TA:103:123:Allen B. Cipher:/usr/abc:/bin/ksh
1 :      2      ,      : 3 : 4 :      5      :      6      :      7
```

The fields are:

1. **login name:** The login name by which the user is known to the system. It should be no more than 8 characters, with no uppercase characters (since */etc/getty* may think the user is logging in on an uppercase-only terminal).
2. **password and password aging:** The encrypted form of the password, if any, associated with the login name in the first field. All encrypted passwords occupy 13 bytes. If you are using password aging (described in the next section), a comma and four characters follow the encrypted password. The first character indicates the maximum number of weeks a password can be kept; the second character indicates the minimum number of weeks. The last two characters are base-64 representations of the week the current password was assigned. This figure is assigned by the operating system, based on a calendar beginning January 1, 1970. Refer to the "Shadow Password File" section in Chapter 2.
3. **user id:** The user ID number (UID) is used by the system to identify the user. Its value must be between 0 and 59999 and must not include a comma. Numbers 0 through 99 are reserved for system logins; UID 0 is reserved for the superuser (**root** login). Each user should have a unique UID.
4. **group id:** Indicates the first group to which a user belongs and determines the default group for files created by the user. The GID matches the group ID established in the */etc/group* file. Numbers 0 through 99 are reserved for system groups. Each group should have a unique GID, but many users may share the same GID. If you are not using groups, you can assign GID 1 ("other") to user IDs.
5. **account:** Has the user's actual name and optional additional information about that user, such as address and phone number. There is no required format for this field.

6. **home directory:** The home directory designates where the user is placed upon logging in. The name is usually the same as the login name, preceded by a parent directory such as */usr/abc*. The home directory is the origination point of the user's directory tree.
7. **program:** This field defines the program to invoke when the user logs in. Usually this is the default shell interpreter for the user, such as */bin/sh* (Bourne shell), */bin/ksh* (Korn shell), or */bin/csh* (C shell). Restricted shells are also supported (*/bin/rsh* and */bin/rksh*). If the field is empty, the default program is */bin/sh*. Any executable program can be used in this field; if you use a program other than a shell, logging in will begin execution of the program; terminating the program logs the user off.

Password Aging

Password aging is an optional security feature that forces a user to change his or her password periodically, then prevents that user from changing it again for a specified period. You determine the times to be used. Refer to the "Shadow Password File" section in Chapter 2.

Password aging is implemented after you have created the user's */etc/passwd* entry and, if desired, put a password in. You then edit the *passwd* entry, adding a comma and two characters to the end of the encrypted password in the second field:

1. first character: the maximum number of weeks the user can keep the password.
2. second character: the minimum number of weeks the user can keep the password.

These times are specified in weeks by a 64-character alphabet. Table 3-1 shows the relationship between the numerical values and character codes. Any of the character codes may be used in the four fields of the password aging information.

Table 3-1. Code for Password Aging

char	# weeks	char	# weeks	char	# weeks	char	# weeks
.	0	E	16	U	32	k	48
/	1	F	17	V	33	l	49
0	2	G	18	W	34	m	50
1	3	H	19	X	35	n	51
2	4	I	20	Y	36	o	52
3	5	J	21	Z	37	p	53
4	6	K	22	a	38	q	54
5	7	L	23	b	39	r	55
6	8	M	24	c	40	s	56
7	9	N	25	d	41	t	57
8	10	O	26	e	42	u	58
9	11	P	27	f	43	v	59
A	12	Q	28	g	44	w	60
B	13	R	29	h	45	x	61
C	14	S	30	i	46	y	62
D	15	T	31	j	47	z	63

Password aging can be used in a number of ways, as illustrated by the following list. The samples show only the second field of the */etc/passwd* line.

1. **User to supply password on first login:** `:,: :`

Create the */etc/passwd* line with only a comma and two dots in the password field. The two dots represent zeroes (0). The user needs no password to log in the first time, but will be thrown into the **passwd** program to create a password. After the user creates a password, no further password aging is implemented. Note that there is an inherent security risk to having logins that require no password; see #2 for a more secure alternative.

2. **User to supply old password and change to new password:** `:RTKESmMOE2m.E, . . :`

After creating the encrypted password for the user with the **passwd** command, edit the *passwd* file to add the comma-dot-dot after the encrypted password. You tell the user what the password is, which reduces the risk that someone else will use this login to access the system.

3. **User to change password and effect password aging:** `:RTKESmMOE2m.E,M/ :`

After creating the encrypted password, edit the *passwd* file to add the comma and the characters (as shown in Table 3-1) that correspond to the maximum number of weeks a password can be kept, and the minimum number of weeks a password can be kept. The example above forces the user to change the password at least every 24 weeks (represented by the "M") but keep the new password for at least a week (represented by the "/").

When the user first logs in, the system looks for the third and fourth digits that tell when the password was last changed. Because it cannot find them, it assumes the password has expired and forces the user into the **passwd** command, which then calculates and supplies the third and fourth characters to the field.

4. **Only root can change the password:** `:RTKESmMOE2m.E, ./ :`

If the value of the second character is greater than the value of the first character, the user is not allowed to change the password. In the example, . is zero, and / is one.

Adding Users

User login IDs can be added, changed, and deleted either by editing the `/etc/passwd` file or through the **sysadm**(1M) menus. If you choose to edit the `/etc/passwd` file, you will have to use `w!` to write the file, since the editors do not “understand” superuser privileges. Often the most convenient way to do this is to copy an existing line and modify it for the new user. If you do this, be sure to change all fields in the line.



Before editing the `/etc/passwd` file, we suggest you make a copy of it to protect yourself against file damage from editing errors.

The second field in the `/etc/passwd` file holds an encrypted password, and so cannot be changed in the editor. When creating a new record, fill in all other fields, then run the **passwd** command to create a password for the user. To implement password aging, use the editor to add a comma and the two characters that correspond to the password aging policy you wish to effect.

If you are a novice administrator and need to provide some access to the system rapidly, you may prefer to create user IDs using **sysadm**.

- ❑ We suggest that you put the system in single-user state (use **init s** to go from multi-user to single-user state). This is not absolutely essential, but it does prevent contention problems that can arise if users try to modify their own passwords at the time you are creating new user IDs.
- ❑ The `/usr` file system must be mounted (use the **mount** command with no arguments to check what is mounted); if you have not moved the `/usr` file system, it can be mounted with the **mountall** command.
- ❑ The file system on which the home directories for the users IDs being created must be mounted. See Chapter 4 for instructions on mounting file systems.

To invoke **sysadm**, you can either log in as *root* (you will need to know the *root* password) and execute **sysadm** or you can log in as **sysadm** (again you will need to know the password if one was assigned). **sysadm** is a menu-driven interface for system administration; select the User Management menu (**usermgmt**) from the System Administration menu.

The default login (home) directory for user IDs created with **adduser** is in the `/usr` file system. This default enables you to create new user IDs when you first install the system, before you have created additional file systems. It is not ideal for normal use, however, because there are serious ramifications if the `/usr` file system runs out of space. Moreover, future operating system releases will include new `/usr` file systems; having login IDs in `/usr` will complicate the upgrade procedure considerably. Therefore, we suggest that you change the default login (home) directory before creating user IDs.

To do this, select **modadduser** from the User Management menu. **modadduser** will show you the current defaults for **adduser**. A unique ID should be assigned to each person who will be using the system. Shared logins have implicit security risks. If several users need to access the same files, you can set up group privileges for those files or link files together for users located on the same file system. User IDs can range from 100 to 59999, and group IDs can be 1 or 100 to 59999.

For an example, let's change the default group ID to 400 and the default home (parent) directory to */people* (this file system has to exist and be mounted). Under **modadduser**, you will be asked to respond to the following:

```
Do you want to change the default group ID? [y, n, ?, q]  y
Enter group ID number or group name [?, q]  400
Do you want to change the default parent directory? [y, n, ?, q]  y
Enter parent directory name [q]:  /people
```

The new defaults will be displayed on your screen. Verify that they are correct.

```
Do you want to keep these values? [y, n, q]  y

Press the RETURN key to see the usermgmt menu [?, ^, q]:  <CR>
```

Now that you have changed your default home directory, you are ready to add new users. Select **adduser** from the User Management menu.

For an example, we will fill in the information for user John Q. Public.

```
Enter user's full name [?, q]:  John Q. Public
Enter user's login ID [?, q]:  jqp
Enter user ID number (default 324) [?, q]:  <CR>
Enter group ID number or group name
(default 400) [?, q]:  <CR>
Enter user's login (home) directory name.
(default '/people/jqp') [?, q]:  <CR>
```

Verify the new user information and if correct, type **i** for install. You will be asked if you want to give the new user a password. The password should be at least six characters, one of them a numeral.

```
Do you want to install, edit or skip this entry [i, e, s, q]?  i

Do you want to give the user a password? [y, n]  y
New password:

Re-enter new password:

Do you want to add another login? [y, n, q]  n
```

sysadm adduser does not allow you to implement password aging. After creating the user with **adduser**, you can edit the */etc/passwd* file to activate this feature.

sysadm adduser does not allow you to specify a shell or other program (the seventh field of */etc/passwd* will be left blank, so the user will get the default shell, */bin/sh*). After creating the user with **sysadm adduser**, you can either use **sysadm moduser chgshell** or edit the */etc/passwd* file.

sysadm adduser also creates a *.profile* file for the user by copying the */etc/stdprofile* file to the newly created \$HOME directory. Each user should have a *.profile* file before logging in; users will probably change their *.profile* file later to customize their working environment. However, you may want to modify the */etc/stdprofile* file to include standard features used in your environment.

If you want to force a user to change the password during the first login session, you need to edit the */etc/passwd* file directly to implement password aging. This can be implemented whether or not you supplied a password for the user ID in the **sysadm** session. If you supplied a password, the user will have to supply that password to log in; otherwise, no password is required for the first login. Some administrators use the person's employee ID for the initial password, then force the user to change the ID on the first login. This avoids the security risk of having logins assigned with no passwords and yet keeps the transaction simple.

Records in */etc/passwd* have seven fields, separated by colons. The second field is a scrambled version of the password you supplied above. By adding the characters ",.." (comma-dot-dot) to the end of the password you created, you force the user to change the password during the first login. The first line shown is the */etc/passwd* as created by **sysadm**; the second line shows it as edited to force the user to change the password.

```
jqp:yuSF8C7EJU8dQ:46145:1:John Q. Public:/people/jqp:
jqp:yuSF8C7EJU8dQ,..:46145:1:John Q. Public:/people/jqp:
```

If the user subsequently forgets the password, you cannot find out what it is, but as *root* you can change it to some other password, which enables the user to access the account.

Special Administrative and System Logins

On the REAL/IX Operating System, you can establish a special password for any command. This feature, if used wisely, enables you to restrict access to certain commands. During system setup, you are asked to assign passwords to the system logins listed in Table 3-2 (see the *Software Installation Guide*). If you do not do this, these logins have no passwords and pose a serious security risk.

Table 3-2. System Logins Defined During Setup

Login	Program	Use
root	/bin/sh	has no restrictions and overrides all other logins, protections, and permissions. It allows the user access to the entire operating system. The password for the root login should be carefully protected.
daemon	/bin/sh	the system daemon that controls background processing.
bin	/bin/sh	has the power of a normal user login over the files it owns, which are in /bin .
sys	/bin/sh	has the power of a normal user login over the files it owns, which are in /usr/src .
adm	/bin/sh	has the power of a normal user login over the files it owns, which are in /usr/adm .
uucp	/bin/sh	owns the object and spooled data files in /usr/lib/uucp .
nuucp	uucico	used by remote machines to log into the system and initiate file transfers via /usr/lib/uucp/uucico .
lp	/bin/sh	owns the object and spooled data files in usr/spool/lp .
setup	/usr/admin/setup	automatically invokes the setup script.
sysadm	/usr/admin/sysadm	automatically invokes the sysadm package.
listen	/usr/net/nls/listen	monitors the network for UUCP requests.

To change these passwords or to assign passwords to other administrative logins, log in as **root** and execute the **passwd** command for that login (for instance, "**passwd sys**"); you will not have to supply the old password.

You can also prevent a system login from having a password by assigning "NONE" to the password field in that login's record in the **/etc/passwd** file. By using NONE, you do not have to remember extra passwords. If you are logged in as **root**, you can access this login with **su** without being prompted for a password. In addition, no one can access this login directly. An example of the **/etc/passwd** record for **adm** that contains the NONE password is:

```
adm:NONE:4:4:0000-Admin(0000):/usr/adm:
```

Adding Realtime Users

To assign realtime privileges, execute the **setrtusers** command. **setrtusers(1M)** populates the realtime privilege table with the login names or user IDs found in */etc/passwd*. You must have root privileges to execute **setrtusers**. The command format follows:

```
setrtusers (user 1, ...user n)
```

You can use **setrtusers** to add one user ID, or several together, for example:

```
setrtusers john jane bob
```

Each time you execute **setrtusers**, it supersedes the previous list. *S32setrtusers* is a file in the */etc/rc2.d* directory that automatically executes, setting the realtime user privileges, when the system goes to multi-user state. You can preserve your realtime users list by typing the names of the realtime users into the *S32setrtusers* file. Then simply edit the file when you want to add or delete realtime users.

The **rtusers(1)** command lists users with realtime privileges (but not the realtime users that have superuser privileges). You can also use **rtusers** with **setrtusers** (type commands one after the other on the same command line) to add or delete realtime users without retyping every name in your list. When used with **setrtusers**, the **rtusers** command prints out the realtime users names from a previous **setrtusers** command:

```
setrtusers jill  
Makes jill a realtime user.
```

```
setrtusers `rtusers` tom  
Sets tom as a realtime user along with jill, who is already listed under rtusers.
```

```
setrtusers `rtusers` harry sally  
Sets harry and sally as realtime users along with the other realtime users, jill and tom.
```

```
setrtusers `rtusers` | grep -v tom  
Removes tom from the realtime users list but keeps the other realtime users, jill, harry, and sally.
```

```
rtusers  
Prints out the names (one name per line) of the current realtime users, jill, harry, and sally.  
The user ID will be printed out instead of the name if the name cannot be found.
```

Changing User IDs

To change any information (except the password itself) in the user's */etc/passwd* record, you can edit the file. Because the password field of */etc/passwd* is encrypted, you cannot just edit it to change it. Instead, use the **passwd** command to change the password. As the superuser, you can use this command to change any password on the system, and you will not have to supply the old password like a regular user would. For example, to change the password of user "abc", log in as *root* and enter the command:

```
# passwd abc                (Must be logged in as root)
New password: passwd9       (Password entered is not echoed)
Re-enter new password: passwd9
```

The command changes abc's password to **passwd9**. You can force the user to change the password at the next **login** session by editing */etc/passwd* and adding the characters **..** after the encrypted password.

The */etc/passwd* records can also be modified with the screens under the **sysadm moduser** menu:

```
change user's login ID name:    sysadm chgloginid
change user's password:        sysadm chgpaswd
change login program or shell:  sysadm chgshell
```

Deleting User IDs

When deleting a user ID manually, use the following procedure:

- ☐ Delete the appropriate line from the */etc/passwd* file. You may want to lock the ID for a period of time before deleting it; this is discussed in the following section.
- ☐ Remove references to the user from the */etc/group* file.
- ☐ Archive the user's directory tree to a backup media.
- ☐ Delete the user's files, home directory, and */usr/mail* file.

The **sysadm deluser** script deletes the user's */etc/passwd* listing, mail file, and all files under the user's \$HOME directory.

Listing User IDs

The */etc/passwd* file is an ASCII file that can be viewed at any time. Most often, administrators use **grep** to print just the record or records that interest them. The **sysadm lsuser** command lists the login names and information from field 5.

Locking Unused Logins

If a login is not used or needed, you may want to lock the */etc/passwd* entry for a while before actually deleting the record. This enables other users to move files out of the area, while preventing anyone from actually using the login.

A login is locked by editing the */etc/passwd* file and changing the encrypted password field to contain one or more characters not used by the encryption process. One way to do this is to use the expression **Locked;**. In this expression, the semicolon (;) is an unused encryption character. The text, however, only serves to remind you that the login is locked. You can also use another expression, **not valid**, to prevent the use of a login. The space in this expression is another unused encryption character.

The following line entry from the */etc/passwd* file shows the "locked" **bin** login.

```
bin:Locked;:2:2:0000-Admin(0000):/bin:
```

Creating and Modifying Group IDs

Group IDs are a means of establishing another level of ownership of and access to files and directories. Users with some community of interest can be identified as members of the same group. Any file created by a member of the group carries the group identification number as a secondary identification. By manipulating the permissions field of the file, the owner (or someone with the effective user ID of the owner) can grant read, write, or execute privileges to other group members.

/etc/group

The */etc/group* file is an ASCII file with a one-line record for every group defined on the system. Every existing group must have an entry in */etc/group*, but it is not necessary for every user to have an entry in the */etc/group* file.

Like */etc/passwd*, the */etc/group* file has read-only permissions for all categories of users. The superuser can modify */etc/group* by editing it directly or by using **sysadm(1M)**.

Each */etc/group* record has four fields separated by colons:

owner:password:GID:member1, member2, ... , membern

Each entry is one line; each line has the following fields:

1. **Owner:** The 3- to 8-character name of the group ID. It is most frequently used with the **ls -l** command as the group of a file.
2. **Password:** Accommodates a scrambled password similar to that in the */etc/passwd* file, although no algorithm is provided for placing a password in that field. If you want a password for the group, take the following steps:
 - Create a phantom record in the */etc/passwd* file and run **passwd** to create a password.
 - Copy the scrambled password from the */etc/passwd* file to the correct field of the */etc/group* record.
 - Delete the phantom record in the */etc/passwd* file.

If this field is not empty, users attempting to execute **newgrp** for this group will be prompted for a password.

3. **Group ID (GID):** This is the identifying number for this group, used in the */etc/passwd* and other system files, and accessed by certain system calls.
4. **Members:** The user login IDs of users who are members of this group but whose */etc/passwd* file does not use this GID in field 4.

To better understand how group permissions work, consider the following */etc/group* record:

```
soft::403:bill,ann,terry
```

This entry determines that:

- ❑ Files created by a user whose */etc/passwd* record has "403" in field 4 are created with "soft" as the group. After a file is created, the group can be changed with the **chgrp** command.
- ❑ Any user whose */etc/passwd* record has "403" in Field 4 can access a file whose group is "soft," assuming the file's permissions grant access to the group.
- ❑ bill, ann, and terry can execute **newgrp soft** and then access a file whose group is "soft," assuming the file's permissions grant access to the group.

Adding and Changing Groups

You can edit the */etc/group* file directly to add or change groups (remember, you will need to use **w!** to write the changed file). Alternately, you can add groups by executing **sysadm addgroup**; for an example, we will add the new group **seventy7** and use the default for the group ID number:

```
Enter group name [?, q]:  seventy7
Enter group ID number (default 100) [?, q]:  <CR>
```

Verify the information and if correct, type **i** for install.

```
Do you want to install, edit, or skip this entry [i, e, s, q]?  i
```

```
Do you want to add another group? [y, n, q]  n
```

Note that **sysadm addgroup** does not allow you to specify member names.

Deleting Groups

Records can be deleted from the */etc/group* file by editing the file directly or by executing **sysadm delgroup**. If you execute **sysadm delgroup**, you will be asked to respond to the following prompts:

```
Which group name do you wish to delete? [q]  seventy7
Do you want to delete group name 'seventy7', group ID 100? [y, n, ?, q]  y

Do you want to delete any other groups? [y, n, q]  q
```

Take care when deleting groups that no */etc/passwd* records use that GID and that there are no files on the system that use the group. The following command will list any files associated with the specified group name or GID:

```
find / -group group-name|GID -print
```

For example, to list the files associated with the group named **service**:

```
find / -group service -print
```

To list the files associated with the GID **99**:

```
find / -group 99 -print
```

Listing Groups

Most often, you will use **grep**, **cat**, **pg**, **view**, or some similar command to look at the */etc/group* file. The **sysadm lsgroup** command lists the */etc/group* records in a nicer format, as shown in the example below:

Group Name	Group Number	Logins Permitted to Become Members Using newgrp
adm	4	root,adm,daemon
bin	2	root,bin,daemon
daemon	12	root,daemon
mail	6	root
other	1	
rje	8	rje,shqer
root	0	root
sys	3	root,bin,sys,adm

Establishing the User Environment

Each user's working environment is determined by two files:

- */etc/profile* is the system-wide profile.
- *\$HOME/.profile* is the user's individual profile.

These are ASCII text files that can contain commands, shell procedures, and definitions of environmental variables. Whenever a user logs in, the system executes the */etc/profile* file and then the user's own *.profile* file. Definitions in the user's *.profile* file will override any definition made in the */etc/profile* file.

Most administrators leave the specific contents of *.profile* files up to the individual user, but when you create a new user ID, you should provide a basic *.profile* file that can be modified. The */etc/stdprofile* is a skeleton for user *.profile* files. **sysadm adduser** will automatically copy this file to the user's *.profile*, or you can manually copy it after adding the record to the */etc/passwd* file. You may want to modify this file to include definitions that are appropriate for your environment.

You can view the */etc/profile* file released with the operating system on-line; it includes:

- The **trap** statement. This is a critical part of the script, since it causes the shell to ignore breaks and quits.
- A set of environmental variables are defined. Users should not change these definitions.
- **HISTFILE** is the location and name of the history file maintained for each Korn shell user; **HISTSIZE** is the maximum size of this file. See **ksh(1)** for information on other environmental variables used with the Korn shell.
- The **umask** command determines the permissions on newly created files or directories (before the user executes **chmod(1)** to specify permissions). Users can redefine **umask** in their own *.profiles*. In this sample, **022** removes write permissions for the group and other; files normally created with mode 777 become mode 755; files created with mode 666 become mode 644.
- The */etc/TIMEZONE* file defines and exports the correct time zone information. See Appendix B for information on editing this file to reflect the local time zone.
- The **case** statement lets you specify different actions depending on how */etc/profile* is invoked. In the released script, it is specified that the message-of-the-day (from the */etc/motd* file), a "you have mail" message (if appropriate), and news *headline* (unless the user is *root*) will be displayed at login time and any time one issues the **-sh** command.
- Before the message-of-the-day is sent to the user's terminal, traps are reset. This allows the user to break out of the message-of-the-day, which is particularly useful when the message is long and the user is using a slow line (such as is available over phone lines).

- The `/etc/profile` file is executed when you successfully invoke `-su`, but without notification of message-of-the-day, mail or news.
- At the end of the script, traps are reset so the user can break out of processes while using the system.

Other items that are commonly included in the `/etc/profile` file include:

- A `mesg n` line that prevents other users from writing to the terminal directly with such commands as `write` and `wall`. If the superuser issues a `wall` command, it ignores the `mesg n` setting. Users can open up their own terminals by including a `mesg y` command in their `.profile` file or by issuing the command at the shell.
- Any `stty` value that is not terminal-specific can be set in the `/etc/profile` file; `stty` values that apply to all terminals being used can be set in the `/etc/gettydefs` file. Terminal-specific `stty` values should be set in the individual's `.profile`. If you use an illegal `stty` value in `/etc/profile`, all users will receive a "`stty not a typewriter`" message when they attempt to log in and when running some other processes.

Environment Variables

An array of strings called the environment is made available by `exec(2)` when a process begins. Because `login` is a process, the array of environment strings is made available to it. An example of a typical array of strings is shown below:

```
PS1=$
LOGNAME=abc
PWD=/usr/abc
HOME=/usr/abc
PATH=:/bin:/usr/bin:/local/bin:/usr/local/bin
SHELL=/bin/sh
MAIL=/usr/mail/abc
TERM=vt100
PAGER=/usr/bin/pg
TZ=EST5EDT
TERMINFO=/usr/lib/terminfo
EDITOR=/usr/bin/vi
```

These variables can be set in an individual's `.profile` file. For example, in the list above, the user's terminal is defined as a vt100 (`TERM=vt100`). When the user invokes the editor `vi(1)`, `vi` checks the vt100 file in `/usr/lib/terminfo` subdirectories, where the characteristics of a vt100 terminal (such as the 23-line screen) are defined. New strings can be defined at any time. By convention they are defined with the variable in uppercase, followed by an equal sign, followed by the value. Once defined, you can use the `export` statement to make the environment variable global. Users may configure the `.profile` file to suit their individual needs.

Character Display Options (stty)

Chapter 7 discusses how the default character display options are established for the terminal device. Users can customize their working environment by changing these options through the **stty(1)** command. **stty** can be issued to the shell, but is usually set up in the user's *.profile* file so the options are set during every login session. You may want to set some of these options in the */etc/stdprofile* file; individuals can reset these values later, but this ensures that they will have, for example, the functionality they expect from the BACKSPACE key. A sample **stty(1)** line is:

```
stty cr0 nl0 echoe -tabs erase ^H
```

The options are defined as follows:

- cr0 nl0** No delay for carriage return or new line. Delays are not used on a video display terminal, but are necessary on some printing terminals to allow time for the mechanical parts of the equipment to move.
- echoe** Erases characters as you backspace.
- tabs** Expand tabs to spaces when printing.
- erase ^H** Change the character-delete character to a ^H. The default character-delete character is the pound sign (#). Most terminals transmit a ^H when the BACKSPACE key is used. Specifying this option makes the BACKSPACE key useful.

See **stty(1)** for a complete listing of character display options that can be set with **stty**. Any **stty** values that are not terminal-specific can be set in the */etc/profile* file; **stty** values that apply to all terminals being used can be set in the */etc/gettydefs* file discussed in Chapter 7. Terminal-specific **stty** values should be set only in the individual user's *.profile* file. If you use an illegal **stty** value in */etc/profile*, all users' will receive the message "**stty not a typewriter**" when they attempt to log on and when running some other processes.

umask

A system default controls the permissions mode of any files or directories created by a user. The REAL/IX Operating System is shipped with 644 default values for files and 755 for directories. That means that while everyone automatically gets file read permission, only the owner has write permission. For directories, everyone gets read and execute permission, only the owner has write permission. (Execute permission on a directory means the ability to **cd** to the directory and to copy files from it.)

Users frequently set up user mask in their *.profile* by means of the **umask** command. **umask** alters the default permission levels by a specified amount. For example,

```
umask 024
```

leaves the permission level for owner unchanged, lowers the permission level for group by 2, and reduces the permissions for others to zero. The system is shipped with 644 file permissions; this user mask changes it to 620, which translates into read and write permission for the owner, write permission for the group, and no permission for others.

There may be a **umask** command in */etc/profile*. If there is, it does not change a user's ability to put one in *.profile*.

Default Shell and Restricted Shell

Generally, when a user logs in the default program */bin/sh* is initiated. There may be cases, however, where a user needs a restricted shell.

A restricted shell does not allow the user to:

- ☐ change directories
- ☐ change the value of \$PATH
- ☐ specify pathnames or command names containing a slash (/). That is, the user of a restricted shell may not access files or directories other than the present working directory or those included in \$PATH.
- ☐ redirect output

The restrictions are enforced after *.profile* has been executed.

The administrator can use a restricted shell strategy to limit the execution of select commands or programs. By setting up a special directory for executables (*/usr/rbin*, for example), and controlling PATH so it only references that directory, the administrator can restrict the user's activity in whatever way is appropriate.

User Communications Services

A responsible administrator makes every attempt to keep computer users informed. For example, users need to know when downtime is scheduled or when new features have been added to the system.

There are several ways to communicate electronically with users, including:

- ☐ Pre-login Message: a brief message displayed prior to the login prompt.
- ☐ Login Message: a brief message displayed automatically every time a user logs in.
- ☐ Broadcast: a method to communicate instantly with all users who are logged on.
- ☐ **mail** and **mailx**: standard electronic mail commands that can be used by administrators to communicate with individual users and groups of users.
- ☐ News: general system information that a user can access by typing **news**. A "headline" for unread news notifies users of news items when they log in.

These facilities are discussed below.

Pre-Login Message

It is sometimes necessary to communicate electronically with users even before they login to the system. For example, before beginning backups, you should warn users because some may not want to access files while backup operations are being performed.

To activate this feature, create a file called `/etc/issue` that contains a short message. As long as this file exists, users will receive the message before their **login** prompt.

Login Message (Message of the Day)

The login message is printed out after the user logs in but before the first shell prompt appears. It is generally used to inform users of scheduled downtime, the need to clean up files, or security issues.

To create the message, you must have superuser privileges. Edit the `/etc/motd` file so it contains the message exactly as it should appear. Try to keep the message short (under 15 lines, if possible), especially if you have users logging in on 300 or 1200 baud terminals.

Remove the messages when they are no longer needed. Most users become annoyed (and justifiably so) when they get login messages about downtime that happened two days ago. This can cause real confusion if the message reads, for instance, "System coming down at noon today." It is a good idea to include date-specific information for clarity, such as "...today (Wednesday)".

Broadcast to All Users

When you must communicate immediately with all users who are logged in, use the **wall** command. **wall(1)** is similar to **write(1)**, except that it ignores **mesg n** permissions and goes to all active terminals.

To use **wall**, you should be logged in as **root**. Other logins can use **wall**, but the message will reach only those terminals with messaging enabled (**mesg y**).

A typical use of the **wall** command is to warn users that the system will be shut down or that a file system is out of space. You type:

```
wall
/usr1 is out of space.
Please clean up files.
<^d>
```

The command reads whatever you type in after the **wall** command line until it reads an end-of-file (indicated by typing a CTRL-d). The message you type is sent immediately to the terminals of all users logged in. It is preceded by:

Broadcast message from root (console):

mail and mailx

The operating system offers two electronic mail utilities through which users can communicate among themselves. If your system is connected to others by networking facilities, **mail(1)** and **mailx(1)** can be used to communicate with persons on other systems.

mail is the basic utility for sending messages. **mailx** uses **mail** to send and receive messages, but adds to it a multitude of extras that are useful for organizing messages into storage files, adding headers, and many other functions.

One of the features of **mailx** is the ability to create mail groups. As administrator, you might want to set up your **.mailrc** file to include groups of users to whom you may need to send the same mail. For instance, a mail group of users on one file system could be used to notify those users that the file system is being unmounted at some time, or that users need to delete unneeded files to make space.

news

To get information to your users that is important but not critical, use the **news** facility, a sort of electronic bulletin board. To create a **news** message:

1. Log in as **root** and change to the **/usr/news** directory.

2. Create a file with a name that describes the contents. The name of the file is displayed as the "headline" when the user logs in.
3. Fill in the file with information the users need to know.

When a user logs in, */etc/profile* checks if the user has seen the **news** items and gives the headlines of items the user has not seen. Item names are displayed only for current items, that is, items added to the */usr/news* directory since the user last looked at the news. When you read a news item, an empty file named *.news_time* is written in your login directory. As with any other file, *.news_time* carries a time stamp indicating the date and time the file was created. When you log in, a comparison is made between the time stamp of your *.news_time* file and time stamp of items in */usr/news*.

When the user issues the **news** command, the "headlines" of all unread items print on the screen. Users can make copies of the files in */usr/news* if they need this information for reference. The **news -a** command displays all news files in */usr/news*, including those the user has already read. Unlike the Message of the Day where users have no ability to turn the message off, with **news** users have a choice of several possible actions:

- | | |
|-------------------|---|
| read everything | If the news command is entered with no arguments, all news items posted since the last time the user typed in the command are printed on the user's terminal. |
| select some items | If the news command is entered with the names of one or more items as arguments, only those items selected are printed. |
| read and delete | After the news command has been entered, the user can stop any item from printing by pressing the DELETE key. Pressing the DELETE key twice in a row stops the program. |
| ignore everything | If the user is too busy to read announcements at the moment, they can safely be ignored. Items remain in <i>/usr/news</i> until removed. The item names will continue to be displayed each time the user logs in. |
| flush all items | If the user simply wants to eliminate the display of item names without looking at the items, a couple of techniques will work: |

```
$ touch .news_time
```

updates the time-accessed and time-modified fields of the *.news_time* control file.

```
$ news > /dev/null
```

prints the news items on the null device.

4. Creating File Systems

Chapter 4

Creating File Systems

During the software installation procedure, the *root* and */usr* file systems were created. The *root* file system should be on the first slice of the boot device; other than this, you have some flexibility about how many file systems you create and the size of each. The information in this chapter is offered only as suggestions to help novice users create some basic file systems that should provide a satisfactory environment in which to familiarize yourself with the system. Later on, you can change these file systems to suit your needs.

We suggest that you create additional file systems so that there is less activity in the *root* and */usr* file systems. If these file systems fill up (which is more likely if temporary files, spooling files, dumps, user files, and so forth are in them), it seriously degrades performance and can cause some system processes to stop. For example, file systems could be created for:

- ❑ user logins. Although users' home directories can be put in the */usr* file system, we recommend that you create at least one file system for this purpose. This keeps user files from filling up the */usr* file system.
- ❑ storing memory dumps. This is especially important in a development environment (especially if you are doing kernel development, such as writing drivers and system calls). If you mount this file system on something other than */usr/dumps*, you must modify the */etc/savedump* line in */etc/rc2.d/S02PUTBUF* and the */dumpfilesys* line in */etc/rc2.d/S01MOUNTUSR*. The *S02PUTBUF* script copies memory dumps taken when the system panics from the system *swap* device to a file where it can be analyzed.
- ❑ */tmp* and */usr/tmp*, rather than leaving them as part of the *root* and */usr* file systems, respectively. This keeps temporary files from filling the *root* and */usr* file systems.
- ❑ */usr/spool*, used for spooler programs, such as spooling for the **lp** spooler or, if you implement it, **USENET**. This keeps spooler programs from filling the *root* and */usr* file systems.
- ❑ production programs and data bases. This simplifies the task of saving an image of the system just before and after you install new software.
- ❑ source code that can be viewed but should not be modified. This file system can then be mounted as "READ ONLY".
- ❑ add-on packages.
- ❑ backing up production file systems. Disk-to-disk copies are generally faster than disk-to-diskette copies, so many administrators run backups by copying a file system to a scratch file system, then copy the copied file system to diskette. This file system may also be used for reorganizing and resizing file systems.

Before creating file systems, you must create the special device files and perform other tasks. To create file systems, you may wish to put the system in single-user mode. This is not always necessary but does prevent contention problems that may arise while performing some tasks, such as modifying a user password or moving user files. Do this by issuing an **init s** command from the console (with superuser privileges) when the system is in multi-user mode. Creating new file systems then involves the following steps:

1. Use the **mkfs(1M)** command to create the file systems.
2. Use the **labelit(1M)** command to label file systems created with **mkfs**.
3. Use the **mount(1M)** command to mount file systems created.
4. Use the **mklost+found(1M)** command to create a *lost+found* directory in each file system created.
5. If you created a user file system to which you are moving existing login directories, follow the instructions later in this chapter.

These steps are discussed in this chapter, along with information on monitoring file system usage and reorganizing file systems. Appendix A explains special device files for disk; Chapter 5 discusses backing up, checking, and repairing file systems. If you are creating a file system that uses a logical block size larger than the system default logical block size, you will need to reconfigure the buffer cache as discussed in Chapter 6. The buffer cache must contain buffers as large or larger than the file system logical blocks in order to do any I/O operations for the file system. See *Concepts and Characteristics* for a detailed discussion of the different file system architectures supported on the REAL/IX Operating System.

Disk and File System Overview

Before discussing how to create file systems, you should understand the subtle relationship between special device files, disk partitions (or *slices*), and file systems. You should also understand the internal structure of the two file system architectures supported on the REAL/IX Operating System and the relationship between the logical block size of a file system and the system buffer cache. This is summarized here.

File Systems and Special Device Files

Appendix A discusses special device files and the specific meaning of the special device files for disk devices that are associated with file systems. The relationship between disk slices or partitions, special device files, and file systems can be summarized as follows:

- ❑ The installation boot procedure creates eight raw and eight block special device files for the PC's hard disk device. These correspond to seven potential disk partitions plus one special device file that represents the entire disk (in other words, the sum of the other seven partitions). See Appendix A for an explanation of special device files and the naming conventions for disk special device files.
- ❑ Partitions are areas of disk space that are used to hold file systems. Each disk can be formatted to hold seven partitions (1-7); an extra partition (partition 0) represents all usable space on the disk.
- ❑ File systems are structures defined by the operating system to hold data that is accessible by the operating system commands. Each file system is assigned to one disk partition.
- ❑ Disk space for paging is called the *swap area*. Each system must have at least one swap area, but more can be defined. Each swap area occupies a disk partition to which no file system has been assigned.

About the F5 File System Architecture

The F5 file system architecture is an extension of the UNIX System V file system architecture (S5). Both file system architectures are supported on the REAL/IX Operating System. The F5 enhancements were added to provide the I/O throughput required for realtime applications, but provides better performance for virtually any application. Unless you require the S5 file system architecture for compatibility reasons, we recommend that you use the F5 architecture for all file systems.

All system calls that work on S5 file systems provide the same functionality on F5 file systems. Data can be transferred between F5 and S5 file systems using the `dd(1M)` command, as discussed later in this chapter.

See *Concepts and Characteristics* for a more detailed discussion of the F5 file system architecture.

About Logical Blocks and the Buffer Cache

The REAL/IX extensions to UNIX System V allow you to select from a wide range of logical block sizes for the file system, and to configure the system buffer cache appropriately. Before creating file systems, you should understand how logical block sizes relate to the system buffer cache, since you may need to reconfigure the buffer cache to support the file systems you have created.

The data content of files is stored in file system data blocks, each of which is one logical block long. While the actual physical disk blocks used by the file system are a fixed size (512 bytes each), the logical block size for the file system can range from 512 bytes to 128 Kbytes. The logical block size of the file system determines the size of a read or write operation. For instance, consider a file that contains 4000 bytes (characters). If this file is in a file system that uses 1-Kbyte logical blocks (1024 bytes), each read/write request for the entire file takes four separate operations. If the file system uses 2-Kbyte logical blocks, two operations are required, and if the file system uses 4-Kbyte logical blocks, only one operation is required.

Conversely, each file created occupies at least one logical block on the file system. So, a file that contains 10 bytes consumes 1 file system data block regardless of the size of that data block. File systems that use larger logical block sizes can improve the speed of I/O requests for large files, but can also waste disk space for smaller files. Figure 4-1 illustrates the number of logical blocks of different sizes allocated for 32 physical blocks (512 bytes each) of disk space, and the number of logical blocks of various sizes that are created for a file system that occupies 29168 physical disk blocks.

<----- 32 physical blocks (512 bytes each) ----->																	
1-Kbyte logical blocks	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	14584 logical blocks in the file system
2-Kbyte logical blocks	1		2		3		4		5		6		7		8		7292 logical blocks in the file system
4-Kbyte logical blocks	1				2				3				4				3646 logical blocks in the file system
8-Kbyte logical blocks	1								2								1823 logical blocks in the file system
16-Kbyte logical blocks	1																911 logical blocks in the file system

*Depending on the logical block size specified, **mkfs** allocates an appropriate number of logical blocks to fit in the number of physical blocks allocated for the file system. Larger logical blocks give better performance for large files, but can waste disk space if used for smaller files.*

A similar diagram could be made for the buffer cache, where larger buffers give better I/O performance for large files, but can waste memory resources if used for smaller files.

Figure 4-1. Logical Blocks in a 29168 Block File System

Use the `icount(1M)` command for a summary of file I/O operations; this summary shows the number of I/O operations (actually, file close operations) for each mounted file system, sorted by the various logical block ranges. Using this information, you can determine which file systems should appropriately use different logical block sizes. The same information can be obtained by using the `icount -i` command under the `crash` utility. See the "Tunable Parameters" section in Chapter 6 for more information on how to modify the configuration of the buffer cache.

Most file I/O operations use buffers in the system buffer cache for transfers to and from the actual device. The buffer used must be at least as big as the logical block of data that is being transferred. To support the variety of logical block sizes, the system buffer cache can be tuned to contain buffers of different sizes. The default logical block size is 1024 bytes. The operating system automatically reserves 10% of real system memory for the buffer cache and divides this up into buffers, each of which is the default logical block size. Refer to page 6–50 for more information about configuring the buffer cache.

Before you can create file systems that use logical blocks sizes other than the default for your system, you should reconfigure the buffer cache to have the appropriately sized buffers available. Choosing the appropriate logical block size for each file system requires careful monitoring of disk usage and I/O operations over time. File systems that are tied to specific applications may have pre-defined logical block size needs, but other file systems are less easily defined. We recommend that you avoid using the very large logical block sizes that are supported until you see that they are truly needed.

For an example, let's assume you want to create a 2-Kbyte file system and the system default size is 1 Kbyte. Do this by running `sysgen` and changing the value of the tunable parameters associated with the buffer cache:

- ❑ Set "Number of 2-Kbyte buffers" (**NBUF2K**) in the "Configuring the Buffer Cache" section of Chapter 6 to 128. This is a reasonable starting place, although you may choose a different value later on.
- ❑ Set "Number of 1-Kbyte buffers" (**NBUF1K**) to the value that is the number of 1-Kbyte buffers automatically configured minus double the number of 2-Kbyte buffers configured. This keeps the buffer cache at the same size (proportion of memory), which is a good idea at this point, although not necessary. So, for a system configured with 832 1-Kbyte buffers (displayed when system is booted), the value for **NBUF1K** is $[832 - (2 * 128)]$ or 576.

By setting a non-zero value for **NBUF2K** (or any of the **NBUF*K** parameters), you have turned off the automatic buffer sizing. This is why you now have to specify the number of 1-Kbyte buffers to configure. Chapter 6 discusses how these tunable parameters are used to define the size of the buffer cache.

After `sysgening` the system and rebooting, you can create a file system with 2-Kbyte logical blocks. For more information on `sysgen(1M)`, see Chapter 9.

Formatting the Disk

This type of formatting refers to a low-level format of the hard drive. Follow the disk manufacturer's directions for performing such a format. Normally the disk manufacturer does not recommend that you perform this type of format yourself; instead you are usually required to send the disk back to the manufacturer for low-level formatting at the factory.

Using mkfs to Create a File System

Use **mkfs(1M)** to create a file system. **mkfs** uses four pieces of information:

- ❑ An optional parameter, preceded by a hyphen (-), that specifies the file system architecture (-f for F5, -s for S5) and the logical block size. Unless you have specific needs for a file system that uses the 64-byte inodes of the S5 architecture, we suggest you use the F5 architecture for all file systems. The logical block size is expressed in number of Kbytes, with valid values ranging from .5 through 128. So, to specify an S5 file system with 2-Kbyte logical blocks, the argument is **-s2k** and to specify an F5 file system with 8-Kbyte logical blocks, the argument is **-f8k**.
- ❑ The raw special device file that corresponds to the partition where the file system will reside.
- ❑ The number of physical blocks to be allocated for the file system. The operating system automatically determines the appropriate number of logical blocks for the file system. See Figure 4-1 for a graphical representation of the number of logical blocks of different sizes allocated.



When using logical blocks larger than the system default size, you must adjust the size of the buffers in the system buffer cache so that you have some buffers as large or larger than the logical block size of the file system. This is done using a set of tunable parameters. See the discussion at the end of the section "About Logical Blocks and the Buffer Cache" for more information.

- ❑ An optional parameter, preceded by a colon (:), that specifies the number of inodes to allocate for this file system. If not specified, the operating system will allocate 1 inode for every four logical blocks.

Each individual file in the file system has one inode, but may have any number of blocks. If you create a file system that is only going to have one file in it (no matter how large the file), you would theoretically only need two inodes for that file system, one for the root inode of the file system and one for the file. You can see the number of free blocks and free inodes for all mounted file systems with the **df** command; if you find you are running out of inodes faster than you are running out of available blocks, you can recreate the file system with more inodes.

mkfs lists two more optional arguments, *gaps* and *blocks-per-tracks*. These are vestiges from older disk devices and should not be used.

Before creating any file systems that use logical blocks larger than the system default size, you must configure some buffers that are that large. You do this by running **sysgen** and changing the value of the tunable parameters associated with the buffer cache. See the section "About Logical Blocks and the Buffer Cache," earlier in this chapter.

After you modify the appropriate tunable parameters, you can create a file system with 8-Kbyte logical blocks as follows. In this case, we are letting the system default to one inode per four logical blocks.

```
mkfs -f8k /dev/rdisk/a710_00s4 320704
```

Use a similar procedure to create file systems that use 1-Kbyte, 2-Kbyte, 16-Kbyte, 32-Kbyte, 64-Kbyte, or 128-Kbyte logical blocks.

As another example of **mkfs**, the following line defines an F5 file system on partition 1 (which occupies 58336 physical blocks), using 16-Kbyte logical blocks (so 1823 logical blocks will be allocated), and allocating 911 inodes (one inode per every two logical blocks).

```
mkfs -f16k /dev/rdisk/a710_00s1 58336:911
```

Note that **mkfs** has a built-in delay of 10 seconds before it starts to construct the file system. During this delay, the command can be aborted by pushing the DELETE key. The wait period can be eliminated with the **-q** option.

mkfs can also be used with a prototype file that defines directories to be created in the file system as well as the file system specifications. Refer to **mkfs(1M)**.

Labeling the File System

mkfs(1M) creates the file system structure on the disk partition, but it does not assign a name to the file system. All file systems (except *root*, which cannot be labeled) should have a name written into the superblock and a disk label assigned; this is done with the **labelit(1M)** command.

- The file system name can be up to six characters, and should correspond to the name of the mount-point directory. Do not use the full path name of the file system; its relationship to root will be established when it is mounted.
- The volume ID (**vol-id**) is required by such commands as **mount**, **volcopy**, **crash**, **fsck**, and **fsdb**; it can be up to six characters. Other than the length, there are no specifications for disk labels. The operating system release media uses the corresponding software release letter and number as the disk label for the */usr* and *root* file systems. It is a good idea to use some convention that identifies the disk and partition number or some other piece of meaningful information, and to use different disk labels for each file system.

As an example, to label the file system created by the sample **mkfs** command on the previous page, naming the file system "stuff" and assigning the volume ID "d00s1", the command is:

```
labelit /dev/rdisk/a710_00s1 stuff d00s1
```

labelit will then take 10 seconds to execute. You can abort **labelit** by pressing the DELETE key during this time.

Mounting the File System

For a file system to be available to users, the operating system has to be told to "mount" it. This connects the block physical device and the file system name to the operating system. When a file system is mounted, an entry is made in an internal system table called the *mount table*.¹ A version of this table is written to disk in the */etc/mnttab* file.

Before mounting the file system, the mount point directory must exist. It can be created with the **mkdir** command. Note that any files created in this directory before the file system is mounted will be inaccessible after the file system is mounted. To create the */people* directory:

```
cd /
mkdir people
```

File systems are mounted with the **mount** command, which requires two arguments:

- the block special device file on which the file system is located
- the full path name of the mount-point directory for the file system

To mount the */people* file system on disk *a710_00*, partition 2, the command would be:

```
mount /dev/dsk/a710_00s2 /people
```

A file system (other than *root* or */usr*) can be mounted in "read-only" state, meaning that files on that file system can be read but not written, regardless of the access permissions. This is useful for file systems that need to be read or executed but should not be changed, such as a file system that holds the executable modules and source code for your production application programs. Files can be copied out of a read-only file system, but nothing can be written. To mount a file system as read-only, use the **-r** option, as in the following:

```
mount -r /dev/dsk/a710_00s3 /usr/oursrc
```

A file system can also be mounted as a synchronous file system, meaning that the disk will be updated immediately whenever data is written to a file in the file system rather than waiting for the periodic flushing done by the **bdflush** daemon. This option should be used only when necessary; while it speeds the time of individual I/O operations to that file system, it degrades overall file write time on the system. To mount a file system as synchronous, use the **-s** option, as in the following:

```
mount -s /dev/dsk/a710_00s2 /usr/rtdat
```

Note that a synchronous file system should never be mounted read-only.

¹ The tunable parameter **NMOUNT** determines the number of entries in the mount table, which determines how many file systems can be mounted simultaneously on your system. The mount table is searched linearly, so the value should be kept as low as possible to save overhead. See the table on page 6-45.

The **mount** command used without arguments shows all currently mounted file systems, according to what is in the */etc/mnttab* file. The **crash** utility's **mount** command lists the actual system mount table; this is useful after a system panic, when it is possible for */etc/mnttab* to be inaccurate.

The *root* file system is mounted as part of the boot procedure. When the system is being brought up to multi-user mode, the */etc/rc2.d/MOUNTFSYS* file automatically mounts all file systems listed in the */etc/fstab* file. When you create other file systems, modify the */etc/fstab* and */etc/checklist* files to include these file systems and they will automatically be checked and mounted when the system goes to multi-user state. (To ensure that other file systems remain mounted when going to single-user mode, use **init s** rather than **init 1**.)

For more information on the */etc/checklist* file, see page 5-16 and **checklist(4)**.

The */etc/fstab* file contains one line for each file system to be mounted. Each line contains the block special device file for the partition on which the file system is located, followed by the full path name for the mount point directory and any arguments. For example, the *fstab* file that mounts all the file systems used in the previous examples would look like the following:

```
/dev/usr /usr
/dev/dsk/a710_00s2 /usr/rtdat -s
/dev/dsk/a710_00s3 /usr/oursrc -r
```



*The order in which file systems are listed in the *fstab* file is important because the file system that contains the mount-point of another file system must be mounted first. For example, in the listing above, */usr* must be mounted before */usr/rtdat* and */usr/oursrc* because the last two file systems have their mount points in the */usr* file system.*

Unmounting File Systems

File systems are automatically unmounted during the system shutdown process. File systems must also be manually unmounted before running any programs that use the raw special device file of the disk partition, such as the commands for backing up and restoring files and file systems, reorganizing or repairing file systems.

Before unmounting user file systems, use the **wall** command to notify users that the file system will be unavailable and allow them time to close files.

The **umount(1M)** command that unmounts a file system requires one argument, the block special device file of the partition on which the file system is located:

```
umount /dev/dsk/dev_partition
```

Creating a lost+found Directory

Each file system requires a *lost+found* directory in its mount point directory with blocks allocated to it. The file system checking program, **fsck(1M)**, places orphaned files and directories (allocated but unreferenced) in this directory. Because the file system is unmounted when **fsck** is run, space cannot be allocated at that time.

Use **mklost+found(1M)** to create the *lost+found* directory. You must supply the full path name of the file system and the block special device file for the partition where it is located. To create a *lost+found* directory for the */people* file system, the command would be:

```
mklost+found /people /dev/dsk/a710_00s1
```

Moving Users to a File System

Many new file systems you create will house user home directories and files. You must modify the */etc/passwd* listings to reflect this, either by editing the file directly or running the **sysadm moduser** command, as described in Chapter 3. Field 6 of */etc/passwd* gives the location of each user's home directory. Be sure the file system is mounted when you change this information through **sysadm**.

If you are moving existing users to a new file system, you must also move their files. Users should be notified before they are moved. Be sure none of the users being moved are logged on when you move their files; ideally, you should do this when the system is in single-user state.

File links work only when all linked files reside in the same file system. If you move users with linked files, the links will be lost unless all the files are moved to the new file system at the same time and you use the **-l** option to the **cpio** command to preserve the links. To check for linked files, look for numbers other than 1 in the second column of the **ls -l** command output, ignoring directories. You could write a shell script to find all such files in the file system.

Use the **find(1)** and **cpio(1)** commands to move the user files. As an example, the following series of commands move users *userx* and *usery* from *fsys1* to *fsys2*:

```
cd /fsys1
find userx usery -print | cpio -pdlm /fsys2
```

After you are sure the user directories were successfully copied to */fsys2*, you can remove them from *fsys1*:

```
rm -rf /fsys1/userx /fsys1/usery
```

Always make an entry in the System Log when users are moved, including information on when and why. Be sure to notify the users by mail that they have been moved and explain why.

Expanding and Reconfiguring File Systems

If a user file system is consistently running out of space, the best solution is to create another user file system and move some users to it as described in the previous section. If a system file system (such as *root* or */usr*) or a database file system is consistently running out of space, you may need to expand it. If you expand the *root* file system, you should boot off another device or off the release media; otherwise you will destroy your *root* file system and will not be able to reboot. You can also reconfigure a file system to change the file system architecture, logical block size, or number of inodes.

Any time you expand or reconfigure a file system, make a notation in the System Log book. Before expanding a file system, be sure you have an up-to-date backup version of the file system. The system should be in single-user state and the file system to be expanded must be unmounted.

To expand or reconfigure a file system (other than *root*):

1. Backup the file system to diskette or another disk partition using the **cpio(1)** command as described in Chapter 5; you may want to use the **dcopy(1M)** command and reorganize the file system at the same time. Check your backup to make sure it is sound.
2. Unmount the file system.
3. Edit the file */etc/partitions* to reflect the new disk configuration (see **mkpart(1M)** for details).
4. Run **mkpart(1M)** to delete and/or add partitions as needed.
5. Create and label the file system using **mkfs(1M)** and **labelit(1M)** as described earlier in this chapter.
6. Mount the expanded file system.
7. Use **mklost+found(1M)** to create a *lost+found* directory for the file system.
8. Copy the data from the backup media or from the alternate disk partition to the expanded file system with **cpio**.

Note that **dcopy**, **dd(1M)**, and **volcopy(1M)** copy the file system super block as well as the data in the file system; if you use one of these commands to move the file system to the expanded or reconfigured file system, the file system will have all the same attributes (size, logical block size, and so forth) as it did before, because these attributes are all stored in the super block. The **sysadm filemgmt** commands for backup (discussed in Chapter 5) call **cpio** and so can be used to copy the files to the expanded file system.

Monitoring Disk Usage

After you make a file system available, you need to monitor its usage and growth regularly. Otherwise, the percentage of disk space used increases until the allocated space is used up. For example, certain files, such as logs, grow as a result of normal system use and must be purged. Also, users tend to be careless about deleting files that are no longer needed. When the allocated space is used up, it is almost impossible to run anything from that disk space, especially if the *root* or *usr* file systems run out of free space. In addition, no one can write a file, and it takes so long to execute a process (because the system must look around for some free space) that cleaning up files is a laborious task.

Administrators have three major tasks in monitoring disk usage:

1. **Be aware of the free space available before it causes a crisis.** Many administrators create shell scripts run by **cron** that check file system space periodically and send mail to the administrator if space falls below a certain threshold.
2. **Free up space regularly.** System log files and other files that grow should be cleaned up regularly, usually by shell scripts executed by **cron**.
3. **Identify and remove large or inactive files when file space is limited.** Shell scripts executed through **cron** may identify files that are candidates for removal, but most administrators do the actual removal manually.

The following sections discuss the commands used to track disk usage and suggested methods for cleaning up file systems.

Monitoring File System Space

The **df** command issued with no options reports the number of free blocks and free inodes in all mounted file systems. A sample output is:

/	(/dev/dsk/a710_20s1):	34952 blocks	6512 i-nodes
/usr	(/dev/dsk/a710_20s2):	134688 blocks	25280 i-nodes
/people	(/dev/dsk/a710_00s1):	28244 blocks	40484 i-nodes
/rtdat	(/dev/dsk/a710_00s2):	19276 blocks	2514 i-nodes
/oursrc	(/dev/dsk/a710_00s3):	17938 blocks	2531 i-nodes

As a general guideline, a file system should have 10 to 20 percent of its blocks free at the beginning of the day. These numbers vary from installation to installation. When the */* or */usr* file system has fewer than 2500 free blocks, your system performance will be noticeably impaired. If either of these file systems goes below 500 free blocks, you will be unable to run most processes.

Many administrators create a shell script to monitor file system space and send them mail only if the free blocks or free inodes fall below a certain threshold; such a shell script can be run through **cron** at regular intervals.

Cleaning Up Log Files

The system makes entries in certain files as part of its routine activity. This information is useful, but if you do not watch these files carefully, they can grow to unreasonable sizes. Some of the log files to watch are:

<i>/etc/wtmp</i>	daily accounting information
<i>/usr/adm/sulog</i>	log of su commands
<i>/usr/adm/putbuf</i>	log of system error messages written to the console
<i>/usr/lib/cron/log</i>	log of cron jobs
<i>/usr/adm/pacct*</i>	daily accounting information

Other files that can grow rapidly are files in the */usr/mail* directory (that hold user mail that has not been read and deleted or copied to another file) and users' individual files that hold copies of outgoing **mailx** mail. While it is not usually advisable for the administrator to delete other people's mail files, you can ask the user to clean them up. You should also watch the *lost+found* directories in each file system; these contain sections of files that were salvaged by **fsck(1M)**. They should either be recovered by the user or deleted to ensure that there is adequate space in the directory for future **fsck** runs.

Rather than just deleting these files (and recreating them with the **echo > filename** command), you may want to use the **tail(1)** command to save the last several entries in the file. For instance, to save the last 50 lines in */usr/lib/cron/log* and delete the rest, the command sequence is:

```
tail -50 /usr/lib/cron/log > /usr/lib/cron/tmpfile
mv /usr/lib/cron/tmpfile /usr/lib/cron/log
```

This sequence puts the last (most recent) 50 lines of the */usr/lib/cron/log* file into a temporary file, then replaces the original logfile with the contents of the temporary file, effectively reducing the original logfile to the 50 most recent lines. If you create shell scripts that backup your file systems, you can add the lines that clean up the log files to the end of the sequence that backed up the file system. That way, you have a backup copy of the log if you need to reference it.

Identifying Large and Inactive Files

If you find that users are beginning to fill all the available disk space, you need to look for large files that are not currently being used. Some files can be fairly large and of little use to the owner after a few days, so can be deleted. Other large files that have been inactive for a long period of time can be archived and removed from the system.

Use the **find**(1) command to identify files that have not been accessed for a specified number of days (**-mtime** **+number-of-days**) or whose size exceeds a specified number of blocks (**-size** **+number-of-blocks**). **find** looks through all files under the specified directory and prints the full path name of files that meet the specified criteria. Usually, you are only interested in regular files, which is specified with **-type f**.

For example, the following command identifies all regular files in the */usr1* file system that have not been accessed in 90 days:

```
find /usr1 -type f -mtime +90 -print > /tmp/deadfiles &
```

To identify all regular files in the */usr1* directory that are larger than 50 blocks, use the following command:

```
find /usr1 -size +50 -print
```

Note that the **-size** value is specified in terms of 512-byte logical blocks rather than logical file system blocks. So, if you specify **-size +10**, **find** will list all files that are larger than 5120 bytes. On a 1-Kbyte file system, a 5120-byte file consumes 10 logical blocks, but will consume less than one block on file systems that use 8-Kbyte or larger logical blocks.

It is important to be specific in the size and time parameters you specify to **find**. If you select a high-level directory such as */* or */usr* and specify too few blocks or too few days, you may get a much longer listing than you wanted.

To narrow down the search for information, use the **du -s *** command from the root directory of a user file system to see which users are using a great deal of file space, then run **find** commands from their *\$HOME* directories to identify the large files.

Reorganizing the File System

Daily activity on a file system gradually causes gaps in the physical file organization, which can substantially impair access time for that file system. We suggest that you reorganize all file systems periodically and keep a log of this activity.

dcopy(1M) takes about 15 minutes to reorganize a 1-Kbyte file system that consumes 92160 physical blocks; smaller file systems and file systems that use larger logical blocks will take less time, while larger file systems will take longer. Before running **dcopy**, check the file system with **fsck(1M)** and back it up. The system should be in single-user state, with the file system unmounted.

Follow this procedure for reorganizing a file system:

1. Unmount the file system you are reorganizing.
2. Run **fsck** on the file system to make sure it is error free.
3. Backup the file system as a precautionary measure.
4. Verify that the partition to which you are copying has enough free space.
5. Execute **dcopy** as follows:

```
/etc/dcopy [options] /dev/rdisk/partition /dev/dsk/temp_partition
```

dcopy, used without options, removes vacant entries from the file system and spaces consecutive blocks in a file by the optimal rotational gap. All subdirectories are moved to the beginning of the directories; the inode list sizes of the reorganized file systems are the same as those in the old file system. Note that **dcopy** reads from a *raw* device and writes to a *block* device. See **dcopy(1M)** for more information.

6. As **dcopy** runs, it catches interrupts and quits, and reports on its progress. To terminate **dcopy** send a quit signal.
7. If **dcopy** terminates abnormally, you probably have a bad receiving medium. Try using a different mask.
8. Copy the reorganized file to the original file system name with the **dd** command. **dd** writes every block in the file system, eliminating possible marginal blocks.
9. Mount the file system.
10. Use the **mklost+found** command to allocate space in the *lost+found* directory.

Reorganizing the *root* file system is slightly more complicated because you cannot unmount *root* with the operating system running. If you are configured with an alternate disk boot device, you can copy *root* to the first partition of the alternate boot device and boot off the alternate boot device. If you do not have an alternate disk boot device, you can boot off the release media. Once you have booted the system using some other device, you can reorganize *root* as if it were a user file system, then reboot the system using the reorganized *root*.



5. Maintaining File Systems

Chapter 5

Maintaining File Systems

This chapter focuses on what the administrator needs to know about maintaining file systems. The following topics are reviewed:

- ☐ Backup and restoring file systems and files
- ☐ Using **fsck(1M)** to check and repair a file system
- ☐ Using **fsdb(1M)** to repair a damaged file system

Chapter 4 discusses how to create file systems and make them available, how to monitor file system usage, and how to reorganize file systems.

Backup and Restore Procedures

All active file systems should be backed up regularly. These backup copies can be used to restore files inadvertently removed by a user, files lost when the system crashes, or entire file systems damaged or destroyed as a result of hardware or power failures. You will need to determine when to perform full and incremental backups, how long to keep the backup media, and how to store the backup media.

Three general categories of backup operations are discussed:

- ☐ **full backups of a file system** involve making a literal copy of the entire file system.
- ☐ **incremental backups of a file system** involve copying only those files in the file system that have been modified since the last full backup. An incremental file system backup is faster than a complete system backup because only files that have changed since the last backup are copied. If you need to restore the entire file system, you can restore the full backup and then apply the incremental backups over it.
- ☐ **copies of specified files and directories** are used for archiving files before removing them from disk, or for holding a safety copy before installing new application software that modifies the files.

Scheduling Backups

Deciding when to do full and incremental backups is critical in a system backup plan. Such decisions should be made carefully, considering the particular needs of your environment. The following backup schedule is recommended for most sites:

- ☐ Weekly: run a full backup of all file systems once a week. Keep these backup media for eight weeks.
- ☐ Daily: run an incremental backup of all active file systems. Keep the backup copies at least a week before reusing the media; a month is better.
- ☐ Monthly: run a full backup of all file systems. Keep these backups "forever," recopying them once a year.

We strongly recommend that a **complete** backup of the system be performed as soon as you have configured it to your satisfaction. It is wise to store this backup in a safe place so that it, rather than the release media, can be used to restore the base system if necessary.

General Guidelines

Regardless of the method you use for running backups, we suggest you consider the following:

- ☐ The discussions in the following sections imply that you would be typing all these commands at the console, although in reality, most routine backups are run with shell scripts written by the administrator to meet the specific needs of the installation.
- ☐ If you have adequate disk space, you may want to copy file systems to scratch partitions, then back up the scratch partitions to another media (such as floppy diskette). Because disk-to-disk copying is faster than copying from disk to diskette, this reduces the amount of time the file system is unavailable to users.
- ☐ Keep a list in a section of the System Log of all backups made, recording the command used and the date. Label all backup media thoroughly and accurately, remembering that someone else may need to use them to restore files when you are not available. When backing up a full file system with a command that does not save the super block, the label or log should record all the information required to make and label a new file system (F5 or S5 architecture, logical block size, number of blocks, number of inodes, and the volume ID).
- ☐ Make a paper copy listing of the contents of the backup media. This copy can be fastened to the media itself or stored in a special binder. Such logs are extremely useful for finding specific files and verifying restore operations. If you can afford the disk space, it is handy to also keep these logs on-line where you can access them through **grep**.

- ❑ After making a backup, we suggest you verify that the backup media is readable. The most effective method is to run `dd`, using the backup media as the input device and `/dev/null` as the output device. While this takes extra time, it is much better to discover that the backup is unreadable before you need to use it to restore the files.
- ❑ Backup media should be stored in a secure area that is climate-controlled. You may want to consider storing copies of some backups off-site or in a data storage vault that protects their contents against fire and water damage.

Summary of Backup and Restore Commands

The REAL/IX Operating System provides a number of ways to backup file systems, directory trees, and files. Syntax and other specific information on each command is discussed in the manual pages.

Each of the following commands has advantages and disadvantages; most administrators use a combination of these commands, selecting the one that is most appropriate for each situation. The following sections discuss how to use these backup and restore commands to do specific tasks.

Command	Backup Type	Comments
cpio (1) with find (1)	full, incremental, or partial	Very flexible; does not copy super block of a file system, so a new file system must be created, labeled, and mounted before restoring from a cpio media; preserves contiguous extents; not as fast as volcopy when restoring an entire file system.
dcopy (1M)	copies and reorganizes file system	Super block is copied; preserves contiguous extents; cannot be used for partial and incremental backups; individual files and directories can be recovered by restoring the entire file system to a scratch partition and using cp or cpio to move specific files.
dd (1M)	exact (byte-for-byte) copy of tape or disk partition	Can read or write media from "foreign" operating system; can convert between ASCII and EBCDIC format; used if you need a duplicate copy of a backup media; if volcopy is used to restore a file system to a scratch partition, use dd to copy the file system to the partition where it belongs; writes every block in the partition (whether it has data or not) and will detect marginal blocks on the disk; preserves contiguous extents when used to move a whole partition.
finc (1M)	incremental	Fast backups of files accessed, files changed, or files whose inode has been changed within a specified period; preserves contiguous extents.
frec (1M)	restores files	Backup media must have been run with either finc or volcopy .
sysadm backup sysadm restore	full or incremental	Menu-driven backup and restore utilities that call cpio with appropriate options; easy and versatile for the novice; more experienced administrators may prefer to write their own backup and restore scripts using other commands.
sysadm store	backup and restore utility	Menu-driven; calls cpio with appropriate options to copy specified files and directories (rather than entire file system).
tar (1M)	copies all data in directory tree or file system	Old command supported by virtually all UNIX operating systems (software distribution media from other companies may be in tar format); does not copy super block; easy to use; less flexibility and slower than cpio .
volcopy (1M)	full only	Entire file system and super block are copied; preserves contiguous extents; very fast; uses volume ID (created through labelit) to check that you are not writing in the wrong place; frec can be used to restore individual files from a volcopy media.

Backing Up an Entire File System

A full backup of a file system can be run using **volcopy**, **cpio**, **sysadm backup** (which calls **cpio**), **tar**, **dcopy**, or **dd**. We recommend that you use **cpio** or **dd** to back up *root* because these are the only backup and restore commands available if you have to boot off the release media.

volcopy

The fastest method is **volcopy**. To backup a file system using **volcopy**, the file system must be unmounted. **volcopy** requires the following information:

1. Name of the file system to be backed up
2. Character special device file for the disk partition on which the file system is located (source)
3. Volume ID (assigned through **labelit**) of the file system being backed up
4. Character special device file for the media to which the file system is being copied (destination); this is normally a scratch disk partition
5. Volume ID of the backup media. This is the equivalent of the volume ID supplied with **labelit(1M)**; you supplied this on the **volcopy** command line to label the backup media. This label can be up to six characters; use whatever conventions are used for your archive library. Here we have used **d** for disk, a disk and partition number, and a sequential number.

For example:

```
volcopy -a /usr /dev/rdisk/0s3 USR /dev/rdisk/1s3 d4s011
```

The file system being backed up is */usr*, which resides on partition *0s3* and has the volume ID "USR." It is being copied to the disk scratch partition *1s3*, whose volume ID is "4s011".

To make a listing of the contents of the backup, use the **ff** command on the mounted disk version of the file system (do this either just before or just after running **volcopy**). **ff** makes a listing that includes inumbers for all files, so you can recover individual files from the backup media with **freec(1M)**. For example:

```
ff -lsu /dev/rdisk/0s3 | lp
```

With these options, **ff** generates a supplementary list of all path names for multiply linked files and prints the file size and owner for each file.

To restore a file system from a backup made with **volcopy**, the source file is now the disk scratch; partition the file system was backed up to; the destination file is the disk partition to

which the file system is being restored. For example, to restore the file system backed up in the previous example, unmount the */usr* file system and execute the following command sequence:

```
volcopy usr /dev/rdisk/1s3 d4s011 /dev/rdisk/0s3 USR
```

Use **freec(1M)** to restore individual files from a **volcopy** backup.

cpio and sysadm backup

cpio(1) can also be used to backup an entire file system. It is not as fast as **volcopy** and does not preserve the super block (which specifies the configuration of the file system), but has the advantage of allowing you to restore individual files and directories from the backup media at a later date. Moreover, because **cpio** is such a versatile command, many administrators like to use it for all backup/restore operations rather than remembering the syntax and peculiarities of multiple commands.

cpio is run against a mounted file system. To ensure that the backup is completely sound, make sure that no users are writing to the file system during backups. This can be done by putting the system in single-user state (**init s** puts the system in single-user state while leaving all file systems mounted) or by mounting the file system read-only.

For example, the following command sequence makes a backup copy of all files in the */people* file system:

```
cd /
find people -print | cpio -ovBc > /dev/rmt/a710_40t
```

To restore these files, be sure the */people* file system is mounted, and execute the following command sequence:

```
cd /
cpio -ivBcd < /dev/rmt/a710_40t
```

sysadm backup and **sysadm restore** call **cpio** and have the same advantages and drawbacks. You will have to supply the **sysadm** password to execute them if you are not logged in as the superuser, and otherwise the scripts are self-explanatory. For novice administrators, these menu-driven programs provide a simple way of backing up the system: they supply the appropriate options to **cpio** for the type of backup you request as well as ensuring that the file systems are mounted.

cpio can also be used to write multiple **cpio** archive files to one backup media. Refer to **cpio(1)** for examples of that usage.

tar

tar(1) can be used to backup an entire file system. It is slow and somewhat inflexible, and does not preserve the contiguous extents allocated to files, but is one of the easiest commands to use. **tar** is run against a mounted file system. It copies all files descending from the specified directory; to backup an entire file system, you should **cd** to the mount point directory of the file system. So, to backup the */people* file system, the command sequence is:

```
cd /people
tar -cvf /dev/rmt/a710_40t *
```

tar requires a specification of the floppy drive because it defaults to */dev/mt/0m*. Always use the **-f** option to specify the correct device, or link the unknown device of your choice as default.

After **tar** completes successfully, you can check the backup media by running the following command:

```
dd if=/dev/rmt/a710_40t of=/dev/null
```

The **dd** command simply lists the number of records in and records out. The number of records in and out should match. A more extensive test for checking your backup is to make a listing of the contents of the backup media (and check it against the actual files backed up) as follows:

```
tar -tvf /dev/rmt/a710_40t
```

To restore the */people* file system from a **tar** backup, you must first make, label, and mount the file system. Then restore the data files with the following command:

```
cd /people
tar -xvf /dev/rmt/a710_40t
```

To restore individual files or directories from a **tar** backup, run the following command (where *filename* is the specific file or directory to be restored):

```
tar -xvf /dev/rmt/a710_40t filename
```

dcopy

dcopy(1M) is primarily used to reorganize file systems and is discussed in Chapter 4. It is seldom used strictly for backing up file systems, but may be used to reorganize the file system before it is backed up. For instance, before running monthly or yearly backups of a file system, it is useful to use **dcopy** to reorganize the file system to a scratch partition, then use **volcopy** to make the actual backup of the file system. If this was not done, when restoring a file system, you might use **volcopy** to restore the file system to a scratch partition, then use **dcopy** to reorganize the file system while copying it to the partition where it resides.

dd

The **dd(1M)** command can be used to copy a file system that is less than or equal in size to the media to which you are copying. To use the **dd** command, you must supply two arguments and various options. The two arguments are:

1. Name of the input file (**if=**); in other words, the special device file for the partition that contains the file system to be backed up.
2. Name of the output file (**of=**); in other words, the special device file for the floppy drive or disk partition to which the file system is to be copied.

For example: `dd if=/dev/rdisk/a710_00s1 of=/dev/rmt/a710_40t`

A number of options are available; see **dd(1M)** for a full listing. Not all of these options are required to backup a file system, although they may improve the speed of a backup operation. Usually, the block size is set to 1024 (**bs=1k**).

After **dd** completes successfully, you can check the backup by running the following command:

```
dd if=/dev/rmt/a710_40t of=/dev/null
```

dd can be used to verify formatted media and backups by checking for read errors. If **dd** reads an entire backup volume without a read error, then the medium or backup is good. However, you may want to do a more extensive test of your backup to ensure that it is good.

To restore a file system from a **dd** backup, use the following command:

```
dd if=/dev/rmt/a710_40t of=/dev/rdisk/a710_00s1
```


Running Incremental Backups

Incremental backups can be run using the **find** | **cpio** combination, **sysadm backup**, or **finc**. Backups run using **finc** are restored with the **freec** command.

These commands allow you to specify the time increment in number of days. All files that have been modified¹ in that number of days are included in the backup. You must decide whether to use:

- ❑ **days since last full backup:** the backup media will get progressively larger and may take longer to run, but if you need to restore the entire file system, you only need to restore the full backup and one incremental backup.
- ❑ **days since last incremental backup:** the backup operations will probably be faster, but you may need to run several restore operations to recover an entire file system. For instance, if you run a full backup on Fridays, and incrementals to the last incremental media on other days, you could lose a file system on Thursday night or Friday morning and need to restore the full backup plus the four incrementals run Monday through Thursday to recover the file system. If you only run full backups once a month or less, the restoration time is even longer.

You will need to set your own policies based on company needs, how much activity there is on a file system, and whether it is more important to save time when making backups or be able to recover a file system quickly.

cpio and sysadm backup

The **cpio** command can be used to do incremental backups or to backup one particular directory for a user. Often administrators use **cpio** to copy files to a backup file system, then copy the backup file system to diskette using **cpio** or one of the other backup commands.

As an example of running an incremental backup to another file system (in this case, the */bck* file system residing on partition *a710_20s2*), the following command sequence copies all files in the */people* file system that have been modified in the last two days:

```
/etc/mount /dev/dsk/a710_20s2 /bck
/etc/mount -r /dev/dsk/a710_20s3 /people
cd /people
find . -mtime -2 -print | cpio -pdm /bck
```

The **find** command searches the current directory (.) for all files that have been changed in the previous two days (**mtime -2** option). The output of this command is then piped to **cpio**, which copies all files that met **find**'s criteria to the */bck* file system. Note that this was run against the block device; the file system must be mounted.

¹ When using **finc** you can specify all files that have been accessed rather than all files that have been modified.

Even though **cpio** is run against a mounted file system, it is best that the file system be mounted read-only or that the system be in single-user state to prevent users from updating the file system while **cpio** is running. Any files that are in the process of being updated when **cpio** copies them cannot be restored, and the job may fail if a file is deleted after **find** has located it but before **cpio** copies it. For incremental backups, some administrators are willing to take the small risks associated with this, especially if backups can be run at a time when the system is quiet. Full backups should always be run when the file system is mounted read-only or the system is in single-user state.

To restore all files from this backup partition, use the following command sequence:

```
/etc/mount /dev/dsk/a710_20s2 /bck
cd /bck
find . -print | cpio -pdm /people
```

Add the **cpio -u** option, which means to copy unconditionally, if you want to overwrite any files on the working disk with older files on the backup medium.

The following command sequence copies all files in the */people* file system that have been modified in the last two days:

```
/etc/mount -r /dev/dsk/a710_20s3 /people
cd /people
find . -mtime -2 -print | cpio -ovBc > /dev/rmt/a710_40t
```

After **cpio** completes successfully, you can check the backup with the **dd(1M)** command. For the proper command sequence, see the discussion on **dd** earlier in this chapter.

To restore all the files in the */people* file system from a **cpio** backup, use the following command sequence:

```
cd /people
cpio -ivuB < /dev/rmt/a710_40t
```

To restore an individual file from a **cpio** backup, specify the *directory/filename* after the **cpio** options.

finc(1M)

finc(1M) is the fast incremental backup utility. It is often used for incremental backups that supplement full backups done with **volcopy(1M)**.

You must run **labelit(1M)** to label the diskette before running **finc**. The file system should be mounted read-only, although **finc** can be run against a file system that is mounted normally; see the discussion in the **cpio** section. So, to do an incremental save of the */people* file system that saves all files that have been modified in the last three days, the command sequence is:

```
labelit /dev/rmt/a710_40t t011
finc -m 3 /people /dev/rmt/a710_40t
```

Instead of the **-m** option, you can use the **-a** option (save files that have been accessed in *n* days) or the **-c** option (save files whose inode has been changed in *n* days). The **-n file** option is particularly useful for shell scripts; **finc** saves all files that have been modified more recently than *file*. If you set up a file that is modified (can be a true modification, or use **touch(1)** to change the date of last modification) only when you run a full backup on the file system, then **finc** can always be run against this file.

Run **ff(1M)** on the file system either just before or just after running **finc**. This gives you a full listing of all files on the file system with the inode and, optionally, the owner and size of each file. The inode will be required to restore individual files from the **finc** backup. **ff** takes the same arguments for time as **finc**, and should be used with exactly the same arguments so the listing matches. For instance, to run **ff** for the backup in the example above, if */people* is mounted on */dev/dsk/a710_20s3*, the command is:

```
ff -m 3 -lsu /dev/dsk/a710_20s3
```

Individual files can be restored from a **finc** backup with the **frecc(1M)** command; use **volcopy** to restore the entire **finc** backup.

frec(1M)

frec(1M) is used to restore individual files from a **volcopy** or **finc** backup. You must specify the inode of the file on the backup media and the full path name to which the file should be written. If a file of that name exists when you run **frec**, it will be overwritten. For example, the following command restores the specified files:

```
frec /dev/rmt/a710_40t 367:/people/john/program 1114:/people/sue/data
```

The command used in this fashion is useful to restore only a few files. If you have a large number of files to restore, you can set up a file such as the following:

```
367:/people/john/program
1114:/people/sue/data
5467:/people/lee/memos
```

If the name of this file is *fixfiles*, you can then restore all these files with the following command:

```
frec -f fixfiles /dev/rmt/a710_40t
```

Backing Up Specific Directory Trees and Files

There are times when you need to backup just specific files or directories. For example:

- ❑ A user has left the company. Before deleting the user's directory tree, you want to backup all files in the directory in case they are needed at some future time by another user on the same project.
- ❑ A data file (or set of files) have grown large. Before cleaning out old data or deleting the files, you want to preserve an intact copy of the file(s).
- ❑ Files such as the **sar** data files are overwritten once a month. If you need to record system usage statistics over a long period, you might want to backup the `/usr/adm/sa` directory once a month.

cpio

The **find** | **cpio** combination can also be used to backup a specific directory tree or set of files. As an example of backing up a specific directory for a user, the following command sequence backs up the files in the `/people/abc/data` directory to the device `/dev/rmt/a710_40a`:

```
cd /people/abc/data
find . -print | cpio -ovBc >> /dev/rmt/a710_40a
```

This will save a copy of all files and directories under `/people/abc/data`.

The command sequence to restore these files is:

```
cd /people/abc/data
cpio -icBvdm < /dev/rmt/a710_40a
```

sysadm store

sysadm store is a menu-driven utility that calls **cpio** to backup the files and directories of your choice. You provide a base directory and selection criteria, then any descendant files of that directory matching the criteria are saved in the archive. The selection criteria can be a name-matching pattern (in the style of shell pattern matching), a file modification date more recent than a given point in time, a specific file owner, or any of a variety of other criteria. **sysadm store** also provides the option of listing all the files copied.

Using fsck to Check and Repair the File System

The *root* file system is automatically checked by the */etc/bcheckrc* file whenever the system is booted. File systems that are mounted when the system goes to multi-user state (in other words, those file systems listed in the */etc/fstab* file) are checked if the sanity flag indicates that they need checking. You can use the **fsck(1M)** command to check a file system at other times. Appendix D describes the checks that **fsck** does and the error messages it gives, with guidelines on responding to the messages.

If file system checking indicates there is a problem, fix it immediately. The **fsck** program can repair many problems it finds, or you may elect to fix them yourself (either by restoring the file system from a backup copy or running **fsdb(1M)** as described later in this chapter). File system problems can rapidly escalate into major corruption if they are not corrected. Whenever the system is rebooted, it checks the file system integrity and runs **fsck** if necessary.

If a file system suffers major damage, the best solution is often to restore it from a backup copy. If the backup copy is not recent, a great deal of work could be lost. In this case, it is sometimes worthwhile to let **fsck** attempt to fix what it can, so that you might salvage some files. You may also be able to repair the file system using the **fsdb** utility described at the end of this chapter. You should always make a copy of a badly corrupted file system before running a file system check; you may also be able to recover files from this copy.

Running fsck

The **fsck(1M)** file system check utility interactively checks and repairs the file system. **fsck** uses the structural information in the file system to perform consistency checks. If an inconsistency is detected, a message describing the problem is displayed. Note the following:

- ❑ When **fsck** is run during system startup, it uses the **-D** and **-y** options. **-D** checks directories for bad blocks; **-y** automatically answers yes to all questions. It is essential to use **-y** during system startup so that the automatic reboot will not halt waiting for operator response. Some administrators always use the **-y** option so they can walk away while **fsck** runs; others prefer to respond to the prompts on an individual basis.
- ❑ When running **fsck**, file systems are unmounted and checked as raw devices. It is faster to run **fsck** on an unmounted file system and, if you let **fsck** do repairs to the file system, it is important that no users are trying to write to the file system at the same time.
- ❑ Before running **fsck**, be sure the file system has a *lost+found* directory. **fsck** writes salvageable pieces of files to *lost+found* where they can be identified by the owner. Use the **mklost+found(1M)** command to create a *lost+found* directory.
- ❑ It is occasionally useful to run **fsck -n** on a mounted file system, such as when you have system problems and want to see if they are being caused by a corrupt file system before you make users log off. Using the **-n** option ensures that **fsck** will not modify the file system; the program will terminate as soon as it finds one problem for which the prompt

is **CONTINUE?**. If you get such a message, you should unmount the file system and do a regular file system check for a complete listing of difficulties.

- ❑ To check and repair the *root* file system, put the system in single-user state and unmount all file systems except *root*. Run **fsck -b**; if **fsck** finds problems in *root* that it can fix, it makes the modifications then automatically remounts the repaired *root*.
- ❑ **fsck -p** performs parallel file system checks. **fsck** uses the information in the */etc/checklist* file to determine which disks to run in parallel. You must also specify either the **-n** or the **-y** option. Never run a parallel check on the *root* file system or on the file system where your system dumps are saved to. (See the note in the */etc/init.d/MOUNTUSR* file for more information.)

In order for a parallel check to run after a system reboot, an entry needs to be added to the */etc/init.d/MOUNTFSYS* file. For example:

```
echo "Doing parallel fsck"
/etc/fsck -t /fstmp -y -p -D
```

When used with the **-p** option, **-t** specifies the prefix used for the temporary file. The **-t** option specifies that the following argument, in this example */fstmp*, is to be used as the scratch file. This file should not be on the file system being checked and if it did not already exist, will be removed when **fsck** completes.

- ❑ **dfsk** checks two file systems at a time. If the two file systems are on different disk drives, this can sometimes save time. See **fsck(1M)** for instructions on using **dfsk**.
- ❑ On the REAL/IX Operating System, file systems are usually created without specifying the blocks-per-cylinder, rotational gap, and blocks-per-track. If you specified these (**-s** option to **mkfs**), you *must* use the **-s** or **-S** option with the same specification every time you run **fsck** against the file system. **-s** causes the free list (S5 architecture) or free bitmask (F5 architecture) to be rebuilt whether or not **fsck** finds corruption; **-S** causes the free list or free bitmask to be rebuilt only if **fsck** finds corruption.
- ❑ **fsck -f** does a partial file system check, running only Phase 1 (block and size check), Phase 5 (free list or free bitmap check) and, if Phase 5 detects corruption in the free list or free bitmap, Phase 6 (reconstruct free list or free bitmap).
- ❑ If you are checking a large file system on a configuration that has little available memory (either because you have a small memory configuration or because you have allocated most of the memory for other purposes), **fsck** may need additional memory to run. If **fsck** has inadequate space in main memory and all configured **swap** devices, it will prompt you for the name of a file to use for additional scratch space. If this happens, you can use the **-tfile** option to specify a file to use (such as */fsck.tmp*) the next time you run **fsck**. Some administrators prefer to always use this option. Note that the **fsck** scratch file must be on a mounted file system; *root* is a good file system to use if there is adequate space, since it is already checked and mounted whenever **fsck** is run on other file systems.

The **fsck** command operates in 5 phases (plus a possible sub-phase to check on duplicate blocks). A sixth phase is run only if the free block organization is corrupt and you want to salvage it. As each phase is completed, a message is displayed. At the end of the program, a summary message is displayed, showing the number of files (inodes), blocks, and free blocks.

fsck -p does not display the phase messages. However, errors are still reported.

/etc/checklist

The */etc/checklist* file is used to define a default list of file system devices to be checked by */etc/fsck* and */etc/ncheck* when the system is in single-user mode. If you run **fsck -y** without specifying any file system names, all file systems listed in */etc/checklist* are checked. The character (raw) device partition for the file system should be identified, except for *root*, where the block device is identified. When the system is delivered, this file contains an entry for the *root* and */usr* file systems (*/dev/root* and */dev/usr*). As you create new file systems and add them to */etc/fstab*, you should also add them to */etc/checklist*, so that it contains a listing for each file system that is mounted when the system is in multi-user mode.

If you want to use the parallel option (**-p**) of the **fsck** utility, an additional piece of information needs to be added to the end of each line of the */etc/checklist* file. Each physical disk on the system must be assigned a number, starting with zero, as shown in the following sample */etc/checklist* file:

```
/dev/dsk/a710_10s0    0
/dev/dsk/a710_10s2    0
/dev/dsk/a710_20s0    1
/dev/dsk/a710_30s0    2
/dev/dsk/a710_30s2    2
```

In this sample, there are disks at addresses 10, 20, and 30. Disks 10 and 30 each have two file systems, and disk 20 has one file system. **fsck** will check one partition from each physical disk in parallel. When one file system check is complete, the next file system on that disk is checked.

All file systems on the same disk must have the same number. While no harm is done in assigning them different numbers, the time to **fsck** a disk will increase if two file system checks occur on the same disk at the same time because of the extra disk head movement.

The partitions */dev/root* and */dev/usr* must be removed from */etc/checklist* when running **fsck** with the parallel option.

If you have a large number of disks, you may want to group physical disks into different groups and assign each disk group a number. This may be desirable to avoid swapping. If you do this, one file system from each group is checked in parallel.

Using fsdb to Repair Damaged File Systems

When logged in as *root*, you can use the file system debugger, **fsdb(1M)**, to correct discrepancies that exist in a file system, as well as to view the structure of the file system. **fsdb** functions like an editor; you can use it to print and alter any part of a file system as if it were a file. However, **fsdb** does not have a capability to "undo" a command, so use it with extreme caution. Before you change any information on the disk, research the problem thoroughly.

Although **fsdb** is not difficult to use, be very careful. You are editing a raw disk and the slightest error can have severe consequences. You should understand file system structures before you edit the raw disk. In addition, use the following precautions:

- Always make a backup copy of the file system (using the **volcopy** or **dd** command) before you begin **fsdb**; then you can always "start over" if you make a mistake.
- Always run **fsdb** from a hard-copy terminal or keep a complete log of all the issued commands.

The command to run **fsdb** is:

```
/etc/fsdb /dev/rdisk/special_device_file
```



fsdb can be run on a mounted file system. Be sure to use the **-r** read-only option so that you do not modify anything in the file system. You may note a few inconsistencies during the session as other processes modify the file system, but these should not interfere with your ability to study the file system structure.

Once **fsdb** is initialized, you can move around in the file system with the Action Characters listed in Table 5-1. The debugger knows its current location, and unless you specify another spot in the file system, it acts on the current inode.

To print portions of the file system, use the following command format:

```
address.format
```

To modify a portion of a file, use the following command format:

```
[address.]member=value
```

The operational characters used in these strings are listed in Table 5-1. The next two sections give examples of how to use them.

Table 5-1. fsdb Command Summary

Initialize fsdb:		<i>/etc/fsdb /dev/rdisk/special_device_file</i>
Display portion of file:		<i>address.format</i>
Modify portion of file:		<i>[address.]member=value</i>
Action Characters	! escape to shell q quit p general print facilities >,< save, restore and address = numerical assignment =+ incremental assignment =- decremental assignment =" character string assignment O error checking flip flop X radix flip flop (hex/octal)*	
Address	xi for inumber 3i is inode 3 xb for block number 5b is block 3 xB for byte number 512B is byte 512 x# for absolute address (Difficult to use)	
Format	fformat print first block Use format as specified i print as i-nodes d print as directories o print as octal words e print as decimal words c print as characters b print as octal bytes s print as superblock S print as superblock	
Member	md:d mode is directory Should be assigned as octal number md:f mode is regular file ln link count # of directories pointing at this inode uid user ID number Name stored in <i>/etc/passwd</i> gid group ID number Name stored in <i>/etc/group</i> sz file size in bytes a# data block numbers 0 - 12 points to data block maj major device number See <i>/etc/master.d</i> files min minor device number Specific device ct change time Last time inode modified at access time Last time file read mt modify time Last time file modified d# directory slot Contains inode number d#.nm name field in directory slot Value in quotes unless first char alpha e[0..3]o print extent offset 0-3 e[0..3]s print extent sum 0-3 lw last written fl flags	

* To indicate the block address in hexadecimal (radix flip flop= on), additional field separators are required. *a0b* (octal) is equivalent to *a0.b* (hexadecimal).

Using fsdb to View a File System

You can use the **fsdb** command to view and repair the internal structure of a file system. Ideally you should start by using **fsdb** to view a non-damaged file system. The following pages give a sample **fsdb** viewing session. Viewing a file system with **fsdb** is a good way to get a full understanding of file system structure, even if you never use it to make repairs.

Starting the fsdb Session

After you invoke `/etc/fsdb /dev/rdisk/a710_00s3`, some basic file system information is displayed, as illustrated in Figure 5-1.

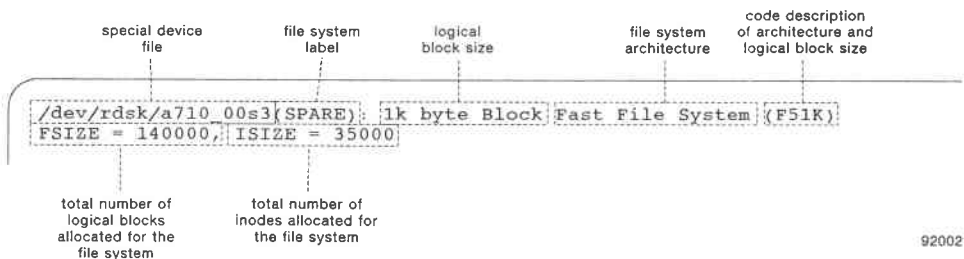


Figure 5-1. Basic File System Information Displayed when fsdb is Invoked

Similar information is displayed for an S5 file system, but the architecture is defined as "Standard File System" and the code description is S51K.

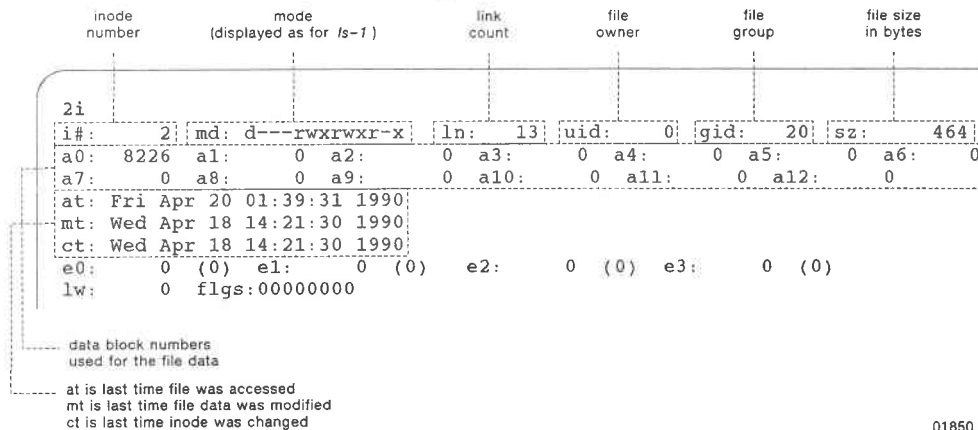


fsdb displays the number of logical blocks because internally this is how the system sees the file system. Most commands (such as **mkfs**) use physical blocks rather than logical blocks. In other words, the number of physical blocks for the file system shown in the example above would be 280000 because each 1-Kbyte logical block equals two 512-byte physical blocks.

Viewing the Structure of an inode

To display a particular inode, provide the inode number and the format information. Similar information is displayed each time you move to a new inode.

Figure 5-2 illustrates the output of the `2i` command that prints inode #2.



01850

Figure 5-2. Display of inode 2

Inode #2 is the inode for the `.` and `..` files in the file system. Note the following:

- ❑ The data blocks referenced in the `a0` and `a1` fields of the display contain a listing of the files and directories located in the first level below the root directory of the file system. For regular files, the data blocks are the blocks that contain the actual data.
- ❑ The UID and GID displayed correspond to the owner and group displayed for `.` and `..` in an `ls -lai` listing.
- ❑ The number of bytes displayed in the `sz` field and the number of data blocks used will increase as more files are created in the file system, but are not affected by the size of the sizes of the individual files and directories that are created. This is true of any inode that is associated with a directory-type file.
- ❑ The `e0` through `e3` fields list extents, and will always be 0 for file systems on S5 file systems and non-preallocated files, as well as for inode #2. The `flgs` field will display the value of 0 unless extents were allocated for the file.
- ❑ The `lw` field will often contain a non-zero value on F5 file systems. This value represents the last data block allocated to the inode. The system uses this information to preserve contiguity in the file system.

The **e0** – **e3** and **flgs** fields of the display are meaningful only when the inode refers to a preallocated file. For example, a preallocated file might include these two lines in the display of the inode for a preallocated file:

```
e0:      300 (10) e1:      100 (60) e2:      500 (62) e3:      200 (162)
lw:      6789 flgs:60000001
```

For each extent, **fsdb** displays the offset and the running sum of the offset sizes. This extent list could be graphically illustrated as shown in Figure 5-3.

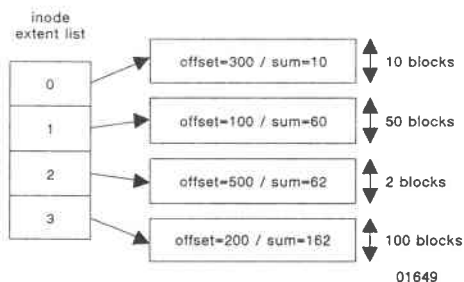


Figure 5-3. Graphic Representation of Extent List

The **lw:** field gives the data block number last allocated to the file. In this case, it is block 6789, so the system will attempt to write to allocate 6790 (or another block close by) on the next write, to preserve contiguity in the file.

The **flgs:** field indicates the flags for the file. Table 5-2 shows the values associated with each flag or command option.¹ In the example above, the extents were created with the **-gz** options.

Table 5-2. Flag Values for Preallocated Files

Flag prealloc(2)	option prealloc(1)	Numerical value
F5NOGROW	-g	00000001
F5SHRINK	-s	00000002
F5EXT		40000000
F5NOZERO	-z	20000000

Note that the **F5NOCHANGE** flag (**-c** option) causes an immediate action but is not stored in the inode's flag list.

¹These flags are defined in the `/usr/include/sys/fcntl.h` header file.

Viewing the Data for an inode

To view a particular file with **fsdb**, you must know the inode number of the file, which can be obtained by invoking the **ls -ild** command before invoking **fsdb**. The inode of "." in the directory is a good place to start, since its data blocks contain a listing of all files in the directory. For example, if you **cd** to a directory and run **ls -lai** and get the following display:

```
9632 drwxr-xr-x 5 thl cecw 224 Sep 18 11:30 .
```

The leftmost column gives the inode of the file, which is 9632. After you invoke **fsdb**, issue the **9632i** command to move to inode 9632; the structure of the inode will be displayed. To print the contents of the first data block as directories, issue the command **fd**. The output would look similar to the following sample:

```
d0: 9632 .
d1: 1349 .
d2: 9633 . p r o f i l e
d3: 9634 . m a i l r c
d4: 9635 . e x i t p r o f i l e
d5: 9636 m a i l
d6: 9645 b i n
d7: 9687 . n e w s _ t i m e
d8: 9688 r j e
d9: 9689 . j x o u t
d10: 9690 . e x r c
d11: 9691 . j x e r r
d12: 9692 m b o x
d13: 0 h i d e . m a i l r c
d14: 0
:
d127: 0
```

The second and third columns of the sample list the inode and file name, respectively, for each file in the directory. Let's say we want to examine the *.exitprofile* file. The sample shows that the inode is 9635, so issue the **9635i** command to move to inode 9635.

The **fc** command is one way to look at the contents of the first data block associated with this inode. The **c** means print the data as characters. Sample output from issuing an **fc** command is shown below. The contents of the first data block associated with the current inode (9635), are printed as ASCII characters. The left column gives the physical address from the beginning of the file system, in octal bytes.

```

174547000: $ { H O M E } / b i n / F o r w
174547020: a r d   t o   h r c c a ! t h l
174547040: c s   e c h o   S e s s i o n
174547060:   f r o m   $ { S T A R T _ T I
174547100: M E } t o   ' d a t e ' + %
174547120: r % a % D ' '   s y n c s
174547140: l e e p 5 / b i n / s t t y
174547160: 0   <   / d e v / t t y
174547200:
:
```

You can also look at the file contents by referring to the actual block address and specifying how much of the output to print. The number after **a0:** in the initial inode display (Page 5-20) shows the number of the first data block in the file. Examples of commands that could be used are:

312b.p0200c Prints the first 128 (which is 0200 octal) bytes of the 321st block of the file system.

63847b.p0c Print contents of entire data block.

A sample display of **312b.p0200c** is shown below. This display is similar to the **fc** display.

```

174547000: $ { H O M E } / b i n / F o r w
174547020: a r d   t o   h r c c a ! t h l
174547040: c s   e c h o   S e s s i o n
174547060:   f r o m   $ { S T A R T _ T I
174547100: M E } t o   ' d a t e ' + %
174547120: r % a % D ' '   s y n c s
174547140: l e e p 5 / b i n / s t t y
174547160: 0   <   / d e v / t t y
174547200:
:
```

Using fsdb to View the Super Block

Viewing the super block with **fsdb** is a good way to truly understand the file system structure described in *Concepts and Characteristics*. It may be useful to reread that material in conjunction with the following pages if you are not completely familiar with the file system structure. In viewing the super block, the differences between the two file system architectures become quite clear as well, which is why our discussion below will include samples for both file system architectures.

The file system super block begins at the 512th byte; you would print the super block in decimal, with the command **512B.p0S**.¹ The sample on page 5-25 illustrates the superblock for an S5 file system. Note the following:

- ❑ The display includes two separate sections. The "Superblock" section shows the information that is stored on disk in the superblock; the "Extended Superblock" is an incore-only data structure of common sub-expressions calculated from information in the superblock.
- ❑ The field names displayed are defined in the `/usr/include/sys/fs/s5filsys.h` file. You can look there for definitions of each field.
- ❑ The **isize** displayed here is the number of the last block used for the *ilist*, so is the number of blocks used for the *ilist* plus two (for the two blocks at the beginning of the file system that contain the superblock). Contrast this to **ISIZE** displayed when **fsdb** is first invoked; **ISIZE** represents the total number of inodes allocated to the file system when it was created.

The number of inodes stored per block in the *ilist* can be calculated from the number of bytes in each inode (on S5 file systems, each inode consumes 64 bytes, and on F5 file systems, each inode consumes 128 bytes) and the size of the logical block (1 Kbyte in this example). Each 1-Kbyte block on an S51K file system can store 16 inodes ($16 \times 64 = 1024$) and each block on an F5 file system can store 8 inodes ($8 \times 128 = 1024$).

- ❑ The "Free block list" displayed here is the cache of 50 free blocks that are held in the superblock of S5 file systems. It is not a list of all free blocks available for the file system. F5 file systems use a bitmask rather than the free block list to allocate data blocks, so all values in the free block list will be 0 for F5 file systems, as shown on page 5-27.
- ❑ **nfree** is a pointer to the last free block number used. This value is always 0 on F5 file systems, as shown on page 5-27.
- ❑ **magic** is the magic number for the file system architecture. A different magic number is used for F5 file systems, as shown on page 5-27.
- ❑ The number 0x2 shown in the **type** field tells us that this is a 1-Kbyte block file system. For a 2-Kbyte block file system, this number would be 0x3, for a 4-Kbyte block file system, this number would be 0x4, and so forth, up to a 128-Kbyte block file system, which would display 0x9.

¹ You could also use **512B.p0e** to print the super block information. Most users find the output more difficult to interpret than that produced by the **S** format.

512B.p0S

```

----- Superblock -----
isize: 0xbc7(3015)      fsize: 0x2f134(192820)  nfree: 0x26(38)  ninode: 0x62(98)
Free block list:
0: 0x229e 1: 0xbd2e 2: 0x21a51 3: 0x299b8 4: 0x1ffb8 5: 0x219d0
6: 0xfbc1 7: 0xfbc2 8: 0xfbc3 9: 0xfbc4 10: 0x319d 11: 0x29395
12: 0x318d 13: 0xbd0 14: 0xbcf 15: 0xbef 16: 0x1f104 17: 0x1f105
18: 0x3195 19: 0x3196 20: 0x3197 21: 0x7a0d 22: 0x1c516 23: 0x2032
24: 0x19690 25: 0x1a1fc 26: 0x1112a 27: 0x1904e 28: 0x2068 29: 0x2036
30: 0x2031 31: 0x65dd 32: 0x1c51c 33: 0x1eb99 34: 0x196e5 35: 0x2079
36: 0x206b 37: 0x2060 38: 0x22c0 39: 0x22be 40: 0x22bd 41: 0x22bc
42: 0x22bb 43: 0x291c1 44: 0x229c 45: 0x22a2 46: 0x22a1 47: 0x291eb
48: 0x22a0 49: 0x229f
Free inode list:
0: 0x72 1: 0x71 2: 0x70 3: 0x6f 4: 0x6e 5: 0x6d
6: 0x6c 7: 0x6b 8: 0x6a 9: 0x69 10: 0x68 11: 0x67
12: 0x66 13: 0x65 14: 0x64 15: 0x63 16: 0x62 17: 0x61
18: 0x60 19: 0x5f 20: 0x5e 21: 0x5d 22: 0x5c 23: 0x5b
24: 0x5a 25: 0x59 26: 0x58 27: 0x57 28: 0x56 29: 0x55
30: 0x54 31: 0x53 32: 0x52 33: 0x51 34: 0x50 35: 0x4f
36: 0x4e 37: 0x4d 38: 0x4c 39: 0x4b 40: 0x4a 41: 0x49
42: 0x48 43: 0x47 44: 0x46 45: 0x45 46: 0x44 47: 0x43
48: 0x42 49: 0x41 50: 0x40 51: 0x3f 52: 0x3e 53: 0x3d
54: 0x3c 55: 0x3b 56: 0x3a 57: 0x39 58: 0x38 59: 0x37
60: 0x36 61: 0x35 62: 0x34 63: 0x33 64: 0x32 65: 0x31
66: 0x30 67: 0x2f 68: 0x2e 69: 0x2d 70: 0x2c 71: 0x2b
72: 0x2a 73: 0x29 74: 0x28 75: 0x27 76: 0x26 77: 0x25
78: 0x24 79: 0x23 80: 0x22 81: 0x21 82: 0x20 83: 0x1f
84: 0x1e 85: 0x1d 86: 0x1c 87: 0x1b 88: 0x1a 89: 0x19
90: 0x18 91: 0x17 92: 0x16 93: 0x15 94: 0x10 95: 0x13
96: 0x14 97: 0x12 98: 0xf 99: 0x9
flock: 0 ilock: 0 fmod: 0 roonly: 0
time: Tue Apr 17 10:56:31 1990
dinfo: 012000000 tfree: 9887 tinode: 48190
fname: dumps fpack: 1d0s3
state: 0x5e72d81a magic: 0xfd187e20 type: 0x2
----- Extended Superblock -----
bsize: 0x400(1024) bmask: 0x3ff bshift: 10 ltopshift: 1
bpb: 0x2000(8192) bpbmask: 1fff bpbshift: 13
nindir: 0x100(256) nmask: 0xff nshift: 8
inopb: 16 imask: 0xf ishift: 4 irnd: 0x1f(31) isiz: 0x40(64)
bbsize: 0xbc7(3015) bbstart1: 0xbc7(3015) bbstart8: 0xbc7(3015)

```

01800

The superblock of an F5 file system looks a bit different when displayed with **fsdb**; page 5-27 shows an example of an F5 file system. Note the following:

- ❑ The free block list shows 0's in place of block numbers. F5 file systems do not use the free block list to allocate blocks, but instead use a bitmap that is located between the end of the *ilist* and the beginning of the list of free data blocks.
- ❑ The bitmask begins in the block following *isize*, and ends in the block specified by *bbsize* in the Extended Superblock portion of the display.

A graphic illustration of the F5 file system, showing the field names as well as the corresponding values from page 5-27 is shown here:

<i>offset 512 bytes</i>	Superblock
<i>logical block 2</i>	65496 Inodes (128 bytes each)
<i>logical block 8188 (isize-1)</i>	
<i>logical block 8189 (isize)</i>	Bitmap
<i>logical block 8225 (bbsize-1)</i>	
<i>logical block 8226 (bbsize)</i>	Data Storage Blocks
<i>logical block 302049 (fsize-1)</i>	

- ❑ **nfrees**, which points to the last free block used in the file system, always has the value of 0 for an F5 file system because the free block list is not used.
- ❑ **magic** has a different value for an F5 file system than for an S5 file system.

512B.p0S

```

----- Superblock -----
isize: 0x1ffd(8189)      fsize: 0x49be2(302050)  nfree: 0(0)  ninode: 0x60(96)
Free block list:
0: 0 1: 0 2: 0 3: 0 4: 0 5: 0
6: 0 7: 0 8: 0 9: 0 10: 0 11: 0
12: 0 13: 0 14: 0 15: 0 16: 0 17: 0
18: 0 19: 0 20: 0 21: 0 22: 0 23: 0
24: 0 25: 0 26: 0 27: 0 28: 0 29: 0
30: 0 31: 0 32: 0 33: 0 34: 0 35: 0
36: 0 37: 0 38: 0 39: 0 40: 0 41: 0
42: 0 43: 0 44: 0 45: 0 46: 0 47: 0
48: 0 49: 0
Free inode list:
0: 0x47b 1: 0x214c 2: 0x214b 3: 0x214a 4: 0x2149 5: 0x2148
6: 0x2147 7: 0x2146 8: 0x2145 9: 0x2144 10: 0x2143 11: 0x2142
12: 0x2141 13: 0x2140 14: 0x213f 15: 0x213e 16: 0x213d 17: 0x213c
18: 0x213b 19: 0x213a 20: 0x2139 21: 0x897 22: 0x899 23: 0x898
24: 0x896 25: 0x89c 26: 0x89d 27: 0x89e 28: 0x89f 29: 0x8a0
30: 0x8a1 31: 0x8a2 32: 0x8a3 33: 0x8a4 34: 0x8a5 35: 0x8a6
36: 0x8a7 37: 0x8a8 38: 0x8a9 39: 0x8aa 40: 0x8ab 41: 0x8ac
42: 0x89b 43: 0x8ad 44: 0x89a 45: 0x8af 46: 0x8b1 47: 0x8b2
48: 0x8b3 49: 0x8b4 50: 0x8b5 51: 0x8b6 52: 0x8b8 53: 0x8b9
54: 0x8ba 55: 0x543 56: 0x8bb 57: 0x8bc 58: 0x8bd 59: 0x8be
60: 0x8bf 61: 0x8c0 62: 0x8c1 63: 0x8c2 64: 0x8c3 65: 0x8c4
66: 0x8c5 67: 0x8c6 68: 0x8c7 69: 0x8c8 70: 0x8c9 71: 0x8ca
72: 0x8cb 73: 0x8cc 74: 0x8cd 75: 0x8ce 76: 0x8cf 77: 0x8d0
78: 0x694 79: 0x79b 80: 0x8d1 81: 0x8d2 82: 0x8d3 83: 0x8d4
84: 0x8b7 85: 0x8d5 86: 0x8d6 87: 0x8d7 88: 0x8d8 89: 0x8d9
90: 0x8da 91: 0x8de 92: 0x8df 93: 0x8db 94: 0x8dc 95: 0x8dd
96: 0x8b0 97: 0x8e1 98: 0x8e2 99: 0x8e3
flock: 0 ilock: 0 fmod: 0 ronly: 0
time: Fri Apr 20 08:45:31 1990
dinfo: 012000000 tfree: 66630 tinode: 49301
fname: SPARE fpack: 1d2s3
state: 0x5e72d81a magic: 0xfd187e3d type: 0x2
----- Extended Superblock -----
bsize: 0x400(1024) bmask: 0x3ff bshift: 10 ltopshift: 1
bpb: 0x2000(8192) bpbmask: 1fff bpbshift: 13
nindir: 0x100(256) nmask: 0xff nshift: 8
inopb: 8 imask: 0x7 ishift: 3 irnd: 0xf(15) isiz: 0x80(128)
bbsize: 0x2022(8226) bbstart1: 0x2022(8226)

```

01780

Example of Repairing a File System with fsdb

You can fix most file system corruption by letting **fsck** handle it. However, you can use **fsdb**:

- ❑ to determine if a non-empty inode that **fsck** wants to clear should be (or can be) saved.
- ❑ to confirm the accuracy of either **fsize** or **isize**. The numbers for **fsize** and **isize** are determined when the file system was created and should not change.
- ❑ when either the entry "." or ".." in a directory type file has been corrupted. **fsck** usually cannot repair this sort of damage, but it can be repaired rather easily with **fsdb**.

If the entry "." or ".." has been corrupted in a directory, it will be difficult to access the files in that directory, even though the data may be good. The error message may be:

cannot access .

This condition can occur from the following sequence of events. You issue an **ls** command to look at the */install/bin* directory; the **ls** command works from the */install* directory specifying *bin*, but when you **cd** into that directory, the **ls** command lists no files. However, if you issue an **ls** command that specifies the file names, the names are displayed.

```
# cd /install

# ls -axF
adm.jab bin/  int21 -5c jim

# ls -axF bin
/ ../ Lock* ksh* logout* src/ train* train2*

# cd bin

# pwd
/install/bin

# ls
. not found

# ls -xF Lock ksh
Lock ksh

# ls *
* not found

# ls -xF src
dclock.c hanoil.c
```

Issuing some other `ls` commands gives messages that `.` is not found:

```
# ls -la
. not found

# ls -ld
. not found

# ls -la src
total 17

drwxr-xr-x  2 jab other    64 Oct  4 10:19 .
drwxr-xr-x  3 jab other   128 Oct  4 10:19 ..
-rw-r--r--  1 jab other  2630 Nov 21 1984 dclock.c
-rw-r--r--  1 jab other  4483 Nov 27 1984 hanoil.c

# cd src
src: bad directory

# cd ./src
./src not found

# ls -ail
total 530

  2 drwxr-xr-x   5 jab other    544 Oct  4 10:52 .
  2 drwxrwxr-x  16 root sys     320 Oct  2 11:26 ..
 17 -rw-rw-rw-   1 jab other    192 Oct  3 11:03 .exrc
 19 -rw-----   1 jab other   1558 Oct  4 10:18 .histfile
  7 -rw-r--r--   1 jab other     93 Oct  3 09:10 .jxerr
  6 -rw-r--r--   1 jab other      0 Oct  3 09:10 .jxout
 42 -rw-r--r--   1 jab other   2546 Sep  9 16:57 adm.jab
  8 drwxr-xr-x   3 jab other    128 Oct  4 10:19 bin
 44 -rw-rw-rw-   1 jab other  24909 Oct  2 15:42 int21-5C
 32 -rw-r--r--   1 jab other      0 Mar 14 1985 jim
```

The directory `/install/bin` has problems when one refers to `."`, or if one tries to execute the command `ls`, or if one tries to use metacharacters. The files in `/install/bin` can be accessed with fully qualified pathnames, but not from the `/install/bin` directory. The command `ls -ail` in the parent directory of `/install/bin` shows the inode number of the `bin` directory is 8.

To fix the problem, unmount the file system and execute `fsdb`.

The system displays basic information about the file system, for example:

```
/dev/rdisk/a710_20s3 (install) : 2-Kbyte Block Standard File System (S52K)
FSIZE = 711, ISIZE = 192
```

Because the inode on the suspect directory is 8, type the **8i** command to show the suspect inode. The system responds with basic information about the inode:

```
i#:      8      md: d---rwxr-xr-x  ln:    3  uid:  101  gid:   1  sz:   128
a0:     28    a1:    0  a2:    0  a3:    0  a4:    0  a5:    0  a6:    0
a7:      0    a8:    0  a9:    0 a10:    0 a11:    0 a12:    0
at: Fri Oct  4 15:00:03 1985
mt: Fri Oct  4 10:19:28 1985
ct: Fri Oct  4 10:19:28 1985
```

01499

The first data block this inode points to is "28" and the previous **ls bin** showed us only 8 entries in the directory, so the command **28b.p10d** will print the first 10 entries in the first block pointed to by this inode as directory entries.

```
d0:      8
d1:      2      .
d2:      9    l o g o u t
d3:     10    L o c k
d4:     11    k s h
d5:     12    t r a i n
d6:     13    s r c
d7:     16    t r a i n 2
d8:      0
d9:      0
```

Notice the name field on the "top" (d0) entry is blank. The top entry of any directory is always ".", so the name field of the entry should be changed to "." (dot).

To change this, the command is:

```
d0.nm="."
```

The system responds:

```
d0: 8 .
```

Issue a **q** to quit **fsdb**, run **fsck**, mount the file system, and test the same commands which caused problems before:

```
# /etc/mount /dev/dsk/a710_20s3 /install
# cd /install/bin
# pwd
/install/bin
# ls -axF*
./ ../ Lock* ksh* logout* src/ train* train2*
# ls -axF*
Lock* ksh* logout* train* train2*

# ls -axF src
./ ../ dcllock.c hanoil.c
# cd ./src
# pwd
/install/bin/src
```

The problem is now resolved.

Another possible problem is when the **."** entry points to some other directory than itself. In this example, the **."** directory points to the sub-directory **src**. This could cause problems like the following. An **ls** command issued within the **/install/bin** directory gives a different listing than an **ls bin** command issued from within the **/install** directory:

```
# cd /install/bin

# pwd
/install/bin

# ls -C
dcllock.c hanoil.c

# cd ..

# pwd
/install

# ls -C bin
Lock logout src train train2
ksh

# pwd
/install

# cd bin

# ls Lock
Lock

# ls ./Lock
./Lock not found
```

```
# ls -C
dclock.c hanoil.c

# ls *
* not found

# ls L*
L* not found

# ls Lock
Lock
```

The problem is not yet totally obvious, but it seems that any time one tries to reference anything in */install/bin*, the files are there. Looking at the inode of this directory from different locations shows the problem.

From the */install* directory, the listing looks like this:

```
# pwd
/install
# ls -ail bin
total 261

13 drwxr-xr-x 2 jab other 64 Oct 4 10:19 .
2 drwxr-xr-x 5 jab other 544 Oct 4 10:52 ..
10 -rwxrwxrwx 1 jab other 2496 Oct 3 12:05 Lock
11 -rwxr-xr-x 1 root other 119500 Oct 26 1984 ksh
9 -rwxrwxrwx 1 jab other 6036 Oct 4 1984 logout
13 drwxr-xr-x 2 jab other 64 Oct 4 10:19 src
12 -rwxr-xr-x 1 jab other 97 Dec 10 1984 train
16 -rwxr-xr-x 1 jab other 183 Dec 13 1984 train2
```

But if one goes into the */install/bin* directory, one gets a different listing:

```
# cd bin
# ls -ail
total 17

13 drwxr-xr-x 2 jab other 64 Oct 4 10:19 .
8 drwxr-xr-x 3 jab other 128 Oct 4 10:19 ..
14 -rw-r--r-- 1 jab other 2630 Nov 21 1984 dclock.c
15 -rw-r--r-- 1 jab other 4483 Nov 27 1984 hanoil.c
```


If one now issues a `cd` command to move to the directory above (*/install*), the inode for *bin* is not the same as the inode for the "." file that shows when listing *bin*:

```
# cd /install
# ls -ail
total 530

 2 drwxr-xr-x   5 jab   other    544 Oct  4 10:52 .
 2 drwxrwxr-x  16 root   sys     320 Oct  2 11:26 .
17 -rw-rw-rw-   1 jab   other    192 Oct  3 11:03 .exrc
19 -rw-----   1 jab   other   1558 Oct  4 10:18 .histfile
 7 -rw-r--r--   1 jab   other    93 Oct  3 09:10 .jxerr
 6 -rw-r--r--   1 jab   other     0 Oct  3 09:10 .jxout
42 -rw-r--r--   1 jab   other   2546 Sep  9 16:57 adm.jab
 8 drwxr-xr-x   3 jab   other    128 Oct  4 10:19 bin
44 -rw-rw-rw-   1 jab   other  24909 Oct  2 15:42 int21-5C
32 -rw-r--r--   1 jab   other     0 Mar 14 1985 jim
```

The inode of */install/bin* is reported in one place as 8 and in another place as 13. Since the listing of *bin* in its parent directory shows the inode as 8, this is the correct value.

To fix this, `cd` to */*, unmount the file system and execute `fsdb`:

```
# cd /
# /etc/umount /dev/dsk/a710_20s3
# /etc/fsdb /dev/rdisk/a710_20s3
/dev/dsk/a710_20s3(install): 2-Kbyte Block File System
FSIZE = 711, ISIZE = 192
```

Issue the `8i` command to look at inode 8 of the */install* file system. The basic descriptive version of the inode is printed:

```
i#.      8 md: d---rwxr-xr-x ln:  3 uid: 101 gid:  1 sz: 128
a0:      28 a1:  0 a2:  0 a3:  0 a4:  0 a5:  0 a6:  0
a7:      0 a8:  0 a9:  0 a10: 0 a11: 0 a12: 0
at: Fri Oct 4 15:12:23 1985
mt: Fri Oct 4 10:19:28 1985
ct: Fri Oct 4 10:19:28 1985
```

01599

Issue the **28b.p10d** to look at the 28th block, printed as directories. The following is displayed:

```
d0: 13 .
d1: 2 .
d2: 9 l o g o u t
d3: 10 L o c k
d4: 11 k s h
d5: 12 t r a i n
d6: 13 s r c
d7: 16 t r a i n 2
d8: 0
d9: 0
```

Notice that the display says that the inode of the "." file is 13. To fix this, issue the **d0=8** command. The system responds:

```
d0: 8 .
```

To check the results, issue the same **28b.p10d** command you did before. The output now looks like:

```
d0: 8 .
d1: 2 .
d2: 9 l o g o u t
d3: 10 L o c k
d4: 11 k s h
d5: 12 t r a i n
d6: 13 s r c
d7: 16 t r a i n 2
d8: 0
d9: 0
```

You can quit **fsdb** with a **q** command, run **fsck**, and mount the file system:

```
# /etc/mount /dev/dsk/a710_20s3 /install
```

6. Performance Management

Chapter 6

Performance Management

Successful performance management is an art; we can give you guidelines and tell you how to use the tools that measure system activity. Beyond that, you must use your own judgement to obtain optimum system efficiency for your particular environment.

This chapter discusses:

- ❑ how the administrator establishes a high-performance environment.
- ❑ tasks that should be performed routinely to maintain optimum system performance.
- ❑ guidelines for analyzing and resolving performance problems that may arise on the system.
- ❑ tools that can be used for checking and analyzing system performance.
- ❑ tunable parameters that can be modified to customize the operating system to the needs of your environment, tells how to modify them, and specifies the "rules" that apply to the values of the parameters.

The philosophies of performance management are different for systems running time-sharing *versus* systems running realtime applications: in a time-sharing environment, performance management is usually a matter of ensuring that all executing processes get an equitable portion of system resources, whereas in an realtime environment, performance management may be a matter of ensuring that critical realtime processes get the resources they need to react rapidly to external events¹, even if it means that other executing processes get no system resources for a period of time. This chapter discusses both performance goals.

Older UNIX system releases suggested some performance guidelines that are not relevant to the REAL/IX Operating System. These are listed here to benefit administrators who have worked on these other systems.

- ❑ When using `ksh(1)`, the length of user PATHs is no longer a serious performance issue because of the "path caching" feature. The first time a user issues a command, the entire PATH is searched from left to right. The full path name of this command is then "remembered," so that subsequent executions of that command do not require a search of the PATH directories.

¹ This is often done in the code itself by preallocating resources and holding on to them whether they are being used or not.

- The "path name caching" feature further speeds the time to access the executable file. Commonly used files (such as */usr/bin/vi*) and directories are remembered in a fast lookup table in the kernel, in anticipation of future heavy use. They are pushed out of that table only when they are no longer heavily used.

This chapter summarizes the issues an administrator must consider and the tasks that must be done to keep the system performing well. As an administrator, these fundamental concepts are important when considering performance management:

- The REAL/IX Operating System can be customized to serve the performance needs of your particular environment. Many system facilities use data structures that are always resident in memory. On some operating systems, the operating system always reserves a specified amount of memory for these structures, which might be too small for some environments and a waste of memory on others. But on the REAL/IX system (like all UNIX operating systems) you can modify the sizes of these structures, expanding those that are used heavily and shrinking or even eliminating those that are unused.
- Performance management is a constant task for the administrator. Performance degrades if file systems become disorganized or are filled with unneeded files. Moreover, the activities of your users will change over time: memory tables that were once quite adequate will overflow as new users and applications become active, or user file systems and swap space may need to be expanded as new users are added.
- Performance management involves a number of tradeoffs that must be carefully weighed and balanced. For example, one must balance reliability needs with performance needs.
- When realtime applications are involved, the philosophy of performance management changes. The general approach is to ensure that critical realtime processes get all the resources required to execute as quickly as possible, often by preallocating resources and holding on to them for long periods of time when they are not needed so that they are available when they are needed. This will sometimes mean that other processes perform very poorly. The alternative is to distribute resources equitably to all executing processes, but risk having indeterminate response for the critical realtime processes.

Creating an Efficient Environment

Your first performance duty is to create an efficient environment. This includes:

- ❑ selecting an appropriate hardware configuration for your computing needs
- ❑ modifying the tunable parameters (described later in this chapter) to values that are appropriate for your configuration and applications
- ❑ arranging file systems and paging areas on disk for optimal load distribution
- ❑ utilizing an appropriate file system architecture
- ❑ helping users establish efficient environments

The following sections discuss some specific aspects of these topics.

Hardware Configuration

The first step of performance management is to ensure that the hardware configuration is adequate for your needs. Consider the following:

- ❑ In general, it is best to have general users and development work executing on a system separate from the production machine on which realtime processes are executing.
- ❑ For environments that require large amounts of file I/O operations, be sure to configure an adequate amount of disk storage. Otherwise, system performance may be degraded by file systems that are filling up. Also, with adequate disk storage, backups can be run to a scratch disk partition which is then backed up to another media; disk-to-disk copies are faster than copying from disk to diskette, so you can minimize the time the file systems are unavailable to users.

Shifting Workload

Periodically, you should examine the */usr/spool/cron/crontabs* and */usr/spool/cron/atjobs* directories to see if tasks are being run during prime time that could be run at times when the system is idle. Use the **ps(1)** and **systat(1)** commands to determine what processes are heavily loading the system. Encourage users to run large, non-interactive commands (such as **cc** or **make**) during non-prime time with the **at** command.

File System Structure

The REAL/IX Operating System offers two file system structures and a variety of logical block sizes. Selecting the appropriate structure for your file systems can significantly improve system performance.

- ❑ The fast file system architecture (F5) provides faster file access than the S5 file system architecture for most files. In addition, applications accessing files in F5 file systems can preallocate contiguous data storage blocks (extents) that permit large multiple-block transfers directly to user space.
- ❑ The overhead for allocating a file on an S5 file system is lower than on an F5 file system. For this reason, file systems such as */tmp* and */usr/tmp* that constantly allocate small files that are short-lived should use the S5 architecture.
- ❑ Dynamic allocation of file data blocks on F5 file systems supports "clustering," which results in files that are less scattered on the disk, and hence are accessed more efficiently.
- ❑ For large files, file systems that use larger logical block sizes provide better performance than file systems that use smaller logical block sizes. REAL/IX file systems can use logical block sizes ranging from 512 bytes to 128 Kbytes. Tools that can help determine the appropriate logical block size for each file system are discussed later in this chapter.

It is best to spread user directories over file systems located on different disks if they do not need to be able to link files. This reduces the performance degradation that can result from contention of everyone accessing the same disk.

Choosing the Size and Number of Buffers

The REAL/IX Operating System allows you to define a wide variety of buffer sizes to support the wide variety of file system logical block sizes available and to determine the number of each size of buffer. These values should be adjusted over time as the system utilization and configuration changes. Use the **sar -b** and **bfree(1M)** commands to study buffer usage on your system.

Disk I/O Balancing

For optimal system performance, disk activity should be spread equally over all the disks and disk controllers. To check this, study the **sar -d** report, especially the **%busy** (percent of time the device was busy with I/O) field. These values should be roughly similar for all disk devices. If they are not, you should adjust the locations of swap areas and heavily used file systems until the values are similar. To determine whether the swap device or file systems are causing the imbalance, run **sar -w** and look at the **swpin** (transfers from *swap* to memory) and **swpot** (transfers from memory to *swap*) fields.

Partitions located toward the center of the disk are accessed fastest. For this reason, swapping areas and heavily used file systems should be located there as much as possible.

If the `%busy` on all disks is greater than 50% and you are not satisfied with the level of response on your system, you may need to add more disk drives to optimize your disk I/O.

Setting Text Bit (Sticky Bit)

Setting the text bit (commonly called the sticky bit) on a select group of frequently used commands can improve performance, unless you overuse the feature. Without the sticky bit, a region table entry is deallocated when the last process exits and frees the `pregion` entry that points to that region. If the sticky bit is set, the region table entry for text pages of "sticky" commands are kept resident in memory, even when the process terminates. This can reduce the setup time on an `exec(2)` for the text pages of a process.

On systems that usually run a light workload and are seldom in a tight-memory situation, setting the text bit for frequently used commands (such as `vi(1)` and `cc(1)`) can cause a significant improvement in performance. On systems with limited memory or a heavy workload, however, the text bit should not be used. If too many files have the sticky bit set, there may not be enough physical memory for users to execute other programs.

Chapter 10 discusses the accounting system; command summaries identify the most actively used processes. These are the most likely candidates for the sticky bit. The superuser can set and unset the sticky bit with the `chmod +t` command:

```
chmod +t /path/program
```

To identify processes that have the sticky bit set, use the following command:

```
find / -perm -1000 -print
```

Note that a "sticky" command can be swapped out of memory, as can associated data. For critical application programs, use the realtime memory management facilities that enable you to preallocate memory for the process and lock the process in main memory. This is discussed in the *Programmer's Guide*. In most cases, the sticky bit should not be set on systems running programs that lock processes in memory with `plock(2)` and `resident(2)`.

Arranging the Multiple Swap Areas

In addition to the `/dev/swap` area required for booting, the REAL/IX Operating System allows you to create up to 99 additional swapping areas. Swap areas are disk memory regions used to store process pages that `vhand` pages out to disk. Proper utilization of this feature can improve your system performance.

A good starting place for general purpose systems is to allocate at least 1000 blocks of swap space for each user on your system. The exact amount of swap space required varies according to the efficiency of the application programs and the degree of system usage by each user. If users report that programs are failing because of inadequate memory, you should allocate more swap devices or increase the size of the existing swap devices.

Many realtime environments are configured with large amounts of memory relative to the needs of the applications running. Much of the application code is locked into preallocated memory with the **plock(2)** and **resident(2)** system calls, and consequently seldom use the swap device. This may affect the amount of swap space required:

- If most processes executing on the system are locked in memory, smaller amounts of disk space are required for swapping.
- If your system is running a mixture of processes that are locked in memory and processes that are not locked in memory, it may be necessary to allocate larger amounts of swap space to compensate for the amount of available memory that is preallocated to the processes that are locked in memory.

If you have a large amount of paging activity, it is better to allocate multiple swap areas than to allocate one large area. These swap areas can be spread across different disks and should be located in the portions of the disk that are accessed most rapidly. The section that starts on page 6-11 discusses how to determine the amount of paging on the system and steps that can be taken to reduce the amount of paging activity.

Reducing Memory Usage

Changes to kernel tunable parameters are made by using the **sysgen(1M)** command. (See page 6-43 and Chapter 9 for instructions on using **sysgen**.) The kernel reserves blocks of memory for those items and their parameters that are configured under the **sysgen** collection screen. It is possible to improve system performance by reducing the amount of memory reserved for some kernel parameters and eliminating unused collections from the system configuration.

Table 6-1 lists the collections and parameters that are most often used to reduce memory usage. **NBUFK1**, **NPROC**, **NINODE**, **PHYSBSIZE**, and **NCLIST** are actual parameters. All other entries in the "Parameter" column are device names from the **sysgen** collection screen because the particular collection has multiple parameters or because the collection is not required for certain configurations. The "Collection" column is the line item from the **sysgen** collection screen. The table is sorted according to the order of potential savings. The potential saving for each item, in kilobytes, is given in the last column of the table. If this number is preceded by a "<" symbol, the item is a collection whose parameters can be reduced or a kernel parameter whose value can be reduced. If the number is not preceded with a "<" symbol, the item is something that can be deconfigured in its entirety. The number represents a maximum theoretical savings. Note that actual savings may be less than the theoretical number due to memory boundary alignments and differences in the various **REAL/IX** Operating System releases.

Table 6–1. Reducing Memory Usage through Kernel Parameters

Parameter	Collection	Definition	Savings (Kbytes)
NBUF1K	Kernel and Paging Parameters	Number of buffers in buffer cache. May be reduced if not much normal file I/O.	<800
NPROC	Kernel and Paging Parameters	Number of concurrent processes.	<300
NINODE	Kernel and Paging Parameters	Number of inode structures, normally a function of NPROC. May be reduced if few files open.	<200
strbufs	STREAMS Parameters	Streams plus socket and pseudo TTY support. Required for Ethernet support; also may be used for other data communications.	102
PHYSBSIZE	Kernel and Paging Parameters	Size of buffer required by certain drivers. May be eliminated if drivers not present.	64
evt	Events Parameters	Default reserved space for events may be reduced according to need.	<45
kprof	Kernel Profiler	Driver for profiler; may not be required.	32
NCLIST	Kernel and Paging Parameters	Number of structures for TTY system. Normally depends on NPROC. May be reduced if few terminals.	<32
bs	Binary Semaphore Parameters	Default reserved space for binary semaphores may be reduced according to need.	<16
sem	Semaphore Parameters	Deconfigure if System V semaphores are not required.	13
msg	Message Parameters	Deconfigure if System V message passing is not required.	11
shm	Shared Memory Parameters	Deconfigure if System V shared memory is not required. (Probably needed for most realtime applications.)	6

Remember that oftentimes kernel parameters interact with each other in order to build default values. Modifying some of these parameters may cause other default values to change. If you encounter system error or panic messages, you should re-evaluate and/or reconfigure any system parameters you modified.

Maintaining System Efficiency

To maintain a high-performance environment, the following activities are recommended:

- ❑ reorganize file systems regularly
- ❑ schedule jobs for non-prime time whenever possible
- ❑ monitor disk usage regularly
- ❑ repair or disable hardware devices that are defective
- ❑ monitor the **sar** and **systat** reports frequently. This familiarizes you with the information they give, shows you what normal values are for your environment, and keeps you aware of performance trends. The **sar** reports are described later in this chapter.
- ❑ encourage programmers to write efficient application programs

File System Organization

As file systems and areas with temporary work space (primarily */tmp* and */usr/tmp*) are used, they tend to become physically scattered around the disk and I/O becomes less efficient. File systems should be reorganized periodically to restore good file system organization:

- ❑ Use **dcopy(1M)** to remove gaps in the physical file system organization¹. Note that **volcopy(1M)** and **cpio(1M)** do some reorganization of file systems, but not as thoroughly as **dcopy**.
- ❑ Use **fsck -s** to reorganize the free list on S5 file systems.²

↑
Critical realtime processes that have predetermined file space needs can define extents for their files and further improve access time.

¹**dcopy** merges all the extents of a file into a single extent. It does not, however, move any regular data blocks appended to the end of a contiguous file into the contiguous region. In most cases, **dcopy**'s reorganization of extents will not impact the ability of programs to access the extent and may improve the performance of such programs.

²The free list array is used to access free data storage blocks on S5 file systems. The difference between successive block numbers in the free list is the rotational gap selected by the system when the file system is created. For example, a file created on a system with a rotational gap of 24 may consist of blocks 500, 524, and 548. When the file is read, I/O requests are sent to the disk drive to read blocks 500, 524, and 548. As soon as the drive finishes reading block 500 and has started to process the second request, block 524 will be moving over the read/write head just as the drive is ready to read that request. This leads to more efficient I/O operation.

As you change and remove files, efficiency starts to decrease. When several files are created at once, they contend for blocks from the free list. Some of the blocks allocated to the files will be out of sequence and the free list also becomes scattered about the disk. This is less of a problem with F5 file systems, because the architecture uses a bitmask to access the free list and dynamically allocates blocks with the proper gap (based on cylinder size and, optionally, a track size).

Cleaning Up Log Files

The operating system keeps a number of log files during normal operations. Chapter 4 discusses the primary log files and gives instructions for cleaning them up on a regular schedule. If these are not cleaned up regularly, they can eat up a great deal of file system space, which may degrade performance. Moreover, updating large files takes longer than updating small files. So, for example, some process accounting files are updated every time a process starts or exits. When these files grow large, general system performance is degraded.

Directory Organization

Directory organization also affects I/O performance. When a file is removed from a directory, the inode number is nulled out. This leaves an unused slot of 16 bytes in the directory. Reorganizing the file system solves this problem, although just copying the directory is a good "stop-gap" measure.

Directory searches of very large directories are less efficient because of file system indirection. The **find(1)** command can be used to identify directories with large numbers of entries. For example, for a file system that uses 2-Kbyte logical blocks, directories with more than 1280 entries (20480 bytes, or 40 physical blocks that are 512 bytes each) use indirection;¹ they can be identified with the following command:

```
find / -type d -size +40 -print
```

Directories identified as being this large should be broken up into smaller directories. Take care to notify users when moving executable files and data files that are shared among a number of users.

Free Blocks and Inodes

When file systems have few free blocks or free inodes, an excessive amount of system time is devoted to finding available space for writing. As a general guideline, a file system should have at least 10% of its blocks free at the beginning of the day.

If any file system has fewer than 5000 free blocks, your system performance may be notably impaired; if a file system has fewer than 2500 free blocks, users may not be able to create new or expand existing files. If the *root* or */usr* file system (or separate file systems that contain */tmp*, */usr/tmp*, or */spool*) goes below 2500 free blocks, you may be unable to run some processes; when one of these file systems goes below 500 free blocks, accounting shuts down.

To check the free blocks and free inodes available on the file systems, use the **df(1)** command.

¹ File systems that use larger logical blocks can support larger directories without requiring indirect access of the data.

Solving Performance Problems

Poor response time at your terminals or recurring error messages at the console indicate that your system is not performing as well as it could. To define the problem more specifically, use the tools discussed in the section starting on page 6-16, then take appropriate action to resolve it.

Some of the major areas for action are:

- ☐ improve disk utilization
- ☐ improve system usage patterns
- ☐ reallocate memory resources by modifying the tunable parameters

Changes to the tunable parameters and other major adjustments to the system should be made only if your system is performing unsatisfactorily. When making changes to the system, especially when modifying tunable parameters, make small adjustments, then monitor performance carefully for several days. An adjustment that solves one performance problem may create a problem in a different area; as long as the adjustments are gradual, it is unlikely that the ramifications will be severe. Be sure to record all actions taken in the System Log, along with comments on the subsequent performance.

Be aware that the following will degrade system performance in general, especially realtime response:

- ☐ Running the debug kernel
- ☐ Device errors
- ☐ Running kernel profiling or anything that accesses */dev/prf* (which lengthens clock interrupts)
- ☐ Synchronous and raw read/write operations may degrade the performance of other operations on the system, although they are themselves usually faster than non-synchronous read/write operations.

Note, also, that poor response for some processes may be attributable to higher-priority realtime processes that are preventing them from executing. This is one of the hazards on systems running realtime applications; check that processes have appropriate priorities, and consider that some applications may need to be moved to auxiliary systems.

Table 6-2 gives an overview of the problems that can impair system performance, how to identify them, and some possible solutions. Each of these is discussed in more detail in the following sections. If your **sar** reports show values in the unacceptable range but your performance is acceptable, it is not necessary to take corrective action.

Table 6-2. Overview of Performance Problems

Category	Symptoms	Corrective Action
System Paging	swpin/s > 1.00 and %idle > 0 on sar -wu report	Reduce size of buffer cache and other system tables Improve structure of application programs Adjust paging parameters Increase memory configuration Move some users to another system
Disk or Swapping Bottleneck	%wio > 10% on sar -u report %busy > 50% on sar -d report	Use larger logical block sizes on file systems Reorganize file systems Increase buffers Balance disk load Reorganize <i>/tmp</i> and <i>/usr/tmp</i> Create more or larger swap areas Move swap areas to inside of disks Allocate swapping areas across all disks Add more disk controllers
Device Interrupts	mdmin > 0 on sar -y report	Repair modems, ports, terminals, and/or lines
Table Overflows	Warning message on console Table overflows show on sar -v report	Increase table sizes
System Usage	%idle < 10% on sar -u report swpin/s > 1 and %idle = 0 on sar -wu report	Reduce number of users Reschedule large jobs to run at non-prime time Move some users to another system

Check for Excessive Paging

The transfer of pages between memory and the disk is costly in both disk and CPU overhead; excess paging can severely impede system performance. To see whether this is the cause of your degraded performance, look at the following:

- ❑ Page faults (**vflt/s**) greater than 100 on the **sar -p** report.
- ❑ Free memory (**freemem**) less than 100 pages on the **sar -r** and **systat** reports.
- ❑ Pages read from the swap device (**swpin/s**) greater than 1 and idle time (**%idle**) greater than 0 on the **sar -wu** report.
- ❑ CPU utilization on the **sar -u** and **systat(1)** reports. This report identifies the percent of time the CPU was occupied by system overhead (**%sys**), by user processes (**%usr**), waiting for physical I/O (**%wio**), and the percent of time the CPU was idle (**%idle**).

As a general guideline, if %sys is consistently higher than 60, or if %wio is greater than 0 and swpin/s on the `sar -w` report is greater than 1.00, the system may be paging excessively.

Excessive paging happens when you run out of memory, so that pages of executing processes must be paged out to disk memory. To correct the problem, you must free up some memory. Some ways to do this are:

- ❑ Try to reduce the number of processes executing during peak times by scheduling some in non-prime time.
- ❑ Improve the efficiency of application programs, especially their locality of reference, working set, and size. The `gprof(1)` and `prof(1)` commands let you profile the structure of a program for which you have the source code.
- ❑ Check if sticky bits are set. If so, undo some of them.
- ❑ Check if applications are unnecessarily locking processes in memory with `plock(2)`.
- ❑ Check that the system buffer cache is not consuming more memory than necessary. This is an especially critical issue when using large buffer sizes: an excess number of 128-Kbyte buffers can waste much more memory than an excess number of 1-Kbyte buffers would. If the buffer-hit cache ratio¹ is consistently high for a given buffer size (the `bfree(1M)` command shows the cache hit rate for each size of configured buffer), you might be able to decrease the number of buffers of that size² (specified by the kernel tunable parameters discussed in the "Configuring the Buffer Cache" section on page 6-50), thus returning memory space to the pool of available memory.
- ❑ Check that processes are not allocating unnecessarily large shared memory regions or leaving unused shared memory regions allocated when they exit abnormally. Use `iperm(1)` to remove an unused shared memory region from the system; see the *Programmer's Guide* for instructions on writing applications that clean up shared memory regions when they exit on error.
- ❑ Check that the various system tables are not consuming more memory than necessary. If the peak number of entries shown as being used on the `sar -v` report are consistently much less than table size, you can decrease table sizes and regain some memory.
- ❑ If you continue to page excessively, you may need to increase your memory configuration or move some users/applications to a second computer. Remember that the values of the tunable parameters may require adjustment after you add more memory.

¹ The `reache` and `wcache` columns on the `sar -b` report give the buffer-hit cache ratio. This is the composite hit rate for all buffers of all sizes; if your buffer cache contains buffers of various sizes, you can determine the cache hit rate for each size of buffer by examining the `crash` utility's `bfree` report.

² The optimal cache hit rate varies according to the size of the buffers in question. See page 6-42 for guidelines.

Disk or Swapping Bottleneck

Disk input/output can cause a bottleneck in system performance. Compare the **sar -d** reports for all disks. If the value of **%wio** is consistently greater than 10% and the **%busy** is greater than 50% for any disk drive, the system has a disk bottleneck on that drive. If the **%busy** field has radically different values for different disks, your I/O activity is not evenly balanced across disks.

The following list suggests some ways to solve a disk bottleneck.

- ☐ Use the F5 file system architecture for all file systems. The F5 file system uses a bitmask to access the free data blocks in the file system rather than a free list, which improves the time required to identify and allocate free data blocks to a file.
- ☐ Study the **icount(1M)** report (see page 6-40) to see if some file systems should be converted to use larger logical block sizes. Both file system architectures support a range of logical block sizes from 512 bytes to 128 Kbytes.
- ☐ Reorganize the file systems regularly with **dcopy(1M)**.
- ☐ Reorganize **/tmp** and **/usr/tmp**. These should be separate file systems, use the S5 architecture, and be located on separate drives when possible.
- ☐ Increase the number of buffers of an appropriate size if you have sufficient memory. Use the **bfree(1M)** report (see page 6-42) to study the cache hit rate for each buffer size that is configured in the system.
- ☐ Balance the **swap** devices and most active file systems across all disks and controllers on the system.
- ☐ Keep adequate free blocks and inodes in the file systems, especially **root** and **/usr**.
- ☐ Create more swap areas. Study **sar -r** and **systat** reports to determine how much swapping is occurring and whether more swap space is needed.
- ☐ Move most active data to the partitions in the center of the disks. If you have a lot of swapping activity, the swap areas should be put in the middle of the disks; otherwise, file systems that contain frequently used binary files should get the favorable location.
- ☐ Allocate the most heavily used file systems (or swapping areas, if your system has a lot of swapping activity) across disks.
- ☐ Consider adding more memory or moving some users to another system if this is a recurring situation. Additional memory reduces paging traffic and allows an expanded buffer pool, thus reducing the number of user-level reads and writes that need to go out to disk.
- ☐ Consider adding more disk drives and controllers if you continue to have problems.
- ☐ Set sticky bit for appropriate commands, if you have enough memory.

Improving System Usage Patterns

Many performance problems are caused by careless system usage patterns. These include:

1. Less critical or noncritical jobs interfering with critical jobs.
2. Running unnecessary processes that consume system resources.
3. Large, noninteractive jobs that could be rescheduled for non-prime time.
4. Application programs that have an inefficient structure.

Efficient Application Programs

Carelessly constructed application programs can seriously impede system performance. The following checklist summarizes some of these issues for administrators. See the *Programmer's Guide* and the *Kernel Programming Guide* for specific information on writing efficient programs.

- ❑ To identify the system resources being consumed by a particular program, run the command with **timex(1)** (described in the "Measuring Performance" section on page 6-16) on an otherwise quiet system.
- ❑ To check the efficiency of C language programs for which you have source code, use the **gprof** command. See **gprof(1)** for detailed instructions.

The **gprof** command interprets the profile file by reading and correlating the symbol table in the executable object file with the profile file. For each external text symbol, **gprof** prints the percentage of time spent executing between the address of that symbol and the address of the next. It also gives the number of times that function was called, who called it, and the average number of milliseconds per call.



*If you compare successive identical runs with **gprof**, the call counts should all match. The time given for various processes may vary by as much as 20% because of varying cache-hit ratios caused when the cache is shared with other processes and interrupts that cause a context switch in the middle of a clock tick.*

*Even if this program seems to be the only one executing, hidden background or asynchronous processes may blur the data. In rare cases, the measurements may be grossly distorted because the clock ticks that initiate recording of the program counter may disproportionately coincide with some locations in the program. Because of this, you may want to compare a few identical runs with **gprof** when the timing information is critical.*

Any of the following in application programs can have a negative impact on system performance:

- ❑ Poor locality of reference, such as an excessive number of **go to** statements, which cause extra paging.
- ❑ CPU loops waiting on a condition.
- ❑ Relative path names used instead of full path names to identify files.
- ❑ Using too many system calls to transfer information. For instance, 2048 reads of 1 character each are less efficient than 1 read of 2048 characters.
- ❑ Calling library routines rather than system calls.
- ❑ Excessive use of shell programming can degrade system performance.

User-installed kernel code (system calls and drivers) can seriously degrade system performance. See the *Kernel Programming Guide* for more information on performance issues. Note especially:

- ❑ Drivers installed under the CPU affinity compatibility mode, which disables preemption.
- ❑ Large sections of critical kernel code that is protected either by **spl(D3X)** or **spsema(D3X)**. Each **spl** call degrades the system's interrupt latency; note that most **spl** calls can be removed from drivers installed under the major or minor device semaphoring compatibility modes.
- ❑ Using one kernel semaphore to protect a large pool of resources, which leads to contention for the semaphore. The **semstat(1M)** tool that is available under the debug kernel provides information on semaphore contention.

Measuring Performance

The REAL/IX Operating System supports a number of tools that can help measure performance. The tools discussed here are:

kernel profiler	a group of commands that provide a report detailing the percentage of time a process spends executing kernel-level routines
ps(1)	report execution priority and CPU time used by each executing process
sar(1M) and sar(1)	report various statistics on system use of resources
timex(1)	report same information as sar , but for run of individual process
systat(1)	screen-oriented report of current system activities
icount(1M)	report number of disk writes by size for each file system
bfree(1M)	report access statistics for each size buffer configured on the system

The **strstat(1M)** tool, which is part of the unsupported tools package, reports utilization statistics for all sizes of STREAMS buffers. Customers who have configured a communications controller and are running Internet protocols should monitor this report to see if an appropriate number of STREAMS buffers are configured.

These tools do not provide absolute answers, but are useful for spotting general performance trends and for identifying the source of the problem when system performance is unacceptable. In addition, the **errpt(1M)** tool and the */usr/adm/utbuf* file that are used in trouble analysis can be used to identify system errors that may be degrading performance.

Note that **sar** and **timex** are functional only if the **sadc** data collection tool (see **sar(1M)**) is executing. This tool is executed with the **sa1** command through **cron(1)**; as released, the **sa1** lines in **cron** are commented out.

These performance tools were designed for time-sharing computers on which all executing processes should get an equitable share of system resources when the system is properly tuned and configured. When executing realtime processes on the system, the system should be configured to provide the performance required by the realtime processes, which in some cases will severely degrade the performance of non-realtime processes.

Kernel Profiler

The kernel profiler is a group of commands used to facilitate an activity study of the operating system. The activity study involves detailing the percentage of time that a process spends executing kernel-level routines. The study of the resultant data may indicate areas where modifying tunable parameters could improve system performance, or where inefficient kernel code (i.e. device drivers) could be re-written to improve system performance. The kernel profiler uses */dev/prf* which reduces system performance, therefore the kernel profiler should not be set up to run continuously in a production environment.

Note that the kernel profiler commands are discussed in the order in which they should be executed. For command syntax, refer to the **profiler** manual page or the individual command's manual page.

The **prfld(1M)** command is used to initialize the profiler. You must specify the executable kernel to be tested if it is other than */realix*.

After the profiler is initialized you should execute the **prfstat(1M)** command. **prfstat** has two options: *on* or *off*. When the *on* option is specified, status posting by the profiler is enabled. When the *off* option is specified, the profiler disables status posting.

Actual collection of the data is accomplished by the **prfsnap(1M)** or **prfdc(1M)** commands. **prfsnap** provides a one-time snapshot of the system activity and writes that data to a specified. **prfdc** allows you to specify a snapshot interval to the profiler and a turnoff time for the data collection. For example, the command string **prfdc filename 60 8** would execute a system activity snapshot every 60 seconds, post that data to the file called *filename*, and would continue to execute until 8:00 AM.

Once the profiler is initialized and is permitted to post status you would execute the performance benchmark or execute the application program that appears to cause system performance problems. By executing the **prfsnap** or **prfdc** commands during the performance run, the data is collected for later review. At the completion of the benchmark or performance testing you would turn off status reporting and format the profiler data by using the **prfpr(1M)** command.

prfpr formats the data from the specified file and outputs the results to the standard output device. Figure 6-1 shows an example of the **prfpr** output.

```
06/12/91 07:51:35
06/12/91 07:52:35

Samples      6412
idleloop     83.8
_getblk      1.4
_namei       3.5
user         1.4
other        6.4
```

Figure 6-1. Profiler Statistics

This example of the **prfpr** output indicates that the time period between samples is 1 minute. During this minute the system was sampled 6412 times. During the sample period the CPUs spent 83.8% of their time idling, 1.4 % of their time in the `_getblk`, and 3.5% of their time in the `_namei` kernel routines. Also, 1.4% and 6.4% of CPU time was spent in user space and in other routines, respectively.

The percentage of time spent indicated by `user` indicates the amount of time processes were executing in user space. The symbol tables for user space are inaccessible to the kernel so all "user" time is reported as a lump sum.

The percentage of other time will depend on an option of the **prfpr** command. The **prfpr** command allows you to specify a cutoff percentage below which you are not interested in reviewing. For example, the command string **prfpr filename 10** would format a report using *filename* and would report only those processes using more than 10% of CPU time. The processes using less than 10% of CPU time are reported as `other`. The cutoff percentage defaults to 1, so the example's 6.4% for `other` indicates the sum of all processes using less than 1% of CPU time.

When evaluating the report, you would examine the reasons for calling and the methods used to call the routine that uses the highest percentage of CPU time. If the routine is a call from your code, you may choose to restructure your code so that your calls are more efficient. If the routine with the highest percentage of CPU time is one that you've written, then you may wish to rewrite code to improve the performance of your routine.

ps

The **ps(1)** command reports information about active processes.¹ The command gives a "snapshot" picture of what is going on, which is useful when you are trying to identify what processes are loading the system.

The **ps -efl | grep -v getty** report provides full information on all executing processes, including the CPU time consumed and the execute priority. The **getty** processes are processes associated with idle terminals. They consume minimal system resources until a user accesses the terminal device, and so are seldom worth looking at for performance management.

- By looking at the TIME (minutes and seconds of CPU time used by processes) and STIME (time when process started) columns, you can identify processes that are using an unreasonable amount of system resources.

Such processes could be programs with bugs or programs running on bad data. You should check with the user to see if this is the case. If so, either the owner or the superuser can terminate the process with the **kill -9 pid** command. If the process is legitimately hogging resources, you might consider running it with **at** during off hours.

- By looking at the PRI column, you can see the process priorities. Priorities in the range 0 to 127 are realtime priorities that are not adjusted by the operating system. A higher priority job may be preventing a lower priority job from running; for instance, a process running at priority 25 will, if active, prevent a process at priority 40 from running. Priorities 254 and 255 are non-migrating priorities for processes that should execute only when all other processes are inactive.

Assuming you have a terminal logged in at a higher priority than any application program (which is a good idea), you can change the executing priorities of realtime processes with the **setpri(1R)** command.² A run-away process executing at a high priority should be killed or have its priority lowered if there are other critical realtime processes that are being prevented from running.

- By looking at the WCHAN column, you can determine which processes are blocked and the address of the semaphore on which they are blocked. If this field is blank, the process is not blocked. By using the **crash** utility's **sema** command, you can locate the closest symbol whose value is less than or equal to the WCHAN value. Note that some WCHAN values can be identified with the **nm** listing. However, large WCHAN values are usually in the buffer area, which is not included in the **nm** listing.

The **crash** utility's **proc** command displays similar information.

¹This same information is reported on the **crash** utility's **proc** output.

²Note that the default priorities for a number of system processes are tunable. This is discussed on page 6-59.

sar

Internal activity is measured by a number of counters contained in the operating system kernel. Each time an operation is performed, an associated counter is incremented. The **sadp(1M)** process is run periodically by **cron**,¹ to sample the value of the counters and store the results in files in the */usr/adm/sa* directory. The files in this directory have names of the form **sa nn** , where nn represents the day of the calendar month (for instance, the data collected on the fifteenth of any month is stored in the **sa15** file, overwriting information from the previous month). When **sar** is run, it formats the information gathered every 15 minutes since midnight, and shows the average value for all samplings.

Table 6-3 lists the **sar** options used to specify the system activity to be reported.

Table 6-3. sar Options

Option	Reports on:
sar -A	composite of all sar reports
sar -a	file access operations
sar -b	system buffer activity
sar -B	binary semaphore activity
sar -c	system call activity
sar -C	connected interrupt activity
sar -d	block disk devices
sar -E	common event mechanism activity
sar -l	asynchronous I/O activity
sar -m	user-level semaphores, shared memory, and message activity
sar -p	page fault activity
sar -q	run queue activity
sar -r	amount of main memory and disk memory free
sar -T	process interval timer activity
sar -u	CPU utilization
sar -v	status of process, inode, file, file/record lock, and stream-head tables
sar -w	swapping and switching activity
sar -y	terminal device activities

Values you receive may be quite different from values in the examples, depending on your application or benchmark. When tuning your system, it is recommended that you use a benchmark or have the system under normal load for your environment to allow you to tune directly toward your specific application.

¹By default, the **sadp** process is turned off. To turn it on, modify the */usr/spool/cron/crontabs/sys* file to remove the **#** character at the beginning of the **sa1** lines.

sar -a

The **sar -a** option reports the use of file access operations. The operating system routines reported are as follows:

iget/s	Number of files located by inode entry per second.
namei/s	Number of file system path searches per second. namei calls iget , so iget/s is always larger than namei/s .
dirbk/s	Number of directory block reads issued per second.

An example of **sar -a** output follows:

12:41:40	iget/s	namei/s	dirbk/s
12:44:10	18	13	14
12:47:40	90	40	112
12:50:10	11	63	37
:	:	:	:
Average	10	14	16

The larger the values reported, the more time the kernel is spending to access user files. This indicates how heavily programs and applications are using the file system(s). If you suspect that poor file system organization is affecting performance, refer to the "Solving Performance Problems" section on page 6-10 for appropriate corrective actions.

In general, if the ratio of **iget/s** to **namei/s** is greater than 5 and **namei/s** is greater than 30, the file access of operations on your system are inefficient. However, if system performance is acceptable with these values, you need not make any adjustments to the system. You should, however, save the current results for comparison at a later time when system performance becomes unacceptable.

sar -b

The **sar -b** option reports the following buffer activity.

bread/s	Average number of physical blocks into the system buffers from the disk (or other block devices) per second.
lread/s	Average number of logical blocks read from system buffers per second.
%rcache	Fraction of logical reads found in buffer cache (100% minus the ratio of bread/s to lreads/s).
bwrit/s	Average number of physical blocks written from the system buffers to disk (or other block devices) per second.
lwrit/s	Average number of logical blocks written to system buffers per second.
%wcache	Fraction of logical writes found in buffer cache (100% minus the ratio of bwrit/s to lwrit/s).
pread/s	Average number of physical read requests (including asynchronous I/O) per second.
pwrit/s	Average number of physical write requests (including asynchronous I/O) per second.

The entries that you should be most interested in are the cache hit ratios **%rcache** and **%wcache** which measure the effectiveness of system buffering. If you are using all 1-Kbyte buffers and **%rcache** falls below 90, or **%wcache** falls below 65, it may be possible to improve performance by increasing the number of buffers. If you are using different buffer sizes, use the **crash** utility's **bfree** command to study the cache hit rate for each buffer size. See page 6-42 for information on using **bfree** and recommended cache hit rates for different buffer sizes.

Note that this reports the cache hit rate for all buffers in the buffer cache. If you have configured the buffer cache to use buffers of different sizes, this report does not tell you the cache hit rate for different sized buffers. See the end of this section for information on using the **crash** utility's **bfree** command to see similar information, displayed separately for each configured buffer size.

An example of **sar -b** output follows:

16:30:57	bread/s	lread/s	%rcache	bwrit/s	lwrit/s	%wcache	pread/s	pwrit/s
17:00:07	2	17	87	1	3	71	0	0
17:30:17	18	141	88	6	35	84	0	0
18:00:27	8	305	98	10	90	89	0	0
:	:	:	:	:	:	:	:	:
Average	10	91	90	3	14	79	1	1

This example shows that the buffers are not causing bottlenecks because all data is within acceptable limits.

sar -B

The **sar -B** option reports on the activities of the binary semaphore mechanism in the following categories:

bsemas	Number of unique binary semaphores currently initialized (BSCNTSYS).
bsemaprocs	Number of processes currently using binary semaphores (BSMAXPRC).
maxbsemas/proc	Number of active bsget operations for the biggest user (BSCNTPRC).
bsgets	Number of active bsget operations currently initialized. This number is normally larger than the value of bsemas , because three processes using one binary semaphore to coordinate access of a shared resource will cause the value of bsgets to be 3, while bsemas will be 1.

An example of **sar -B** follows:

	bsemas	bsemaprocs	maxbsemas/proc	bsgets
09:20:08				
09:40:12	3	8	3	21
10:00:03	3	5	3	13
10:20:07	0	0	0	0
:	:	:	:	:

To interpret this report, compare the values for each field to the maximums set by the related tunable parameter (see page 6-66). For example, the default value for BSMAXPRC on the released system is 20; if the value shown is representative and you do not anticipate installing additional programs that use the binary semaphore mechanism, you can regain some memory by setting BSMAXPRC to a lower number.

If no applications on the system use the binary semaphore mechanism, you can regain some memory by tuning all the binary semaphore tunable parameters (see page 6-66) to a value of 1. On development systems, you may want to tune the binary semaphores parameters to low values but greater than 1, to allow developers to execute small test programs that use binary semaphores.

sar -c

The **-c** option reports system calls in the following categories:

scall/s	Number of all types of system calls per second.
sread/s	Number of read system calls per second.
swrit/s	Number of write system calls per second.
fork/s	Number of fork system calls per second. This number will increase if shell scripts are running.
exec/s	Number of exec system calls per second.
rchar/s	Number of characters (bytes) transferred by read system calls per second.
wchar/s	Number of characters (bytes) transferred by write system calls per second.

Typically, reads plus writes account for about half of the total system calls, although this varies greatly with the activities that are being performed by the system.

An example of **sar -c** output follows:

	scall/s	sread/s	swrit/s	fork/s	exec/s	rchar/s	wchar/s
18:33:04							
19:03:35	104	33	15	1.29	3.77	23077	4452
19:34:05	500	53	50	7.50	10.00	260095	2393
20:03:35	136	48	8	0.28	0.68	12846	4549
:	:	:	:	:	:	:	:
Average	104	33	5	0.98	0.95	15237	4109

If **scall/s** is greater than 300 over an extended period of time, **%sys** on the **sar -u** report is probably also large. This may indicate that application programs are causing system degradation.

sar -C

The **sar -C** option reports on the activity of the connected interrupt mechanism in the following categories:

cintrpts	Number of connected interrupts active system wide (CIMAXSYS).
ciprocs	Number of processes using connected interrupts (CIMAXPROC).
maxcis/proc	Number of connected interrupts being used by the largest user (CICNTPROC).

An example of **sar -C** output follows:

	cintrpts	ciprocs	maxcis/proc
12:41:40			
12:44:10	5	5	1
12:47:40	5	5	1
12:50:10	5	5	1
:	:	:	:

To interpret this report, compare the reported values to the values of the associated tunable parameters (see page 6-72) to see if values could be tuned down to regain some memory. A report like the one shown above reflects a system that has five processes that use connected interrupts executing, each of which is using one connected interrupt. If this load is never exceeded, the values of CIMAXSYS and CIMAXPROC could be tuned down to 5 or 6 (from the release default of 25) and the value of CICNTPROC could be tuned down to 1 or 2 (from the release default of 5).

If no applications on the system use the connected interrupt mechanism, you can regain some memory by tuning all the connected interrupt tunable parameters (see page 6-72) to a value of 1. On development systems, you may want to tune the connected interrupt parameters to low values but greater than 1, to allow developers to execute small test programs that use connected interrupts.

sar -d

The **sar -d** option reports the activity of block devices.

device	Name of the block device(s) that sar is monitoring.
%busy	Percent of time the device was servicing a transfer request.
avque	The average number of requests outstanding during the period of time (measured only when the queue is occupied).
r+w/s	Number of read and write transfers to the device per second.
blks/s	Number of 512 byte blocks transferred to the device per second.
await	Average time in milliseconds that transfer requests wait idly in the queue (measured only when the queue is occupied).
avserv	Average time in milliseconds for a transfer request to be completed by the device (for disks this includes seek, rotational latency, and data transfer times).

An example of **sar -d** follows:

	device	%busy	avque	r+w/s	blks/s	await	avserv
08:00:01	a147-0	2	1.8	1	40	14.7	18.0
08:15:01	a147-2	2	2.2	1	35	17.7	14.6
18:30:00	a147-0	25	6.5	16	521	84.8	15.4
	a147-2	2	2.3	1	45	15.6	12.4
18:45:03	a147-0	14	1.9	3	82	12.9	13.7
	a147-2	4	1.9	3	82	12.9	13.7
:	:	:	:	:	:	:	:
Average	a147-0	3	3.3	2	53	34.9	15.2
	a147-2	11	2.3	6	201	22.2	17.1

Queue lengths and wait times are measured while the queue had something in it. If **%busy** is small, large queues and service times probably represent the periodic **sync** efforts by the system to ensure that altered blocks are written to disk in a timely fashion.

Note that these values represent the activity when the data collector ran. If the data collector is running every 15 or 30 minutes and the disk load is variable, individual numbers can be misleading. The averages over a period of days, however, can indicate how well the load is balanced between disks on the system.

sar -E

The **sar -E** option reports the following information relevant to the common event notification mechanism:

events	Number of events being used system-wide (EVTMAXQUE).
evtprocs	Number of processes with event identifiers initialized (EVTMAXPRC).
maxevts/proc	Number of events posted for the process using the most events (EVCNTPRC). This reflects the number of event identifiers actually being used, not the pool of identifiers allocated for the process when the first evget(2) call is issued.

An example of **sar -E** output follows:

12:41:40	events	evtprocs	maxevts/proc
12:44:10			
12:47:40			
12:50:10			
:	:	:	:

To interpret this report, compare the reported values with the current values of the associated tunable parameters (see page 6-72). The values reflect the activity for all types of events (connected interrupts, process interval timers, resident memory violations, user-defined event subsystems, and so forth). In many cases, the values of these parameters can be set to lower numbers to regain some memory.

If no applications on the system use the common event notification mechanism, you can regain some memory by tuning all the common event tunable parameters (see page 6-66) to a value of 1. On development systems, you may want to tune the common event parameters to low values but greater than 1, to allow developers to execute small test programs that use the common event notification mechanism.

sar -I

The **sar -I** option reports the following information relevant to asynchronous I/O operations:

aioblks	Number of asynchronous I/O structures initialized system-wide (AIOMAXAIO).
aioprocs	Number of processes with asynchronous I/O structures initialized (AIOMAXPRC).
maxaioblks/proc	Maximum number of asynchronous I/O structures being used by any process that is currently executing (AIOCNTPRC).

An example of **sar -I** output follows:

12:41:40	aioblks	aioprocs	maxaioblks/proc
12:44:10	61	15	15
12:47:40	55	12	13
12:50:10	64	19	14
:	:	:	:

The **sar -I** report provides information about the asynchronous I/O activity that can be used to determine appropriate values for the asynchronous I/O tunable parameters (see page 6-71). For the sample shown above, if the system is running with the released values, you might want to tune the values of AIOMAXAIO (default 64) and AIOCNTPRC (default 16) up, since the current usage is pushing the maximums and values could be higher in the period between **sadc** samplings than what is reflected here.

If no applications on the system use the asynchronous I/O mechanism, you can regain some memory by tuning all the asynchronous I/O tunable parameters (see page 6-71) to a value of 1. On development systems, you may want to tune the asynchronous I/O parameters to low values but greater than 1, to allow developers to execute small test programs that use asynchronous I/O operations.

sar -m

The **sar -m** option reports on the interprocess communication activities. Message and semaphore calls are reported as follows:

msg/s	Number of message operations (sends and receives) per second.
sema/s	Number of user-level semaphore operations per second. This does not include binary semaphore activities.

An example of **sar -m** output follows:

13:50:58	msg/s	sema/s
13:53:23	12.3	29.2
13:56:43	15.2	28.6
13:56:43	11.8	26.8
14:48:48	11.8	26.8
14:51:02	17.6	24.7
14:54:22	18.4	23.1
14:57:13	16.9	22.5
15:00:00	15.8	27.4
15:03:35	15.3	22.2
:	:	:
Average	15.4	25.5

If you are not running application programs that use messages and semaphores, these figures will all be equal to zero. If you do not require this feature, you can set the tunable parameters to minimum values and save some memory.

If you are using these interprocess communications facilities and either **msg/s** or **sem/s** is greater than 100, the application is not using the system efficiently. The "Tunable Parameters" section on page 6-43 discusses the tunable parameters that can be adjusted for these drivers.

sar -p

The **-p** option reports paging activity. The following page rates are recorded:

vflt/s	Number of address translation page faults per second (valid page not present in memory).
pflt/s	Number of page faults from "copy-on-write" operations. pflt/s may occasionally result from illegal accesses to a page.
pgfil/s	Number of vflts per second satisfied by a page-in from the file system. (Each pgfil causes two lreads ; see sar -b).
rclm/s	Number of valid pages per second that the system has reclaimed (added to list of free pages).

An example of **sar -p** output follows:

08:00:00	vflt/s	pflt/s	pgfil/s	rclm/s
08:15:01	1.13	1.42	0.09	0.00
08:30:00	15.95	10.00	0.36	0.00
08:45:03	4.04	2.72	0.48	0.00
09:00:01	3.31	3.77	0.60	0.00
:	:	:	:	:
Average	4.54	9.37	0.10	0.00

High values (over 100) for **vflt/s** can indicate that:

- ☐ application programs are not efficient for a paging system (poor locality of reference)
- ☐ paging tunable parameters may need adjustment
- ☐ the memory configuration is inadequate for the system load

sar -q

The **sar -q** option reports the average run queue length while the queue is occupied and the percentage of time that it is occupied.

runq-sz	Average number of processes available in memory and ready to run; typically, this should be less than 2. Consistently higher values mean you are CPU-bound.
%runocc	The percentage of time that the run queue is occupied; the larger this value is, the better. Note that the idle task is excluded from this count, since it is always on the run queue at priority 255, to run if nothing else is runnable.
swpq-sz	unused
%swpocc	unused

An example of **sar -q** follows:

11:00:56	runq-sz	%runocc	swpq-sz	%swpocc
11:01:07	2.1	39		
11:01:17	1.5	31		
11:01:27	1.6	37		
:	:	:		
Average	1.4	45		

If **%runocc** is greater than 90 and **runq-sz** is greater than 2, the CPU is heavily loaded and response is degraded. In this case, additional CPU capacity may be required to obtain acceptable system response.

This report can also be used to identify peak usage times and times when system usage is low. Scheduling jobs during periods of low system usage will improve the perceived performance of the system during peak usage times.

sar -r

The **sar -r** option records the number of memory pages and *swap* file disk blocks that are currently unused. The following are recorded:

freemem	Average number of 8-Kbyte pages of memory available to user processes over the intervals sampled by the command.
freeswap	Number of disk blocks of all sizes available for process paging.

An example of **sar -r** output follows:

07:45:00	freemem	freeswap
08:00:01	400	34656
08:15:01	384	34720
08:30:00	351	34784
08:45:03	330	34784
:	:	:
Average	388	26848

This report should be interpreted in relationship to the amount of available memory on your system (displayed whenever you boot the system). If **freemem** is consistently more than 20% of available memory, you can afford to set the sticky bit on some frequently-run programs. If **freemem** decreases and the **vflt/s** field on the **sar -p** report increases dramatically, the sticky bit is set on too many processes.

The **systat(1)** utility displays similar information.

Anytime **freemem** is less than 10% of available memory, the **vhand** daemon pages processes out of main memory. If **freemem** is consistently less than 10% of available memory, you can improve system performance by making more memory available. This can be accomplished by modifying tunable parameters to reduce the size of system tables, rescheduling jobs for off-hours, and unsetting the sticky bit on some processes. If the performance of your system is substandard, you may need to add more memory to your configuration.

Note that free memory is mostly a performance concern for general purpose and development environments; on machines dedicated to one realtime application, you may have critical programs resident in memory and require smaller amounts of free memory.

sar -T

The **sar -T** option reports on the activity of structures associated with the process interval timer mechanism:

timers	Number of process interval timers in use system-wide (ITIMAXSYS).
timprocs	Number of processes with process interval timers initialized (ITIMAXPROC).
maxtims/proc	Number of process interval timers being used by the process that is the biggest user of process interval timers (ITICNTPROC).

An example of **sar -T** follows:

	timers	timprocs	maxtims/proc
12:41:40			
12:44:10	2	2	3
12:47:40	2	2	3
12:50:10	4	3	4
:	:	:	:

This report should be interpreted relative to the current values of the associated tunable parameters (see page 6-73). For instance, the default value of ITIMAXSYS on the released system is 55, and the example shows that the largest number shown in the **timers** column is 4. If this reflects the normal (and maximum) usage for your system and you do not anticipate adding programs to the system that use process interval timers, the value of ITIMAXSYS could be tuned down to 5 or 6 to free up memory. The default value of ITICNTPROC is 5; before executing a process that initializes 8 simultaneous process interval timers, the value of ITICNTPROC must be increased to at least 8.

If no applications on the system use process interval timers, you can regain some memory by tuning all the process interval timer tunable parameters to a value of 1. On development systems, you may want to tune the process interval timer parameters to low values but greater than 1, to allow developers to execute small test programs that use process interval timers.

sar -u

The CPU utilization is listed by **sar -u** (default). At any given moment the processor will be either busy or idle. When busy, the processor will be in either user or system mode. When idle, the processor will either be waiting for input/output completion or it has no work to do. The **-u** option of **sar** lists the percent of time that the processor is in system mode (**%sys**), user mode (**%usr**), waiting for input/output completion (**%wio**),¹ and idle time (**%idle**).

An example of **sar -u** follows:

09:20:08	%usr	%sys	%wio	%idle
09:40:12	63	8	26	3
10:00:03	73	11	17	0
10:20:07	2	2	8	89
:	:	:	:	:
Average	55	30	10	5

The numbers shown on the "Average" line are good averages for most systems, although these numbers might be different for special applications without anything being abnormal. Some general guidelines for interpreting this report are:

- ❑ In typical time-sharing environments, **%usr** and **%sys** are both between 40 and 60%; this shows that user and system programs are each getting about half of the CPU time. If **%sys** is consistently greater than 60, your computer is running system bound. Check programs for efficiency and distribute the system load to off-peak hours.
- ❑ Any time **%wio** is over 0, your CPU is sitting idle waiting for disk I/O. If this is consistently over 30, you have a disk bottleneck. If performance is a problem, try increasing the system buffers, reorganizing and relocating the files on disks, or adding disk hardware.
- ❑ Time spent waiting for memory is attributed to **%idle**; if **%idle** is low and response time is degraded, you may have memory constraints. Times when **%idle** is over 0 (especially if it is 50 or more) are good times to schedule work, since computer resources are available at this time.

The **systat(1)** utility reports the current value of these statistics.

¹Note that **%wio** represents times that the I/O transfer is blocked with **lowait(D3X)**, which are most frequently disk updates done through the file system. It does not represent waits on other devices such as terminals and network devices.

sar -v

The **-v** option reports the status of process, inode, file, shared memory record, and file/record lock tables. The tunable parameters that determine the table sizes are given in parentheses after the definition. From this report you know when the sizes of these system tables need to be modified.

proc-sz	Number of process table entries being used/allocated in the kernel (NPROC).
inod-sz	Number of inode table entries being used/allocated in the kernel (NINODE).
file-sz	Number of file table entries being used/allocated in the kernel (NFILE).
ov	Number of times an overflow occurred. (One column for each of the above three items.)
lock-sz	The number of file/record lock table entries being used/allocated in the kernel (FLCKREC).
fhdr-sz	Currently unused.
stream-sz	Number of "stream-head" structures being used/allocated in the kernel (NSTREAM).

Each number has two parts separated by a slash: the first number is the average number of entries used, the second number is the number of table entries available. An example of **sar -v** follows:

	proc-sz	ov	inod-sz	ov	file-sz	ov	lock-sz	fhdr-sz	stream-sz
07:45:00									
08:00:01	42/128	0	94/512	0	61/512	0	0/ 50	0/ 0	20/120
08:15:01	43/128	0	101/512	0	69/512	0	0/ 50	0/ 0	36/120
08:30:00	42/128	0	104/512	0	67/512	0	0/ 50	0/ 0	38/120
08:45:00	53/128	0	133/512	0	90/512	0	0/ 50	0/ 0	39/120
:	:	:	:	:	:	:	:	:	:

Note that this report does not give averages. This example shows that all tables are large enough to have no overflows. Sizes could be reduced to save main memory space if peaks are consistently less than table size.

Table sizes define the robustness of the system. Larger tables mean a more robust system, but consume more memory. In determining the appropriate sizes for tables, you must balance considerations of robustness for free memory.

The **stream-sz** information should be used in conjunction with **strstat(1M)**, which reports utilization statistics for each size of STREAMS buffer. **strstat(1M)** is part of the unsupported tools package.

sar -w

The **-w** option reports paging activity. The following are some target values and observations.

swpin/s	Number of transfers into memory per second.
bswin/s	Number of 512-byte-block units (blocks) transferred for swap-ins (including initial loading of some programs) per second.
swpot/s	Number of transfers from memory to the disk swap area per second. If greater than 1, memory may need to be increased or buffers decreased.
bswot/s	Number of blocks transferred for swap-outs per second.
pswch/s	Process switches per second.

An example of **sar -w** output follows:

	swpin/s	bswin/s	swpot/s	bswot/s	pswch/s
07:45:00					
08:00:01	0.0	0.0	0.0	0.0	37
08:15:01	0.1	0.0	0.0	0.0	7
08:30:00	0.0	0.0	0.0	0.0	163
:	:	:	:	:	:
Average	0.0	0.0	0.0	0.0	38

This example shows that there is sufficient memory for the currently active users, since very little paging is occurring.

sar -y

The **-y** option monitors terminal device activities. You can use this report to determine if there are any bad lines. Activities recorded are defined as follows:

rawch/s	Input characters (raw queue) per second.
canch/s	Input characters processed by canon (canonical queue) per second.
outch/s	Output characters (output queue) per second.
rcvin/s	Receiver hardware interrupts per second.
xmtin/s	Transmitter hardware interrupts per second.
mdmin/s	Modem interrupts per second.

The number of modem interrupts per second (**mdmin/s**) should be close to 0, and the receive and transmit interrupts per second (**xmtin/s** and **rcvin/s**) should be less than or equal to the number of incoming or outgoing characters, respectively. If this is not the case, check for bad lines.

An example of **sar -y** output follows:

	rawch/s	canch/s	outch/s	rcvin/s	xmtin/s	mdmin/s
07:45:00						
08:00:01	0	0	0	0	0	0
08:15:01	1	0	34	1	34	0
08:30:00	1	0	13	1	13	0
:	:	:	:	:	:	:
Average	0	0	3	0	3	0

- ❑ The third column of the upper portion of the display shows the percentage of the CPU utilized by user and kernel activities (excluding I/O and memory waits) for the last 1, 5, and 15 minutes.
- ❑ The upper left column displays memory allocation statistics, broken into five categories. The numbers represent clicks, or 8-Kbyte pages of memory.

maxmem Amount of memory available after the kernel is loaded (but not initialized).

freemem Amount of memory available to user processes, after all kernel data structures have been allocated. This number will increase and decrease as you modify tunable parameters to modify the size of various tables. The **sar -r** report also reports **freemem**, with the same meaning as here. Anytime **freemem** is less than 10% of available memory (the sum of **availrmem** and **availsmem**), the **vhand** daemon pages out processes out of main memory. An asterisk (*) appears anytime **vhand** is active.

availrmem Amount of resident memory available. This is the number of pages available for processes locked in memory with **plock(2)** and **resident(2)**, plus the number of pages required for page tables.

availsmem Amount of virtual memory available (**availrmem** plus all disk swap space).

free res Free memory available to resident processes. This is the value of **availrmem** minus the value of **MINPAGMEM**, the minimum number of pages that must be kept available for paging.

Use **systat** to view ongoing activities to give you a better understanding of the summary statistics shown in the **sar** report.

timex

The **sar** reports reflect information gathered at regular times on the system. The **timex(1)** command times a command and reports the system activities that occurred just during the time the command was executing. If no other programs are running, then **timex** can give a good idea of which resources a specific command uses during its execution. System consumption can be collected for each application program and used for tuning the heavily loaded resources.

timex can be used in the following way:

```
$ timex -s application program
```

Your application program will operate normally. When you finish and exit, the **timex** result will be printed on your screen, giving you a clear picture of system resources used by your program.

icount(1M)

The **icount(1M)**¹ command reports the number of files of each size range closed, by file system, since the system was last booted. File systems are listed by special device file; use the **df** command to get a listing of file systems in order to relate to this listing. A sample **icount** report is:

DEVICE NAME	1->1K	->2k	->4k	->8k	->16k	->32k	->64k	>128k	>256k	->inf
/dev/dsk/m327_00s0	4556	17K	6010	63	182	137	96	45	464	0
/dev/dsk/m327_00s1	9043	2255	3313	244	170	143	1776	14	17	0
/dev/dsk/m327_20s0	10k	2812	2302	1456	955	427	388	286	946	2
/dev/dsk/m327_20s1	2413	1407	808	866	790	1047	692	292	20	0
/dev/dsk/m327_20s2	10k	4153	1254	1920	2629	1054	581	441	369	54
	:	:	:	:	:	:	:	:	:	:
TOTALS:	45k	36k	17k	7323	6486	3591	3820	1185	2107	56

Each column lists the number of regular files closed in that size range (for instance, the "->4k" column lists the number of close operations on files between 2048 bytes and 4096 bytes).² This information, gathered over time, can help determine which logical block size is appropriate for each file system. Note that this report tells the number of files that were closed, not the number of buffered I/O operations. For instance, on a 1-Kbyte logical block file system, each write of a 128-Kbyte file may entail 128 I/O operations.³ Performance is affected not only by the number of I/O operations required to read or write the file, but also (except for data in file extents) by the indirect data address blocks involved in accessing files that are larger than 10 logical blocks. See *Concepts and Characteristics* for a discussion of indirect data address blocks.

The values could be plotted to form a bell curve; if you can afford the memory and the disk space, use the peak of the bell as the logical block size; if memory and disk space are tight, it may make sense to select a logical block size that is half the peak (so, if the peak is at 4 Kbytes, use 2-Kbyte logical buffers). For some applications, you might find that the peak is broad (for example, the values in the 2-Kbyte, 4-Kbyte, and 8-Kbyte columns may all be high and roughly equal). In this case, you might want to favor a larger logical block size.

Different applications have different file access attributes. One way to get maximum performance without wasting inordinate amounts of disk space and memory is to make separate file systems for separate applications. For instance, say you have a file system shared by an application that uses mostly small files and an application that uses mostly larger files. Using a larger logical block size will improve performance on the application that uses the larger files, but every I/O operation that uses a small file wastes disk space and memory. If the two applications are separated onto separate file systems, you can create each file system with the appropriate logical block size for the application.

¹The **crash** utility's **icount** command reports the same information as **icount(1M)**, except that it reports this information by mount table entry.

²To pack as much information as possible per line, overly long values are scaled down using the following notation: k=10³; m=10⁶; g=10⁹; t=10¹². For example, 40k means 40,000.

³The multi-block I/O capabilities can reduce the number of I/O operations by reading or writing several blocks at a time.

You will need to allocate kernel buffers at least as large as each logical block size you use. The numbers in the TOTALS row at the bottom, combined with your knowledge of the size and activity on the file systems created with each logical block size, should provide some guidance on selecting an appropriate initial number of buffers of each size; you can then use the **bfree(1M)** command as discussed below to monitor buffer performance and determine whether the buffer cache configuration needs to be adjusted.

The numbers in the sample are typical for a development environment. Assuming that the numbers in the report above are consistent for the system, the 1-Kbyte logical block size is appropriate for virtually all the file systems. If you can afford the memory and disk space, you might improve performance by making File System 0 into a 2-Kbyte logical block file system.

bfree(1M)

The **sar -b** command summarizes the cache hit rate for the entire buffer cache. As long as the system is configured to use a buffer cache of buffers that are all the same size, this information is adequate to determine whether the size of the buffer cache is appropriate. The **sar -b** report is accurate for buffer caches that contain various sizes of buffers, but it does not identify if you have too few or too many of a certain size of buffer. The **bfree(1M)**¹ command reports the same information as **sar -b**, but broken down for each buffer size configured in the system. It also reports the number of buffers of that size configured. So, for a system that uses 1-Kbyte and 2-Kbyte buffers, a sample report is:

SIZE	FCOUNT	BREAD	BWRITE	LREAD	LWRITE	RCACHE	WCACHE
1k	536	480701	160462	2588659	771647	82%	80%
2k	100	1437	515	12070	3849	89%	87%

The FCOUNT column shows the number of buffers of this size configured. The other columns show, in order, the number of block read and block write operations, the number of logical read and logical write operations, and the read and write cache hit ratios. See **sar -b** for information on how to interpret these numbers. Note two significant differences between this report and **sar -b**:

- **sar -b** reports the average number per second for each type of I/O operations. **bfree** reports the total number of each type of I/O operation since the system was booted.
- **bfree** does not report on physical read/write operations, since on modern devices, these usually use on-board memory for buffering and there is little one can do when tuning the system to modify the performance impact of such operations.

The RCACHE and WCACHE columns can be used to determine whether the size of the buffer cache needs to be adjusted. In general, the higher the cache-hit rate, the better performance you get on I/O through the buffer cache. However, because the larger buffers consume more memory, it is generally best to allow lower cache-hit rates on the larger buffers. The following values are recommended for various buffer sizes:

Buffer Size	Optimal Cache Hit Rates	
	RCACHE	WCACHE
1K	85-90%	80-85%
2K	80-85%	75-80%
>2K	75-80%	70-75%

In most cases, the buffer cache should favor the most active buffer size. If the numbers reported in the sample above are typical for the system, performance could be improved by increasing the number of 1-Kbyte buffers (not only because they have a lower cache-hit ratio, but also because there is significantly more I/O activity that is using 1-Kbyte buffers than is using 2-Kbyte buffers). The number of 2-Kbyte buffers could be reduced slightly to avoid devoting more memory to the buffer cache.

¹ The **crash** utility's **bfree** command reports the same information as **bfree(1M)**.

Tunable Parameters

Tunable parameters are used to set various table sizes and system thresholds to handle the expected system load. These are set to the default value on the operating system release media. Depending on your particular application and environment, you may need to modify some of these.

Modifying Tunable Parameters

Tunable parameters are modified using the **sysgen(1M)** command, which is discussed in Chapter 9.

1. Run **sysgen** without the **-d** option. Type **o** to open the *Configuration* to be modified, then the *Collection* with which the tunable parameters are associated. The *Items* screen will list all tunable parameters in the *Collection*.
2. Move the cursor to the line of the tunable you want to change, type **o** and you will get a screen that shows the default value and the current value.
3. Move the cursor to the **Value** field, type **c** and the new value. Note that, if the value appears in double quotes or parentheses, you must type the double quotes or parentheses around the new value.
4. Type **q** to close that screen and return to the *Items Screen*. You can then select another set of tunable parameters to modify, or type **q** to return to the *Collection* screen and then the *Configuration* screen. Type another **q** to exit **sysgen**.
5. When you exit **sysgen** by answering **y** (yes) to the four questions, the operating system is rebuilt with the new information. When you reboot the system, the new configuration is in place, with a copy of the previous system preserved in the */oldrealix* file.
6. If you are unable to boot using the new */realix* file, boot on the */oldrealix* file and try running **sysgen** again.
7. If you are able to boot but the system is unstable, boot on the */oldrealix* file but save the "bad" */realix* file to use with **crash(1M)** in analyzing the problems.

Memory Usage Parameters

The *cf/descriptions/kernel* file contains a number of tunable parameters. Some of these never need to be modified, others are modified fairly regularly. These tunable parameters will appear on the **sysgen** screen in alphabetical order; to aid readability, the kernel tunable groups are discussed here in groups according to what they do.

The default values are appropriate starting points for a development environment. We suggest that you monitor system activity for the first few weeks after setting up the system, then modify parameters as appropriate to support the particular needs of your environment. For instance:

- ❑ If the machine is being used for development and you have a large number of users (especially if they use shell layers or REAL/VU® windows), it may be appropriate to increase the size of the process table by increasing the value of NPROC. Note that, in this release, the value of several other tunable parameters will automatically be adjusted if you modify the value of NPROC.
- ❑ If the machine is dedicated to a realtime application, it may be appropriate to decrease the value of NPROC.
- ❑ Other parameters should be modified according to applications running on the system.

The tunable parameters listed in Table 6-4 determine the size of memory-resident tables and other similar entities. The table shows the default value, recommended values by memory configuration to use as guidelines, and the range into which values may fall.

Table 6-4. Memory Usage Parameters

Parameter	Definition	Recommended Value					Comment
		8 Mb	16 Mb	24 Mb	32 Mb	40 Mb	
CDLIMIT	Maximum file write address			32768			
F5CLUSIZ	F5 contiguous cluster attempt size			8			
FLKREC	Maximum file record count			50			
MAXPMEM	Maximum number of physical memory pages			0			
MAXPRBUFS	Maximum number of printfs active			100			
MAXSLICE	Max process time slice (ticks)			6 ticks			
MAXUMEM	Maximum number of mem pages in user virtual space	1024	2048	3072	4096	5120	Defaults to 2048 Max Value: 32768 or total virtual memory
MAXUP	Maximum number of processes per user			25 75			Defaults to 75 Min Value: 15
† NCLIST	Character list buffer count			256			
† NCALL	Call-out table size			NPROC+30			
NFILE	Open file table size			NPROC*4			Usually 4*NPROC
NINODE	Inode table size			NFILE			Min Value: NFILE
NSINODE	Number of system V inodes			NINODE			
NMOUNT	File system mount table size			50			
NOFILES	Maximum number of open files per process			80			Min Value: 20; Max Value: 100
NPROC	Process table size	128	160	192	224	256	Defaults to 128 Min Value: 64; Max Value: maxmem/5
NREGION	Region table size			NPROC*3			Min Value: 2, 5*NPROC
PUTBUFSZ	Console circular buffer size			2000			
SHLBMX	Maximum number of shared libraries			0			Defaults to zero
SPTMAP	System page table map of virtual space			200			

† controls structure which, if too small, will cause system to panic

System Tables for All Processes

This set of kernel tunable parameters determines the size of a set of system tables that impact all executing processes on the system. *Concepts and Characteristics* discusses how these tables work and should be studied in conjunction with this specific information on how to select values for the parameters. Setting the size of these tables to an appropriate size for your environment is important: values that are too small will seriously degrade system performance, cause processes to fail or, in some cases, cause the system to panic, whereas values that are too large consume unnecessary amounts of memory and may cause performance degradation as more paging operations are necessary.

- | | |
|------------------|--|
| FLCKREC | Specifies the number of records that can be locked by the system. The number of file/record lock table entries currently being used is reported in the lock-sz field of the sar -v report. |
| MAXPMEM | Specifies the maximum amount of physical memory to use in pages. The default value of 0 specifies that all available physical memory be used. |
| MAXPRBUFS | Specifies the maximum number of kernel error messages generated by cmn_err(D3X) calls that can be active at one time. The default value for this parameter is adequate for most environments. However, if you increase the value of MAXPRBUFS , you should also increase the value of PUTBUFSZ . |
| NCLIST | <p>Specifies how many character list buffers to allocate. Each buffer contains up to 58 bytes of data and occupies 64 bytes. The buffers are dynamically linked to form input and output queues for the terminal lines controlled by the serial driver (the serial ports COM1 and COM2) and other slow speed devices that may be configured on the system. The system will panic if it runs out of CLISTs.</p> <p>The average number of buffers needed per terminal is in the range of 5 to 10, plus 5 to 10 CLISTs for each <i>sxt</i> device (for shell layers, shl(1)); up to 8 <i>sxt</i> devices can be associated with each terminal device. The default value is set high enough to prevent system panics if there is not heavy use of shell layers.</p> |
| NCALL | Specifies how many callout table (timeout table) entries to allocate. Each entry represents a function to be invoked at a later time by the clock handler portion of the kernel. If the callout table overflows, the system panics. |

NFILE Specifies how many open file table entries to allocate. Each entry represents an open file. NFILE must be less than or equal to NINODE. The NFILE control structure operates in the same manner as the NINODE structure. The **file-sz** field on the **sar -v** report tells how many open file table entries are being used; if the table overflows, it is listed in the **sar** report and a message is written to the system table.

NINODE Specifies how many inode table entries to allocate. Each table entry represents an incore inode which represents an active file, directory, open file, or mount point. The file control structure is modified when changing this variable.

The number of entries used depends on the number of opened files, but are usually in the range of 100 to 400. NINODE must be equal to or greater than NFILE. The **inod-sz** field on the **sar-v** report tells how many inode table entries are being used; if the inode table overflows, it is listed in the **sar** report and a warning message is output on the system console.

NSSINODE should have the same value as NINODE. The two file system architectures supported on the REAL/IX Operating System are implemented (using the File System Switch) internally as one architecture, so there should be a 1:1 correspondence for entries in these two tables.

NMOUNT Specifies how many mount entries to allocate. Each entry represents a mounted file system. The **root (/)** file system is always the first entry. When full, the **mount(2)** system call returns the error EBUSY. Since the mount table is searched linearly, this value should be as low as possible.

NPROC Specifies how many process table entries to allocate. Each table entry represents an active process. The **onesec** process is always the first entry and **/etc/init** is always the second entry. The number of entries depends on the number of terminal lines available and the number of processes spawned by each user. The **proc-sz** field on the **sar -v** report tells how many process table entries are being used. When the process table is full, the **fork(2)** system call returns the error EAGAIN and an overflow is logged for the **sar** report.

For general purpose systems, a good starting value for NPROC is five times the number of concurrent users on the system. NPROC should not exceed **maxmem/5**; to get the current value of **maxmem**, use the **crash** utility's **rd -d maxmem** command.

NREGION Specifies how many region table entries to allocate. Most processes have 3 regions: text, data, and stack. Additional regions are needed for each shared memory segment and shared library (text and data) attached. However, the region table entry for the text of a "shared text" program will be shared by all processes executing that program. Each shared memory segment attached to one or more processes uses another region table entry.

This value is automatically modified when NPROC is changed. On systems that use shared memory and run with the process table nearly full, it might be appropriate to set this to $\lceil (3.5) * \text{NPROC} \rceil$. If the system runs out of region table entries, an error message is output on the system console.

PUTBUFSZ Specifies the size of a circular buffer, **putbuf**, that is used to contain a copy of kernel error messages generated by **cmn_err(D3X)** calls. The contents of **putbuf** can be viewed using **crash(1M)**. Making this buffer smaller or larger affects the number of messages that are written to **putbuf** before the system begins overwriting existing messages. The default value of this tunable is adequate for most environments. However, if you increase the value of PUTBUFSZ, you should also increase the value of MAXPRBUFS.

SHLBMAX Specifies the maximum number of shared libraries that may be used. Note that if this value is anything other than the default, you may need to increase the value of NREGION.

SPTMAP Specifies the number of entries in the map used to allocate kernel virtual address space.

Resources for User Programs

The next set of kernel tunable parameters defines resource allocation for individual user processes.

- | | |
|-----------------|--|
| CDLIMIT | Specifies the maximum size (in 512-byte blocks) of any file. This limit prevents a runaway job that is writing to a file from filling up the partition. If you are running an application that uses large, contiguous files, you may need to increase this parameter. For general purpose applications, this parameter is seldom modified. Individual programs can override this limit with the ulimit(2) system call. |
| MAXSLICE | Specifies in clock ticks the maximum time slice for user processes. After a process executes for its allocated time slice, that process is suspended. The operating system then dispatches the highest priority process and allocates to its MAXSLICE clock ticks. ¹ |
| MAXUMEM | Specifies the maximum size of a user's virtual address space in pages. Even if less than the hard-coded upper limit of 32768 pages, this number must always be smaller than the number of pages in virtual address space (physical memory pages plus total <i>swap</i> space allocated). We recommend that you never set MAXUMEM to a value smaller than the default of 2048. Setting MAXUMEM to a smaller value may impact an existing binary program's ability to run. |
| MAXUP | Specifies how many concurrent (child) processes a non-superuser is allowed to run. This value should be at least 10% less than the value of NPROC. This value is per user identification number, not per terminal. For example, if 12 people are logged in on the same user identification, the default limit would be reached very quickly. |
| NOFILES | Specifies the maximum number of open files ² per process. Processes using standard C library I/O subroutines (fread , fwrite , etc.) are limited to 20 open files, regardless of the value of NOFILES. Values higher than 20 are accessible only to processes using system calls (write , read , etc.). Unless an application package recommends that NOFILES be changed, the default setting should not be modified. |

If the configured value of NOFILES is greater than the maximum (100) or less than the minimum (20), the configured value is set to the default (100), and a NOTICE message is sent to the console.

¹ Realtime processes scheduled with the fixed-priority scheduler can set their own time slice (which is inherited by child processes) with the **setslice(2)** system call. The maximum slice is 11 years.

² Open files per process include the standard files: **stdin** (0), **stdout** (1), and **stderr** (2). This leaves the number of data files that can be opened per process as NOFILES-3. It is possible to close the standard files, but this is not recommended and should be done with extreme caution.

Configuring the Buffer Cache

On many UNIX operating systems, one tunable parameter determines the number of buffers in the cache; all buffers are the same size, and that size is determined by the operating system developers, not the administrator. The REAL/IX Operating System allows you to configure the buffer cache to use a mix of buffer sizes ranging from 1 Kbyte to 128 Kbytes. Table 6-5 summarizes the tunable parameters used to do this.

Table 6-5. Buffer Cache Parameters

Parameter	Definition	Default
DEFBUFSIZ	Default buffer size	4096
NBUF1K	Number of 1-Kbyte buffers	0
NBUF2K	Number of 2-Kbyte buffers	0
NBUF4K	Number of 4-Kbyte buffers	0
NBUF8K	Number of 8-Kbyte buffers	0
NBUF16K	Number of 16-Kbyte buffers	0
NBUF32K	Number of 32-Kbyte buffers	0
NBUF64K	Number of 64-Kbyte buffers	0
NBUF128K	Number of 128-Kbyte buffers	0
NHBUF	Number of hash buckets to allocate	0

As long as all these have a value of 0, the buffer cache is automatically configured to use buffers of the size specified by DEFBUFSIZ.

The buffer cache should be configured to provide optimum values for the following:

1. **Number of buffers:** The system buffer cache is a memory array containing disk file information. Improvements in the hit rate of this cache increases with the number of buffers. Cache hits reduce the number of disk accesses and thus improve overall performance. The number of buffers allocated is reported on the system console every time the system is booted.
2. **Size of each buffer:** If you use file systems with logical block sizes larger than the system default size, you must reconfigure the system buffer cache to include buffers the same size or larger as a file system logical block. As well, if you use file systems with logical block sizes smaller than the default size for your system, you will most likely want to reconfigure the system buffer cache. All buffered I/O operations transfer one block of data at a time, using one buffer for each transfer. The operating system can use a buffer that is larger than the data block for the transfer (for instance, a 2-Kbyte buffer can be used to transfer a 1-Kbyte block of data), but it cannot allocate two contiguous 1-Kbyte buffers to transfer a 2-Kbyte block of data.

If the buffer cache has too few buffers of a needed size, processes will get fewer cache hits (times when the required data is already in a buffer and does not require a disk access operation) and may block awaiting an empty buffer. However, using a larger buffer than necessary wastes memory, as does configuring too many buffers. Wasting memory for the buffer cache can degrade performance because there is less space for executing user code. This becomes especially critical when using the large buffers.

There are two modes of determining the size of the buffer cache:

1. **automatic mode:** Keep the NBUF?K tunable parameters set to 0. The operating system allocates a percentage of available memory for the system buffer cache, with each buffer the size specified to DEFBUFSIZ. For large memory configurations, the system uses 12.5% of the first 8 Mbytes of available memory and 5% of the rest of available memory for buffers. As released, this makes a buffer cache of 4-Kbyte buffers, for compatibility with the porting base. For many environments, a buffer cache of 2-Kbyte buffers (with user file systems created to use 2-Kbyte logical blocks) provides better performance; to automatically configure a cache of 2-Kbyte buffers, change the value of DEFBUFSIZ to 2048.
2. **specified buffer configuration mode:** Assign positive, non-zero values to some mix of the NBUF1K through NBUF128K parameters that correspond to the number of buffers of that size to be created. The value of DEFBUFSIZ is ignored, and the buffer cache is configured to contain the specified number of each size buffer. This mode can also be used to create a smaller or larger buffer cache of same-sized buffer.

Assign negative, non-zero values to some mix of the NBUF1K through NBUF128K parameters, which will allocate that percentage of available memory for the system buffer cache. The value of DEFBUFSIZ is ignored.

For example, let's say we have 15,000,000 bytes of available memory, and we change the default settings on the following buffer cache parameters:

```
NBUF1K   = -8
NBUF2K   = -2
NBUF64K  = 4
```

8.0% of available memory will be allocated for 1-Kbyte buffers (8.0% of 15,000,000 is 1,200,000 bytes, divided by 1024, and rounded up yields 1172 buffers)

2.0% of available memory will be allocated for 2-Kbyte buffers (2.0% of 15,000,000 is 300,000 bytes, divided by 2048, and rounded up yields 147 buffers)

4 buffers of 64 Kbytes will be allocated.

It is impossible to give firm rules on how to decide the configuration of the buffer cache. Unless your applications require some specific logical block or buffer size, or you have created file systems whose logical block sizes are different than the default size for your system, we suggest you begin by using the automatic mode.

See page 6-40 for a discussion of how to use the `icount(1M)` data to determine the appropriate logical block size for various file systems; page 6-42 explains how to use the `bfree(1M)` data to judge whether the buffer cache contains an appropriate mix of buffer sizes. The following guidelines can guide your decisions for configuring the buffer cache:

- ❑ **sar -b** summarizes the number of cache hits on read and write operations for the buffer cache. It does not break this information down by buffer size, so if your buffer cache uses a variety of buffer sizes, use this information in conjunction with the **bfree** output, which reports the cache hit rate for each size configured buffer.
- ❑ In general, cache hit rates should fall somewhere around 80–85%, adjusted up or down according to buffer size and I/O activity on that buffer (see page 6–42 for more information); lower hit rates (higher contention rates) indicate that performance could be improved by allocating more buffers of that size, higher hit rates (lower contention rates) indicate that you might be able to allocate fewer buffers of that size to free up memory without adversely impairing performance on buffered I/O operations.
- ❑ If you are using a wide variety of logical block sizes in the file system, it may make sense to allocate "alternate" sizes of buffers. For instance, if you are using 1-Kbyte, 2-Kbyte, 4-Kbyte, 8-Kbyte, and 16-Kbyte logical blocks, configure the buffer cache with 1-Kbyte, 4-Kbyte, and 16-Kbyte buffers. The amount of memory wasted when a 2-Kbyte transfer uses a 4-Kbyte buffer or an 8-Kbyte transfer uses a 16-Kbyte buffer is apt to be less than the amount of additional memory that would be consumed by having a set of buffers of all sizes.

Hash Buckets

The **NHBUF** parameter specifies how many "hash buckets" to allocate for the buffer cache. These are used to locate a specific buffer (given a device number and block number) to avoid the overhead that would be associated with a linear search through the entire list of buffers. This value must be a power of 2, and the operating system will adjust it from whatever value you provide.

The operating system is capable of automatically calculating an appropriate number of hash buckets to allocate. To take advantage of this feature, keep the value of **NHBUF** set to 0. The system will perform this calculation by selecting the nearest power of 2 that is equal to or less than the total number of buffers, halving that value, and allocating the resultant number of hash buckets. Consider the following examples.

If the buffer cache contains 1596 buffers and **NHBUF** is set to 0, the system will select 1048 (the nearest power of 2 equal to or less than 1596), halve this value, and thus allocate 512 hash buckets.

If the buffer cache contains 500 buffers and **NHBUF** is set to 0, the system will select 256 (the nearest power of 2 equal to or less than 500), halve this value, and thus allocate 128 hash buckets.

Tunable Parameters for Multi-Block Transfers

All supported disk devices for the REAL/IX Operating System support multi-block data transfers. This means that sequential read and write operations can transfer several contiguous blocks of data to and from the device without returning in between. This takes advantage of contiguity in F5 files by using single I/O operations to move multiple disk blocks, thus significantly reducing the system load. Table 6-6 summarizes the tunable parameters that control this feature.

Table 6-6. Multi-Block Transfer Parameters

Parameter	Definition	1-Kbyte Buffers	Recommended		
			2-Kbyte Buffers	4-Kbyte Buffers	Maximum
F5CLUSSIZ	F5 contiguous cluster attempt size	8	8	4	16
MBMAXBLKS	Multi-block max # blocks per transfer	8	8	4	16
MBCNT	Multi-block max # transfers in system	150			{# buffers} / MBMAXBLKS
MBMAXSZ	Multi-block max byte count per transfer	32768			16 * buffer size

These parameters should not be changed except for systems that have extremely tight performance requirements. The parameters are defined as:

F5CLUSSIZ

Specifies the number of logical blocks that the system will attempt to cluster when allocating file space dynamically on F5 file systems. Instead of allocating one block at a time (which leads to fragmented files and, consequently, longer file-access times), the system will cache F5CLUSSIZ blocks when it allocates a new block to the file. If the operation subsequently needs another data block, it is taken from this cached set of blocks; if the blocks are not used by this operation, they are freed when the process closes the file.

The default value of 8 has proven to be a good one for general-purpose and development applications. If your system has limited disk space, you may want to decrease the value of this tunable. If you are running applications that warrant a different "cluster size," such as one that uses 9-block records, you may want to increase the value of this parameter.

Note that F5CLUSSIZ affects only file allocation for the non-extent based portion of F5 file systems. It has no affect on preallocated file space or on files in S5 file systems.

MBMAXBLKS

The maximum number of logical file system blocks that can be transferred at one time. On systems that use large amounts of sequential I/O and have a large number of buffers configured, this value could be increased.

MBCNT	Determines the number of entries in the array of structures used for multi-block transfers, thus limiting the number of multi-block transfers that can be active on the system at one time. The value may be increased on systems that have a large number of buffers configured.
MBMAXSZ	The maximum number of bytes that can be transferred with each multi-block transfer operation.

For sequential read operations, the system reads many contiguous blocks in a single operation, on the assumption that all blocks will eventually be required. The more blocks read, the less likely it is that this assumption will prove true. The unwanted blocks use up buffers, so that the data in that buffer is overwritten. By setting these parameters appropriately, you limit the number of blocks that will be read unnecessarily.

The MBMAXBLKS and MBMAXSZ parameters control the amount of data that can be transferred in one multi-block I/O operation. The default values are set to appropriate values for systems where the majority of I/O operations use 1-Kbyte buffers. Some performance gains may be realized by modifying these parameters on systems where most I/O operations use larger buffers; Table 6-6 shows suggested values for larger buffers. F5CLUSSIZ specifies the number of blocks that will be clustered on the file system.

The default values are effectively the same for 1-Kbyte file systems, but for larger buffer sizes, MBMAXSZ may limit the number of blocks below the value of MBMAXBLKS. For instance, with 2-Kbyte file systems, the 8192 value of MBMAXSZ limits multi-block operations to 4 blocks, even if MBMAXBLKS is set to 8. Having these two values provides good support for systems that use more than one buffer size, in that there are usually fewer large buffers than small buffers.

Multi-block I/O capability can be deconfigured for a specific disk device by removing the "Has a multi block strategy handler" for the driver associated with that device. In most cases, this is not recommended. Set all values to 0 to remove multi-block capabilities from the system. This is appropriate in the following cases:

- ☐ The system is configured with very few buffers. Small buffer caches do not work well with multi-block I/O, because the read-ahead blocks may have been flushed from the cache before the process reads them.
- ☐ The main pattern of the supported application (for machines that are dedicated to one application) is to do a small amount of sequential I/O then skip to another part of the file. With a large number of buffers configured, it may be sensible to use the multi-block facility with a reduced value for MBMAXBLKS.

Tunable Parameters for Disk Updates

Table 6-7 summarizes the tunable kernel parameters that control disk updates through the buffer cache and the physical I/O buffer.

Table 6-7. Disk Update Parameters

Parameter	Definition	Default
BDFLUSHR	Disk Buffer Flush Rate (Seconds)	60 secs
NAUTOUP	Rate for flushing the buffers	30 secs

All file I/O operations (except those that bypass the buffer cache) are done with the buffer cache; the **bdflush** daemon periodically writes the contents of these buffers to disk. Two tunable parameters define how often the disk is updated from the buffers. **BDFLUSHR** determines how often the daemon runs. When **bdflush** runs, it checks all buffers and flushes those that have been memory-resident for the number of seconds specified by **NAUTOUP**. Specifying a smaller value for **NAUTOUP** increases system reliability by writing the buffers to disk more frequently and decreases system performance; specifying a larger limit increases system performance at the expense of reliability.

For normal system operations, the default values of **BDFLUSHR** and **NAUTOUP** provide a good balance between system performance and system reliability. It may make sense to adjust these values to flush the buffer cache more often when you are load testing new kernel code on a production system, or if there is some other reason to expect that the system may be unstable (for instance, if your building is suffering frequent power interruptions). In such cases, it may make sense to tune **BDFLUSHR** to 15 - 30 seconds, and **NAUTOUP** to 5 seconds. Increasing the values of these tunable parameters is not recommended except on a production system that has almost no critical tasks that write to the buffer cache.

For critical I/O operations, writes to the disk can be done immediately (rather than waiting for the **bdflush** daemon to run) through one of the following:

- ❑ Mount the file system with the **-s** option. Each time a buffer associated with a file in this file system is updated, it will immediately update the associated disk file.
- ❑ Augment critical **write(2)** operations with an **fsync(2)** operation. The buffers associated with this file will immediately be flushed to disk.

See the *Programmer's Guide* for more information on synchronizing disk access for critical realtime application programs.

Note that realtime processes executing at priorities higher than **PRIBDFLUSH** may cause the **bdflush** daemon to execute less often than is specified by the value of the tunable.

Tunable Parameters for Physical I/O

The physical I/O buffer is used as a temporary holding area for physical I/O operations to devices that cannot support direct transfers from the device to scattered user pages. Physical I/O is used when accessing the character special device file associated with a driver that also supports a block special device file, such as when accessing a raw disk device. The characteristics of this buffer are controlled by three tunable parameters:

Table 6–8. Physical I/O Parameters

Parameter	Definition	Default
NPBUF	Physical I/O Buffer Count (headers)	50
PHYSBSIZE	Raw I/O Static Buffer Size	64*1024
PHYSCNT	Raw I/O Buffer Number	1

- NPBUF** Specifies how many physical input/output buffer headers to allocate. One input/output buffer is needed for each physical read or write active. Note that devices that do not use the kernel's physical I/O buffer (because they have on-board memory to use) may use a buffer header.
- PHYSBSIZE** The size of the static buffer in bytes. This value is expressed as a multiple of 1024, so the default 64*1024 allocates 64 1-Kbyte blocks for the buffer. This number can be adjusted up or down depending on how many devices are using this buffer cache.
- PHYSCNT** The number of raw I/O buffers to allocate. Each buffer is PHYSBSIZE big (by default, 64 Kbytes). This value is seldom changed unless you are adding devices that require this buffer; it must be non-zero.

AUTODUMP and SYSRESET Parameters

In general, you should control whether the "autoboot on panic" feature is active. The two tunable parameters associated with this feature are not normally modified by customers:

- ❑ **SYSRESET** determines whether a panic results in an automatic system reset and reboot after a panic. If it is set to 0 (off), the system will loop and not reboot.
- ❑ **AUTODUMP** determines whether an automatic dump of core memory¹ is taken after a system panic. We strongly recommend that you not disable this feature; if disk space is at a premium, you can delete these files shortly after a reboot, but failure to save the core memory image can seriously damage your ability to analyze a system panic.

¹ and the associated */realix* file, required to run **crash(1M)**

- The automatic dump (**sysdump(1M)**), executed after a panic, writes the memory data and a copy of the operating system to */dev/rdump* (typically, the swap partition). The amount of core memory dumped is controlled by the **WHOLEDUMP** tunable parameter. When **WHOLEDUMP** is enabled (i.e., equal "1"), all of memory is dumped; otherwise only the system pages are dumped (as the system default, **WHOLEDUMP** = 1). Refer to **sysdump(1M)** for more information.
- When the system reboots and executes */etc/rc2.d/S02PUTBUF*, the **savedump(1M)** command copies the information in */dev/rdump* to the */usr/dumps* directory or to a tape device. This information is used by the **crash(1M)** command to assist you in analyzing the failure. Refer to **savedump(1M)** for more information.

Each of these tunable parameters can be set to one of the following values:

- 0 Do not take any action.
- 1 Take specified action only if the **bootedOK** program (run by the script */etc/rc2.d/S90noteOKboot*) was run the last time the system was booted. This is the default value for **SYSRESET** and **AUTODUMP**.

This program is run each time the system boots successfully (whether manually or automatically). By setting the **SYSRESET** and **AUTODUMP** parameters to 1, you request that the system be automatically rebooted and a memory dump be taken only if the previous boot was successful. If the system panics and the reboot is not successful (e.g., if another panic occurs before the boot is completed), the system does not attempt to reboot again.

- 2 Take specified action whether or not the **bootedOK** program executed the last time the system was booted.

The **SYSRESET** tunable affects the production kernel only. When running the debug kernel, the system will return to the **kdb(1M)** monitor after any panic regardless of how **SYSRESET** is set.

Paging Parameters

Table 6-9. Paging Parameters

Parameter	Definition	Recommended Value					Range	
		8 Mb (default)	16 Mb	24 Mb	32 Mb	40 Mb	Minimum	Maximum
GPGSLO	VHAND Free Memory Low Water Mark (%)	1%					1	GPGSHI-1
GPGSHI	VHAND Free Memory High Water Mark (%)	5%					GPGSLO+1	25% of available memory
MAXSWAPLIS	Max No. of pages swapped out together	8					Do not change these values	
MINARMEM	Min # of Memory Pages for User Processes	8						
MINASMEM	# of System Reserved Memory/Swap Pages	8						
MINPAGMEM	Min # of Memory Pages for Paging (not available for resident programs)	NPROC*2					NPROC*2	
VHANDR	VHAND Maximum Run Rate (Seconds)	2 secs					1	300
VHNDFRAC	Memory Fraction when page aging starts	10					1	24

The **vhand** daemon frees up memory as needed, according to the values of these tunable parameters. Many of these values express percentages of the available user memory reported when the system is booted.

- When the amount of free memory on the system falls below 1/VHNDFRAC of available memory, the **vhand** daemon begins executing every VHANDR seconds.¹ Each time it checks the amount of free memory and takes aging statistics of the pages of memory being used for the text and data of user processes.

Lower values for VHNDFRAC make **vhand** become active when more memory is available; lower values for VHANDR make **vhand** execute more often.

- When the amount of free memory falls below GPGSLO percent of available user memory, **vhand** begins paging out "least recently used" pages of active processes to disk in units of no more than MAXSWAPLIST pages.
- When the amount of free memory is over GPGSHI percent of available user memory, **vhand** stops paging out pages of processes.

In most environments, the value of these tunable parameters is not changed. If the system is paging excessively, memory should be freed up. If a process attempts to access free memory and there is none available, the system may panic.

MINPAGMEM may be decreased to allow more physical memory to be used for resident realtime processes, at the expense of more paging by non-resident processes.

¹ **vhand** runs at priority 95, so may run less often if a higher-priority realtime process is executing. The priority of **vhand** can be modified with the PRIVHAND tunable parameter discussed in the next section.

Process Priority Parameters

A number of REAL/IX system processes and daemons have tunable parameters to determine their runtime priority. In most cases, these priorities should not be changed, although for some special applications it may be necessary to change them. The priorities assigned by these tunable parameters are all realtime priorities, meaning they will not be adjusted by the operating system, will preempt processes scheduled at non-realtime priorities, and will themselves be prevented from running by realtime processes scheduled at higher priorities.

Parameter	Default	Controls
PRIBDFLUSH	95	Priority of bdflush daemon
PRONESEC	95	Priority of onesecc daemon
PRIPGRPSIG	0	Priority of pgrpsig daemon
PRIPRFD	95	Priority of prfd , the print daemon
PRISTREAMS	95	Priority of streams daemon
PRIHITIMED	0	Priority of hitimed daemon
PRLOTIMED	90	Priority of lotimed daemon
PRITTYD	95	Priority of tyd daemon
PRIVHAND	95	Priority of vhand

The default values for the priorities are either 0 (which is the highest realtime priority) or 95 (the lowest realtime priority is 127). Proper values must be determined according to the needs of the applications running on the system. The ramifications are discussed below:

PRIBDFLUSH

The **bdflush** daemon flushes I/O buffers that have not been used recently. Most realtime I/O operations do not use buffers and so should execute at a higher priority than **bdflush**. If you have realtime operations that do large amounts of buffered I/O, this process should run at a higher priority.

PRONESEC

The **onesecc** daemon maintains free page counts, calculates new process priorities for time-slice processes, and unblocks other daemons. The priority of **onesecc** is not normally changed.

PRIPGRPSIG

The **pgrpsig** daemon is used to send signals to the process groups.

PRIPRFD

The **prfd** daemon writes error messages from kernel processes to the console and */usr/adm/putbuf* file. The priority of **prfd** is not normally changed.

PRISTREAMS

The **STREAMS** daemon controls the interface mechanisms used for a **STREAMS** driver. It should run at a high enough priority to ensure that false timeouts are not detected.

PRIHITIMED The **hitimed** daemon manages realtime interval timer expirations, delays, alarms, and all other timer services for high priority processes.

PRILOTIMED The **lotimed** daemon manages realtime interval timer expirations, delays, alarms, and all other timer services for low priority processes.

PRITTYD The **ttyd** daemon manages line discipline interrupts for the driver that controls the console and serial ports on the CPU; it must run at a higher priority than any process that accesses a driver that uses the line discipline or the system can deadlock. The appropriate priority for **ttyd** must be determined in consideration of the following:

- ❑ **ttyd** runs for a few hundred microseconds every 16 milliseconds. This could impact the performance of critical realtime processes running at a lower priority
- ❑ realtime processes running at a higher priority than **ttyd** may prevent it from running. This could delay the output of data to the console, including error and panic error messages; if delayed more than 58 milliseconds, some of this data could be lost. Also, there is no guarantee that you will be able to use an "emergency shell" running on the console or one of the terminals to regain control of the system if a high-priority process goes into an endless loop.

Note that there are some possible hardware ramifications if the priority of this daemon is changed.

PRIVHAND The **vhand** process controls paging operations as discussed in the previous section. If realtime applications running at priorities higher than 95 may require paging, this priority should be changed. If **vhand** cannot run and a process cannot access adequate memory without something being paged out, the system will panic. Note that **vhand** may run less often than **VHANDR** seconds if a realtime process is executing at a higher priority.

RESIZESWAP Parameter

The RESIZESWAP parameter determines whether to enable the feature that automatically resizes misconfigured *swap* areas for paging. To turn off the resizing feature, set the RESIZESWAP parameter to a value of 0. In most environments, this is not advised. When RESIZESWAP is enabled (set to a value of 1), the system automatically adjusts the system *swap* device defined at system configuration time and the additional *swap* areas created with the *swap(1M)* command for the following:

- ❑ ensures that no *swap* area begins at the beginning of the partition (block 0). Each *swap* area must be offset at least one 512-byte block from the beginning of the partition.
- ❑ checks that the size specified for a *swap* area fits within the defined partition and, if not, sizes it down. The system will get an I/O error on the *swap* device and panic if the *swap* area extends beyond the end of the partition and the system tries to access *swap* area beyond the end of the partition.
- ❑ adjusts the size of all *swap* areas up or down so that the starting block number is a multiple of 16.¹

NODE Parameter

The NODE parameter establishes the system name for communications facilities such as UUCP. The *sysgen* parameter is overwritten by the value supplied to the *uname -S* command (executed by a script in the */etc/rc2.d* directory when the system goes to multi-user state). The *sysadm setup* scripts executed when you set up the system modify the value of *uname*, so this parameter is unaffected. Once established, avoid changing the system node name if users on other systems routinely send files or mail to your system.

¹*swap* areas should be page-aligned. The starting block number of the *swap* area should be a multiple of the page size (specified in 512-byte blocks). In this release, each page is 8192 bytes, or 16 blocks.

Other Kernel Parameters

The following tunable parameters should never be changed by customers. They are listed here with their values for reference only.

Parameter	Definition	Value
BADDISKS	# of bad disk track mappings supported	4
CPUBOARD	CPU Board Type	0
GPGSMASK	Paging Daemon Aging Mask	0x00000108
MACHINE	Machine Type	i386
MAXRDAHEAD	Max readahead for consecutive file blocks	4
MAXTREQ	Number of tty daemon requests per port	8
NTTYDREQ	Number of tty daemon request blocks	50
REL	Operating System Release	e.g., V.3-E.0
SYS	System Name	realix
VERSION	Version	e.g., E.0

STREAMS Parameters

The STREAMS interface is supported for the TCP/IP networking package with the socket and TLI interfaces. A number of tunable parameters control how STREAMS works; most of these are under the *strbufs* description file, except for MAXSEPGCNT, which is part of the *kernel* description file. If you have not configured any Ethernet controllers and are not using the TCP/IP feature, these values can be tuned down to reclaim memory for other uses. These values can be modified to support the particular needs of your applications and your configuration.

Table 6-10 lists the initial values of the general STREAMS parameters. These values are adequate for a configuration that has 32 users and one Ethernet board.

Table 6-10. STREAMS Parameters – General

Parameter	Definition	Initial Value	Range	
			Minimum	Maximum
MAXSEPGCNT	# of Additional Event Cell Memory Pages	1		
SPCNT	# of STREAMS Pipe Modules	10		
NUMTIMOD	# of Transport Interface Modules	10		
NSTREVENT	Initial Number of STREAMS Event Cells	256		
STRLOFRAC	Low-Priority Block Allocations Cutoff	80	0	STRMEDFRAC
NSTRPUSH	Maximum # of Modules Pushed on Stream	9		
STRMEDFRAC	Medium Priority Block Allocations Cutoff	90	STRLOFRAC	100
NMUXLINK	Multiplexer Link Count Maximum	87		
NSTREAM	Number of "Stream-Head" Structures	120		360+SPCNT
NQUEUE	Number of STREAMS Queues	(5*NSTREAM)		
NLOG	Number of minor devices for log driver	3		
STRCTLSZ	STREAMS Message Control Maximum Size	1024		
STRMSGSZ	STREAMS Message Data Maximum Size	4096		

MAXSEPGCNT

The number of additional pages of memory that can be dynamically allocated for event cells. If this value is 0, only the allocation defined by NSTREVENT is available for use. If the value is greater than 0, when the kernel runs out of event cells, it will attempt to allocate an extra page of memory from which new event cells can be created. MAXSEPGCNT places a limit on the number of pages that can be allocated for this purpose. Each new page can provide 166 event cells. Once a page has been allocated for event cells, it cannot be recovered later for use elsewhere. We recommend that the NSTREVENT value be set to accommodate the normal load conditions, and that MAXSEPGCNT be set to 1 to handle exceptional loads when they arise.

SPCNT	Number of STREAM pipe devices. Any two minor STREAM devices can be opened and connected to each other, so that each user is at the end of a single stream. This provides a full-duplex communications path and supports the passing of file descriptors as well.
NUMTIMOD	Number of TLI modules that can be configured. If you are running many TLI-based applications, increase this value accordingly.
NSTREVENT	The initial number of stream event cells to be configured. Stream event cells are used for recording process-specific information in the poll(2) system call. They are also used to implement the STREAMS I_SETSIG ioctl and in the kernel bufcall(D3X) function. The minimum value to configure is the expected number of processes expected to be using STREAMS concurrently. Note that this number is not necessarily a hard upper limit on the number of event cells that will be available on the system (see MAXSEPGCNT).
STRLOFRAC	The percentage of data blocks of a given class at which low-priority block allocation requests are automatically failed. For example, if STRLOFRAC is 80 and there are 48 256-byte blocks, a low-priority allocation request will fail when more than 38 256-byte blocks are already allocated. The parameter is used to help prevent deadlock situations by starving out low-priority activity. The recommended value works well for current applications.
NSTRPUSH	The maximum number of modules that can be pushed onto a stream. This prevents an errant user process from consuming all available queues on a single stream. By default, this value is 9, but in practice, existing applications seldom push more than four modules on a stream.
STRMEDFRAC	The percentage cutoff at which medium priority block allocations are failed (see STRLOFRAC discussion above). The recommended value works well for current applications. Note that there is no cutoff fraction for high-priority allocation requests; it is effectively 100.
NMUXLINK	The maximum number of multiplexer links to be configured. One link structure is required for each active multiplexer link (STREAMS I_LINK ioctl).
NSTREAM	<p>The number of "stream-head" (stdata) structures to be configured. One is needed for each stream opened, including both streams currently open from user processes and streams linked under multiplexers.</p> <p>This value should be a lump sum of all STREAMS driver's individual maximum number of end-points configured. The sar -v report (see page 6-35) reports the number of stream-head structures used relative to the number configured.</p>

NQUEUE

The number of STREAMS queues to be configured. Queues are always allocated in pairs, so this number should be a multiple of 2. A minimal stream contains four queues (two for the stream head, two for the driver). Each module pushed on a stream requires two additional queues, so the default value allows half of the STREAM-queue pairs to have a module pushed in association with them. This is generally a good value if most of the networking applications use the socket interface, which typically does not push modules. If you are using the TLI interface, you may want to set this value to $6 * \text{NSTREAM}$ or $8 * \text{NSTREAM}$. If you run out of queues, an "out of queues" message will appear on the console and the affected operation will fail.

NLOG

The number of STREAMS error loggers that can be pushed. This facility is not accessed by any module or driver supported in the current release of the operating system.

STRCTL SZ

The maximum allowable size of the control portion of any STREAMS message. The control portion of a `putmsg(2)` message is not subject to the constraints of the min/max packet size, so the value entered here is the only way of providing a limit for the control part of a message. The recommended value of 1024 is more than sufficient for existing applications.

STRMSG SZ

The maximum allowable size of the data portion of any STREAMS message. This should usually be set just large enough to accommodate the maximum packet size restrictions of the configured STREAMS modules. If it is larger than necessary, a single `write(2)` or `putmsg(2)` can consume an inordinate number of message blocks. The recommended value of 4096 is adequate for existing applications.

The `NBLK*` parameters determine the number of STREAMS data blocks and buffers to be allocated for each size class. Table 6-11 shows the initial values for these tunable parameters. These default values are generous for most environments; you may be able to save memory by reducing some values.

Table 6-11. Parameters to Control STREAMS Data Blocks and Buffers

Parameter	Definition	Initial Value	Minimum
NBLK4	No. of 128-byte buffers for stream msg	512	0
NBLK16	No. of 16-byte buffers for stream msg	256	0
NBLK64	No. of 64-byte buffers for stream msg	256	0
NBLK128	No. of 128-byte buffers for stream msg	128	0
NBLK256	No. of 256-byte buffers for stream msg	48	0
NBLK512	No. of 512-byte buffers for stream msg	40	0
NBLK1024	No. of 1024-byte buffers for stream msg	48	0

Table 6–11. Parameters to Control STREAMS Data Blocks and Buffers (cont.)

Parameter	Definition	Initial Value	Minimum
NBLK2048	No. of 2048-byte buffers for stream msg	40	0
NBLK4096	No. of 4096-byte buffers for stream msg	12	0
NBLK8192	No. of 8192-byte buffers for stream msg	0	0
NBLK16384	No. of 16384-byte buffers for stream msg	0	0
NBLK32768	No. of 32768-byte buffers for stream msg	0	0

Message block headers are also allocated based on these numbers; the number of message blocks is 1.25 times the total of all data block allocations. This provides a message block for each data block, plus some extras for duplicating messages. The optimal configuration depends on both the amount of primary memory available and the intended application. The default values are adequate to support a moderately-loaded configuration using UUCP over TCP/IP.

If no STREAMS driver is configured on your system, you can regain memory by setting the value for all these tunable parameters to 0.

If your application consistently transfers large buffers and you have a large memory configuration, it may be appropriate to increase the number of message buffers that are used, especially if more than one STREAMS driver is configured. If your application transfers a large number of small data packets, it may be appropriate to increase the number of some smaller buffers. Use the **strstat(1M)** utility (part of the unsupported tools package) to monitor the buffer use.

Interprocess Communications Parameters

The tunable parameters discussed in this section are associated with the interprocess communication facilities: user-level semaphores (**sem**) and binary semaphores (**bs**), messages (**msg**), and shared memory (**shm**).

The initial values assigned for these parameters were selected to enable customers to begin testing code that uses these facilities. Most sites will need to adjust these values, either to lower values to reclaim memory or to higher values to meet the needs of the applications being run. These tunable parameters can be set to 1 (minimum value) if no applications are using the associated facilities.

Binary Semaphore Parameters

The tunable parameters defined in the **bs** description file control the binary semaphore mechanism. Note that binary semaphores use a small segment of shared memory, so shared memory parameters must never be set to 1 if you are using binary semaphores. The following table lists the initial value of these tunable parameters.

Parameter	Description	Initial Value
BSCNTPRC	Maximum # of binary semaphores per process	32
BSCNTSYS	Maximum # of binary semaphores in system	300
BSMAXPRC	Maximum # of processes using binary semaphores	20

BSCNTPRC The maximum number of active **bsget(2)** operations allowed at one time. The values in the **maxbsemas/proc** column on the **sar -B** display show the number of active **bsget** operations for the largest user on the system; if that number is significantly smaller than the initial value, you should set this tunable to a lower value.

BSCNTSYS The maximum number of unique binary semaphores that can be initialized on the system at one time. The **bsemas** column on the **sar -B** display shows the number of these entries being used.

BSMAXPRC The maximum number of processes that can use the binary semaphore mechanism at one time. The **bsemaprocs** column on the **sar -B** display shows the number of processes using the binary semaphore mechanism.

If a process attempts to issue a **bsget(2)** call that would result in one of these values being exceeded, the **bsget** will fail with an **ENOSPC** error number.

Message Parameters

The tunable parameters defined in the *msg* description file control the message facility. See **msg(4)** for information on the structures controlled. The following table lists the initial value of these tunable parameters.

Parameter	Description	Initial Value	Max
MSGMAP	Size of control map for message segments	100	
MSGMAX	Maximum size of a message	2048	65535
MSGMNB	Maximum length of a message queue	4096	
MSGMNI	Max number of message queues	50	
MSGSEG	Number of message segments in the system	1024	NOTE
MSGSSZ	Size, in bytes of a message segment	8	NOTE
MSGTQL	Number of message headers in the system (# of outstanding messages)	40	

NOTE: [MSGSSZ*MSGSEG] must be less than or equal to 131,072 bytes (128 kilobytes).

If a process issues a **msgget(2)** call with the **IPC_CREAT** flag to initialize a message queue, or a **msgsnd(2)** call to send a message to an existing queue that would exceed these limits, the process

will block or the call will fail, depending on whether the `IPC_NOWAIT` flag is set. Use the `sar -m` report to determine whether the current values for these tunable parameters are appropriate for your environment.

- MSGMAP** Determines the number of entries in the map that controls message queues. Each message queue that is initialized requires one entry in this map. To allow for holes in the map, MSGMAP should be set to double the value of MSGMNI, to ensure that MSGMNI rather than MSGMAP limits the number of message queues.
- MSGMAX** Determines the maximum size of each message queue created without preallocation (the `IPC_PREALC` flag). If message queues are preallocated, the `msgsz` parameter to the `msgget(2)` system call determines the maximum size of each message queue, overriding limits set by this parameter.
- MSGMNB** Sets the maximum number of bytes in all messages queued to a single message queue (`msgqid_ds`).
- MSGMNI** The maximum number of message queues that can exist on the system at one time.
- MSGSEG** The maximum number of messages that can be posted to all message queues on the system.
- MSGSSZ** Determines the maximum number of bytes allowed for a single message. `MSGSEG * MSGSSZ` determines the size of the data portion map space, which is the space allocated for message bodies.
- MSGTQL** Maximum number of message headers on all non-preallocated message queues on the system. One header is required for each message sent, to define the message type and so forth.

Semaphore Parameters

The tunable parameters defined in the `sem` description file control the user-level semaphore mechanism. See `sem(4)` for information on the structures controlled. The following table lists the initial value of these tunable parameters.

Parameter	Description	Initial Value	Max
SEMAEM	Adjustment on exit for maximum value	16384	16384
SEMMAP	Size of semaphore set control map	10	
SEMMNI	Number of semaphore Id's in the kernel	10	
SEMMNS	Number of semaphores in the system	40	
SEMMNU	Number of undo structures in the system	15	
SEMMSL	Maximum number of semaphores per identifier	25	
SEMOPM	Maximum number of semaphore operations	8	
SEMUME	Maximum number of undo entries	8	
SEVMX	Maximum value a semaphore can have	32767	32767

If a process issues a call that would exceed one of these values, the process will block or the call will fail, depending on whether the `IPC_NOWAIT` flag is set. Use the `sar -m` report to gauge whether the current values are appropriate for your environment.

- SEMAEM** Maximum adjustment to semaphore value made when the process exits. Customers should never modify this value.
- SEMAP** Determines the number of entries in the map that controls semaphore arrays. Each semaphore array that is initialized requires one entry in this map. The value of **SEMAP** must be \geq **SEMMNI**.
- SEMMNI** Sets the maximum number of semaphore arrays that can exist on the system.
- SEMMNS** Sets the maximum number of semaphores that can exist on all semaphore arrays in the system. Note that this number is set significantly lower than **SEMMNI*SEMMSL**, because most environments do not use many large semaphore arrays.
- SEMMNU** Undo structures are required because operations on semaphore sets must be atomic. If a process locks A and fails to unlock B, an undo structure is required to unlock A. If `semop(2)` calls fail with an `ENOSPC` error, this value needs to be increased.
- SEMMSL** Maximum number of semaphores in each semaphore array.
- SEMOPM** Determines the largest value that the `nsops` argument to `semop(2)` may take.
- SEMUME** Maximum number of entries allowed in an undo structure. The value of **SEMUME** must be \geq (**SEMMSL** - 1)
- SEMVMX** Limits the value that the `semval` member of the `sems` structure can have. It is not usually changed.

Shared Memory Parameters

The tunable parameters defined in the `shm` description file control the shared memory facility. See `shm(4)` for information on the structures controlled. The following table lists the initial value of these tunable parameters.

Parameter	Description	Initial Value	Max
SHMBRK	Number of pages reserved above data segment	512	
SHMALL	Max # of in-use shared memory text segments	512	
SHMMAX	Maximum shared memory segment size	512*1024	
SHMMIN	Minimum shared memory segment size	1	
SHMMNI	Maximum number of shared memory identifiers	30	
SHMSEG	# of attached shared memory segs per process	6	15

If a process issues a call that would exceed any of the limits set by these tunable parameters, the process will block or the call will fail depending on whether the `IPC_NOWAIT` flag is set. Note that shared memory is required by the binary semaphore mechanism, so these tunable parameters cannot be set to a minimal value if binary semaphores are being used.

SHMBRK	This number is used to calculate the address where <code>shmat(2)</code> will attach the first shared memory segment that uses 0 for the second parameter. The minimum value allowed is 512 pages.
SHMALL	Maximum number of shared memory regions that can exist on the system at one time.
SHMMAX	The maximum size of one shared memory segment, in bytes. If a <code>shmget</code> call uses a larger value for the second parameter (<i>size</i>), the call will fail. Note that <code>shmget</code> specifies the size in bytes.
SHMMIN	The minimum size of one shared memory segment, in pages. If a <code>shmget</code> call uses a smaller value for the second parameter (<i>size</i>), the system will automatically round the size up to the size specified here.
SHMMNI	The maximum number of shared memory segments (<code>shm_ds</code> structures) that can exist on the system at one time.
SHMSEG	The maximum number of segments to which one process can attach with <code>shmget(2)</code> at one time. The default value of 6 is large enough for most applications because the normal usage is to use one shared memory segment for several purposes.

Tunable Parameters for Realtime Facilities

Some of the REAL/IX realtime facilities have tunable parameters associated with them. The appropriate values for these tunable parameters must be determined according to the applications being run; the default values are provided as good starting points.

Three tunable parameters are associated with most of these features. They can be set to 1 for any facility not being used:

1. **Maximum number of processes using the feature.**
2. **Maximum number of operations per process.** This value determines the maximum number of structures that can be initialized and owned by one process at a time.
3. **Maximum number of operations in system.** This value determines the amount of memory that is reserved for control structures for the feature.

Note that the maximum number of operations in the system is often smaller than the maximum number of processes using the facility multiplied by the maximum number of operations per process. For instance, with the default parameters, 64 asynchronous I/O operations can exist in the system at one time, which is significantly smaller than the 480 operations implied by 30 processes each executing 16 asynchronous I/O operations.

Tunable Parameters for Asynchronous I/O

The tunable parameters for asynchronous I/O are listed under "Asynchronous I/O Parameters".

Parameter	Definition	Initial Value
AIOCNTPRC	Maximum # of AIO operations per process	16
AIOMAXPRC	Maximum # of processes using AIO	30
AIOMAXAIO	Maximum # of AIO operations in system	64
AIOMAXDAEM	Maximum # of AIO daemons in system	32

The **sar -I** report displays statistics for asynchronous I/O operations.

- | | |
|-------------------|---|
| AIOCNTPRC | Compare this value to the values displayed in the maxaioblks/proc field of the sar -I report to see if this value is appropriate for your environment. |
| AIOMAXPRC | Compare this value to the values in the aioprocs field of the sar -I report to determine if this value is appropriate for your environment. |
| AIOMAXAIO | Compare this value to the values in the aioblks field of the sar -I report to determine if this value is appropriate for your environment. |
| AIOMAXDAEM | Usually, one daemon per inode is used when performing an asynchronous I/O emulation through a daemon. Note that aread(2) , awrite(2) , arinit(2) , and awinit(2) will return an EAGAIN error condition if a daemon cannot be created. |

Tunable Parameters for Connected Interrupts

Tunable parameters for the connected interrupt mechanism are listed under "Connected Interrupt Parameters".

Parameter	Definition	Initial Value
CICNTPROC	Maximum # Cinterrupts per process	5
CIMAXSYS	Maximum # of Cinterrupts system wide	25
CIMAXPROC	Maximum # processes using Cinterrupts	25

The **sar -C** report gives usage statistics for connected interrupt operations.

- CICNTPROC** Compare this value to the value of **maxcis/proc** on the **sar -C** report to determine if the current values are appropriate for your environment.
- CIMAXSYS** Compare this value to the value of **cintrpts** on the **sar -C** report to determine if the current values are appropriate for your environment.
- CIMAXPROC** Compare this value to the value of **ciprocs** on the **sar -C** report to determine if the current values are appropriate for your environment.

Tunable Parameters for Common Event Mechanism

The tunable parameters for the common event mechanism are under "Events Parameters".

Parameter	Definition	Initial Value
EVCNTPRC	Maximum # of posted events per process	32
EVTMAXPRC	Maximum # of processes using events	30
EVTMAXQUE	Maximum # of events per process	64

The **sar -E** report gives usage statistics for the common event mechanism.

- EVCNTPRC** Compare this value to the value of the **maxevts/proc** field of the **sar -E** report to determine if the current values are appropriate for your environment.
- EVTMAXPRC** Compare this value to the value of the **evtprocs** field of the **sar -E** report to determine if the current values are appropriate for your environment.
- EVTMAXQUE** Compare this value to the value of the **events** field of the **sar -E** report to determine if the current values are appropriate for your environment.

The maximum numbers specified include asynchronous I/O completion events, timer expiration events, connected interrupt notifications, and resident process violations as well as user-posted events.

Tunable Parameters for Timer Subsystem

The tunable parameters that control the timer subsystem are listed under "Timer System Parameters".

Parameter	Definition	Initial Value
ADJTFDELTA	adjtime(2) delta for fast slew, millisecs	1000
ADJTFAST	adjtime(2) slew rate if > ADJTFDELTA	5000
ADJTRATE	adjtime(2) slew rate, microsecs/sec	500
ITIMAXSYS	Maximum # of interval timer entries system wide	ITICNTPROC * ITIMAXPROC + ITIMAXK
ITICNTPROC	Maximum # of interval timers per process	5
ITIMAXK	Maximum # of interval timers for system use	5
ITIMAXPROC	Maximum # of processes using interval timers	10
NCALL	Maximum # of timeout(D3X) calls	NPROC+30
CLOCKRES	System clock rate, ticks per second	0
TMR_UPDLOW	Use low priority time daemon for update	1
TMR_PRILOW	Use low priority time daemon if pri >=	128

Parameters Controlling Process Interval Timers

The **sar -T** report gives usage statistics for the process interval timers.

ITIMAXSYS The maximum number of interval timer entries system wide will, by default, be a function of the other process interval timer parameters, **ITICNTPROC** * **ITIMAXPROC** + **ITIMAXK**. This allows for the worst case usage, where each process using process interval timers uses the maximum available to it. If it is known that this is an overestimate and the system is short of memory, reducing **ITIMAXSYS** will save some memory. Compare this value to the value displayed in the **timers** field of the **sar -T** report to see if the current values are appropriate for your environment.

ITICNTPROC This will vary according to the needs of the realtime applications that use process interval timers. The default number of 5 is expected to be sufficient for most cases. Compare this value to the value displayed in the **maxtims/proc** field of the **sar -T** report to see if the current values are appropriate for your environment.

ITIMAXK It is possible for device drivers to make use of the process interval timer mechanism; however, such use will be rare.

ITIMAXPROC This will vary according to the needs of the realtime applications that use process interval timers. The default number of 10 is expected to be sufficient for most cases. Compare this value to the value displayed in the **timprocs** field of the **sar -T** report to see if the current values are appropriate for your environment.

NCALL NCALL gives the total number of internal kernel calls to the **timeout(D3X)** and **timeouts(D3X)** functions. It is unlikely that you would want to change this parameter. The default is large because a system panic with the message "Timeout table overflow" will result if insufficient timer blocks have been reserved via NCALL.

The **c** command to **crash(1M)** will display all the timer blocks that have ever been used since system boot. Each timer block takes 64 bytes, so a few kilobytes might be saved by reducing NCALL.

TMR_UPDLOW It is unlikely that you would want to change this parameter. The system obtains the current time by applying a delta to the recorded elapsed time since boot. Obtaining this delta when an **adjtime(2)** call has caused a skew of the clock rate is a fairly lengthy procedure. This delta is calculated afresh whenever an accurate reading of the time is required. A coarse value of the current time, in seconds, is also maintained for use in time stamping operations where low overhead is desirable and precision unnecessary. This coarse setting is updated once a second by one of the time daemons.

If **TMR_UPDLOW** is non-zero, the low priority time daemon performs the update, otherwise, this is the responsibility of the high priority daemon.

The default value of 1 causes the low priority time daemon to be used for updating the coarse timestamp. This prevents the high priority time daemon from blocking other high priority processes while it performs this trivial piece of housework.

It may be worthwhile to set **TMR_UPDLOW** to 0 if:

- ☐ The low priority time daemon may be blocked by higher priority processes for extensive lengths of time
- ☐ File timestamps must never lag
- ☐ Accurate time values are not being requested predictably and frequently
- ☐ It is acceptable for the high priority time daemon to have an extra 100 microseconds or so of processing to perform every second (the exact overhead is hardware-dependent)

TMR_PRILOW It is unlikely that you would want to change this parameter. Processes of priority equal to or lower than TMR_PRILOW will have their timeout handling performed via the low priority daemon.

The default value is chosen so that all non-realtime processes are serviced by the **lotimed** daemon, leaving **hitimed** for the realtime applications that may be running.

Parameters Controlling adjtime(2)

The **adjtime(2)** system call allows a small change (delta) to be made to the system clock in a gradual manner, making it appear as if the clock is running a little faster or slower, until the change has been accomplished.

Parameters control the rate of change. If the desired delta is small, a small rate of change is applied. This is given by ADJTRATE. If the desired delta is more than ADJTFDELTA, then a faster rate of change is used, given by ADJTFAST. It is unlikely that you would want to change these parameters.

CLOCKRES This parameter determines the resolution of the system clock as seen by the timer subsystem. A value of 0 implies the default setting. Other values specify the number of clock interrupts per second that the system clock should generate. There is a limited range of valid rates. A value for CLOCKRES that does not correspond to a valid rate will be rounded up as appropriate. Valid rates for CLOCKRES are hardware platform dependent. For higher portability of your program, use HZ as defined in the file */usr/include/sys/param.h*.

Tunable Parameters for Disjoint I/O

The tunable parameters for disjoint I/O are under "Kernel and Paging Parameters."

Parameter	Definition	Initial Value
DJNTCNT	Max # of disjoint I/O operations in system	AIOMAXAIO+NPBUF
DJNTMAXSZ	Max size of a disjoint I/O transfer	64*1024

The two parameters associated with disjoint I/O operations determine how many disjoint I/O operations can concurrently execute on the system and the maximum size of a disjoint I/O transfer. Disjoint I/O is most often used with asynchronous I/O; setting the value at NPBUF more than the value of AIOMAXAIO allows some other processes to also use disjoint I/O.

If you are not using the asynchronous I/O facility, DJNTCNT can be sized down to conserve memory. (Changing the value of AIOMAXAIO automatically reduces the size of DJNTCNT.) However, as released, the value of DJNTCNT causes less than 4 Kbytes of memory to be dedicated, so the memory saving is small. We recommend that you never set the value of DJNTCNT to be less than 10.

DJNTMAXSZ must be changed to 128*1024 if you want to use 128-Kbyte logical blocks for any file system. If the value of DJNTMAXSZ is not changed, **mkfs** will fail if used as follows:

```
mkfs -f128k /dev/rdsk/special-file
```


7. TTY Management

Chapter 7

TTY Management

TTY devices include user terminals, printers, and networking devices that use a `tty` structure to control character display processing. This chapter discusses how to set up the system files that are used to identify these devices to the operating system. It also includes information on how the TTY subsystem works, how the login process works, and how block mode display processing works. Chapter 8 discusses how to set up the LP Spooler system that controls the use of the line printer.

Login Cycle

Figure 7-1 illustrates the login cycle. *Concepts and Characteristics* explains what happens from a user viewpoint. From a system standpoint, the steps are:

1. Each idle terminal port (in other words, a port that no one is currently using) is running a `getty(1M)` process. This `getty` process is controlled by the `init(1M)` process according to information in the `/etc/inittab` file. The `getty` lines in `inittab` define the special device number for the terminal port and the default baud rate for the line.
2. When a user requests a login prompt (by sending an interrupt signal to the port), `getty` uses the baud rate specified for that line in `inittab` as an index into the `/etc/gettydefs` file, from which it gets default character display settings for the terminal. See `termio(7)` for a description of these character display settings; users can reset these settings with `stty(1)` commands issued either in their `.profile` files or at the shell.

If `getty` detects that the baud rate of the actual terminal device being used does not match the default baud rate specified for that terminal special device file, it waits for the user to press `BREAK` (send another interrupt signal), and then tries another baud rate. The next baud rate to try is also specified in the `/etc/gettydefs` line. This process continues until `getty` is satisfied that the baud rate of the actual terminal device matches the baud rate of the `gettydefs` line.

3. `getty` sets the initial-flags character display settings (field 2) specified in the `gettydefs` file and sends a prompt to the user terminal, according to what is specified in field 4 of the `gettydefs` line. By default, this is `"login:"`, but it can be changed by modifying the `gettydefs` line.
4. When the system receives the login name typed by the user, `getty` sets the final-flags (field 3) defined in `gettydefs` and calls the `login(1)` process, which sends a `"Password:"` prompt to the terminal.

5. When a user enters a password, **login** checks it against what is in the `/etc/passwd` file (field 2) for that user (internally, **login** calls an encryption program and compares the encrypted password to field 2 in `/etc/passwd`). See Chapter 3 for more information on the `/etc/passwd` file. If the password does not match what is in the file, **login** posts a "Login incorrect" message and returns control to **getty**, which resets the terminal to the initial-flags settings and posts another "login:" prompt.

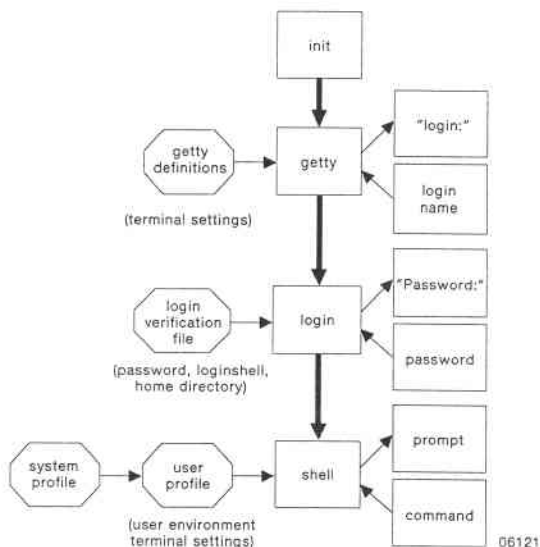


Figure 7-1. The login Cycle

6. When the user enters the correct password, **login** executes the system's `/etc/profile` file (same for all users), then the user's individual `.profile` file. Either of these files may contain **stty** command lines that reset the terminal's character display processing mode from what **getty** has set. See Chapter 3 for a discussion of these files. The user's `.profile` file usually sets and exports the **TERM** variable that defines the *terminfo* file that defines the block-mode display processing for the terminal being used.
7. The final step of the **login** procedure is to initialize a shell or other initial program for the user. Several shells are available on the system; **login** uses the shell specified in the last field of the `/etc/passwd` line for that user, unless the user specifies a different shell in the `.profile` file. If the last field of the `/etc/passwd` line is blank, **login** spawns `/bin/sh`.

Basic TTY administration involves configuring three sets of information:

- special device files in the */dev* directory for all TTY devices. The system comes with the special device files configured for two serial ports (COM1 and COM2) and seven virtual terminals.

See Appendix A for more information on terminal special device files.

- */etc/getty* lines in the */etc/inittab* file for all serial TTY ports configured for user terminals and logins used by incoming network devices. The system comes with these lines set up for the serial ports listed above; you must turn on the lines that match the boards configured on your system. *getty* lines for serial ports used for printers and networking devices remain "OFF".
- */etc/gettydefs* lines for all baud rates supported on the system. The released operating system includes an adequate set of *gettydefs* lines to get you started, although you may eventually want to add lines to satisfy the special needs of your installation.

The */etc/getty* lines in the */etc/inittab* file and the lines in the */etc/gettydefs* files can be modified through the *sysadm ttygmt* menus or by editing the files directly.

***/etc/getty* Lines in the */etc/inittab* File**

Appendix C discusses the structure of the */etc/inittab* file and how it is used to control transitions between machine states. Here we will discuss the */etc/getty* lines that control TTY devices. There is one *getty* line for each serial port. There are no *getty* lines for parallel ports because they are not used for login terminals.

Like all lines in the *inittab* file, the *getty* lines have four fields plus an optional comment:

```
id:run—state:action:process      # comment
```

The following discussion explains how these fields are populated for *getty*.

A sample *getty* line:

```
co:234:respawn:/etc/getty console console # console terminal
```

As shown in the sample line, the rules for the four fields in the *getty* lines in the *inittab* file are:

1. **id** is "co" for the console. For all other TTY ports, it is a two digit number.
2. **run-state** is always "2" for multi-user state. If 1 (for single-user state) were included, users would be able to log in when the system is in single-user state.

3. **action** is usually either "**off**" or "**respawn**" for TTY devices. In the released system, all TTY lines are set to **off**; you will have to activate the appropriate lines by changing this action to **respawn** for the ports that correspond to boards configured in your system. **getty** lines for TTY ports on boards you do not have configured should be left **off**.

off is also used for serial ports used by printers and for outgoing network lines. Ports used for incoming network lines (in other words, lines that can be dialed by other machines through UUCP) must be set to **respawn**. **off** is also used to disable a port that is malfunctioning until it can be repaired.

4. **process** is **/etc/getty**, which requires two arguments: the special device number of the terminal port and a label that serves as an index into the **/etc/gettydefs** file described below. The label used should be the one that matches the needs of the TTY device that most often uses that port. As released, most terminal lines are set up for 9600 baud terminals; if you are using something else, modify the label or else users must sit and press the break key trying to get the **gettydefs** setting they need.

Options can be added from those listed on **getty(1)**; a common one used is **-t nn** , which tells **getty** to hang up if it does not receive a response to the **login:** prompt within **nn** seconds. Using **-t** for dial in lines is a good security feature.

5. **comment** follows the **#** sign on each line. We recommend using these comment fields to add identification information for all lines. For instance, if your terminals are hard-wired to the computer, the comment could identify the room number or user who has that port.

Lines in the /etc/gettydefs File

The *label* given as an argument to the */etc/getty* line serves as an index into the */etc/gettydefs* file, which specifies the **termio(7)** flags to use initially (to send the login prompt to the user terminal) and finally (when calling the **login** program). Each *gettydefs* line has five fields, separated with # signs. These fields are:

label#initial—flags#final—flags#login—prompt#next—label

The following lines illustrate a sample */etc/gettydefs* file:

```
19200# B19200 HUPCL # B19200 SANE IXANY TAB3 HUPCL #login: #9600
9600# B9600 HUPCL # SANE IXANY TAB3 HUPCL #login: #4800
:
1200UUCP# B1200 # B1200 SANE IXANY TAB3 HUPCL #login: # #1200UUCP
```

These five fields are populated as follows:

1. **label** must match what is used by the **getty** lines in the *inittab* file. There are no rules for labels, but by convention most labels define the baud rate. This is not necessary, but it makes it easier to look at the *inittab* file and identify the baud rate for each line.
2. **initial—flags** are the character display settings (as defined in **termio**) necessary for **getty** to send the login prompt to the terminal. The baud rate must be specified. The other common flag to specify is HUPCL (hang-up-on-close), which causes the line to be disconnected when the last process with the line open closes it or terminates. The released *gettydefs* files includes lines for all baud rates with and without HUPCL as an initial flag.

The lines for the console also include two additional initial flags: OPOST (postprocess output) and ONLCR (map new-line to carriage-return and new-line on output).

The **pc220** line for personal computer devices includes a number of other initial flags that are required for the system to communicate with personal computers used as terminal devices. Depending on what sort of personal computers is being used, you may need to create other *gettydefs* lines that define the appropriate initial flags (and final flags) for the devices.

3. **final—flags** are set after the user responds to the login prompt and before **getty** calls the **login** process. These are the flags that are in affect for the user unless they are modified by **stty** commands in the */etc/profile* or user's *.profile* file.

For a description of all possible *gettydefs* flags, refer to **termio(7)**.

Each */etc/gettydefs* entry is followed with a blank line. After editing the file, run the command:

```
/etc/getty -c /etc/gettydefs
```

This causes **getty** to scan the file and print the results on your terminal. If there are any unrecognized modes of improperly constructed entries, they are reported.

Using sysadm to Set Up TTYs

The */etc/getty* lines in the */etc/inittab* file and the */etc/gettydefs* file can be modified either by editing the file directly or using the **sysadm ttygmt** menus. To edit the files directly, you must either be the superuser or supply the **sysadm** password.

The **sysadm ttygmt** menu provides three options that do the following:

1. **lineset** displays the information from the */etc/gettydefs* file.
2. **mklineset** allows you to modify the information in the */etc/gettydefs* file.
3. **modtty** allows you to view and modify the */etc/getty* lines in the */etc/inittab* file.

sysadm lineset

The **lineset** option displays the */etc/gettydefs* definitions.

Each line setting is just a name used to identify a set of TTY line characteristics. During the **login** process, a **BREAK** signal is used to look for a compatible computer connection. Line settings on one line "hunt" from left to right, moving from one setting to the next on receiving this signal. The rightmost setting on each line hunts to the first one, forming a circular hunt sequence.

You can look at one line setting in detail by following the menu and typing in the line setting number when prompted. For example:

```
Select one line setting to see it in detail [?, q]: 1200
```

sysadm lists the initial and final flags from the *gettydefs* file with a brief description of each, the text to be used for the login prompt, and the next setting to be used.

The following flags are used for the normal terminal lines in the released *gettydefs* file:

baud	same rate as specified in initial-flags
SANE	defines a set of "normal" characteristics. SANE is defined in the getty code as:
IGNBRK	ignore break condition
BRKINT	signal interrupt on break
ISTRIP	strip character
ICRNL	input carriage-return character for new-line character
IXON	enable start/stop output control. When on, ctrl-s stops output to the screen; ctrl-q restarts the output
OPOST	postprocess output
ONLCR	map new line to carriage-return and new-line
ISIG	enable signals
ICANON	canonical input (in other words, process erase and kill characters before transmission)
ECHO	enable echo
ECHOK	echo new-line after kill character
CS7	7 bits
CREAD	enable receiver
PARENB	enable parity
IXANY	enable any character to restart output
TAB3	expand tabs to spaces for display
HUPCL	hang up on close

4. **login-prompt** is the character string posted to user terminals to begin the login process. Usually this is left as **login:** (except for console devices, where **Console Login:** is used), but it can be changed if you like. We suggest you avoid changing this prompt for ports used by incoming UUCP lines because other machines that call your machine may expect the **login:** prompt.

sysadm mklineset

You can use the **sysadm mklineset** screen to create *gettydefs* lines that define new line settings and hunt sequences. **mklineset** prompts you for all the information required for a new *gettydefs* entry. For an example, we will enter two new line settings. If you type a ? for help in selecting a baud rate, you will get a listing of all the available baud rates.

```

Enter the name of the new tty line setting [?, q]: 1200300
Select a baud rate [?, q]: ? (to ask for help)

Select a baud rate [?, q]: 1200

Enter the login prompt you want (default = "login: ") [?, q]: <CR>

Do you want to add another tty line setting to the sequence? [y, n, q] y

Enter the name of the new tty line setting [?, q] 3001200
Select a baud rate [?, q]: 300
Enter the login prompt you want (default = "login: ") [?, q]: <CR>
Do you want to add another tty line setting to the sequence? [y, n, q] n

```

The line setting sequences you created will be displayed. If all the information is correct, install the new line settings.

```

Do you want to install this sequence? [y, n, q] y

Press the RETURN key to see the ttygmt menu [?, ^, q]: q

```


sysadm modtty

sysadm modtty provides screens to let you update the */etc/getty* lines in the */etc/inittab* file. The first screen displayed lists the special device files for each port that has a **getty** line in the *inittab* file, followed by the prompt:

```
Select the tty you wish to modify,
or enter ALL to see a report of all ttys [?, q]:
```

If you select ALL, you see a summary display of all the **getty** lines. For example:

TTY	State	Hangup Delay	Line Setting	Description
console	on	off	console	cpu card (Serial Port 1/Console)

The information displayed is:

1. **TTY:** the special device file for this **getty** line. This is the name you will specify to change one of these lines.
2. **State:** the **action** field for the *inittab* line. **sysadm** allows you to set the line to either "off" or "on"; "on" is interpreted as **respawn**.
3. **Hangup Delay:** the value for the **-t** option to **getty**.
4. **Line Setting:** the label used as an index into the */etc.gettydefs* file. In other words, a line setting as defined in **lineset** or **mklineset**.
5. **Description:** the comment section of the *inittab* lines.

If you specify one line, you will get that line's current characteristics, the available states, and the following prompts enabling you to change both the state and hangup delay:

```
Select a state (default: off) [?, q]: on
```

```
Enter a hangup delay, in seconds, or 'off' (default: off) [?, q]: 45
```

A list of available line settings will be displayed along with a prompt to select a line setting:

Select a line setting (default: 9600) [?, q]: 19200

The current description will be displayed along with another prompt to enter a new description:

Enter a new description (default: current description) [?, q]: <CR>

The current or new description will be displayed along with a list of the new characteristics. Verify that the new characteristics are correct and answer the following prompt:

Do you want to install these new characteristics? [y, n, q] y

Controlling Block-Mode Display Processing

Block-mode display processing is required for commands like `vi` and any applications that use **courses** for screen formatting. Such commands are very sensitive to the internal formatting specifications of the terminal. To enable the same program to run on a variety of terminal types, the `TERM` environmental variable is set and exported (usually in the user's `.profile` file).

In general, users are responsible for setting their own `TERM` environmental variables, but administrators may be called on to help novice users. If your site uses only a few types of terminals, you may want to include a shell script in the `/etc/sdprofile` file that prompts users to specify the terminal type and then establishes `TERM` appropriately; this profile is copied to create a default `.profile` file for new users. It is possible to set and export `TERM` in the `/etc/profile` file, but this is not recommended, even if all your terminals are the same type.

`TERM` references a "*terminfo*" file that defines the characteristics of the terminal. All *terminfo* files are stored under the `/usr/lib/terminfo` directory. This directory has 39 subdirectories, named *1*, *5*, *a*, *b* and so forth. Each terminal type has a name, such as *tvi950* for the TeleVideo® 950 and *vt100* for the DEC™ VT-100; the first letter of that name determines the subdirectory in which the terminal description is stored. So, the file for the TeleVideo 950 is `/usr/lib/terminfo/t/tvi950`. This file is linked to the `/usr/lib/terminfo/9/950` file, which is for the same terminal type, to provide an alias name.

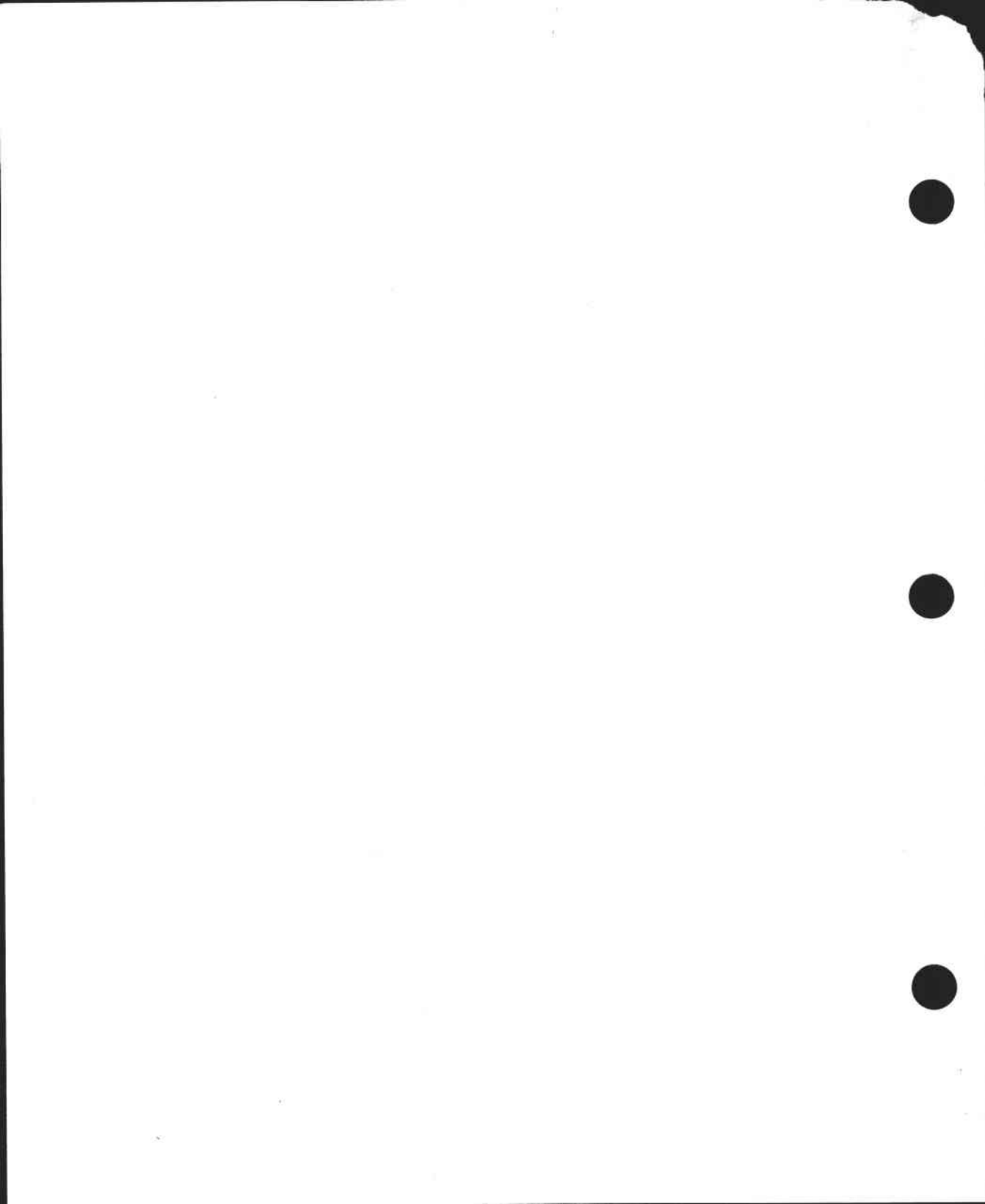
Names can be given suffixes that begin with a hyphen to indicate modes for the hardware or user preferences. Table 7-1 defines the conventional suffixes.

Table 7-1. terminfo Suffixes

Suffix	Meaning	Example
<code>-w</code>	Wide mode (more than 80 columns)	<code>vt100-w</code>
<code>-nam</code>	Without automatic margins	<code>tvi950-nam</code>
<code>-n</code>	Number of lines on the screen	<code>aaa-60</code>
<code>-na</code>	No arrow keys (leave them in local mode)	<code>c100-na</code>
<code>-np</code>	Number of pages in memory	<code>c100-4p</code>
<code>-rv</code>	Reverse video	<code>c100-rv</code>

Most terminals you will want to use are already defined in the *terminfo* files provided with the system. If you do not know the *terminfo* name for the terminal in question, look for files named for the model number or vendor initials. To confirm that this is the correct file, read the compiled description file with the `cat -v` command. Much of the output from this command will be nonsensical, but the full terminal name is displayed in the first line, along with synonymous names.

You can create new *terminfo* files for terminals not already defined. The bibliography in *Concepts and Characteristics* references articles that discuss how to do this.



8. Line Printer Administration

Chapter 8

Line Printer Administration

This chapter discusses the following points:

- ❑ How the LP Spooling system works
- ❑ How to find instructions for installing the LP Spooler
- ❑ The commands used to administer the system
- ❑ Printer interface programs
- ❑ LP Spooler files and directories

LP spooling means that you can send a file to be printed while you continue with other work. The term LP originates from Line Printer and includes many other types of printing devices as well. LP Spooling system software performs the following functions:

- ❑ receives and schedules files that users want to print
- ❑ starts programs that interface with the printer(s)
- ❑ tracks job status and issues error messages

The LP Spooler has five user commands (see the appropriate manual pages) as shown in Table 8-1.

Table 8-1. User Commands for the LP Spooling System

Command	Description
enable	Activate the named printer(s).
cancel	Cancel a request for a file to be printed.
disable	Deactivate the named printer(s).
lp	Send a file or files to a printer.
lpstat	Print the status of the LP system.

In addition to sending requests to the LP Spooling system, checking request status, and canceling requests, users can disable and enable a printer. A user can turn off a malfunctioning printer without calling the administrator.

The **lp.cnfg(1M)** command provides an easy interface to the LP Spooling system, but does not support all printer types. **lp.cnfg** can only be run by the superuser; see **lp.cnfg(1M)** for information on using this tool.

Administrative Commands

Separate commands for the LP administrator are shown in Table 8-2. These commands are found in the */usr/lib* directory and documented in the manual pages. If you expect to use them frequently, you might find it convenient to include that directory in your *PATH* variable. To use the administrative commands, log in using either the **root** or the **lp** system login.

Table 8-2. Administrative Commands for the LP Spooling System

Command	Description
accept	Permit job requests to be queued for a specified destination.
reject	Prevent jobs from being queued for a specified destination.
lpadmin	Set up or change the LP configuration.
lpmove	Move output requests from one destination to another.
lpsched	Start the LP scheduler.
lpshut	Stop the LP scheduler.

In Table 8-2 the administrative commands are listed in the order they occur in the manual pages. In the sections that follow, we describe the commands in the order they are logically used, followed by an example of how these commands might be used to set up the printer on your system.

lpadmin(1M)

The **lpadmin(1M)** command is used to add a new printer to the system, assign *classes* of printers, name or remove a default destination, and specify a model interface program. Do not use **lpadmin** when the LP scheduler, **lpsched(1M)**, is running, except if you are specifying the **-d** option.

Include one of the following three options on the command line when you execute **lpadmin**:

- d[dest]** creates a new default destination; the printer specified must have been previously defined with the **-p** option. The default destination determines where a file printed with the **lp(1)** command goes if the user does not specify an alternate destination and no value is assigned to the *LPDEST* environmental variable. If *dest* is not specified, there is no default destination. No other options can be used.
- xdest** remove this destination from the LP system. If this is the only member of a class, the class will be deleted also. No other options can be used.

- pprinter** names a new printer; the following options may be supplied to describe the printer (options may appear in any order):
 - cclass** assigns the printer to the specified *class*.
 - eprinter** use the appropriate interface program specified for this printer.
 - h** printer is connected to a parallel port.
 - iinterface** calls a new interface program for the printer specified in the **-p** option. *interface* is the path name of the new program.
 - l** printer is connected to a serial port (most often used for hard-copy terminals).
 - mmodel** identifies the printer model. Model interface programs reside in the */usr/spool/lp/model* directory.
 - rclass** remove this printer from the specified *class*.
 - vdevice** add this printer to the LP system. *device* is the full path name of the special device file for the port where the printer is attached (i.e., */dev/ttyxxx*).

The following examples illustrate how to use **lpadmin**.

- ❑ You must be the superuser to run **lpadmin**.
- ❑ In all but the first example, the LP scheduler must have been stopped with the **lpshut** command.

The first command makes printer *lp_1* the default printer on the system:

```
lpadmin -dlp_1
```

The next command adds a new printer called *lp_2* and associates it with the */dev/ttyxxx* device. The printer uses the **lp** model interface program:

```
lpadmin -plp_2 -v/dev/ttyxxx -mlp
```

The following example creates a hardwired printer called *lq_2* and associates it with */dev/ttyzzz*. The printer is added to a new class called *cl1*, and uses the same model interface program that is used with printer *lq_1*.

```
lpadmin -plq_2 -v/dev/ttyzzz -elq_1 -cc11
```

To change the model interface program for printer *lq_2* to the **lp** program, the command is:

```
lpadmin -plq_2 -mlp
```

To add printer *lp_2* to class *cl1*, the command is:

```
lpadmin -plp_2 -ccl1
```

To remove printers *lp_1* and *lp_2* from class *cl1*, the commands are:

```
lpadmin -plp_1 -rccl1  
lpadmin -plp_2 -rccl1
```

To remove the destination printer *lp_3*, the command is:

```
lpadmin -xlp_3
```

No printer can be removed if it has pending requests; pending requests must first be either canceled with the **cancel(1)** command or moved with the **lpmove(1M)** command.

lpsched(1M)

The **lpsched(1M)** command starts the LP scheduler. The LP scheduler takes the top job request off the queue and "hands" it to the appropriate interface program to be printed on a printer. The LP scheduler keeps track of the job progress, and as soon as the job is completed, it takes the next job request off the queue and repeats the process. As long as the LP scheduler is running, jobs requested by **lp** will be printed. If the scheduler is not running, jobs will not be printed.

The LP scheduler is started automatically each time the system is turned on. This is done through a shell script called **S38lpsched** in the */etc/rc2.d* directory.

Every time the scheduler is started, **lpsched** creates a file called **SCHEDLOCK** in the */usr/spool/lp* directory. As long as the **SCHEDLOCK** file is present, the system will not allow another scheduler to run. When the scheduler is stopped under normal conditions, either with **lpshut(1M)** or as part of the normal shutdown procedure, the **SCHEDLOCK** file is removed. However, if the system comes down abnormally, there is a possibility that the **SCHEDLOCK** file may not get removed. To ensure that the **SCHEDLOCK** file does not exist, */etc/rc2.d/S38lpsched* contains a command line to remove **SCHEDLOCK** first before it attempts to start the scheduler.

The command is entered without arguments. To verify that the scheduler is running, use the **lpstat** command with the **-r** option.

```
lpstat -r  
scheduler is running
```



*If many job requests are queued, there may be a delay before the **lpstat** command reports that the scheduler is running.*

lpshut(1M)

Two of the three **lpadmin** command options (**-x** and **-p**) cannot be executed unless the LP scheduler is stopped. The **lpshut(1M)** command stops the LP scheduler and terminates all printing activity. All requests that were in the middle of printing will be reprinted in their entirety when the scheduler is restarted. The command is entered without arguments, and responds when the scheduler is stopped.

```
lpshut
scheduler stopped
```

lpmove(1M)

Occasionally, you may find it necessary to move output requests from one destination to another. For example, if you have a printer that was removed for repairs, you will want to move all the pending job requests to a destination with a working printer. This is done using the **lpmove(1M)** command. Be aware that job requests routed to a destination without a printer are automatically rejected.

Another use of the **lpmove** command is to move specific requests from one destination to another. When this is done, **lp** will no longer accept requests for the original destination (this is the same effect as a **reject** command). **lpmove** refuses, however, to move requests while the LP scheduler is running.

The general format of the **lpmove** command is as follows:

```
lpmove requests dest
```

requests are the request identification numbers (request IDs) of jobs waiting to be printed, and *dest* is the destination to which the requests are to be moved. The destination can be a printer or a class of printers.

As an example of how **lpmove** is used, the following command moves all the requests for printer *lp1* to printer *lp2*. Moving the requests renames the request IDs from *lp1-*nnn** to *lp2-*nnn**. After the requests are moved, **lp** will no longer accept requests for *lp1* (this is the same effect as a **reject lp1** command issued after the **lpmove**).

```
# lpmove lp1 lp2
```

The following command line moves requests *lp1-54* and *lp2-55* to printer *dqp10_1*.

```
# lpmove lp1-54 lp2-55 dqp10_1
total of 2 requests moved to dqp10_1
#
```

The two requests are now renamed *dqp10_1-54* and *dqp10_1-55*.

accept(1M)

The **accept(1M)** command allows job requests to be placed in a queue at the named destination(s), where destination is the name of a printer or class of printers. The general format of the **accept** command is as follows:

```
accept destination(s)
```

As an example, the following line allows printer *dqp10_1* to start receiving requests:

```
# /usr/lib/accept dqp10_1
destination "dqp10_1" now accepting requests
```

reject(1M)

Sometimes it is necessary to stop **lp** from routing requests to a destination. For example, if a printer has been removed for repairs, or if too many requests are building at a destination, you may want to prevent new jobs from being queued at this destination. The **reject(1M)** command performs this function.

Requests in the queue when the **reject** command is invoked will be printed as long as the printer is enabled. After the condition that led to denying requests has been corrected, use the **accept** command to allow requests to be received again. The general format of the **reject** command is:

```
reject [-r[reason]] destinations
```

The **-r** option is used to let users know why requests are being rejected by the specified destination. *reason* is a brief explanation of the purpose for rejecting requests. If the reason consists of more than one word, enclose it in double quotes ("). The *destinations* are the printers that are not to accept requests any longer.

For example, you might enter the following for a printer, *lqp40_1*, that is being repaired. While *lqp40_1* is out of service you want to prevent **lp** from routing requests to it.

```
reject -r "printer lqp40_1 under repair" lqp40_1
destination "lqp40_1" is no longer accepting requests
```

Users who try to route jobs to *lqp40_1* will receive the following message:

```
lp -dlqp40_1 filename
lp: can't accept requests for destination "lqp40_1" -
printer lqp40_1 under repair
```

Setting Up the Printer

The standard printer for this system is the Hewlett Packard LaserJet Printer. It is already set up in the installation media, but the following instructions show how you would set it up if you had to. If you are using an alternative printer, you will need to modify some of the specifics but the general process will remain the same.

1. Turn off the LP scheduler: `/usr/lib/lpshut`
2. Configure the printer destination **hpjet** to the LP Spooler system, for example:

```
/usr/lib/lpadmin -phpjet -l -mhpjetserial -v/dev/ttyxxx
```

The options given define the printer as a serial interface device using the model interface program `/usr/spool/lp/model/hpjetserial`. A number of model interface programs are supplied in this directory; if you are using a printer that does not conform to any of these, you will have to write your own interface program as discussed in the following section.

3. Define **hpjet** to be the default destination for **lp** requests: `/usr/lib/lpadmin -dhpjet`
4. Allow queueing for **hpjet**: `/usr/lib/accept hpjet`
5. Activate **hpjet**: `/usr/bin/enable hpjet`
6. Turn on the LP scheduler: `/usr/lib/lpsched`
7. Print status of the LP system: `/usr/bin/lpstat`

Replace Steps 2 and 3 above with the following commands to configure two printers to a device, with **lp** requests queued to one of two printers, depending on availability and load:

1. Configure each destination with the appropriate port and make both printers members of **class1**:

```
/usr/lib/lpadmin -phpjet1 -l -mhpjetserial -v/dev/ttyxxx -cclass1
/usr/lib/lpadmin -phpjet2 -l -mhpjetserial -v/dev/ttyzzz -cclass1
```

2. Make **class1** the default destination for **lp** requests: `/usr/lib/lpadmin -dclass1`

Printer Interface Programs

Printers that are used as LP Spooling printers must have a printer interface program. Every print request made with the **lp** command is routed through the appropriate printer interface program before the request is printed on a line printer. The printer interface program to use is specified by **lpadmin(1M)**.

Model Interface Programs

Each type of printer requires its own interface program. Several, referred to as "model" interface programs, are furnished with the LP Spooling Utilities. The model interface programs are written as shell procedures, but they can be written as C programs or any other executable program.

Model interface programs are located in the */usr/spool/lp/model* directory. In order for the LP Spooler system to access it, the model interface program must have the owner and group set to "bin" and the mode set to 775.

Writing Interface Programs

If you have a printer that is not supported by one of the model programs, you will have to furnish an interface program for it. The shell script for a "dumb" printer interface program (a model program) is shown on page 8-10. This program may be used as a guide if you have to provide one of your own. This is followed by a discussion of the model interface program for the HP™ LaserJet, which illustrates a model interface for a more sophisticated device.

When the LP scheduler routes an output request to the printer, the interface program for the printer is invoked in the directory */usr/spool/lp/admins/lp/interface* as follows:

```
/usr/spool/lp/admins/lp/interface/P id user title copies options file ...
```

Arguments for the interface program are:

<i>P</i>	printer name (the name of the interface program is the name of the printer); \$0 in shell scripts
<i>id</i>	request id returned by lp; \$1 in shell scripts
<i>user</i>	logname of user who made the request; \$2 in shell scripts
<i>title</i>	optional title specified by the user; \$3 in shell scripts
<i>copies</i>	number of copies requested by user; \$4 in shell scripts
<i>options</i>	blank-separated list of class or printer-dependent options specified by user. These must be defined in the model interface program for the printer to be functional. \$5 in shell scripts
<i>file(s)</i>	full path name of the file(s) to be printed

When the interface program is invoked, its standard input comes from */dev/null* and both the standard output and standard error output are directed to the printing device. Interface programs format their output based on the command line arguments. Be sure that the interface program has the proper stty modes (terminal characteristics such as baud rate, output options) by adding **stty(1)** command lines of the form:

```
stty mode options < &1
```

Since **stty** commands affect standard input and output, this command line directs the standard input for the **stty** command to the device. An example of an **stty** command line that sets the baud rate at 1200 and sets some of the option modes is shown below.

```
stty parenb parodd 1200 cs8 cread clocal ixon 0<&1
```

Because different printers have different numbers of columns, make sure the header and trailer for your interface program correspond to your printer. When printing is complete, your interface program should exit with a code that tells the status of the print job. Exit codes are interpreted by **lpsched**(1M) as follows:

Code	Meaning to lpsched
0	The print job has completed successfully.
1 to 127	A problem was encountered in printing this particular request (for example, too many nonprintable characters). This problem will not affect future print jobs. The lpsched command notifies users by mail (1) that there was an error in printing the request.
greater than 127	Reserved for internal use by lpsched . Interface programs must not exit with codes in this range.

When problems occur that may affect future print jobs (for example, a device filter program is missing), it is wise to have your interface program disable printers so that print requests are not lost. When an active printer is disabled, the interface program can be halted with signal 15 (see **kill**(1) and **signal**(2)).

Sample Interface Programs

The code sample on the next page is the `/usr/spool/lp/model/dumb` program. It contains minimal functionality: it uses the **banner**(1) command to print the user's name on the first sheet, and accepts requests for multiple copies.

The **hpjetparll** interface program supports two-column output, 132 and 160 column output, in addition to multiple copies. You can view the entire script on-line to understand how other print options are added to the program to support features available on the printer.

/usr/spool/lp/admins/lp

This directory has some subdirectories and may contain some files that have default filters. This filter just provides simple printer fault detection. It does not convert files or handle its special modes.

<i>classes</i>	Contains one file for each LP class. This file identifies each member assigned to the class.
<i>forms</i>	contains a subdirectory for each available form.
<i>interfaces</i>	Contains executable interface programs for each printer. These programs are invoked with their standard error and standard output directed to the printer. Interface programs may be shell procedures or compiled C programs.
<i>printers</i>	Contains one subdirectory for each printer. These subdirectories contain information like configuration, forms, and users allowed and not allowed, respectively.
<i>pwheels</i>	Contains subdirectories for each available print wheel and character set. Print wheels and character sets describe a printers ability to print different font styles.

/usr/spool/lp/model

This is a directory that contains the printer interface programs distributed with the LP Spooling Utilities.

/usr/spool/lp/requests and /usr/spool/lp/temp

These directories contain files that describe each request submitted to the LP print service. The information is split into two directories to put more sensitive information in the */usr/spool/lp/requests* directory where it can be kept secure.

/usr/spool/lp/SCHEDLOCK

The lock file *SCHEDLOCK* is present while the LP scheduler is running to ensure that only one invocation of *lp sched* is active. Unlike other lock files, *SCHEDLOCK* has no expiration time.

Cleaning Out Log Files

As described earlier, when the scheduler is stopped, the *log* file is closed. When the scheduler is restarted, the *log* file is copied to */usr/spool/lp/oldlog*, and a new *log* file is started.

If the scheduler is not stopped for long periods of time and if you have a large number of LP requests, the *log* file can grow to be a large file. You can manually remove the contents of this file, or you can let the system do it for you on a scheduled basis.

To have the system clean out the *log* file, put an entry in the *root* file in the directory */usr/spool/cron/crontabs*. One way to do this is to log in as **root** and use the **crontab(1)** command. The other way is to edit a crontabs directory file.

The example below shows some typical **crontab** command lines. **crontab** adds these command lines to the *root* file in the */usr/spool/cron/crontabs* directory. Every Friday at 11:00 PM **cron(1M)** executes the commands. First, the contents of the *log* file are copied to the *oldlog* file, and then the *log* file is cleaned out.

```
crontab -l
0 23 * * 5 /bin/su lp c "cp /usr/spool/lp/log /usr/spool/lp/oldlog"
1 23 * * 5 /bin/su lp c ">/usr/spool/lp/log"
```



9. sysgen

Chapter 9

sysgen

sysgen(1M) is a screen-oriented utility that is used to modify the system software configuration. It simplifies the tasks of modifying the tunable parameters, adding new drivers to the system, and notifying the operating system of additional boards configured to use the existing drivers. This chapter discusses how to use **sysgen** for administrative tasks, which are defined as the following:

- ❑ Modifying tunable parameter values.
- ❑ Enabling and disabling devices included with the system.

sysgen with the **-d** option is used to configure new drivers and their associated devices into the system; **sysgen** can also be run non-interactively; this is seldom required for administrative tasks and so is not discussed here. Users who need to add drivers or run non-interactive **sysgen** should see the instructions in the *Driver Development Guide* and **sysgen(1M)**.

Running sysgen

Before running **sysgen**, copy the */realix* file to a disk file named something like "good.realix." This preserves a good copy of the operating system to retrofit if you make a critical mistake. While **sysgen** does make a copy of the old */realix* file, this file is overwritten the next time you run **sysgen** to rebuild the operating system.

To run the **sysgen(1M)** program interactively:

1. Login as **root** or execute the **su** command to gain superuser privileges.
2. Type **sysgen** to run the interactive version of **sysgen** to enable/disable collections and devices or modify the value of tunable parameters, then build a production kernel with the new values.
3. Type **sysgen debug** to run **sysgen** as described in step 2 and build a debug kernel with the new values.



*It is usually best to run **sysgen** in single-user mode and then reboot the system on the new kernel immediately. You can run **sysgen** in multi-user mode, but this opens the door to various problems that could occur if someone else modified some file or directory that is used to build the system.*

Notes for Using sysgen

The interactive versions of **sysgen** provide a hierarchical menu system. Table 9-1 summarizes the most commonly-used typing conventions used in **sysgen**. See **sysgen(1M)** for a full list.

Table 9-1. **sysgen** Typing Conventions

Function	Key(s)	Meaning
Responding to question	n	no to sysgen prompt
	y	yes to sysgen prompt
Moving cursor	arrow left <^h>	backspace
	arrow right <^l>	move cursor right
	arrow down j	move cursor down
	arrow up k	move cursor up
Manipulating Screens	a	add line
	c	change field enable/disable line
	d	delete line
	D	duplicate driver entry
	j or k	toggle values
	o	open a new screen
	q	close current screen

Note the following when using **sysgen**:

- ❑ If you want to replace a 2-digit number with a 1-digit number, you must use a leading 0 with the 1-digit number. For example, if the value shown is 12 and you want to change the value to 9, overstrike the "12" with "09". The system will drop the leading 0 on subsequent displays.
- ❑ If the field displayed includes double quotes, be sure the replaced field also has double quotes.
- ❑ If you type an illegal character, the terminal will flash or ding (depending on the terminal characteristics).
- ❑ If you make a mistake on an entry, delete the item and enter it again. The new line will be inserted on the line above where the cursor is; it will be sorted alphabetically by the Comment column (second column from the left) when **sysgen** updates files.
- ❑ If the arrow keys do not work right for moving around in **sysgen**, you may be using the wrong \$TERM definition for your terminal or the cursor control in the *terminfo* file you are using may not match how the terminal is set up. Using the **j**, **h**, **k** and **l** keys to move around the screen is generally more reliable than using the terminal's arrow keys.

- As part of its initialization, **sysgen** reports the directory it is using. Normally, this should be the `/usr/src/uts/realix/sysgen` directory. It is possible to create separate **sysgen** trees if necessary. In this case, use the `-l` option to specify the tree to use.

A Walk Through the Screens

Interactive **sysgen** is a hierarchy of screens that provide access to various entities:

- Configuration Screen* The first screen displayed lists the configurations that are available. On the released system, **Standard** is the only configuration provided. Each configuration corresponds to a tree under the `/usr/src/uts/realix/cf` directory and is controlled by a file in the `/usr/src/uts/realix/cf/machines` directory.
- Collection Screen* The *Collection Screen* lists all collections that are available on the system. A collection is usually the driver (although some non-driver entities also have a collection) with all information required to configure it, including devices controlled by the driver, tunable parameters, etc. Each collection corresponds to a file in `/usr/src/uts/realix/cf/descriptions`. Lines that appear with an asterisk (*) to the left are currently enabled. Administrators can enable and disable *Collections* lines on this screen. See pages 9-7 and 9-7 for information on the various *Collections* that are included in the released system and guidelines on what can be enabled and disabled.
- Item Screen* The *Item Screen* lists all the items that are included in the *Collection*. Lines that appear with an asterisk (*) to the left are currently enabled. Administrators can enable and disable *Devices* lines on this screen, or open the *Parameter Screens*.
- Parameter Screen* Each *Parameter Screen* lists one tunable parameter, its default value, and its current value. Tunable parameters control what resources are allocated to which structures, the maximum number of a certain type of structure that can be allocated per process or for all processes on the system, and other characteristics of the operating system (such as the clock resolution used for process interval timers, the priority or frequency of system daemon execution, and so forth).

To move between screens, use the **o** (open) key to open a screen that is below another screen; use the **q** (quit) screen to move back up the screen hierarchy.



sysgen allows you to make any number of changes in one session. Making a large number of changes in one session saves the time of several reboots, but may also make it more difficult to determine the cause of a failure in either **sysgen** or the reboot. Balance these considerations carefully.

Rebuilding the Operating System

After making modifications to **sysgen** values, you must build a new bootable kernel image that includes those values and reboot the system to have these values in effect. When you type a **q** at the *Configurations Screen*, **sysgen** displays four questions (prompts). Table 9–2 lists these four questions with the files that each updates if you respond **y** to the prompt.

Table 9–2. Updating Files

Question asked:	Files updated by y answer:
Save changes to configuration? [y/n]	<i>descriptions</i> file for the collections modified
Update system configuration files? [y/n]	<i>master</i> , <i>dfile</i> , and other intermediate files used in the kernel build process
Rebuild the operating system? [y/n]	The operating system is built as <i>/usr/src/uts/realix/cf/realix</i> . If the answer to the fourth question is y , the bootable kernel image is copied to <i>/stand/realix</i> .
Install the new operating system to be used on the next reboot? [y/n]	<i>/stand/realix</i> is overwritten

The */realix* file contains the old bootable kernel image until you successfully boot from */stand/realix* and enter the multi-user mode. At that time, */realix* is copied to */oldrealix* and */stand/realix* is relinked to */realix*.

Rebooting the Operating System

To activate the driver and its associated devices, halt and reboot the operating system by typing **shutdown -i6** at the console.

If the system will not reboot, reboot the system from the old kernel file. */oldrealix* is the name of the previous executable kernel that **sysgen** saved automatically, or use the file you saved under a different name (such as */realix.good*).

Creating New Configurations

It is sometimes useful to create alternate configurations, for reasons such as the following:

- ☐ You want to retain an unmodified copy of the released configuration for reference.
- ☐ You need to support different configurations during regular work hours and off hours, perhaps because someone is doing testing during off hours, or because the application requires different devices and/or tunable parameter values for the work done during off hours.
- ☐ You need different tunable values for running the debug and production kernels.

To create duplicate *Configurations*:

- ☐ Move the cursor to the *Configuration* you want to duplicate and type **D**.
- ☐ The cursor will be in the right column of the display for the new configuration; type the short name of the configuration (this will be the name of the new directory) and press RETURN.
- ☐ The cursor will be in the left column of the display; type **c**, the new description of the configuration, and press RETURN.
- ☐ Open the new configuration and alter it as required.

Creating a new configuration creates a new file under the */usr/src/uts/realix/cf/machines* directory with copies of the files that are different from those in the *standard* directory. In most cases, extra configurations consume a small number of disk blocks.

Enabling and Disabling Collections and Devices

It may be necessary to enable and disable collections or individual devices that are part of a collection for reasons such as the following:

- ❑ You are adding or removing hardware from the system. The *Collection* file for most standard drivers includes definitions for the maximum number of boards that could be configured. When adding a board to the system, you only need to enable the description for the device, rather than having to define it.
- ❑ The released system includes drivers that are not being used. In most cases, the drivers can be left in the configuration, but you can regain some memory by removing them.
- ❑ If you think you have a problem with one of your boards, you can try to isolate the problem by configuring the system without that device. This may also enable you to run a stable system (minus that board) until the hardware can be repaired.

Enabling and Disabling Collections

Collections are enabled and disabled on the *Collections* screen. The asterisk (*) to the left indicates that a *Collection* is enabled, meaning it will be included in the configured system. Use the **c** key to toggle between enabled and disabled. Move the cursor until the arrow-cursor points at the collection to be enabled or disabled, following the instructions on page 9-2.

When disabling a device, the following message appears at the bottom of the screen:

De-select this item and its sub-items? [y/n]

Respond **y** to disable the collection.



*Do not use the **-d** option to **sysgen** when enabling and disabling collections. When **sysgen** is executed with the **-d** option, no asterisks appear to the left to indicate enabled collections, and the **c** key cannot be used for toggling between enabled and disabled.*

Table 9-3 lists the optional standard REAL/IX drivers that are included in the system and the conditions under which they can be removed from the configuration.

Table 9–3. Collections that Can be Disabled

Driver Name	Description	Explanation
<i>kdb</i>	Selects optional kdb debugger	Can be deconfigured if not used
<i>msg</i>	Message Parameters	Can be deconfigured or tuned down if not used
<i>nvr</i>	Non-volatile RAM	Can be deconfigured in most cases
<i>sem</i>	Semaphore Parameters	Can be deconfigured or tuned down if not used
<i>shm</i>	Shared Memory Parameters	Can be deconfigured or tuned down if not used
<i>pty</i>	Pseudo TTY devices	Can be deconfigured if no STREAMS are configured
<i>strbufs</i>	Stream buffers	
<i>streams</i>	Stream handlers	
<i>sshm</i>	Special shared memory used for A24 address space allocation	Can be deconfigured if not used
<i>sxt</i>	Shell Layers	Can be deconfigured if not used

Table 9–4 describes the standard REAL/IX drivers that cannot be disabled.

Table 9–4. Collections that Cannot be Disabled

Driver Name	Description	Explanation
<i>aio</i>	Asynchronous I/O Parameters	Required collection**
<i>atcpu</i>	Generic CPU Support	Contains required parameters
<i>bs</i>	Binary Semaphore Parameters	Required collection**
<i>buildparams</i>	Parameters Controlling System Build	Required to build
<i>cintr</i>	Connected Interrupt Parameters	Required collection**
<i>evt</i>	Events Parameters	Required collection**
<i>filsys</i>	File System Handlers	Required for file access
<i>kernel</i>	Kernel and Paging Parameters	Required to configure the kernel
<i>kprof</i>	Kernel Profiler	Always configured
<i>misc</i>	Miscellaneous System Declarations	Required for operating system
<i>sysdev.at</i>	System Devices for booting (root, pipe, and swap)	Must be configured to boot the system
<i>timer</i>	Timer Parameters	Required collection**

** If these mechanisms are not being used by any executing processes, the tunable parameters in these collections can be tuned down to reclaim memory. This is discussed in Chapter 6.

Enabling and Disabling Devices

To enable or disable a device:

1. Open the *Items Screen* for the appropriate collection.
2. Use the **j** and **k** keys to move the arrow cursor to the line of the device to be enabled or disabled.
3. Enabled devices show an asterisk (*) in the left column. Use the **c** key to toggle between enabled and disabled.



Devices must be enabled in numerical order. In other words, do not enable device #3 in a certain collection unless devices #1 and #2 are also enabled.

*Do not disable any other entities listed on the Items Screen. The entire collection can be disabled from the Collections Screen. Disabling the Driver or other entities when associated Devices are configured may cause **sysgen** to fail or, if not, could corrupt the kernel.*

Using the general guidelines above, you may wish to enable or disable a system-specific item; however, we recommend that you not modify any of the settings. Table 9-5 lists some of these items.

Table 9-5. System-Specific Collections

Item	Description
Cpu Board	Defines CPU-specific information such as the board type, CPU type, prefixes for the clock handler and board-level entry routines, console major/minor number, and various vectors and functions supported by the listed CPU.
Filesystem	File system description.
System Devices	Specification of the logical devices that are used for necessary system functions.
Processor	Microprocessor identification declaration.
Probe	Describe how specific register locations are to be accessed and/or initialized at system startup.
Memory Config	Portray system memory so that the system may properly initialize and allocate memory at system startup.

Devices Screens can be opened to see the Interrupt Vector Location, Address, Bus Request Level, and Number of Devices (maximum) for the device. To modify the values of these fields, run **sysgen -d**. See the *Driver Development Guide* for field definitions. Each configured device must have a unique Address and Interrupt Vector Location (unless a Multiple Handler is also defined).

sysgen will fail if you attempt to improperly enable devices that share an address or interrupt vector location, although it may not detect overlapping addresses with different start locations. If you have

installed drivers on your system that are not part of the released operating system, it is possible to run into address and vector conflict problems. If so, contact the organization that provided the driver or MODCOMP Customer Support for assistance. Do not attempt to modify the fields on the *Devices Screen* if you do not fully understand the specifications of the device.

You can also open the *Driver* screen to view how the driver is configured. The value of these fields can be changed only when running **sysgen -d**.

Modifying Tunable Parameters

Tunable parameters enable the administrator to "customize" the system environment to match the needs of the site. Chapter 6 describes the standard tunable parameters that the administrator can change and gives guidelines for determining the appropriate values. The following discussion describes only the mechanics of changing the value of a tunable parameter, not how one determines that a value needs to be modified.

To modify a tunable parameter:

1. Open the *Collection* screen that contains the tunable parameter you want to modify. Use the **j** key to move the cursor over the line that reads "Parameter" in the right column, and open that screen.
2. Use the **j** key to move the arrow cursor to the bottom line on the screen ("Value:").
3. Type **c**, the new value, and press RETURN.
4. Type **q** to exit the *Parameter Screen*.



*Do not modify the value of any tunable parameter without first reading the information about that parameter in Chapter 6. A number of these tunable values have ramifications and dependencies that are not obvious from the brief description given on the **sysgen** screen.*

10. Job Accounting

Chapter 10

Job Accounting

The job accounting facility collects data on system usage by user and by process. Specifically, job accounting records connect sessions, monitors disk usage, and charges fees to specific logins. To help you access this data, a number of C language programs and shell scripts are provided that reduce this accounting data into summary files and reports.

Job accounting is required if you are billing users for computer usage. It also provides information for performance analysis and helps identify hardware problems on the TTY lines configured on the system. Job accounting is an optional feature; as released, the raw statistics (which are also used by other system processes) are gathered, but the scripts that process this data into accounting reports are turned off.

This chapter gives an overview of the job accounting system, how to set it up, and how to use it. For more detailed information on individual files and commands, see the appropriate manual pages.

Overview of Job Accounting

The job accounting system consists of a number of commands, shell scripts, and files. Once you understand the overview of how the system works, you can get the details from the manual pages and by reading the specific shell scripts that are referenced below.

The accounting system has three main phases:

1. During normal system operations (multi-user mode), raw system usage statistics are gathered into a set of files.
2. Once a day (during non-prime hours) these raw statistics are summarized for the day by the **dodisk(1M)** and **runacct(1M)** scripts, and daily accounting reports are printed.
3. Once a month (or once every fiscal period), the cumulative statistics are summarized and printed into monthly accounting reports, including user bills (if set up to do so).

These phases are summarized here; for more details, see the manual pages for the accounting files, commands, and shell scripts in the */usr/lib/acct* directory.

Gathering Raw Statistics

Two main files, *wtmp* and *pacct?*, provide the raw statistics for the connect-time accounting data. These are supplemented by the *fee* file into which administrators can write "special" charges, and the *disktmp* file into which the **dodisk(1M)** script (run overnight by **cron**) writes current disk usage statistics once a day. Table 10-1 summarizes these files that provide the raw accounting data.

Table 10-1. Raw Accounting Data

File	Format	Information	Written By
<i>/etc/wtmp</i>	<i>utmp.h</i>	connect sessions	login(1) , init(1M)
		date changes	date(1)
		reboots	acctwtmp(1M)
		shutdowns	shutacct(1M)
<i>/usr/adm/pacct?</i>	<i>acct.h</i>	active processes	kernel (when process terminates) turnacct(1M) creates a new file when the old one reaches 1000 blocks
<i>/usr/adm/fee</i>		special charges	chargefee (run by administrator to supply charges)
<i>/usr/adm/acct/nite/diskacct</i>	<i>acct.h</i>	disk usage charges	acctdisk(1M) run by dodisk(1M)

Note the following:

- ❑ The */etc/wtmp* file contains information on the run-state of the system; it is a non-ASCII file whose contents can be viewed with the **who -a** command. The system automatically transfers appropriate summary information to the */etc/wtmp* file.
- ❑ The owner and group of the */etc/wtmp* file must be *adm*, and the access permissions must be 664. If you are running accounting, the **runacct(1M)** script controls the size of */etc/wtmp*; otherwise, this file should be cleared or truncated periodically (either manually or by scripts run by **cron** or **rc2**). To truncate the file, use the **tail** command; see **tail(1)** for more information.

Alternatively, the file can be deleted, then recreated with the **> /etc/wtmp** command.

- ❑ If you are running accounting, the **ckpacct(1M)** script controls the size of the */usr/adm/pacct* file, and the **runacct** script deletes old files. If you are not running accounting, you must clean these files out manually, using the same procedures described above for *wtmp*. Even if you are not running accounting, you may want to let **ckpacct** run regularly, to avoid the performance degradation that can occur if the *pacct* file becomes too large.
- ❑ Rather than manually executing **chargefee(1M)**, most administrators write a shell script that includes codes for the various billable services. The operator can then supply a code that identifies the service rendered, and the system tabulates the charge.

Daily Summaries

The **runacct**(1M) script runs overnight to summarize the raw accounting statistics, print daily accounting files, and merge the daily summary information into the cumulative accounting files that will be processed by **monacct**(1M). You can add additional scripts to **runacct** to perform special accounting functions required by your installation.

Monthly Summaries

The **monacct**(1M) script prints monthly reports from the cumulative accounting statistics, then cleans out the cumulative files. You can add additional scripts to the **monacct** to produce user bills and perform special accounting functions required by your installation.

Setting Up Job Accounting

The system is released with job accounting turned off. To turn it on, take the following steps:

1. Sign on the system as superuser.
2. Check that the `/etc/init.d/acct` script is linked to a script in the `/etc/rc2.d` and `/etc/rc0.d` directories. This script turns accounting on when the system goes to multi-user state and turns it off when the system leaves multi-user state.
3. Edit the `/usr/spool/cron/crontabs/adm` file to remove the `#` symbol; this activates the four command lines (the `#` comments them out). You may want to modify the time these scripts run or make other modifications to this file:
 - **ckpacct** should run at least once an hour (it ensures that the `pacct` file does not get too large and turns off the accounting system if `/usr` runs out of space).
 - **dodisk** must run before **runacct** to ensure that the disk usage statistics are available for **runacct**. You may also want to add arguments to the **dodisk** line to specify which file systems should be used, and to specify that accounting is to be done by login ID rather than file system. See **dodisk**(1M) for details.
 - **monacct** must run after **dodisk** and **runacct** have completed, so that the statistics from the last day of the month are included in **monacct**. If you add a large number of extra programs to **runacct**, the hour and fifteen minutes provided may be inadequate. If you want to run **monacct** on some schedule other than monthly, adjust the time in this file.
4. Execute `su adm`, go to the `/crontabs` directory (`cd /usr/spool/cron/crontabs`), and execute `crontab adm`.

5. Modify the `/usr/lib/acct/holidays` file to identify appropriate prime and non-prime time for your site. The first line contains three fields that identify the year, beginning of prime time, and end of prime time. For instance, to establish that this is 1993 and that prime time is from 8:00 AM to 5:00 PM, the first line should be:

```
1993      0800      1700
```

This line is followed by a series of lines that identify company holidays (days when the entire day is considered non-prime time). The format is:

```
Numbered-day-of-year  Month  Day  Description of Holiday
```

The *Numbered-day-of-year* is a number in the range of 1 through 366 (January 1 is "1"; December 31 in a leap year is "366") indicating the day for the corresponding holiday (leading white space is ignored). The other three fields are informative commentary and are not used by the accounting programs. So, you can use any format to indicate the month and day and any meaningful description of the holiday.

Table 10-2 lists common legal holidays for the United States.

Table 10-2. Legal Holidays in the United States

Day of Year	Month	Day	Holiday Name
01	01	01	New Year's Day
Varies	01	Varies	Martin Luther King's Birthday
Varies	02	Varies	President's Day
Varies	05	Varies	Memorial Day
185 (186 Leap Year)	07	04	Independence Day
Varies	09	Varies	Labor Day
Varies	11	Varies	Veteran's Day
Varies	11	Varies	Thanksgiving Day
359 (360 Leap Year)	12	25	Christmas Day

Job Accounting Scripts

When job accounting is active, **cron** runs four main scripts¹ that manipulate the accounting files and produce the accounting reports. These are described here briefly; see the appropriate manual pages for more detailed information.

ckpacct

The **ckpacct**(1M) script performs routine monitoring of the job accounting system. By default, it runs on the hour every hour of every day. It performs two main tasks:

- Every time it executes, it checks the size of the process accounting file, */usr/adm/pacct*. When this file is over 1000 blocks (you can specify a different threshold on the command line in the *crontabs/adm* file), it shuts down accounting, copies the *pacct* file to a sequentially-numbered *pacct* file (*pacct1*, *pacct2*, and so forth), then restarts job accounting. Since this file is updated every time a process terminates, letting it get too large can seriously degrade system performance.
- When **ckpacct** finds that the */usr* file system has less than 500 free blocks, it shuts down job accounting. Once */usr* has more than 500 free blocks, it restarts job accounting.

dodisk

The */usr/lib/acct/dodisk*(1M) script does daily accounting of disk usage. By default, it runs at 2:00 AM every day, so that it is completed before **runacct** starts.

dodisk calls **diskusg**(1M) to summarize the number of disk blocks in use; by default, it summarizes by file system according to the file systems listed in the */etc/fstab* file. By modifying the command line in the *crontabs/adm* file, you can specify the **-o** option (this allows accounting by login directory); in most cases, you should also list the file systems for which disk accounting should be done to override the default. Unless you have user logins in the *root* and */usr* file systems, it does not make sense to run disk accounting on these file systems, and you may have other file systems that should not be included.

dodisk then calls **acctdisk**(1M) to merge the information from **diskusg** with information from the */etc/passwd* file into a */usr/adm/acct/nite/diskacct* file that is used by **runacct**.

¹ These are controlled by the */usr/spool/cron/crontabs/adm* file

runacct

The `/usr/lib/acct/runacct(1M)` script is the daily accounting program. By default, it runs at 4:00 AM every day. **runacct** processes the raw data files that contain information on process accounting, disk usage, fees charged, and connect time. It also prepares daily and cumulative summary files for use by *prdaily* or for billing purposes.

The **runacct** script has a series of protection mechanisms that attempt to recognize an error, provide intelligent diagnostics, and terminate processing in such a way that **runacct** can be restarted with minimal intervention.

- ❑ **runacct** records its own progress by writing descriptive messages into the file *active*.¹
- ❑ All diagnostic messages output during the execution of **runacct** are written into a file called *fd2log*.
- ❑ To prevent multiple invocations (which could happen if two **cron** daemons are spawned), **runacct** will fail if the files *lock* and *lock1* exist when it is invoked.
- ❑ The *lastdate* file contains the month and day **runacct** was last invoked and is used to prevent more than one execution per day.
- ❑ If **runacct** detects an error, it writes a message to the console and the */usr/adm/pubuf* file, sends mail to *root* and *adm*, removes locks, saves diagnostic files, and terminates.

monacct

The `/usr/lib/acct/monacct(1M)` script is the monthly accounting program. By default, it runs at 5:15 AM on the first day of every month. It works on the */usr/adm/acct/sum* files that have been accumulated throughout the previous month, then clears these files and restarts them for the new month. **monacct** first calls **prtaacct(1M)** to format the total accounting files (*tacct*), then calls **acctcms(1M)** to create the monthly command summary reports.

¹ Files used by **runacct** are assumed to be in the */usr/acct/adm/nite* directory unless otherwise noted.

Job Accounting Data Files

The job accounting system includes a large number of files. Earlier in this chapter, the raw data files were discussed. The following sections explain the files that are created for job accounting, including raw files, daily summaries, monthly summaries, and fiscal year summaries.

The *adm* login (UID 4) owns the data collection files, which are located in subdirectories of the */usr/adm* directory. The */usr/adm/acct* directory contains some of the raw data files, plus three subdirectories:

- *nite* contains files that are reused daily by **runacct**(1M)
- *sum* contains cumulative daily and monthly summaries, created by **runacct**(1M) and reported by **monacct**(1M)
- *fiscal* contains fiscal summaries, created by **monacct**.

Files in */usr/adm* Directory

The */usr/adm* directory contains some raw accounting files, plus some temporary files used by the **dodisk**(1M) and **runacct** scripts to produce daily accounting reports.

Table 10-3. Files in the */usr/adm* Directory

File	Contents
<i>diskdiag</i>	diagnostic output during execution of disk accounting programs
<i>dtmp</i>	output from the acctdusg (1M) program
<i>fee</i>	ASCII accounting records for special charges, created by chargefee (1M)
<i>pacct</i>	current process accounting file
<i>pacct?</i>	process accounting files from earlier in the day. ckpacct (1M) creates these so the active file does not exceed 1000 blocks.
<i>Spacct?.MMDD</i>	process accounting files for <i>MMDD</i> (month and day) during execution of runacct

Files in nite Directory

The `/usr/adm/acct/nite` directory contains files created and used by **runacct**.

Table 10-4. Files in /usr/adm/acct/nite Directory

File	Contents
<i>active</i>	used by runacct to record progress and print warning and error messages. <i>activeMMDD</i> same as <i>active</i> after runacct detects an error.
<i>cms</i>	ASCII total command summary used by prdaily
<i>ctacct.MMDD</i>	connect accounting records in <i>tacct.h</i> format
<i>ctmp</i>	connect session records, output of acctcon1 program
<i>daycms</i>	ASCII daily command summary used by prdaily
<i>daytacct</i>	total accounting records for one day
<i>fd2log</i>	diagnostic output during execution of runacct
<i>ldstate</i>	last day runacct executed in <i>date+ %m%d</i> format
<i>lock, lock1</i>	used to control serial use of runacct
<i>lineuse</i>	terminal line usage report used by prdaily
<i>log</i>	diagnostic output from acctcon1
<i>logMMDD</i>	same as <i>log</i> after runacct detects an error
<i>reboots</i>	contains beginning and ending dates from <i>wtmp</i> and a listing of reboots
<i>statefile</i>	used to record current state during execution of runacct
<i>tmpwtmp</i>	<i>wtmp</i> file corrected by wtmpfix (1M)
<i>wtmperror</i>	place for wtmpfix error messages
<i>wtmperrorMMDD</i>	same as <i>wtmperror</i> after runacct detects an error
<i>wtmp.MMDD</i>	previous day's <i>wtmp</i> file

Files in sum Directory

The `/usr/adm/acct/sum` directory contains files created by the daily accounting programs to be used for monthly accounting.

Table 10-5. Files in /usr/adm/acct/sum Directory

File	Contents
<code>cms</code>	total command summary file for current fiscal period
<code>cmsprev</code>	command summary file without latest update
<code>daycms</code>	command summary file for yesterday
<code>loginlog</code>	created by <code>lastlogin(1M)</code>
<code>pacct.MMDD</code>	concatenated version of all <code>pacct</code> files for <code>MMDD</code> , removed after reboot
<code>rprtMMDD</code>	saved output of <code>prdaily</code> program
<code>tacct</code>	cumulative total accounting file for current fiscal period
<code>tacctprev</code>	same as <code>tacct</code> without latest update
<code>tactMMDD</code>	total accounting file for <code>MMDD</code>
<code>wtmp.MMDD</code>	saved copy of <code>wtmp</code> for <code>MMDD</code> , removed after reboot

Files in fiscal Directory

The `/usr/adm/acct/fiscal` directory contains summary files for the fiscal period, created by the `monacct` script.

Table 10-6. Files in /usr/adm/acct/fiscal Directory

File	Contents
<code>cms?</code>	total command summary file for fiscal ?
<code>fiscrpt?</code>	report similar to <code>prdaily</code> for fiscal ?
<code>tacct?</code>	total accounting file for fiscal ?

Troubleshooting the Accounting System

Unfortunately, the accounting system is not entirely foolproof. A file will occasionally become corrupted or lost. Some of the files can be ignored or restored from the `filesave` backup. However, certain files must be fixed in order to maintain the integrity of the accounting system. If the file corruption caused `runacct` to fail, you can restart `runacct`; see `runacct(1M)` for more information.

Fixing wtmp Errors

The *wtmp* files seem to cause the most problems in the day-to-day operation of the accounting system. When the date is changed and the operating system is in multi-user mode, a set of date change records is written into */etc/wtmp*. The **wtmpfix(1M)** program is designed to adjust the timestamps in the *wtmp* records when a date change is encountered. However, some combinations of date changes and reboots will slip through **wtmpfix** and cause **acctcon1** to fail. The following steps show how to patch up a *wtmp* file using the **fwtmp(1M)** command:

```
cd /usr/adm/acct/nite
fwtmp < wtmp.MMDD > xwtmp
ed xwtmp
    delete corrupted records or delete all records up to the date change
fwtmp -ic < xwtmp > wtmp.MMDD
```

fwtmp used without options converts the binary *wtmp* file into an ASCII file that can be edited; used with the **-ic** option, **fwtmp** converts the ASCII file into a binary *wtmp* file.

If the *wtmp* file is beyond repair, create a null *wtmp* file. This prevents any charging of connect time. **acctprc1** cannot determine which login owned a particular process, but it will be credited to the login that is first in the password file for that user ID; if you use a unique UID for each login ID, this will be accurate.

The *wtmp* file collects the data on terminal lines reported in the Daily Report discussed later in this chapter. If one or more terminal lines develop hardware problems, this file may grow very rapidly. In this case, run **acctcon(1M)** to determine which terminal line is causing all the noise, and disable that line by changing the corresponding **getty** line in the */etc/inittab* line to **off** until the line can be repaired. See Chapter 7 for more information on the **getty** lines.

Fixing tacct Errors

If you are using the accounting system to charge users for system resources, the integrity of the *sum/tacct* file is quite important. Occasionally, unusual *tacct* records will appear with negative numbers, duplicate user IDs, or a user ID of 65,535. First, use **prtacct** to check *sum/tacctprev*. If it looks correct, patch the latest *sum/tacct.MMDD* file and then recreate *sum/tacct*. A simple patch procedure would be:

```
cd /usr/adm/acct/sum
acctmerg -v < tacct.MMDD
vi xtacct
    remove the bad records and write duplicate UID records to another file
acctmerg -i < xtacct > tacct.MMDD > tacct
```

Remember that the **monacct(1M)** procedure removes all the *tacct.MMDD* files; therefore, *sum/tacct* can be recreated by merging these files together.

Interpreting Job Accounting Reports

The **prdaily(1M)** script uses the information tabulated by **runacct** and generates four reports:

- ❑ **Daily Report** shows line utilization by TTY number.
- ❑ **Daily Usage Report** indicates usage of system resources by users (listed in UID order).
- ❑ **Daily Command Summary** indicates usage of system resources by commands, listed in descending order of memory usage. In other words, the command that used the most memory for all runs during the day is listed first.
- ❑ **Last Login** shows the last time each user logged in (arranged in chronological order, with the login ID that has been inactive the longest listed first).

The monthly accounting stream, **monacct(1M)**, produces monthly summary reports similar to those produced daily. **monacct** also summarizes the accounting information into the files in the */usr/adm/acct/fiscal* directory. This information can be used to generate monthly billing.

Daily Report

This report gives information about each active terminal line on the system. The **from/to** lines indicate the period covered by the report. It is followed by a log of system reboots, shutdowns, power fail recoveries, and other system occurrences.

The second part of the report is a breakdown of line utilization. **TOTAL DURATION** tells the number of minutes the system was in multi-user state (accessible through the terminal lines). The columns are:

LINE	The terminal line or access port.
MINUTES	The total number of minutes the line was in use during the accounting period.
PERCENT	The total number of MINUTES the line was in use divided by the TOTAL DURATION.
# SESS	The number of times this port was accessed for a login(1) session.
# ON	The number of times the port was used to log on a user. This number should be identical to # SESS.
# OFF	The number of times a user logged off, the number of times the BREAK key was pressed, and any interrupts that occurred on that line. Generally, interrupts occur on a port when the system is brought to multi-user state and getty(1M) is invoked. When the # OFF is more than three times the # ON, it may indicate that the port, modem, or cable is going bad. It can also indicate that there is a bad connection somewhere, such as an unconnected cable dangling from the port.

Daily Usage Report

This report gives a breakdown of system resource utilization by user. The columns contain the following information:

UID	User ID
LOGIN NAME	Login name of the user. A single user ID may have more than one login ID.
CPU (MINS)	The amount of time the user's processes used the CPU. This category is broken down into PRIME and NPRIME (nonprime) utilization. The file <i>/usr/lib/acct/holidays</i> determines how these are defined.
KCORE-MINS	The total amount of memory used by processes executed under this login. This amount represents kilobyte segments of memory used per minute; PRIME and NPRIME amounts are also broken out.
CONNECT (MINS)	Clock time used. In other words, the amount of time that a user was logged into the system. If this time is rather high and the column #OF PROCS is low, it usually indicates a person who logs in first thing in the morning and rarely touches the terminal the rest of the day.
DISK BLOCKS	The average amount of disk storage space in use for that user, usually the amount in use when ddisk ran.
# OF PROCS	The number of processes invoked by the user. Large numbers in this column may indicate that the user had a faulty shell procedure (e.g., a crontab entry trying to execute a user's <i>.profile</i> that prompts the user for information).
# OF SESS	Number of times the user logged into the system.
# DISK SAMPLES	Number of times disk accounting was run to obtain the average number of DISK BLOCKS listed earlier.
FEE	Total amount of special charges made against this account with chargefee . This field is often unused.

Daily Command Summary

This report shows the system resource utilization by command. With this report, you can identify the commands used most frequently and, based on how those commands use system resources, gain insight on how to tune the system. A similar report is produced monthly. See Chapter 6 for more information on tuning the system.

This report is sorted by TOTAL KCOREMIN, which is an arbitrary but useful yardstick for calculating the "drain" on a system.

COMMAND NAME	Name of the command. Unfortunately, all shell procedures are lumped together under the name <code>sh</code> because only object modules are reported by the process accounting system.
NUMBER CMDS	Total number of invocations of this particular command.
TOTAL KCOREMIN	Number of kilobyte segments of memory used by the process per minute of run time.
TOTAL CPU-MIN	Total processing time this program has accumulated.
TOTAL REAL-MIN	Total wall-clock minutes this program has accumulated.
MEAN SIZE-K	Mean of the TOTAL KCOREMIN divided by the number of invocations reflected by NUMBER CMDS.
MEAN CPU-MIN	Mean of the TOTAL CPU-MIN divided by the NUMBER CMDS.
HOG FACTOR	Total CPU time divided by the elapsed time (TOTAL REAL-MIN). This shows the ratio of system availability to system utilization, which gives a relative measure of the total available CPU time consumed by the process during its execution.
CHAR TRANSFD	This column, which may be negative, is the total number of characters transferred by <code>read(2)</code> and <code>write(2)</code> system calls.
BLOCKS READ	A total of the "reads" and "writes" in physical blocks for that process.



11. Administering Internet Protocols

Chapter 11

Administering Internet Protocols

The REAL/IX Operating System supports the Internet networking protocols running over an Ethernet Local Area Network (LAN) controller. This chapter provides information for setting up and maintaining a REAL/IX Internet network on an Ethernet LAN. The following topics are discussed:

- ❑ **Addressing** – Discusses how addresses are determined and how they are used by Internet software.
- ❑ **Network Database Requirements** – The database files that need to be updated to reflect your own network environment. They are used to match symbolic names to those used by lower levels in the system, and for interhost validation.
- ❑ **Interhost Accessing** – Introduces network user access considerations.
- ❑ **UUCP over TCP/IP** – Discusses how to configure UUCP for use over an Ethernet LAN using TCP/IP.

Addressing

Every host on the network can be identified by its Internet address, Ethernet address, or host name. Under normal use, an administrator does not need to be concerned with Ethernet addresses; however, the information is provided here for completeness. The following sections describe how each of these is used in Internet network operations.

Internet Addresses

The Internet protocols arose from a need to connect different networks together. "Internet" is a network of networks. Although TCP/IP protocols were developed for this complex scenario, they have proven popular in cases where there is only a single local network.

Internet addresses are structured into two parts to allow for the interconnection of different networks. One part is a network number used to uniquely identify the network to which the machine is attached, so that it may be accessed from other networks attached to the Internet. The other part of the address is a host number, which must be unique for each host on the network.

If you are connecting the REAL/IX Operating System to an existing network, you need to assign the system a name and address. The address simply uses the already known network number, followed by a host number. The host number must be different from all other host numbers, so that it may uniquely distinguish your machine from all the other machines on the network.

If you are setting up the network for the first time, it is necessary to choose a network number. If you do not intend to connect the local network into the Internet, then the network number portion of the address can be chosen arbitrarily. If you do intend to connect to the Internet, then it is necessary for a unique network number to be assigned. The administration of these numbers is performed by SRI International.

Although it is normal to use only one network number for a particular physical network, it is possible to have multiple "logical" networks coexisting on the same physical network. In this case, each logical network has its own network number. All hosts on the network that wish to communicate directly with each other (without going through a gateway), must be on the same logical network.

Internet addresses are expressed in dotted decimal notation:

a.b.c.d

Each component of the address is expressed as a decimal number ranging from 0 to 255. Each component represents eight of the 32 bits in the Internet address. Part "a" is the most significant byte.

Networks can be one of three classifications, distinguished by the three high-order bits of their address:

- Class A networks have a 7-bit network number and a 24-bit host number. The highest order bit is always 0 (zero). Networks 1 through 127 are the only Class A networks. For example, host number 5 on a Class A network number of 89 is expressed as 89.0.0.5.
- Class B networks have a 14-bit network number and a 16-bit host number. The two highest order bits are set to 10 (binary). Class B network numbers range from 128.1 through 191.254. For example, host number 5 on a Class B network number of 128.18 is expressed as 128.18.0.5.
- Class C networks have a 21-bit network number and an 8-bit host number. The three highest order bits are set to 110 (binary). Class C network numbers range from 192.0.1 through 255.255.254. For example, host number 5 on a Class C network number of 192.5.41 is expressed as 192.5.41.5.

Table 11-1 summarizes how your network class affects your Internet network setup. Note that network fields and hosts containing all zeros or all ones are reserved and cannot be assigned to a real network or host.

Table 11-1. Network Classes

Class	Possible Number of Networks	Possible Hosts Per Network	Notation
A	126	16,777,214	n.h1.h2.h3
B	16,382	65,534	n1.n2.h1.h2
C	2,097,150	254	n1.n2.n3.h

Host Names

Hosts on a network are usually given names by which they can be referred to in user commands, instead of using network addresses. The host name is an alternative name for the Internet address and (like the Internet address) must be unique for each Ethernet interface.

One way to create a unique name for your system is to use names of fictional or mythological characters. If you have multiple controllers on your system, simply add a single digit to the end of the host name to specify which board you are using.

The host name can be up to 128 characters long. Uppercase and lowercase letters, digits, and hyphens are allowed. Aliases can also be specified.

Host names and their aliases are mapped to their Internet addresses in the */etc/hosts* file as discussed in the following sections.

Ethernet Addresses

Each node on an Ethernet has its own Ethernet address. Ethernet packets can be sent to a particular node via its specific Ethernet address, to all nodes via a special "broadcasting address," or to a collection of nodes via a multicast address.

In normal operation, Ethernet packets are sent to specific addresses. The node hardware at all the other nodes will ignore such packets. In contrast, broadcast packets are picked up by the node hardware at all nodes. This causes processing overhead at all nodes even though only one or two nodes may actually need to look at the packet.

The Ethernet address is six bytes long. This length was chosen so that every Ethernet node that has ever existed could have a unique address. This address is a property of the hardware, built in when the hardware is manufactured. If for example, the Ethernet controller is replaced with a different Ethernet controller, then the Ethernet address of the host will change.

Mapping an Internet Address to Its Link Level Address

In order to send data to a particular host, it is necessary to find that host's Link Level (Ethernet) address. This is a general problem when using the TCP/IP protocol suite, not one that is limited to Ethernet networks.

The solution for Ethernet is the "Address Resolution Protocol" (ARP). What happens here is that a node broadcasts to all nodes. The packet contains the sender's Link Level address, the sender's Internet address, and the target host's Internet address. The target host is one of the many hosts that receives the packet. It returns a packet to the original sender, with the reply packet giving the target's Ethernet address.

The Address Resolution Protocol decouples hosts from needing to have built-in knowledge of Ethernet addresses. Ethernet addresses and ARP can be regarded as low level implementation details that need not be dealt with when setting up a network.

The Network Database

The following is a description of the Internet database files. Each of these files consists of simple lines of text. They may be updated or modified using any REAL/IX text editor (e.g., vi editor). Comments may be added to any of these files by entering a line with a # as the first character. Database files are used by the network for mapping symbolic names to actual values. For example, the */etc/hosts* file is used to map host names to Internet addresses.

The *tcip/addrmgmt* subcommands (*addaddr*, *deladdr*, *showaddr*) of *sysadm(1M)* can be used to maintain the database files. However, it is better to access the files directly through *vi(1)* or an equivalent text editor, so that meaningful comments can be recorded.

The */etc/hosts* File

The */etc/hosts* file is used by the network processes to translate a host name into its Internet address. Each file entry includes the host Internet address, host name, and any aliases. All the end-user network applications use */etc/hosts*. In addition, a set of subroutines is also provided so that locally written applications may make use of this facility. (For more information about these subroutines, refer to the *Programmer's Guide*.) Each entry has two required fields and an optional field, plus an optional comment:

```
internet_address host_name [alias] ... [#comment string]
```

The meanings of these fields are:

- internet_address* A 4-byte value specified in decimal, octal, or hexadecimal, using dot notation. Note that in the example below, octal values are preceded by a 0 (zero) and hexadecimal values by 0x (zero,x).
- host_name* A string of up to 128 characters uniquely identifying this host. You can use letters, numbers, and hyphens.
- alias* An optional alias for the named host. You can specify several aliases.

The */etc/hosts* file contains the names of all the hosts on the network with which the local system wishes to have contact, as well as the name of the host where this file resides. You may also include hosts that are currently nonexistent or that are on other networks accessible through gateways. Note that the loopback connection to the local host is specified by the "localhost" entry in the */etc/hosts* file, which always has the decimal address 127.0.0.1 (or its equivalent in octal or hexadecimal). The "localhost" entry must exist in the */etc/hosts* file.

Figure 11-1 shows some possible entries in the */etc/hosts* file. Network numbers are given in decimal, octal, and hexadecimal formats.

```
#
127.0.0.1          localhost
#
# Lab systems
#
130.40.9.1         zeus zoo
130.40.9.2         athena
130.40.9.3         hermes
130.40.9.4         mercury merc
130.40.9.50        Xdevelop xdevelop xdev
130.40.9.60        XStation1 xs1
130.40.9.70        XStation2 xs2
130.40.9.80        XStation3 xs3
#
```

```
#
0177.0.0.01       localhost
#
# Lab systems
#
0202.050.011.01    zeus zoo
0202.050.011.02    athena
0202.050.011.03    hermes
0202.050.011.04    mercury merc
0202.050.011.062   Xdevelop xdevelop xdev
0202.050.011.074   XStation1 xs1
0202.050.011.0106  XStation2 xs2
0202.050.011.0120  XStation3 xs3
#
```

```
#
0x7f.0x0.0x0.0x1  localhost
#
# Lab systems
#
0x82.0x28.0x09.0x01 zeus zoo
0x82.0x28.0x09.0x02 athena
0x82.0x28.0x09.0x03 hermes
0x82.0x28.0x09.0x04 mercury merc
0x82.0x28.0x09.0x32 Xdevelop xdevelop xdev
0x82.0x28.0x09.0x3C XStation1 xs1
0x82.0x28.0x09.0x46 XStation2 xs2
0x82.0x28.0x09.0x50 XStation3 xs3
#
```

Figure 11-1. Sample /etc/hosts Entries

The /etc/hosts.equiv File

The */etc/hosts.equiv* file is used, together with *.rhosts*, to control interhost access. It is used to validate communications between hosts when an r-series (**rlogin**(1), **rccp**(1), and **remsh**(1)) application is invoked (refer also to page 11-12). To allow two hosts to accept r-series requests from each other, each would have to list the other in its *hosts.equiv* file. For example, a user on Host A cannot **rlogin** to Host B without supplying a password, unless Host A is included in Host B's *hosts.equiv* file.

You may restrict equivalence between hosts using *hosts.equiv*. For example, while Host A may include Host B in its *hosts.equiv* file, Host B does not have to include Host A in its file. In this instance, Host B can invoke an r-series application to Host A, but the reverse is not allowed. You may further restrict equivalence by specifying a user along with a host name. For example, if in Host A's *hosts.equiv* file, you make the entry **mercury andy**, Host A will accept only r-series requests from user **andy** at host **mercury**. Requests from other users at host **mercury** will be refused.

Entry Format:

host [*user_name*]

Description:

host Name of the remote host from which local access will be allowed.

user_name Optional. Name of the user from a specified remote host who is allowed access to the local host.

Example:

```
zeus
athena
hermes
mercury andy
```

In this example, all users on the **zeus**, **athena**, and **hermes** remote hosts have access to the local host, while on host **mercury**, only user **andy** has access to the local host.



All users with the same name on different machines can log into any host in which the user has an account. For example, if two different users are named **andy** on machines A and B, and if one or both machines list the other in its *hosts.equiv* file, both users can log into the other's machine. To avoid this situation, use unique user account names on the network.

The .rhosts File

While */etc/hosts.equiv* can be used to permit host-to-host access, individual users can include a file named *.rhosts* in their login directory to specify account-to-account access privileges. This file is identical in format to */etc/hosts.equiv*. It is used by **rcp(1)**, **remsh(1)**, and **rlogin(1)** as described on page 11-12.

Entry Format:

host [user]

Description:

<i>host</i>	Name of remote system from which users can gain access.
<i>user</i>	Optional. Name of the user on the specified host permitted access to a local user's account.

Example:

```
XStation1 andy
XStation1 mike
Xdevelop jeff
zeus
```

In this example, the user on the local host permits users **andy** and **mike** on the remote host **XStation1**, and **jeff** on the remote host **Xdevelop**, access to his account on the local host. Since no name is given after the **zeus** entry, it is assumed that the user with the *.rhosts* file on the local host will be using the same user name to log in from the **zeus** system.

The /etc/services File

Details of this file are given here for completeness; you should not have to modify this file.

The Internet protocol suite allows for many different application programs on one machine to share access to the network. The method by which they are multiplexed onto the network is to have unique "port" numbers for each application. The TCP and UDP protocol packets include 2-byte fields for source and destination ports. In this way incoming packets can be routed to the correct application.

The */etc/services* file lists all the servers for local applications and their respective port assignments. Network applications use this file to determine the appropriate ports to use.

There is a conventional use of ports. Some "well known" port numbers are reserved and are used to address particular services on any host. The remaining port numbers are free for any local use. The numbering scheme reserves the first 1024 port numbers for fixed usage. A set of subroutines is provided to permit locally-written applications to make similar use of */etc/services*. See `getservent(3N)` for more information on these subroutines.

While you are free to make additions to */etc/services*, care should be taken before modifying existing entries. The entries in */etc/services* are used to determine the "well-known" ports used to supply network services. Altering these values will jeopardize the system's ability to communicate with other systems.

Entry Format:

service port/protocol [alias]...

Description:

service Name of service.

port/protocol *Port* is the port number assigned to this service. *Protocol* is either **tcp** for connection-based services or **udp** for message-based services.

alias Optional alias.

Interhost Access Control

While adding your host to a network does not create any new access control issues, it does broaden the user community. A thorough understanding of the tools available to the system administrator for controlling access from the network will assist in preventing potential problems. The following sections describe the various schemes used for network access control, and detail which of the applications provided use each scheme.

telnet

telnet(1) uses a simple access control mechanism. The scheme is to treat users logging in from the network identically to users logging on at local terminals. That is, when a user logs on from a terminal on a remote host, the normal system access mechanism at the remote host is in place, requiring the user to provide a user name and password just as if the user were logging on at a local terminal.

File Transfer Protocol (ftp)

The scheme used by **ftp**(1) is the same as **telnet**, with minor additions. Each incoming **ftp** user must log in using the user name provided by the administrator of that host; the user must have a password or the login attempt will fail. Furthermore, if the user name is listed in the */etc/ftpusers* file, the login will be denied. */etc/ftpusers* is intended to prevent misuse of accounts such as **uucp** which have valid logins but are not associated with any particular user. Note that the file */etc/ftpusers* is not configured as part of your system and if you wish to take advantage of the security it offers, you must create it with one of the system text editors. Finally, if there is an entry in the */etc/passwd* file for a user named **ftp**, it is possible to use either **ftp** or **anonymous** as a valid **ftp** login; any password will be accepted. Since unauthorized users can access your machine using the **ftp** login, its inclusion in the */etc/passwd* file is not recommended.

The r-series Commands

The r-series commands developed by the University of California at Berkeley (UCB) for communications between UNIX system hosts use an access control scheme which is both powerful and flexible. These commands (**remsh**(1), **rcp**(1), and **rlogin**(1)) all share a common mechanism based on unified administration of hosts and privileged TCP ports. By UCB convention, TCP port numbers less than 1024 are privileged and may only be used by the superuser.

When one of the r-series commands establishes a connection to another host, it sends the name of the user establishing the connection and the name to be used on the remote host system. The two names are usually the same. The remote host performs the following validation steps. If any validation fails, the connection will be closed with the message "Permission denied":

1. The remote host checks that the originating host has an entry in its */etc/hosts* file.

2. It then checks that the connection originates from a privileged port.
3. Next it checks that the login name to be used is valid on the remote system. That is, the name must be entered in its */etc/passwd* file.
4. Finally, the remote host verifies system equivalence. It examines the file */etc/hosts.equiv* for an entry consisting solely of the originating system name. If one exists and the originating user name is the same as the name to be used remotely, validation is successful; if not, it looks for an entry consisting of both the originating system name and the originating login name. Failing that, it examines the file *.rhosts* in the home directory of the remote login name. If an entry is found consisting of the originating system name plus the originating login name, the validation succeeds. Refer to page 11-7 for a description of */etc/hosts.equiv* and page 11-8 for a description of the *.rhosts* file.

If the administrator includes an entry in the */etc/hosts.equiv* file specifying a remote user name with a host name in the */etc/hosts.equiv* file, only that remote user will be granted access to the local host (there may be many such entries for each host). All users of a host named in */etc/hosts.equiv* without any accompanying remote user name will be granted access, provided their login name is valid on the target system. In addition, each user can grant similar access to remote users of his or her name. Finally, when using the superuser password, the system does not check the */etc/hosts.equiv* file. In order for access to be granted to a remote superuser, an appropriate entry must exist in *.rhosts*.

The Internet Super-Server

When you bring up the network, a process called **inetd** is started. This is a user-level daemon sometimes called the "Internet super-server." If **inetd** is not up and listening for connection requests, connection requests from other hosts to the applications **inetd** controls cannot be answered.

The **inetd** daemon listens at a variety of predefined ports, determined at system startup by reading a configuration file, */usr/etc/inetd.conf*. When an application is invoked, it looks at the database file */etc/services* to find the service's predefined port number. When a connection request is made on a port on which **inetd** is listening, **inetd** executes the appropriate server program to handle the client. With this method, clients are unaware that an intermediary has played any part in the connection.

The server process spawned by **inetd** takes over administering the connection request while **inetd** returns to its state of listening for requests. Sometimes user connection requests are refused because **inetd** is in transition, in other words, between listens, while establishing a server process to handle a prior connection request. Once a connection is established, the server process performs the requested application activity (such as a file transfer request). After completion, the connection is closed and the server process disappears.

Establishing Connections

Network connections are handled by the network applications: **ftp(1)**, **telnet(1)**, **remsh(1)**, **rcp(1)**, and **rlogin(1)**. These applications were discussed earlier in this chapter.

There are two sides to each application: the client and the server (see Figure 11-4). For example, the **telnet** application software consists of two parts, the **telnet** client and the **telnetd** server. When you start the **telnet** application on the local system, **telnet** sends a connection request to the **telnetd** server on the remote system. (Note that the dotted arrows in Figures 11-4 through 11-9 indicate traversal of the network.)

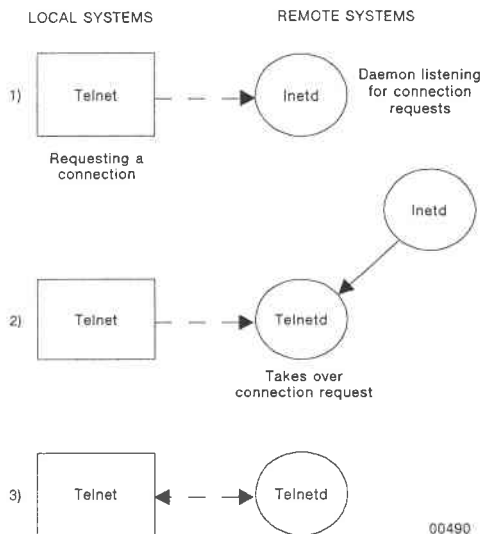


Figure 11-4. Inetd Spawned Server Process to Take Over Connection Request

Making ftp Connections

When the **inetd** daemon receives a connection request for the **ftp(1)** server, it spawns an **ftpd** server process (see Figure 11-5). This process takes over the connection request and, in turn, creates two types of connections to the user **ftp**: a control connection and a data connection. The control connection is used to transmit commands and instructions regarding the data being transferred between hosts. It is bidirectional, allowing control information to move both ways between **ftpd** and the **ftp** user. This connection persists throughout the entire **ftp** session. The data connection is used to actually transfer files or other data. There is a new data connection established for each data transfer command. The data connection is unidirectional and is deleted once a transfer is completed.

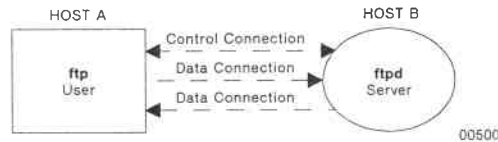


Figure 11-5. ftp Connection

Making remsh Connections

The connection process for **remsh(1)** is similar to **ftp**'s, except that **remshd** creates a shell for processing control information (see Figure 11-6). When the **inetd** daemon receives a connection request for **remsh**, it spawns a **remshd** server process. The server takes over the connection request. In addition to executing a shell for processing commands, the server acts as a conduit for standard input and signal information.

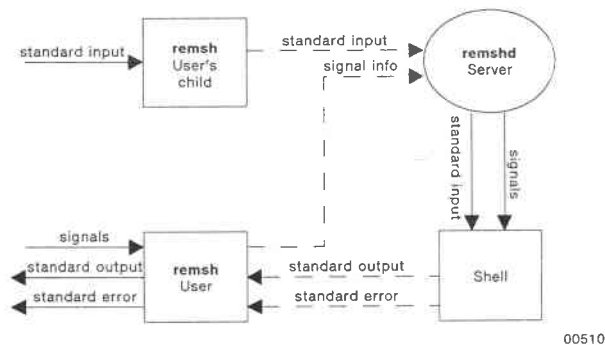


Figure 11-6. remsh Connection

Making rcp Connections

rcp(1) is a particular use of **remsh** (see Figure 11-7). **rcp** connection requests are sent to the **remshd** daemon. When the **inetd** daemon receives a connection request for **remsh** from a **rcp** client, it spawns a server **remshd** process. The server **remshd** does not service the request, but instead creates an **rcp** process on the remote system. The remote and local **rcp** processes then execute the file transfers.

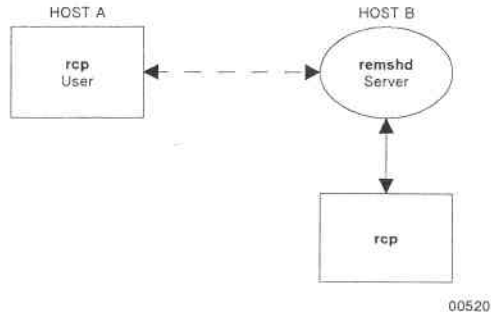


Figure 11-7. **rcp** Connection

Making telnet and rlogin Connections

The processing behind **telnet(1)** and **rlogin(1)** connections is more complicated. Both are designed to allow a remote terminal to appear as though it were on the local system. This is accomplished by passing data through pseudo-TTY device software. As a result, the user program at the remote host is "fooled" into thinking it is passing data to a locally attached terminal, when in fact, it is communicating with another host across the network. When the **inetd** daemon receives a connection request for **telnet** or **rlogin**, it spawns a **telnetd** or **rlogind** server process (see Figure 11-8). This process:

- ❑ takes over the connection request
- ❑ initiates a connection to the pseudo-TTY drivers
- ❑ executes a login at the remote host for the user

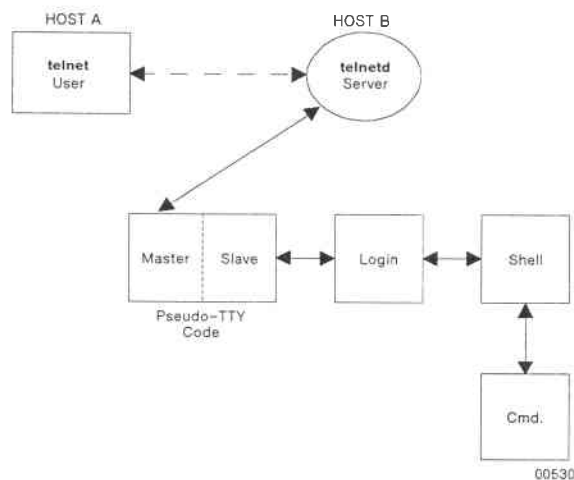


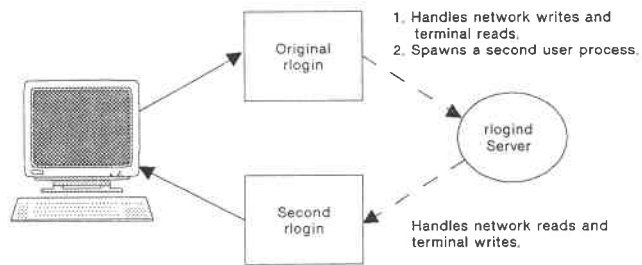
Figure 11-8. Server Side of a telnet or rlogin Connection

The connection to the pseudo-TTY software allows information from the data connection to be processed into a format recognizable to the remote user application. When it goes back out the data connection to the user process, it is again translated into the format appropriate to the sending host. The pseudo-TTY software has two sides: a master and a slave (see Table 11-2). The slave side acts like a TTY driver for it's operating system.

Table 11-2. Pseudo-TTY Master and Slave Entry Points

Master	Slave
/dev/ptyp*	/dev/ttyp*
/dev/ptyq*	/dev/ttyq*

The server is a relay between the network and the shell. The shell is created as a result of the login executed by **rlogind**. Because of the complexity of processing, the **rlogin** user process spawns a second user process (see Figure 11-9). The original user process reads data from the terminal and writes data to the network; the second reads data from the network and writes data to the terminal.



00540

Figure 11-9. User Side of an rlogin Connection

Observing Network Processes

Use the **ps -eal** command to observe the network processes that are executing on your system. Figure 11-10 shows part of a possible display.

	F	S	UID	PID	PPID	C	PRI	NI	ADDR	SZ	WCHAN	TTY	TIME	COMD
1	11	D	0	9	0	0	95	20	dca88	0	1550cc	?	1:28	streamsd
2	10	S	0	592	1	0	188	20	dd188	10	dd238	?	0:08	inetd
3	10	S	0	17	1	0	188	20	dd288	10	11d408	?	0:00	tftpd
4	10	R	0	23840	592	0	188	20	de188	9		?	0:16	telnetd
5	10	S	307	13324	19079	0	156	20	e0d88	22	14adc8	m332x02	0:01	ksh
6	10	S	305	29036	23840	0	188	20	ded88	20	dee08	ttyp8	0:03	ksh
7	10	S	0	19079	592	0	188	20	dfd88	9	ded38	?	0:00	telnetd
8														
9														
10														
11														

Figure 11-10. Observing Network Processes with **ps -eal**

The following information is useful for determining which networking processes are associated with each other:

- ❑ **streamsd** (a kernel-level daemon) and **inetd** (a user-level daemon) execute at all times if networking has been installed and set up on your system. By default, **streamsd** executes at priority 95 (this priority can be modified with the **PRISTREAMSD sysgen** tunable parameter or with the **setpri(1R)** command), and **inetd** executes at a time-sharing priority.
- ❑ When a user issues the **telnet(1)** command, the **inetd** daemon on the remote system receives this request and spawns a **telnetd** daemon. Note that the Parent Process ID (PPID) for each **telnetd** daemon in Figure 11-10 matches the Process ID (PID) for **inetd** (592).
- ❑ The subsequent shell process (in this case, **ksh**) has a PPID that matches the PID of the associated **telnetd** daemon. For instance in Figure 11-10, the PPID of the **ksh** process in line 6 matches the PID of the **telnetd** daemon in line 4, and the PPID of the **ksh** process in line 5 matches the PID of the **telnetd** daemon in line 7.
- ❑ **tftpd** is a stand-alone daemon that **listens(2)** and processes **connect(2)** requests to its own port, and does not use **inetd** for that purpose.

The **netstat(1M)** command also is useful for watching network connection activity. **netstat** symbolically displays the contents of various network-related data structures and connection information. See **netstat(1M)** for details on what information can be displayed.

UUCP Over TCP/IP

UUCP may be configured for use over a local Area Network (LAN) using the TCP/IP protocol. This will provide file transfers, remote command execution, and remote mail facilities over your LAN.

The **sysadm setuptcpip** Command

The following configuration is necessary for using UUCP over TCP/IP:

- ❑ add the local node name and Internet address to the */etc/hosts* file
- ❑ configure and start the TCP/IP network listener
- ❑ add appropriate entries to the */usr/lib/uucp/Systems* file

The first two items are done by executing the **sysadm setuptcpip** command. This should be done only upon initial system setup (see the *Software Installation Guide*) or whenever the Internet address of the local host is changed.

The Network Listener

The **sysadm setuptcpip** command configures and starts the TCP/IP network listener to service network requests for **uucico**. The **sysadm setuptcpip** command executes the following **nlsadmin(1M)** commands (no user intervention is needed):

- ❑ **nlsadmin -i tcpip**
 - initializes the configuration files used by the tcpip listener
- ❑ **nlsadmin -a 101 -c "/usr/lib/uucp/uucico -r0 -i TLIS -u nuucp" -p tirdwr -y uucico tcpip**
 - adds service 101 and makes it available through the tcpip listener
 - executes the command *"/usr/lib/uucp/uucico -r0 -i TLIS -u nuucp"* as the server
 - push tirdwr STREAMS module
 - *"-y uucico"* is a comment
- ❑ **nlsadmin -l \x00020401 <address> tcpip**
 - sets the address to where the tcpip listener listens
 - *"00020401"* is the port ID and *"<address>"* is the Internet address of the local host

❑ **nlsadmin -s tcpip**

- starts the listen process for tcpip

sysadmin setuptcpip creates a startup script in the */etc/rc2.d* directory that will execute the **nlsadmin -s tcpip** command upon entering multi-user state.

To display the current configuration of the tcpip listener, type:

```
nlsadmin -v tcpip
```

For more information, see **nlsadmin(1M)**.

Setting Up the Systems File

You need to add an entry to the */usr/lib/uucp/Systems* file for each machine on your network wanting UUCP service. The format of each entry follows:

```
<node name> <schedule> <device> - <port ID and Internet address>
```

Following is a sample entry for the machine "zeus" with Internet address 130.40.7.3:

```
zeus Any tcpip - \000\002\004\001\202\050\007\003
```

where "zeus" is the node name, "Any" is the time to call, "tcpip" is the device type to use in the */usr/lib/uucp/Devices* file (see the next section), the dash is a place holder, "\000\002\004\001" is the port ID, and "\202\050\007\003" is the Internet address in octal.

```
130 (decimal) = 202 (octal)
40 (decimal)  = 050 (octal)
7 (decimal)   = 007 (octal)
3 (decimal)   = 003 (octal)
```

It may be easier to specify the port ID and Internet address in hexadecimal, in which case the entry above would be:

```
zeus Any tcpip - \x0002040182280703
```

where "00020401" is the port ID and "82280703" is the Internet address.

```
130 (decimal) = 82 (hexadecimal)
40 (decimal)  = 28 (hexadecimal)
7 (decimal)   = 7 (hexadecimal)
3 (decimal)   = 3 (hexadecimal)
```

Use the **sysadm showaddr** command to display the Internet address of a system in the */etc/hosts* file in the hexadecimal format for UUCP. Use **sysadm addaddr** to add systems to the */etc/hosts* file. Note that it is not necessary to have an entry in the */etc/hosts* file for remote systems wanting UUCP service.

The Devices, Dialers, and Devconfig Files

The *Devices*, *Dialers*, and *Devconfig* files in the */usr/lib/uucp* directory are configured for TCP/IP as released and require no additional configuration. Their default configuration for TCP/IP is briefly described here.

The following entry exists in the */usr/lib/uucp/Devices* file:

```
tcpip,e tcpip - - TLIS \D nls
```

This entry defines the device type "tcpip". The ",e" specifies the guaranteed error-free delivery protocol; the second "tcpip" specifies the */dev/tcpip* device; the two dashes are place holders for unused fields; the "TLIS" specifies the Transport Layer Interface with STREAMS; the "\D" specifies to read the port ID and Internet address from the entry in the */usr/lib/uucp/Systems* file; and the "nls" specifies which dialer type to use in the */usr/lib/uucp/Dialers* file.

The following entry exists in the */usr/lib/uucp/Dialers* file for the dialer type "nls" specified in the */usr/lib/uucp/Devices* file:

```
nls " " " NLPS:000:001:101\N\c
```

This tells **uucico** to use the network listener service 101.

In the */usr/lib/uucp/Devconfig* file, the following entry specifies which STREAMS module to push for **uucico** service:

```
service=uucico device=tcpip push=tirdwr
```

APPENDIXES

Appendix A

Special Device Files

This appendix discusses the special device files used on the REAL/IX Operating System with information on how to create each type.

Special device files (also called special files), located under the `/dev` directory, allow communication with hardware and software devices by establishing a connection between a file in `/dev` and a device driver or a piece of hardware. When a program attempts to access a device, it finds the device name in `/dev` and the system determines the appropriate driver and address (hardware or software) from the major and minor numbers used in the entry for the device in `/dev`.

A device driver is a collection of routines used to open, close, read from, and write to a device. It handles all device-specific information and makes the device look like a file to the rest of the operating system. Device drivers are used to abstract devices; most details of the device are "hidden" in the device driver routines. The programmer can treat the device as if it were a file or a stream of bytes. For example, one of the special files for terminals is `/dev/tty001`. To write to that terminal, you would write to the `/dev/tty001` file as you would to any other file.

There are two types of drivers:

- ❑ **hardware drivers:** talk directly to hardware devices
- ❑ **software (or pseudo) drivers:** communicate with other software. The other software may talk to a hardware device.

Device handling routines are built into the operating system at `sysgen(1M)` time. Each driver has a file in the `cf/descriptions` directory. That file identifies the devices that are controlled by the driver, tunable parameters associated with the driver, as well as other information the operating system needs to properly access the device.

Structure of Special Device Files

The inode of a special device file is formatted differently than for regular files. Instead of holding data block addresses, the inode for a special file contains the major and minor device numbers (explained below).

Figure A-1 compares the output of an `ls -l` command for a regular file and a special device file. Two fields are different:

- ❑ The first character of the *mode* field is **c** (character) or **b** (block) for special device files, whereas it is **-** for regular files. Directories under the `/dev` directory have a **d** in this position, just as in other directories.
- ❑ The column showing the number of bytes (1050) in the file for regular files, indicates the major and minor numbers of the device for a special device file.

Regular File	-rw-r-----	1	abc	dsg	1050	Apr 23 08:14	dm.ol
Special Device Files	brw-rw-rw-	1	root	sys	96, 0	Apr 15 10:59	/dev/dsk/a710_00s0
	crw-rw-rw-	1	root	sys	96, 0	Apr 15 10:59	/dev/rdisk/a710_00s0

Figure A-1. `ls -l` Listings for Regular and Special Files

Major and Minor Numbers

The major number of a special device file identifies the specific driver that controls the device(s), and must match the major number assigned to the device in `sysgen`. The meaning of minor numbers is device-specific, but often identifies the specific sub-device being accessed. For example, all terminals controlled by the same driver have the same major device, but each terminal, modem, or printer has a unique minor device. For disk devices, the minor number refers to the disk partition or slice. Major block and major character device numbers are configured into the system.

Block and Character Devices

All special device files are either character (also called "raw") or block. The definitions of these are:

<i>Block Special File</i>	Data flowing to or from a block device goes through the system buffer cache, where the scheduling for actual I/O is done.
<i>Character Special File</i>	Data flowing to or from a character device goes directly to the device, not through the buffer cache.

Input/output operations with a block device are slower than those with a raw device, yet are more efficient because the system can "organize" its I/O operations. The conventions for which devices use character access and/or block access are:

- ❑ Terminal devices are raw, so that terminal response does not go through the system buffer cache.
- ❑ File systems are mounted as block devices, although all disk devices are listed as both block and raw devices. For most operations, you should use the block device, but when doing massive data transfers (such as *fsck* and *dd* do), it is faster and more reliable to use the raw device. Some administrative commands can be issued only against a character device, whereas a number of commands (administrative and regular) can be issued only against a block device.
- ❑ Some system software drivers, such as *root* and *swap*, are block devices, with corresponding raw versions such as *rroot* and *rswap*.

/dev Subdirectories

Some special device files are in subdirectories under */dev*. These subdirectories and their contents are:

<i>/dev/dsk</i>	Block special files for disk devices
<i>/dev/rdsk</i>	Character (raw) special files for disk devices

Many of the special files have links, so that the same device can be referenced by different names. For instance, *console*, *syscon*, and *syssty* all have the same major and minor numbers, and an *ls -l* listing shows them all as having three links. To verify that these are linked, do an *ls -li*; linked files all have the same inode, which shows in the leftmost column of *ls -li*.

Creating and Linking Special Device Files

The system comes configured with most of the special device files you will need. To create additional special device files, use the **mknod**(1M) command, which has the following format:

```
mknod name c major-number minor-number (character special file)
```

```
mknod name b major-number minor-number (block special file)
```

The first argument to **mknod** is the name of the special file. A sample **mknod** command for creating a console character device follows:

```
mknod console c 0 0
```

You must be *root* to create special files with **mknod**, so the default owner of special device files is *root* and the default group is *sys*. You can change the group and owner with the **chgrp**(1) and **chown**(1) commands, respectively, just as for any regular file. The owner, group, and permissions must be set carefully to ensure system security while providing adequate access to the device. The following sections discuss the ramifications of the group, owner, and permissions of each type of special device file.

To link special device files, use the **ln**(1) command as for any other file. As an example, the following commands would create special device files for *syscon* and *contty* that are linked to the *console* special device file created above:

```
ln /dev/console /dev/syscon
ln /dev/console /dev/contty
```

Of course, these files already exist on the release media, so this is not necessary. Special device files are more commonly linked with some of the system special device files discussed at the end of the chapter. For instance, if you move the *swap* area, you must relink the */dev/swap* and */dev/dump* files to associate them with the disk partition where the *swap* area is now loaded.

Any */dev* file can be removed with the **rm**(1) command; to modify a */dev* file, delete it with **rm**, then recreate it with **mknod**.

Special Files for Memory

The system is configured with three special files for memory, */dev/mem*, */dev/kmem*, and */dev/smem*.

/dev/mem is an image of the core memory and can be used to examine and possibly patch the system. Byte addresses are interpreted as memory addresses. An error will be returned when given a bad address.

/dev/kmem is the same as */dev/mem* except that kernel virtual memory, not physical memory, is accessed.

/dev/smem references physical memory but also verifies the memory location as a valid physical RAM memory address before accessing it. Encode drivers that read and write physical RAM memory so that they use the safe memory device */dev/smem*. The standard */dev/mem* is used for accessing I/O space as well as reading and writing memory, hence the driver cannot validate addresses. If a bad address is passed to the driver, it may produce long delays resulting in poor performance or it may panic the system resulting in a crash. */dev/smem* is to be used solely for reading and writing physical RAM memory as it provides a validation routine that will set `u.u_error` to `EFAULT` when given a bad address.

Following is a sample `ls -l` listing for these special files. Note that in this sample, the files have read-only permissions. Changing the permissions could create a major security risk to the system.

```
cr--r----- 1 root sys 3, 0 Oct 4 14:58 /dev/mem
cr--r----- 1 root sys 3, 1 Oct 4 14:58 /dev/kmem
cr--r----- 1 root sys 3, 3 Jan 31 10:53 /dev/smem
```

Special Device Files for Terminals

The system comes configured with the following special device files for terminals:

- ❑ */dev/com1* and */dev/com2* for the two serial ports. These are linked to */dev/tty00* and */dev/tty01* so can be addressed by either name.
- ❑ virtual terminals */dev/vt01* through */dev/vt07*. Virtual terminals are used for network terminal emulation.

If you add more terminals to your system, you will need to create special device files (as well as `getty` lines in the */etc/inittab* file) for each device.

The access permissions, owner, and group of terminal devices are interesting because they change automatically as the terminal is used. Table A-1 summarizes this information. Note that the permissions of an unused terminal device allow read/write access for anyone, but the permissions of an active terminal are under the control of the user.

Table A-1. Permissions for Terminal Devices

	Mode	Owner	Group
As created	crw-rw-rw-	root	sys
When user logs in	mesg y: crw--w--w- mesg n: crw-----	user's ID	user's group
When user logs off	crw-rw-rw-	root	previous user's group

Console Special Files

The name of the special file for the system console is */dev/console*. The console is a character device, similar in nature to other terminal devices, with major number 0 and minor number 0. The *console* special device file is linked to two other special device files: *syscon* and *systty*. The *syscon* file is the virtual console and *systty* is the physical console. All three files have the same major and minor numbers and refer to the same device. The **co** line in the */etc/inittab* file (which calls the **console** line in the */etc/gettydefs* file) controls the baud rate and terminal settings of all three devices.

System Special Device Files

In addition to the special device files for actual devices, there are a number of system special device files used by the operating system for a variety of purposes. Many of these are linked to another special device file. The operating system is delivered with a set of these that may need to be modified if you are moving file systems and such around. Tables A-2 and A-3 summarize the system special device files, what they are used for, and the meaning of their major/minor numbers or to which other files they should be linked. The special device files listed in Table A-2 may need to be modified by the administrator; the special device files listed in Table A-3 should never be modified by the administrator.

Table A-2. System Special Device Files Changed by Administrators

Name	Major/Minor Number	Use
<i>dump</i>	Usually linked to <i>swap</i> .	The disk partition where memory dumps will be written after a system panic. The contents of the dump partition are then saved to a disk file (usually under the <i>/usr/dumps</i> directory) during system initialization, where they can be studied with crash (1M).
<i>rdump</i>	Usually linked to <i>rswap</i> .	
<i>root</i>	Linked to block special device file where <i>root</i> file system is located.	Used as an alias for the special device file where the <i>root</i> file system is located.
<i>rroot</i>	Linked to raw special device file where <i>root</i> file system is located.	
<i>usr</i>	Linked to block special file for disk partition where <i>/usr</i> file system is located.	Used as an alias for the special device file where the <i>/usr</i> file system is located.
<i>rusr</i>	Linked to raw special file for disk partition where <i>/usr</i> file system is located.	



root must always be on partition 1 of the disk. The swap and dump partitions must be on partition 2 of the disk unless you run **sysgen** to modify the System Devices collection. This is a complex operation; if you need to move these devices, contact MODCOMP Customer Support for assistance.

Table A-3 lists system special files that should not be changed by administrators.

Table A-3. System Special Device Files Never Changed by Administrators

Name	Major/Minor Number	Use
<i>debug</i>	Character device. Major number must match what sysgen assigns to "osm" under "Miscellaneous System Declarations." Minor number must be 2.	Used to write kernel messages to the console and the <i>/usr/adm/putbuf</i> file.
<i>error</i>	Character device. Major number matches what sysgen assigns to "errlog" under "Miscellaneous System Declarations." Minor number must be 0.	Software device for system error logging daemon.
<i>kmem</i>	Character device. Major number must match what sysgen assigns to "memory" under "Miscellaneous System Declarations." Minor number must be 1.	Used to read contents of memory, including I/O space.
<i>mem</i>	Character device. Major number must match what sysgen assigns to "memory" under "Miscellaneous System Declarations." Minor number must be 0.	Used to read contents of memory, including I/O space.
<i>null</i>	Major number must be 3. Minor number must be 2.	Identifies the null device.
<i>smem</i>	Character device. Major number must match what sysgen assigns to "memory" under "Miscellaneous System Declarations." Minor number must be 3.	Used to read contents of memory.
<i>swap</i>	Block device linked to block special device file where main swap space is located. Minor number must be 2.	System swap area for paging. One such area is required for booting; multiple swap areas can be allocated with the swap command, but do not have special device files.
<i>tty</i>	Character device. Major number must match what sysgen assigns to "tty" under "Miscellaneous System Declarations." Minor number must be 0.	Supports tty structure that controls terminal settings; see termio(7) .

Appendix B

Administrative Directories and Files

A number of system files and directories control how the REAL/IX Operating System functions. By modifying the contents of these files and directories, you can customize the REAL/IX Operating System for your environment.

After a brief discussion of the file systems included on the release media, this appendix discusses the system directories and files used for:

- ❑ controlling the time zone identifiers used on the system (*/etc/TIMEZONE*)
- ❑ controlling and monitoring access to the **cron** facilities
- ❑ logging access to superuser permissions
- ❑ logging use of **help**(1) and **spell**(1) facilities

Other administrative files are discussed elsewhere in this book:

- ❑ files for managing users and groups: */etc/passwd*, */etc/group* (Chapter 3)
- ❑ files for managing file systems: */etc/fstab*, */etc/checklist* (Chapters 4 and 5, respectively)
- ❑ files for controlling run-state transitions: */etc/inittab* and associated files (Appendix C)
- ❑ files for job accounting: */etc/adm/acct* directory (Chapter 10)

Released File Systems

The REAL/IX Operating System release includes two file systems, *root* and */usr*. Table B-1 describes some of the key directories included in these file systems.

Table B-1. Directories in / and /usr

File System	Directory	Contents
<i>root</i>	<i>bin</i>	public commands
	<i>dev</i>	special device files that define all devices on the system
	<i>diag</i>	terminal and printer diag package
	<i>etc</i>	administrative commands and files
	<i>install</i>	used to mount utilities packages for installation and removal
	<i>lib</i>	public libraries
	<i>local</i>	repository of commands and files that are not part of the released system
	<i>stand</i>	includes copy of realized <i>/realix</i> file and online list of error messages
	<i>tmp</i>	temporary files (often made a separate file system)
<i>usr</i>	<i>adm</i>	error file, log of superuser access, job accounting files
	<i>BOS</i>	contains Bill of Materials used by mkcomply
	<i>catman</i>	"pcattable" manual pages accessed by man
	<i>dumps</i>	memory dumps and associated namelists saved after system panics
	<i>examples</i>	driver examples for reference only
	<i>include</i>	standard system header files
	<i>lbin</i>	repository of commands and files that are not part of the released system
	<i>mail</i>	incoming mail files for all users
	<i>news</i>	news files
	<i>preserve</i>	preserves editing sessions in <i>/tmp</i> when editing terminates abnormally
	<i>spool</i>	spool files for cron , lp , uucp and other utilities
	<i>src</i>	directory reserved for operating system source code
	<i>tmp</i>	temporary files (often made a separate file system)

/etc/TIMEZONE

The */etc/TIMEZONE* file defines and exports the **TZ** (time zone) variable that identifies the time zone in which the computer operates, its relationship to Greenwich Mean Time (GMT), and whether alternate time zones (e.g., Daylight Savings Time) are observed. It is called at boot and shutdown time by */etc/profile* and the **cron** daemon, as well as other processes that need this information.

The contents of a typical */etc/TIMEZONE* file are shown below:

```
# set timezone environment to default for the machine
TZ=EST5EDT
export TZ
```

TZ Environment Variable

All system commands and user applications that need to perform time conversions use functions from the **ctime(3C)** library. The **ctime** library, provided with the REAL/IX Operating System, uses the **TZ** environment variable. If in the current environment, the variable **TZ** is defined and no errors are found in its format, its value will be used by the library. If **TZ** is not defined or its format is incorrect, the library will assume Greenwich Mean Time as the time zone.

The values assigned to **TZ** have the following format:

```
std offset[dst[offset]]
```

See **ctime(3C)** for a detailed explanation of the time zone string format.

Editing the /etc/TIMEZONE File

While the */etc/TIMEZONE* file may be edited manually, it is best to allow the **sysadm(1M)** utility to configure this file for you. The **sysadm** utility ensures that the file is modified following the proper formats expected by **ctime(3C)**.

To modify the file using **sysadm**, you would first become super user and enter the command string **sysadm datetime**. The utility will prompt you for the time and zone information. Upon providing this information, the utility will prompt you for the date and time information. After the last block of information is entered, the utility will exit and return you to the system prompt.

Files that Control cron

The **cron** facility is the clock daemon that allows you to schedule certain commands to be run automatically at a time you select. After an overview of how **cron** works, this section explains the following files:

- ❑ The files in the *crontabs* and *atjobs* directories
- ❑ The *queuedefs* file
- ❑ Files used to restrict (if desired) the use of **at**, **batch**, and **crontab**
- ❑ The **cron** log

Note that realtime processes on the REAL/IX Operating System can be coded to use the realtime timing mechanism, which provides some functionality similar to that of **cron** but with the finer timing resolutions required by some realtime applications. See the *Programmer's Guide* for information on using the realtime timing mechanisms.

The **/etc/cron** daemon is started by the **/etc/rc2** shell script when the system enters the multi-user mode. **cron** accesses files from the *crontabs* and *atjobs* directories under */usr/spool/cron*.

- ❑ The *crontabs* directory contains files created by the **crontab** command; these enable users to schedule tasks to be run at given times on a regular basis such as monthly, weekly, daily, or every half hour.
- ❑ The *atjobs* directory contains files that have been created through either **at** or **batch**; these enable users to schedule a task to be performed only once, at a specified time.

The files in each directory have specific formats that are discussed below.

By default, all users can use all **cron** facilities; it is possible to restrict the use of either **crontab** or **at** and **batch**. It is seldom necessary to restrict use of **at** and **batch**, since by using these commands users can schedule resource-intensive jobs to run during times when the system is quiet. However, some sites restrict the use of **crontab** to a small group of users to avoid frivolous usage.

cron has the ability to keep track of the various requests and execute them at the correct times. A separate queue is maintained for requests made through each of the three commands: **at**, **batch**, and **crontab**. */usr/lib/cron/queuedefs* is the control file that tells **cron** how to manage the two queues associated with **at** and **batch**, "queue a" and "queue b", respectively. Most importantly, the *queuedefs* file instructs **cron** how often to check each queue for tasks to be performed.

Tasks scheduled through the **crontab** command reside in "queue c". However, "queue c" is not governed by the *queuedefs* file. Whenever the **crontab** command is used to add a request to or delete one from "queue c," **crontab** communicates with **cron** through the */usr/lib/cron/FIFO* file. By reading the *FIFO* file, **cron** knows when to perform the tasks in "queue c."

cron maintains a log, */usr/lib/cron/log*, for all tasks scheduled through **at**, **batch**, or **crontab**. Before a command is executed through **cron**, **cron** inserts two lines into the log. When the task is performed, another entry is made indicating completion of the task.

To execute a command, **cron**

- ❑ spawns a shell, always the standard shell **sh**
- ❑ runs **env(1)** to replicate all environmental variables defined for your login
- ❑ executes a **cd(1)** so that its present working directory matches that of the user at the time the job was submitted. For jobs submitted through **crontab(1)**, the user's **\$HOME** directory is used as a present working directory.
- ❑ uses the privileges associated with that user. **cron** can execute a program that requires realtime privileges only if the user who submits the job is listed in the realtime privileges file.

Files in the crontabs Directory

The **crontab(1)** command allows you to schedule a command to be run at a given time on a regular basis, such as daily, weekly, or every half hour. The **crontab** command with its argument creates a file in the */usr/spool/cron/crontabs* directory. Any file present in the *crontabs* directory is considered active and executed by **cron** at the appropriate time. Any file in the *crontabs* directory is no longer owned by the original owner. Rather, the file is owned by **root**, and named after the original owner.

The **crontab** command takes a file as its argument. This file contains one or more of the "crontab lines." Each line has six fields, separated by a tab or one or more spaces. The first five fields specify the time at which the command is to be executed, as shown in Table B-2. The sixth field specifies the command.

Table B-2. Time Specifications for crontabs

Field	Meaning	Range	Examples
1	minute	0 - 59, or *	0 = on the hour
2	hour	0-23, or *	0 = midnight
3	day of the month	1 -31, or *	1 = 1st day of month
4	month of the year	1 - 12, or *	1 = January
5	day of the week	0 - 6, or *	0 = Sunday, 6 = Saturday

As *root*, you can either use the **crontab** command or edit the appropriate file under */usr/spool/cron/crontabs* to make the desired entries. Revisions to the file using an editor take effect at the next reboot, unless you execute **crontab**, in which case they take effect immediately.

The following syntax applies to the fields for time specifications:

- ❑ Two numbers separated by a minus indicates an inclusive range of numbers between the two specified numbers. For instance "1-5" in the fifth field indicates that the job is to be run Monday through Friday, but not Saturday and Sunday.
- ❑ A list of numbers separated by commas specifies all of the numbers listed. For instance, "3,6,9,12" in the fourth field indicates that the job is to be run in March, June, September, and December, but not the other months.
- ❑ An asterisk specifies all legal values. For example, an asterisk in the third field indicates that a job is to be run every day at the specified time.

The sixth field specifies the command. Note the following:

- ❑ The percent sign is translated to a newline character.
- ❑ Only the first line of a command field (character string up to the percent sign) is executed by the shell. Any other lines are made available to the command as standard input.
- ❑ By default, all standard error and standard output messages will be mailed to the user who initiated the **crontab** command. To avoid this, run the command with "1>/dev/null" and "2>/dev/null".
- ❑ Blank lines in the *crontabs* file generate errors.

To understand how **crontab** works, consider the following *crontabs* file that has three lines to control the times when users can run the **chess** program:

```
59 3-18 * * 1-5 /usr/chess/turnoff 1>/dev/null 2>/dev/null
59 19-23 * * 1-5 /usr/chess/turnon 1>/dev/null 2>/dev/null
59 3-23 * * 0,6 /usr/chess/turnon 1>/dev/null 2>/dev/null
```

This is interpreted as follows:

- ❑ The first line prevents users from accessing the **chess** program between 3:59 AM and 6:59 PM (18:59) Monday through Friday, all year.
- ❑ The second line turns the program on at 7:59 PM. Despite what appears in Line 1, this is when the program is really accessible.
- ❑ The third line guarantees that **chess** remains available from 3:59 AM Saturday and Sunday.
- ❑ No **stderr** or **stdout** messages are sent to the user when these commands execute.

The atjobs Directory

The `/usr/spool/cron/atjobs` directory contains files listing requests made through the **at** and **batch** commands. The files are created automatically, and list the environmental variables from the system's `/etc/profile` and user's `.profile`, followed by the command(s) issued. Each file is assigned a unique number as its name and is owned by the user who executed the command.

A sample *atjobs* file follows:

```

: at job
export HOME; HOME='/q1/terri'
export LOGNAME; LOGNAME='terri'
export MAIL; MAIL='/usr/mail/terri'
export PATH; PATH='/q1/terri/bin:/bin:/usr/bin:/usr/sbin'
export TZ; TZ='EST5EDT'
cd /q1/terri
ulimit 2048
umask 022
1st command issued
2nd command issued

```

The queuedefs File

cron uses the `/usr/lib/cron/queuedefs` file to control the queues associated with **at** and **batch**. It contains two lines, each of which contains the following four pieces of information:

1. The name of the queue. **a** is the **at** queue, **b** is the **batch** queue.
2. The maximum number of commands from this queue allowed to run at one time, expressed as a number followed by a **j**. For instance, "2j" allows two jobs at a time, "5j" allows five jobs at a time. If omitted, 100 jobs are allowed to run at a time from the queue.
3. The **nice**(1) value to be applied to the priority of any job executed from this queue, expressed as a number followed by **n**. For instance, "1n" issues a **nice** value of 1. If omitted, the **nice** value is 2.
4. The length of time, in seconds, to wait before executing a command, expressed as a number followed by **w**. For instance, "30w" causes jobs in the queue to wait 30 seconds before executing. If not specified, jobs will wait 60 seconds before executing.

The released *queuedefs* file is:

```

a.4j1n      4 at jobs can run at once, using a nice value of 1 and delaying 60 seconds (default)
b.2j2n90w   2 batch jobs can run at once, using a nice value of 2 and delaying 90 seconds

```

crontabs/sys

The released operating system includes a `/usr/spool/cron/crontabs/sys` file with the following lines commented (#) out:

```
0 * * * 0-6 /usr/lib/sa/sa1
15,30,45 8-17 * * 1-5 /usr/lib/sa/sa1
5 18 * * 1-5 /usr/lib/sa/sa2 -s 8:00 -e 18:01 -i 1200 -A
```

Remove the # characters to activate these processes, which collect data for the `sar(1)` reports. Lines can be added for additional administrative tasks, such as scripts that monitor disk and file usage.

Controlling Use of cron

By default, any user can access the **cron** facility through the **at**, **batch**, and **crontab** commands. You can, however, restrict access to these commands with the files in the `/usr/lib/cron` directory:

- ☐ Access to **at** and **batch** is controlled by the `at.allow` and `at.deny` files.
- ☐ Access to **crontab** is controlled by the `cron.allow` and `cron.deny` files.

When a user tries to submit a **cron** job, the system checks the following:

1. If the `*.allow` file exists and the user's name is in it, the job is accepted.
2. If the `*.allow` file does not exist, check the `*.deny` file. If the user's name is in it, the job is rejected; if the user's name is not in it, the job is accepted.
3. If neither the `*.allow` nor `*.deny` files exist, only `root` can use the facility.

Table B-3 summarizes how to create the files to provide the **cron** access you want. The system as released allows unrestricted use of the **cron** facility. Referring to the table, you can determine how the files are set up to allow this: no `*.allow` files exist; both an `at.deny` and a `cron.deny` file exist, but they contain no entries. If you elect to restrict use of **at** or **cron** by creating an `*.allow` file and placing entries in the corresponding `*.deny` file, you must include the following system entries in the `*.allow` file: `root`, `sys`, `adm`, and `uucp`.

Table B-3. Permissions to Use cron Facility

<code>*.allow</code>	<code>*.deny</code>	User Permissions
does not exist	does not exist	only root can use facility
does not exist	exists but has no entries	allow all users to use facility
exists but has no entries	exists but has no entries	allow all users to use facility
does not exist	exists and has entries	users listed cannot use facility
exists and has entries	exists but has no entries	only users listed can use facility

/usr/lib/cron/log

cron keeps track of every job it executes by maintaining a log file, */usr/lib/cron/log*. Each job has up to three lines associated with it: the first two lines are written before a command is executed, and the third line is added after the command executes.

1. The first of the two lines contains a greater-than sign (>), followed by CMD (for "command"), and the command to be executed.
2. The second of the two lines contains a greater-than sign (>), followed by information on which user requested the command, the process number of the command, and the time the command is to be executed.
3. The third line begins with a less-than sign (<), and contains the user's name, the process number of the command, and the current time, which corresponds roughly to the time the execution was complete.

For example, the three lines might look like this:

```
> CMD: /usr/lib/acct/ckpacct
>root 14324 c Fri Aug 2 03:00:01 1989
<root 14324 c Fri Aug 2 04:00:00 1989
```

In most cases, the third line will be separated from the first two by several other lines. The first two lines are sequential, and the process number can be used to associate the completion line with the two initialization lines.

Two other types of entries in the log file show status information:

1. Lines beginning with three asterisks (***) indicate when **cron** is restarted.
2. Line that begin with an exclamation point contain error messages.

On active systems, the *cron/log* file can grow rapidly. It should be cleaned out periodically, usually with a **crontab** command entered in *root's crontabs* file, although you could also clean it out with a script in the *rc0.d* or *rc2.d* scripts executed when the system changes run states. Rather than deleting the entire file, you can use **tail(1)** to save the end of the file. For instance, the following command saves the last 50 lines in the log:

```
tail -50 /usr/lib/cron/log > /tmp/log; mv /tmp/log /usr/lib/cron/log
```

/usr/adm/sulog

The */usr/adm/sulog* file contains a history of substitute user (*su*) command usage. As a security measure, this file should not be readable by *others*. The */usr/adm/sulog* file should be periodically truncated to keep the size of the file within a reasonable limit. Note that */etc/cron*, */etc/rc0*, or */etc/rc2* can be used to clean up the *sulog* file. To use one of these functions, add the appropriate command line to the */usr/spool/cron/crontabs/root*, */etc/shutdown.d/ . . .*, or *rc2.d*, *rc3.d . . .* file. The following command lines limit the size of the log file to the last 100 lines in the file.

```
tail -100 /usr/adm/sulog > /tmp/sulog
mv /tmp/sulog /usr/adm/sulog
```

An example of a typical */usr/adm/sulog* file is:

```
SU 08/18 12:35 + console root-sysadm
SU 08/18 16:11 + console root-sysadm
SU 08/18 16:16 + console root-sysadm
SU 08/18 23:45 + tty?? root-uucp
SU 08/19 11:53 + console root-sysadm
SU 08/19 15:25 + console root-sysadm
SU 08/19 23:45 + tty?? root-uucp
SU 08/20 10:16 + console root-adm
SU 08/20 10:33 + tty24 rar-root
SU 08/20 10:42 + console root-sysadm
SU 08/20 10:59 + console root-root
SU 08/20 11:01 + console root-sysadm
SU 08/20 12:36 + tty11 bin-bin
SU 08/20 12:37 + tty11 tws-bin
SU 08/20 14:42 - tty24 awa-sys
SU 08/20 14:47 - tty24 awa-sys
SU 08/20 14:48 + tty24 awa-root
SU 08/20 15:44 + console root-sysadm
```

Appendix C

Controlling Run State Transitions

Chapter 2 explains how to boot the system, shut it down, and move between single-user and multi-user state. The system activities that occur when one moves between these run states is determined by a group of files and directories in the */etc* directory. You can modify these files and directories to customize your environment.

The REAL/IX Operating System has 8 run levels (0 – 6 and S or s); each run level is a machine state. These states are used as arguments to the **init** command to move between machine states and are defined as follows:

- 0 Halts the operating system
- s,S Single-user state
- 1 Single-user state
- 2 Multi-user state
- 3,4 Not currently used
- 5,6 Automatic reboot

When a new state is entered, the **init(1M)** process reads the */etc/inittab* file, finds the "instructions" that apply to that run state, and executes those processes. As released, there are shell scripts and, in some cases, directories of shell scripts, that control most processing. Table C-1 summarizes the scripts and directories that control transitions between run states.

Table C-1. Controlling Run States

Run State	Script	Directory	Description
1,s,S (single-user from boot)	<i>shutdown</i>	<i>none</i>	checks permissions to run shutdown, broadcasts warning messages to users, does init s , and calls <i>/etc/rc0</i>
	<i>bcheckrc</i>	<i>none</i>	checks <i>root</i> file system
	<i>sgfusr</i>	<i>none</i>	provides ability to stop in single-user state
	<i>brc</i>	<i>none</i>	mounts <i>root</i> file system
0, 5, 6 (single-user from multi-user)	<i>rc0</i>	<i>rc0.d</i>	scripts prefaced with K and S executed when entering single-user state, 0, 5, or 6
			synchronizes and unmounts file systems
2	<i>rc2</i>	<i>rc2.d</i>	scripts prefaced with K and S executed when entering multi-user state

/etc/init.d Directory

The */etc/init.d* directory contains executable files used in upward and downward transitions to all system run levels. These files are linked to files beginning with S (start) or K (stop) in */etc/rcn.d*, where *n* is the appropriate run level. Files are not executed from this directory. They are only executed from */etc/rcn.d* directories. Files can be added to this directory, but should not be removed; the files in the */etc/rcn.d* directories can be removed if necessary.

Table C-2 lists the files released for the *init.d* directory and the file names under which they are executed from the *rc0.d* and *rc2.d* directories.

Table C-2. Contents of `init.d` Directory

init.d File	Description	rc0.d File	rc2.d File
ANNOUNCE	notification of shutdown to console	K00ANNOUNCE	none
nodename	set <code>nodename</code> for system; change this file to match sysgen <code>NODE</code> parameter	none	S00nodename
LINKBOOT	link most recently booted kernel in <code>/</code> or <code>/stand</code> to <code>/realix</code>	none	S01LINKBOOT
MOUNTUSR	mounts <code>/usr</code> and <code>dump</code> file system if specified	none	S01MOUNTUSR
PUTBUF	start console logging, save contents of <code>putbuf</code> buffer, and save any dumps left on <code>/dev/rdump</code>	K02PUTBUF	S02PUTBUF
SHMGET	set up shared physical memory segment for application programs to use	none	S02SHMGET
MOUNTFSYS	mount all file systems in <code>/etc/fstab</code> that are not already mounted	none	S03MOUNTFSYS
RMTMPFILES	cleanup <code>/tmp</code> and <code>/usr/tmp</code> directories	none	S05RMTMPFILES
RMMAILLOCKS	remove mail locks	none	S07RMMAILLOCKS
syssetup	system setup requirements: print trademark files, create <code>/etc/ps_data</code>	none	S20syssetup
perf	start sadc to produce sar data	none	S21perf
acct	start (or stop) process accounting	K30acct	S30acct
setrtusers	load the system realtime privilege table	none	S32setrtusers
errlog	start and stop error logging	none	S34errlog
ucmex ??	Download Ethernet controller and start <code>inet</code> daemons	K60UCMEX (see note)	S60UCMEX (see note)
lp	start and stop lp sched	K80lp	S80lp
nls_inet	network listener service	K47nls_inet (see note)	S47nls_inet (see note)
uucp	cleanup <code>uucp</code> lock, status, and temporary files	none	S70uucp
cron	start and stop cron daemon	K75cron	S75cron
noteOKboot	boot sequence completed OK; controls autoreboot and autodump	none	S90noteOKboot

Note: These files are present only if your network was configured using **sysadm**.

/etc/inittab

The */etc/inittab* file contains instructions for the */etc/init* command. Each line has four fields, separated by colons. A comment can be added at the end of the line; it is preceded with a *#* sign and may go to the end of the line. The four fields are:

1. *id* is one or two characters that uniquely identifies an entry.
2. *rstate* is zero or more numbers and letters (0 through 6, s or S) that determines in what run-state *action* is to take place. If *rstate* is null, the *action* is taken for transitions to all states.
3. *action* can be one of the following:

sysinit	run <i>process</i> before <i>init</i> sends anything (including a login message) to the system console.
boot	entry is processed only at <i>init</i> 's boot-time read of <i>/etc/inittab</i> . <i>init</i> starts the process but does not wait for it's termination and when the process dies, <i>init</i> does not restart the process. The <i>rstate</i> for this line should be the default or it must match <i>init</i> 's run-state at boot-time.
bootwait	start <i>process</i> the first time <i>init</i> goes from single-user to multi-user state after the system is booted. (If <i>initdefault</i> is set to 2, the process will run right after the boot.) <i>init</i> starts the process, waits for its termination and, when it dies, does not restart the process.
wait	when going to <i>rstate</i> , start <i>process</i> and wait until it is finished.
initdefault	scan this entry only when <i>init</i> is initially invoked. <i>init</i> uses this entry, if it exists, to determine which run-state to enter initially. It does this by taking the highest run-state specified in the <i>rstate</i> field and using that as its initial state. If the <i>rstate</i> field is empty, this is interpreted as 0123456 and <i>init</i> will use run-state 6 as its initial state. If the default run-state is not S or s, <i>init</i> will ask if you want to enter a run-state other than the default. If a RETURN is not entered in response, <i>init</i> uses the run-state specified in the <i>initdefault</i> entry as its initial state. If a RETURN is entered in response, or there is no <i>initdefault</i> entry in <i>/etc/inittab</i> , then <i>init</i> will request an initial run-state from the user at the system console. The number of seconds that <i>init</i> waits for the RETURN response is configurable in the process field of the <i>initdefault</i> entry by specifying # TIMEOUT = <i>n</i> (defaults to 5 seconds if not specified). If the TIMEOUT value is set to zero, <i>init</i> will not prompt you to enter a run-state, and will go directly to the default run-state.
once	run <i>process</i> once and do not start it again if it finishes.

powerfail	tells init to run <i>process</i> whenever a direct powerdown of the computer is requested (for example, by a daemon when a switch to emergency power is detected).
powerwait	tells init to run <i>process</i> only when init receives a power fail signal and wait until <i>process</i> terminates before continuing any processing of <i>/etc/inittab</i> .
respawn	if <i>process</i> does not exist, start it, wait for it to finish, and then start another.
ondemand	synonymous with respawn , but used only with <i>level a, b, or c</i> . (See init(1M) for more information on <i>rstates a, b, and c</i> .)
off	when in <i>rstate</i> , kill <i>process</i> or ignore it.

4. *process* is any executable program, including shell procedures.

The following figure shows the lines (except for the */etc/getty* lines for user terminals) in the released */etc/inittab* file.

```
bchk::sysinit:/etc/bcheckrc </dev/console >/dev/console 2>&1
is:2:initdefault:
p3:s1234:powerfail:/etc/shutdown -y -i0 -g0 >/dev/console 2>&1
s0:056:wait:/etc/rc0 >/dev/console 2>&1 </dev/console
s1:1:wait:/etc/shutdown -y -iS -g0 >/dev/console 2>&1 </dev/console
s2:23:wait:/etc/rc2 >/dev/console 2>&1 </dev/console
s3:3:wait:/etc/rc3 1>/dev/console 2>&1 </dev/console
of:0:wait:/etc/uadmin 2 0 >/dev/console 2>&1 </dev/console
fw:5:wait:/etc/uadmin 2 2 >/dev/console 2>&1 </dev/console
rb:6:wait:/etc/uadmin 2 1 >/dev/console 2>&1 </dev/console
co:234:respawn:/etc/getty console console
```

/etc/getty lines for user terminals

When entering a new machine run-state, the appropriate lines are executed in order. For instance, when booting up the system and going to multi-user state (which is the **initdefault** state), the following commands are executed in this order:

1. **bcheckrc**
2. **sglusr**
3. **brc**
4. **rc2** (which in turn executes the files prefaced by **S** and **K** in the */etc/rc2.d* directory)

You can modify the *inittab* file to support the specific needs of your installation. Use the following guidelines:

- ❑ In general, additional scripts to be executed when the system goes to multi-user state or is shut down should be added to the *rc2.d* and *rc0.d* directories, respectively, rather than being added directly to the *inittab* file.
- ❑ If *inittab* or the associated files start processes that execute at realtime priorities, those processes should begin at lower priorities (even if they subsequently boost their priority to something higher). This gives you a chance to login the console after rebooting the system and kill the process if necessary.

Booting the System

Unless you specifically request otherwise, the system automatically boots to multi-user state. The script prompts you for a carriage return if you want to specify a different run state as discussed in Chapter 2. If you always want to boot to single-user state, modify the **initdefault** line in the */etc/inittab* file.



*If you modify **initdefault** to specify single-user state, the autoreboot will leave the system in single-user state making it unavailable to the user community.*

/etc/rc2 Script and */etc/rc2.d* Directory

When the system goes to run state 2 (multi-user), the **init** process calls the */etc/rc2* script, which in turn executes the scripts in the */etc/rc2.d* directory. Files in the *rc2.d* directory have a one-letter prefix with the following significance:

- K** indicates processes that are stopped
- S** indicates processes that are started
- N** never execute (used to keep the script available for future use)

Following the letter is a two-digit number that determines the order in which scripts will be executed. If two scripts have the same priority, they are executed in alphabetical order (where any uppercase letter precedes any lowercase letter). The order of the scripts is important in some cases; for instance, the file systems must be mounted before scripts that use the file systems execute.

The */etc/rc2* file contains a shell script that is executed by */etc/init* on transitions to run level 2 (multi-user state). Executable files beginning with **K** or **S** in the */etc/rc2.d* directory are executed when */etc/rc2* is run. All files in */etc/rc2.d* are linked from files in the */etc/init.d* directory. The following are descriptions of some of those files:

MOUNTFSYS	Sets up and mounts file systems. Builds the mount table and mounts the <i>root (/)</i> and <i>/usr</i> file systems.
RMTMPFILES	Makes the <i>/tmp</i> and <i>/usr/tmp</i> directories, cleaning up (deleting) any previous files in those directories. In order to ensure that all files (even if they are given an odd name, such as one that begins with ".") are deleted, the system actually removes and recreates the directories. When these become file systems rather than directories in the file system, you will get an error message like the following: mkdir: Failed to make directory "/tmp"; File exists This error message can be ignored. The directories will be purged of all files. (Never put files in these directories or file systems that cannot be deleted.)
SHMGET	Creates a shared segment of physical memory when the system changes states from single- to multi-user. The key (path and id) used in this script should be used by all other programs using the shared memory. Refer to shmget(1) and ftok(3C) for information about the path and id associated with SHMGET .
cron	Starts the cron daemon by executing <i>/etc/cron</i> .
uucp	The uucp file deletes uucp locks (LCK*), status files (STST*), and temporary files (TM*) under the <i>/usr/spool/uucp</i> directory structure.
lp	The lp file removes the line printer spooler lock file and starts the scheduler.

Other files may also be added to the directory as a function of adding hardware or software to the system.

/etc/rc0

The */etc/rc0* file contains a shell script that is executed by */etc/shutdown* for transitions to single-user state, and by */etc/init* for transitions to run levels 0, 5, and 6. Files in the */etc/rc0.d* directory are executed when */etc/rc0* is run. The file **K00ANNOUNCE** in */etc/rc0.d* prints the message 'System services are now being stopped.' Any task you want executed when the system is taken to run levels 0, s, 5, or 6 can be done by adding a file to the */etc/rc0.d* directory.

The */etc/rc0* script does the following:

1. Sends a message to the system console that the system is coming down.
2. Executes all scripts in the */etc/rc0.d* directory, in order.
3. Kills all executing processes not directly related to the shutdown procedure.
4. Flushes all system buffers to disk, then unmounts all file systems except *root*.
5. Flushes all system buffers to disk again, to ensure that the *root* file system is updated.
6. Sends a message to the system console that the system is down.

/etc/rc0.d Directory

The */etc/rc0.d* directory contains files executed by */etc/rc0* for transitions to system run levels 0, 5, and 6. Files in this directory are linked from the */etc/init.d* directory, and begin with either a **K** or an **S**. **K** indicates processes that are stopped and **S** indicates processes that are started when entering run levels 0, 5, or 6.

/etc/shutdown

The */etc/shutdown* file contains a shell script to shut down the system gracefully in preparation for system backup or scheduled downtime. It has a built-in grace period of 60 seconds (you can use the **-g** option to change the grace period); it broadcasts warning messages to all users who are logged in, then, when the grace period has expired, calls *init* to enter the desired run state. By default, it will enter state 0, but this can be changed by using the **-i** option. */etc/shutdown* calls */etc/rc0* before calling run level s (or S). If it calls run level 0, 5, or 6, then */etc/rc0* will be executed by *init* through its entry in the */etc/inittab* file.

Appendix D

fsck Internals and Error Messages

The **fsck** program checks an inactive file system for structural integrity. Each file system has a certain amount of redundant data; by reading or computing this information, **fsck** can identify inconsistencies in the structure. **fsck** is a multipass program. It is usually run only when the file system is unmounted. To check the integrity of the free-block organization, or just to see if the file system is corrupted, it is sometimes useful to run **fsck** with the **-n** option (answer **no** to all questions) on a mounted file system.

Chapter 5 gives instructions for running **fsck**. This appendix describes the checks done by **fsck** and the error messages that may be returned.

File System Components

The following sections summarize the consistency checks applied to each component of a file system. Note that this summary is not in the order that **fsck** checks the file system components. Most of the checks done by **fsck** are appropriate for either the S5 or the F5 file system architecture; those components that exist in only one of the architectures are so marked in the list below. For a full discussion of the file system structure, see *Concepts and Characteristics*.

The Super Block

Every change to the file system blocks or inodes modifies the super block. If the last command involving output to the file system is not an **unmount** command (usually when the system crashes), the super block is almost certainly corrupted.

The super block can be checked for inconsistencies involving the following areas.

File System Size and Inode List Size

The file system size must be larger than the number of blocks used by the super block and the number of blocks used by the list of inodes. The number of inodes must be less than 65,535. The file system size and inode list size are critical pieces of information to the **fsck** program. While there is no way to actually check these sizes, **fsck** can verify whether they are within reasonable bounds. All other checks of the file system depend on the correctness of these sizes.

Free Block List (S5 Architecture Only)

The free block list on S5 file systems starts in the super block and continues through the free-list blocks of the file system. Each free-list block can be checked for a list count out of range, for block numbers out of range, and for blocks already allocated within the file system. A check is made to see that all the blocks in the file system were found.

The first part of the free block list is in the super block. The **fsck** program checks that the list count is greater than or equal to 0 or less than or equal to 50. It also checks each block number to make sure it is within the range bounded by the first and last data block in the file system. Each block number is compared to a list of already allocated blocks. If the free-list block pointer is not zero, the next freelist block is read in and the process is repeated.

When all the blocks have been accounted for, a check is made to see if the number of blocks used by the free block list plus the number of blocks claimed by the inodes equals the total number of blocks in the file system.

If anything is wrong with the free block list, **fsck** can rebuild it, leaving out blocks already allocated.

The free block list is not used with F5 file systems. Instead, a bitmap is used to identify unallocated data blocks in the file system. See below for information on how **fsck** checks the bitmap.

Free Block Count

The super block contains a count of the total number of free blocks within the file system in both S5 and F5 file system architectures. The **fsck** program compares this count to the number of blocks it found free within the file system. If the counts do not agree, **fsck** can replace the count in the super block with the actual free block count.

Free Inode Count

The super block contains a count of the total number of free inodes within the file system. The **fsck** program compares this count to the number of inodes it found free within the file system. If the counts do not agree, **fsck** can replace the count in the super block with the actual free inode count.

The Free Block Bitmap (F5 Architecture Only)

F5 file systems use a bitmap to manage the free data blocks in the file system rather than the free block list described above. This bitmap is a bit array indexed by the block numbers of an F5 file system, and is located right after the inode data structure on the disk. The free block bitmap can be checked for:

- ☐ blocks allocated both to files and marked as free in the bitmap
- ☐ blocks not allocated to either a file or marked as free in the bitmap

Each bit in the bitmap represents one block in the file system. Bits set to 1 indicate that the corresponding data blocks are free; bits set to 0 indicate that the corresponding data blocks are allocated. The **fsck** program checks the bitmap to ensure that the bits not set correspond to the blocks allocated. When all the blocks are accounted for, **fsck** checks to see if the number of blocks in the free block bitmap, plus the number of blocks claimed by the inodes, freemap, and super block, equal the total number of blocks in the file system. If anything is wrong with the bitmap, **fsck** rebuilds it, leaving out blocks already allocated.

The Inode List

The list of inodes is checked sequentially starting with inode 1 (there is no inode 0). Each inode is checked for the consistencies described below.

Format and Type

Each inode contains a mode word that describes the *type* and *state* of the inode. Inodes may be regular, directory, block, character, or pipe (fifo). If an inode is not one of these types, it is illegal.

Inodes can be unallocated, allocated, and partially allocated. This last state indicates an incorrectly formatted inode. An inode can reach this state if bad data is written into the inode list through, for example, a hardware failure. The only corrective action **fsck** can take is to clear the inode.

Link Count

Every inode keeps a record of the number of directory entries linked to it. The **fsck** program verifies the link count of each inode by examining the total directory structure and calculating an actual link count for each inode.

If the link count stored in the inode and the actual link count determined by **fsck** do not agree, the reason may be:

- ❑ stored count not zero, actual count zero (no directory entry appears for the inode). **fsck** will link the disconnected file to the *lost+found* directory.
- ❑ stored count and actual count do not match and neither is zero (directory entry may have been removed without inode update). **fsck** replaces the stored link count with the actual link count.

Duplicate Blocks

Each inode has a list of all the blocks claimed by the inode. The **fsck** program compares each block number claimed by an inode to a list of allocated blocks. If a block number claimed by an inode is not on **fsck**'s allocated blocks list, then it is placed there. If a block number claimed by an inode is already on the allocated blocks list, it is placed on **fsck**'s *duplicate-blocks* list. **fsck** prompts the user to clear both inodes.

Bad Block Numbers

The **fsck** program checks each block number claimed by an inode for a value lower than the first data block, or greater than the last block in the system. If the block number is outside this range, the block number is bad. A large number of bad blocks in an inode may result when an indirect block is not written to the file system. The **fsck** program prompts the user to clear the inode.

Inode Size

Each inode contains a 32-bit **size** field. This **size** indicates the logical size of the file associated with that inode. A directory inode has the directory field bit set in the inode mode word. The directory size must be a multiple of 16 because a directory entry contains 16 bytes (2 for the inode number and 14 for the directory or file name). If the directory size is not a multiple of 16, **fsck** sends a warning. The situation remains unchanged because **fsck** does not have enough information to correct the misalignment.

For a regular file the **fsck** program checks the consistency of the inode **size** field by dividing the number of characters in an inode (represented by the **size** field) by the logical block size, and then rounding up. One block is added for each indirect block associated with the inode. If the actual number does not match up, **fsck** sends a warning. This warning is normal for any file that contains a logical hole (such as a *core* file) or any preallocated file whose logical EOF is less than the physical size of the file.

Extent List (F5 Architecture Only)

fsck checks an **F5** extent list against the *ilist* of each file. If inconsistencies are found **fsck** asks the user if the extent list should be fixed.

Indirect Blocks

Indirect blocks are claimed by an inode. Therefore, inconsistencies in an indirect block directly affect the inode that owns it.

Inconsistencies that can be checked are:

- ☐ blocks already claimed by another inode
- ☐ block numbers outside the range of the file system

The consistency checks under "Duplicate Blocks" and "Bad Block Numbers" are performed the same way for the direct blocks of the inode.

Directory Data Blocks

Directories are distinguished from regular files by an entry in the mode field of the inode. Data blocks associated with a directory contain the directory entries.

fsck (used with the **-D** option) checks directory data blocks for the inconsistencies described below.

Directory Unallocated

If a directory entry inode number points to an unallocated inode, **fsck** can remove that directory entry. This condition occurs when the data blocks containing the directory entries are modified and written out while the inode is not yet written out.

Bad Inode Number

If a directory entry inode number is pointing beyond the end of the inode list, **fsck** can remove that directory entry. This condition occurs if bad data is written into a directory data block.

Incorrect "." and ".." Entries

The directory inode number entry for "." should be the first entry in the directory data block. Its value should be equal to the inode number for the directory data block. The directory inode number entry for ".." should be the second entry in the directory data block. Its value should be equal to the inode number for the parent of the directory entry. If the directory inode numbers for "." and ".." are incorrect, **fsck** can replace them with the correct values.

Disconnected Directories

The **fsck** program checks the general connectivity of the file system. If directories are found that are not linked into the file system, **fsck** links the directory back into the file system in the *lost+found* directory. This condition arises when inodes are written to the file system without the corresponding directory data blocks being written to the file system. When a file is linked into the *lost+found* directory the owner of the file needs to be told about it.

Regular Data Blocks

Data blocks associated with a regular file hold the file's contents. **fsck** does not attempt to check the validity of the contents of a regular file's data blocks.

fsck Phases and Error Messages

The **fsck** program runs in phases. Some phases run only if they are called out in a command line option. When a phase completes, it displays a message. At the end of the program a summary message shows the number of inodes, blocks, and free blocks.

If **fsck** detects a problem in the file system organization, it sends a message and, in many cases, gives a prompt asking you what to do (fix the error, continue, and so forth). You can run **fsck** with the **-y** option, so that all questions are automatically answered **yes**. Similarly, the **-n** option specifies that all answers are to be answered **no**; this latter option is useful when you want to check a mounted file system to see if it needs to be unmounted and repaired.

The following abbreviations are used in the **fsck** error messages:

BLK	block number
DUP	duplicate block number
DIR	directory name
MTIME	time file was last modified
UNREF	unreferenced

The following pages list error messages that may appear when running **fsck**, with information on their meaning. The following variables are used in these messages; when the message appears on your screen, the variables are replaced by the indicated value.

<i>block</i>	block number
<i>name</i>	file (or directory) name
<i>inode</i>	inode number
<i>mode</i>	file mode
<i>uid</i>	user ID of the file's owner
<i>size</i>	file size
<i>time</i>	time file was last modified
<i>count</i>	link count or number of BAD, DUP, or MISSING blocks or number of files
<i>links</i>	corrected link count number
<i>free</i>	number of free blocks

General Error Messages

Three error messages may appear in any phase:

CAN NOT SEEK: BLK *block* (CONTINUE?)

The request to move to a specified block number *block* in the file system failed. The occurrence of this error condition indicates a serious problem (probably a hardware failure) that may require additional help.

CAN NOT READ: BLK *block* (CONTINUE?)

The request for reading a specified block number *block* in the file system failed. The occurrence of this error condition indicates a serious problem (probably a hardware failure or a mismatch between the slice table and the file system) that may require additional help.

CAN NOT WRITE: BLK *block* (CONTINUE?)

The request for writing a specified block number *block* in the file system failed. The disk may be write protected, or the slice table and file system may be mismatched.

Respond with a **y** (yes) to continue the file system check, or a **n** (no) to terminate the program.

These messages, like any **fsck** messages with a "(CONTINUE?)" prompt, may indicate serious file system damage.

- ☐ If these messages are accompanied by error messages from the disk device, you probably have a malfunctioning device. You should attempt to backup the file system (this copy may not be usable, but if you do not have a current backup of the file system, this backup may allow you to salvage some files), then fix the hardware problem before mounting the file system.
- ☐ If these messages are not accompanied by error messages from the disk device, they often mean that the file system was created with more blocks than exist on the physical partition of the disk. A newly created file system does not actually use all the space allocated, so the system allows you to create the file system, but **fsck** will detect the inconsistency.

If **fsck** is able to repair the file system, one or more files will probably be deleted in the process. Usually you will want to let the file checking continue to determine the severity of the problem. If the file check seems to run smoothly after a few of these, you should let it finish, then run another file system check to ensure that the problems are solved. You may then be able to restore from a backup any files that were deleted.

Sometimes **fsck** will produce an endless stream of messages with a "(CONTINUE?)" prompt. In this case, if you have a recent backup of the file system, you may want to terminate the file system check with a **n** response and restore it from the backup. If you do not have a recent backup, you might want to let the consistency check continue, to see if any files can be salvaged. You will still need to restore the file system, then copy in any files that were salvaged.

Initialization Phase

Command line syntax is checked. Before **fsck** performs the file system check it sets up some tables and opens some files. The **fsck** program terminates on initialization errors.

Phase 1: Check Blocks and Sizes

This phase checks the inode list and reports errors that result from:

- ☐ checking inode types
- ☐ setting up the zero-link-count table
- ☐ examining inode block numbers for bad or duplicate blocks
- ☐ checking inode size
- ☐ checking inode format
- ☐ checking **F5** contiguous extents

Phase 1 has three types of error messages:

1. Information messages
2. Messages with a **CONTINUE?** prompt

Use an **n**(no) response to terminate the program. Use a **y**(yes) response to continue the program. A complete check of the file system is not usually possible after such an error is found. See the discussion on messages with a **CONTINUE?** prompt in the "General Error Messages" section in this chapter.

3. Messages with a **CLEAR?** prompt

Use an **n**(no) response to tell **fsck** to ignore the condition (in other words, do not to attempt repair). This is appropriate only if you plan to either repair the file system yourself with **fsdb**(1M) or restore the file system from a backup, or if you are running **fsck** only for information.

Respond **y**(yes) to have **fsck** attempt to fix the problem. The inode is deallocated by zeroing its contents. You may get an "unallocated inode" error condition in Phase 2 for each directory entry that points to this inode.

I-BLOCK CORRUPT BLOCK=*block* INODES=*inode inode* UNRECOVERABLE

This is an informational message that occurs only when the **-r** option is used. The recently redirected block, *block*, listed in the bad block file given by the **-r** option, was found to be in the inode list. As a result, the information for the *inode* contained therein (and, therefore, the files they described) is lost. The list of file/dir names affected will appear in Phase 2 as **UNALLOCATED**.

Phase 1B: Rescan for More Duplicate Blocks

When a duplicate block is found in the file system, the file system is rescanned to find the inode that previously claimed that block. When the duplicate block is found, the following information message is printed:

block DUP I=*inode*

The specified inode contains the specified block number, which is already claimed by another inode. This error condition invokes the **BAD/DUP** error condition in Phase 2. inodes with overlapping blocks may be determined by examining this error condition and the **DUP** error condition in Phase 1.

Phase 2: Check Path Names

This phase removes directory entries pointing to bad inodes found in Phase 1 and Phase 1B. It reports errors that result from:

- ☐ root inode mode and status checking
- ☐ finding directory inode pointers out of range
- ☐ directory entries pointing to bad inodes

Phase 2 has 4 types of error messages:

1. Information messages
2. Messages with a **FIX?** prompt

An **n**(no) response terminates the program. A **y**(yes) response changes the root inode type to "directory". If the root inode data blocks are not directory blocks, a very large number of error conditions are produced.

3. Messages with a **CONTINUE?** prompt

An **n**(no) response terminates the program. A **y**(yes) response says to ignore the **DUPS/BAD** error condition in the root inode of the file system and attempt to continue to run the file system check. If the file system's root inode is not correct, this may result in a large number of other **fsck** error conditions.

The following lists the Phase 1 errors and provides an explanation of the error.

UNKNOWN FILE TYPE I=inode (CLEAR?)

The mode word of *inode* suggests that the inode is not a pipe, special character inode, regular inode, or directory inode.

LINK COUNT TABLE OVERFLOW (CONTINUE?)

An internal table for **fsck** containing allocated inodes with a link count of zero has no more room.

block BAD I=inode

The specified *inode* contains the specified *block* number which has a number lower than that of the first data block in the file system or greater than that of the last block in the file system. This error condition may invoke the EXCESSIVE BAD BLKS error condition in Phase 1 if the inode has too many block numbers outside the file system range. This error condition invokes the BAD/DUP error condition in Phase 2 and Phase 4.

EXCESSIVE BAD BLOCKS I=inode (CONTINUE?)

There is more than a tolerable number (usually 10) of blocks with a number lower than the number of the first data block in the file system or greater than the number of the last block in the file system associated with *inode*.

block DUP I=inode

The specified *inode* contains the specified *block* number, which is already claimed by another inode. This error condition may invoke the EXCESSIVE DUP BLKS error condition in Phase 1 if the inode has too many block numbers claimed by other inodes. This error condition invokes Phase 1B and the BAD/DUP error condition in Phase 2 and Phase 4.

EXCESSIVE DUP BLKS I=inode (CONTINUE?)

There is more than a tolerable number (usually 10) of blocks claimed by other inodes.

DUP TABLE OVERFLOW (CONTINUE?)

An internal table in **fsck** containing duplicate block numbers has no more room.

POSSIBLE FILE SIZE ERROR I=inode

The size of the specified inode does not match the actual number of blocks used by the inode. This is only a warning, and is normal for *core* files and preallocated files. This message is printed only if the **-q** option is not used.

DIRECTORY MISALIGNED I=inode

The size of a directory inode is not a multiple of 16. This is only a warning. This message is printed only if the **-q** option is not used.

PARTIALLY ALLOCATED INODE I=inode (CLEAR?)

The specified *inode* is in an intermediate state.

4. Messages with a REMOVE? prompt

An **n**(no) response causes the error condition to be ignored; it is appropriate only if you intend to take other measures to fix the problem. A **y**(yes) response causes the file or directory to be deleted; after the file system check finishes, you can restore the file or directory from a backup.

The following lists the Phase 2 errors and provides an explanation of the error.

ROOT INODE UNALLOCATED. TERMINATING

The root inode (always inode number 2) has no allocate mode bits. The occurrence of this error condition indicates a serious problem. The program stops.

ROOT INODE NOT DIRECTORY (FIX?)

The root inode (usually inode number 2) is not directory inode type.

DUPS/BAD IN ROOT INODE (CONTINUE?)

Phase 1 or Phase 1B found duplicate blocks or bad blocks in the root inode (usually inode number 2) for the file system.

I OUT OF RANGE I=inode NAME=name (REMOVE?)

A directory entry *name* has an inode number *inode* that is greater than the end of the inode list.

UNALLOCATED I=inode OWNER=uid MODE=mode SIZE=size MTIME=time NAME=name (REMOVE?)

A directory entry *name* has an *inode* number without allocate mode bits. The owner's *uid*, *mode*, *size*, and modify *time* are also printed. If the file system is not mounted and the **-n** option was not specified, the entry is removed automatically if the inode it points to is character size 0.

DUP/BAD I=inode OWNER=uid MODE=mode SIZE=size MTIME=time DIR=name (REMOVE?)

Phase 1 or Phase 1B found duplicate blocks or bad blocks associated with directory entry *name*, directory *inode*. The owner's *uid*, *mode*, *size*, and modify *time* are also printed.

DUP/BAD I=inode OWNER=uid MODE=mode SIZE=size MTIME=time FILE=name (REMOVE?)

Phase 1 or Phase 1B found duplicate blocks or bad blocks associated with file entry *name* and *inode*. The owner's *uid*, *mode*, *size*, and modify *time* are also printed.

BAD BLK B IN DIR I=inode OWNER=uid MODE=mode SIZE=size MTIME=time

This message only occurs when the **-D** option is used. A bad block was found in the specified directory inode. Error conditions looked for in directory blocks are nonzero padded entries, inconsistent "." and ".." entries, and embedded slashes in the name field. This error message means that the user should at a later time either remove the directory inode if the entire block looks bad or change (or remove) those directory entries that look bad.

FILE/DIR CORRUPT *I=inode OWNER=uid MODE=mode SIZE=size MTIME=time FILE/DIR=dev/file*

This message is informational and only occurs when the **-r** option is used. One of the blocks in the bad block file given by the **-r** option was found to be a direct or indirect data block of the indicated file or directory. Some portion of the file or directory has been zeroed by the redirection process.

Phase 3: Check Connectivity

This phase concerns itself with the directory connectivity seen in Phase 2. It reports errors that result from:

- unreferenced directories
- missing or full **lost+found** directories

Phase 3 has 2 types of error messages:

1. Information messages
2. Messages with a **RECONNECT?** prompt

A **y**(yes) response says to redirect the specified inode to the file system in the *lost+found* directory. This may involve *lost+found* error conditions if there are problems connecting the directory inode to the *lost+found* directory. If the link is successful, **fsck** returns a **CONNECTED** information message.

An **n**(no) response says to ignore the error condition. This invokes the **UNREF** error condition in Phase 4. An **n** response is appropriate only if you intend to take other measures to fix the problem.

The following lists the Phase 3 errors and provides an explanation of the error.

UNREF DIR *I=inode OWNER=uid MODE=mode SIZE=size MTIME=time (RECONNECT?)*

The directory *inode* was not connected to a directory entry when the file system was traversed. The owner's *uid*, *mode*, *size*, and modify *time* of the directory's *inode* are also printed. The **fsck** program forces the reconnection of a nonempty directory.

SORRY. NO lost+found DIRECTORY

There is no *lost+found* directory in the root directory of the file system; **fsck** ignores the request to link a directory in *lost+found*. This invokes the **UNREF** error condition in Phase 4. Possible problem with access modes of *lost+found*.

SORRY. NO SPACE IN lost+found DIRECTORY

There is no space to add another entry to the *lost+found* directory in the root directory of the file system; **fsck** ignores the request to link a directory in *lost+found*. This invokes the **UNREF** error condition in Phase 4. Clean out unnecessary entries in *lost+found* or make *lost+found* larger with the **mklost+found** command.

DIR I=*inode1* CONNECTED. PARENT WAS I=*inode2*

This is an advisory message indicating a directory inode was successfully connected to the *lost+found* directory. The inode number of the parent directory for the directory inode is replaced by the inode number of the *lost+found* directory.

Phase 4: Check Reference Counts

This phase checks the link count information seen in Phases 2 and 3. It reports errors that result from:

- ☐ unreferenced files
- ☐ missing or full *lost+found* directory
- ☐ incorrect link counts for files, directories, or special files
- ☐ unreferenced files and directories
- ☐ bad and duplicate blocks in files and directories
- ☐ incorrect total free-inode counts

Phase 4 has 5 types of error messages:

1. Information messages
2. Messages with a **RECONNECT?** prompt

An **n(no)** response says to ignore this error condition. This invokes a **CLEAR** error condition later in Phase 4. A **y(yes)** response says to reconnect the inode to a file in the *lost+found* directory. This can cause a *lost+found* error condition in this phase if there are problems connecting the inode to the *lost+found* directory.

3. Messages with a **CLEAR?** prompt

An **n(no)** response says to ignore the error condition; this is appropriate only if you intend to take other measures to fix the problem. A **y(yes)** response says to deallocate the inode by zeroing its contents.

4. Messages with an **ADJUST?** prompt

An **n(no)** response says to ignore the error condition; this is appropriate only if you intend to take other measures to fix the problem. A **y(yes)** response says to replace the link count of the file with a corrected link count number.

5. Messages with a FIX? prompt

An *n*(no) response says to ignore the error condition; it is appropriate only if you intend to take other measures to fix the problem. A *y*(yes) response says to replace the count in the super block with the actual count.

The following lists the Phase 4 errors and provides an explanation of the error.

UNREF FILE I=*inode* OWNER=*uid* MODE=*mode* SIZE=*size* MTIME=*time* (RECONNECT?)

The specified *inode* was not connected to a directory entry when the file system was traversed. If the *-n* option is omitted and the file system is not mounted, empty files are cleared automatically. Nonempty files are not cleared.

SORRY. NO *lost+found* DIRECTORY

There is no *lost+found* directory in the root directory of the file system; *fsck* ignores the request to link a file in *lost+found*. This invokes the CLEAR error condition later in Phase 4. Possible problem with access modes of *lost+found*.

SORRY. NO SPACE IN *lost+found* DIRECTORY

There is no space to add another entry to the *lost+found* directory in the root directory of the file system; *fsck* ignores the request to link a file in *lost+found*. This invokes the CLEAR error condition later in Phase 4. Check size and contents of *lost+found*.

(CLEAR)

The *inode* mentioned in the immediately previous UNREF error condition cannot be reconnected.

LINK COUNT FILE I=*inode* OWNER=*uid* MODE=*mode* SIZE=*size* MTIME=*time* COUNT=*count* SHOULD BE *links* (ADJUST?)

The link count for the specified *inode* (which is a file) is *count* but should be *links*.

LINK COUNT DIR I=*inode* OWNER=*uid* MODE=*mode* SIZE=*size* MTIME=*time* COUNT=*count* SHOULD BE *links* (ADJUST?)

The link count for the specified *inode* (which is a directory) is *count* but should be *links*.

LINK COUNT *name* I=*inode* OWNER=*uid* MODE=*mode* SIZE=*size* MTIME=*time* COUNT=*count* SHOULD BE *links* (ADJUST?)

The link *count* for the specified *inode* should be *links*.

UNREF FILE I=*inode* OWNER=*uid* MODE=*mode* SIZE=*size* MTIME=*time* (CLEAR?)

The specified *inode* (which is a file) was not connected to a directory entry when the file system was traversed. If the *-n* option is omitted and the file system is not mounted, empty files are cleared automatically. Nonempty directories are not cleared.

UNREF DIR I=*inode* OWNER=*uid* MODE=*mode* SIZE=*size* MTIME=*time* (CLEAR?)

The specified *inode*, which is a directory, was not connected to a directory entry when the file system was traversed. If the *-n* option is omitted and the file system is not mounted, empty directories are cleared automatically. Nonempty directories are not cleared.

BAD/DUP FILE *I=inode OWNER=uid MODE=mode SIZE=size MTIME=time (CLEAR?)*

Phase 1 or Phase 1B found duplicate blocks or bad blocks associated with the specified file *inode*.

BAD/DUP DIR *I=inode OWNER=uid MODE=mode SIZE=size MTIME=time (CLEAR?)*

Phase 1 or Phase 1B found duplicate blocks or bad blocks associated with the specified directory *inode*.

FREE INODE COUNT WRONG IN SUPERBLK (FIX?)

The actual count of the free inodes does not match the count in the super block of the file system. If the **-q** option is specified, the count will be fixed automatically in the super block.

Phase 5: Check Free List or Free Block Bitmap

This phase checks the organization of the file system's free blocks. For the S5 architecture, this means checking the free block list; for the F5 architecture, this means checking the free block bitmap. Phase 5 reports error conditions resulting from:

- ☐ bad blocks in the free-block list or the free bitmap
- ☐ duplicate blocks in the free-block list or the free bitmap
- ☐ unused blocks from the file system not in the free-block list
- ☐ total free-block count incorrect

Phase 5 has 4 types of error messages:

1. Information messages
2. Messages that have a **CONTINUE?** prompt

An **n**(no) response says to terminate the program. A **y**(yes) response says to ignore the rest of the free block list and continue execution of **fsck**. This error condition will always invoke the "BAD BLKS IN FREE LIST" error condition later in Phase 5.

3. Messages that have a **FIX?** prompt

An **n**(no) response says to ignore the error condition; it is appropriate only if you intend to take other measures to fix the problem. A **y**(yes) response says to replace the count in the free block list (S5) or free bitmap (F5) with the actual count.

4. Messages that have a SALVAGE? prompt

An **n**(no) response says to ignore the error condition; it is appropriate only if you intend to take other measures to fix the problem.

A **y**(yes) response causes the free block list or free bitmap to be replaced. This will automatically invoke Phase 6. The new free block list or free bitmap is ordered to reduce the time spent waiting for the disk to rotate into position.

The following lists the Phase 5 errors and provides an explanation of the error.

EXCESSIVE BAD BLKS IN FREE LIST/BITMAP (CONTINUE?)

The free-block list or bitmap contains more than a tolerable number (usually 10) of blocks with a value less than the first data block in the file system or greater than the last block in the file system.

EXCESSIVE DUP BLKS IN FREE LIST/BITMAP (CONTINUE?)

The free-block list or bitmap contains more than a tolerable number (usually 10) of blocks claimed by inodes or earlier parts of the free-block list.

BAD FREEBLK COUNT

The count of free blocks in a free-list block or free bitmap is greater than 50 or less than 0. This error condition will always invoke the BAD FREE LIST condition later in Phase 5.

count BAD BLKS IN FREE LIST

The specified *count* number of blocks in the free-block list or free bitmap has a block number lower than the first data block in the file system or greater than the last block in the file system. This error condition will always invoke the BAD FREE LIST condition later in Phase 5.

X DUP BLKS IN FREE LIST

X blocks claimed by inodes or earlier parts of the free-list block were found in the free-block list. This error condition will always invoke the BAD FREE LIST condition later in Phase 5.

count BLK(S) MISSING

The specified *count* number of blocks unused by the file system were not found in the free-block list (S5) or in the free bitmap (F5). This error condition will always invoke the BAD FREE LIST condition later in Phase 5.

FREE BLK COUNT WRONG IN SUPERBLOCK (FIX?)

The actual count of free blocks does not match the count in the super block of the file system.

BAD FREE LIST (SALVAGE?)

This message is always preceded by one or more of the Phase 5 information messages. If the **-q** option is specified, the free-block list or free bitmap will be salvaged automatically.

Phase 6: Salvage Free List

This phase reconstructs the free-block list (S5 architecture) or the free-block bitmap (F5 architecture). It has only one advisory message:

DEFAULT FREE BLOCK LIST SPACING ASSUMED

This advisory message means: (1) the blocks-to-skip (gap) is greater than the blocks-per-cylinder, (2) the blocks-to-skip is less than 1, (3) the blocks-per-cylinder is less than 1, or (4) the blocks-per-cylinder is greater than 500. The values of 7 blocks-to-skip and 400 blocks-per-cylinder are used.

Because the values used are incorrect for the computer, care must be taken to specify correct values with the `-sX` option on the command line. See `fsck(1M)` and `mkfs(1M)` for further details.

Cleanup Phase

Once a file system has been checked, a few cleanup functions are performed. The cleanup phase displays advisory messages about the file system and status of the file system.

count files links blocks free free

This is an advisory message indicating that the file system checked contained *count* files using *links* blocks, leaving *free* blocks free in the file system.

***** BOOT REALIX (NO SYNC!) *****

This is an advisory message indicating that a mounted file system or the root file system has been modified by `fsck`. If the operating system is not rebooted immediately without `sync`, the work done by `fsck` may be undone by the in-core copies of tables the operating system keeps. If the `-b` option of the `fsck` command was specified and the file system is `root`, a reboot is automatically done.

***** FILE SYSTEM WAS MODIFIED *****

This is an advisory message indicating that the current file system was modified by `fsck`.

***** ROOT REMOUNTED *****

This is an advisory message indicating that the root file system was remounted after `fsck` made modifications.

INDEX

INDEX

- accept(1M), 8-6
- accounting, 10-1 to 10-13
- acct directory, 10-1
- acctcms(1M), 10-6
- acctdisk(1M), 10-2, 10-5
- acctwtmp(1M), 10-2
- addresses, internet, 11-1
- ADJTFAST, 6-73
- ADJTFDELTA, 6-73
- ADJTRATE, 6-73
- adm directory, 10-7
- administrative files and directories, B-1 to B-10
- administrative privileges, 1-2
- AIOCNTPRC, 6-28, 6-71
- AIOMAXAIO, 6-28, 6-71
- AIOMAXDAEM, 6-71
- AIOMAXPRC, 6-28, 6-71
- anonymous, 11-12
- application programs, 6-14, 6-24, 6-30, 6-39
- application programs, efficiency of, 6-12
- asynchronous I/O, 6-22, 6-71
- atjobs directory, B-4, B-7
- AUTODUMP, 6-56

- BACKSPACE key, 3-17
- BADDISKS, 6-62
- bdflush, 6-55, 6-59
- BDFLUSHR, 6-55
- bfree(1M), 6-13, 6-42, 6-51
- /bin/sh, 7-2
- binary semaphore mechanism, 6-23
- binary semaphores, 6-66
- block devices, activity of, 6-26
- block mode display processing, 7-11
- block special device files, A-3
- blocks and inodes, free, 6-9
- blocks, free, 6-13
- boot, C-4
- bootwait, C-4
- BSCNTPRC, 6-23, 6-66
- BSCNTSYS, 6-23, 6-66
- BSMAXPRC, 6-23, 6-66
- BTAPEDEV, 6-62
- buffer cache, 1-4, 4-4, 6-4, 6-12, 6-22, 6-50

- buffers, flushing, 6-55

- C library I/O subroutines, 6-49
- callout table entries, 6-46
- cancel(1), 8-1
- CDLIMIT, 6-49
- character display options, 3-17
- character display settings, 7-1
- character list buffers, 6-46
- character special device files, A-3
- chargefee(1M), 10-2
- chmod, 2-10
- CICNTPROC, 6-72
- CIMAXPROC, 6-72
- CIMAXSYS, 6-72
- ckpacct(1M), 10-5
- classes of internet network, 11-2
- CLOCKRES, 6-73
- Command Summary, Daily, 10-13
- common event mechanism, 6-27, 6-72
- configurations, creating new, 9-5
- configuring the system, 9-1 to 9-9
- connected interrupt mechanism, 6-25
- connected interrupts, 6-72
- cpio(1), 4-12, 5-6
- cpio(1M), 6-8
- CPRAM, 6-62
- CPU utilization, 6-34
- CPUBOARD, 6-62
- crash icount(1M), 4-5
- crash mount, 4-10
- Creating a File System
 - mkfs, 4-6, 4-7
- cron, 6-20, C-2
- cron facility, B-4
- cron/log, 4-15
- crontab(1), B-5
- crontab, to clean out LP spooler log files, 8-13
- crontabs directory, B-4, B-5
- CTAPEDEV, 6-62

- daemons, setting priorities of, 6-59
- Daily Command Summary, 10-13
- Daily Report, 10-11

INDEX [continued]

- Daily Usage Report, 10-12
- dcopy, 4-17
- dcopy(1M), 6-13
- dcopy(1M), 4-17, 5-8
- dd, 5-17
- dd(1M), 5-8
- debug kernel, booting, 2-6
- default file for LP Spooling, 8-11
- DEFBUFSIZ, 6-50
- /dev directory, A-1, A-3
- /dev subdirectories, A-3
- /dev/kmem, A-5
- /dev/mem, A-5
- /dev/smdev, A-5
- device driver, A-1
- devices, enabling and disabling, 9-6
- df, 4-14
- df(1), 6-9
- dfsc, 5-15
- directory organization, 6-9
- disable(1), 8-1
- disabling configurations and devices, 9-6
- disjoint I/O, 6-76
- disk activity, 6-26
- disk bottleneck, 6-13, 6-34
- disk bottlenecks, 6-10
- disk buffers, flushing, 6-55
- disk devices, most efficient areas, 6-13
- disk I/O balancing, 6-4
- disk partitions, 4-3
- disk slice, *see* disk partitions
- disk storage, 6-3
- disk usage, monitoring, 4-14
- disktacct file, 10-5
- diskusg(1M), 10-5
- DJNTCNT, 6-76
- DJNTMAXSZ, 6-76
 - impact on mkfs(1M), 6-76
- dodisk(1M), 10-1, 10-2, 10-5
- driver, A-1
- drivers, enabling and disabling, 9-6
-
- enable(1), 8-1
- enabling configurations and devices, 9-6
-
- error logging, C-2
- /etc/checklist, 5-16
- /etc/fstab, 5-14
- /etc/fstab file, 4-10
- /etc/gettydefs, 3-17, 7-5
- /etc/hosts file, 11-5
- /etc/hosts.equiv, 11-7, 11-13
- /etc/init.d directory, C-2
- /etc/inittab, 7-1
- /etc/inittab file, C-4
- /etc/passwd, 2-13, 7-2
- /etc/password
 - changing or deleting entries, 3-2, 3-3
- /etc/profile, 7-2
- /etc/protocols file, 11-11
- /etc/rc0 file, C-8
- /etc/rc0.d directory, C-8
- /etc/shutdown file, C-8
- /etc/stdprofile, 3-17
- /etc/TIMEZONE, B-3
- /etc/utmp file, 10-2
- /etc/wtmp, 4-15
- /etc/wtmp file, 10-2
- Ethernet, 11-1 to 11-22, *see* Trouble Analysis
- Guide
- events, 6-72
- events, common, 6-27
- EVTCPNTPRC, 6-27, 6-72
- EVTMAXPRC, 6-27, 6-72
- EVTMAXQUE, 6-27, 6-72
- export, 3-16
-
- F5 file system architecture, 1-4, 4-3
- F5CLUSSIZ, 6-53
- fd2log file, 10-6
- FIFO file for LP Spooling, 8-11
- file access operations, 6-21
- file access, measuring for file systems, 6-40
- file system, 5-1 to 5-34
 - architecture, 6-13
 - definition, 4-3
 - logical block size and the buffer cache, 6-50
 - reorganizing, 6-8
 - structures, 6-4

INDEX [continued]

- file system architecture, 1-4, 4-3
- File System Switch, 6-47
- file systems
 - creating, 4-6
 - expanding, 4-13
 - labeling, 4-8
 - monitoring disk usage of, 4-14
 - monitoring space in, 4-14
 - mounting, 4-9, C-2
 - moving users to, 4-12
 - read-only, 4-9
 - reconfiguring, 4-13
 - reorganizing, 4-17
 - synchronous, 4-9
 - unmounting, 4-11
- file table entries, 6-35
- file/record lock table entries, 6-35, 6-46
- files and directories, administrative, B-1 to B-10
- finc(1M), 5-11
- find, 4-12
- fiscal directory, 10-9
- FLCKREC, 6-35, 6-45, 6-46
- frec(1M), 5-6, 5-12
- free memory, 6-11
- free-block bitmap, D-3
- fsck -p, 5-15
- fsck phases
 - check connectivity, D-13, D-14
 - check free blocks, D-14 to D-16
 - check path name, D-9
 - check reference, D-14 to D-16
 - initialization, D-9
 - salvage, D-14
- fsck(1M), 4-11
- fsdb, 5-17 to 5-34
- fsdb fc command, 5-23
- fsdb fd command, 5-22
- ftp(1), 11-12, 11-15
- fwtmp(1M), 10-10
- getty lines in inittab, 7-4
- getty(1M), 7-1
- gettydefs, *see* /etc/gettydefs
- GID, 2-10
- GPGSHI, 6-58
- GPGSLO, 6-58
- GPGSMK, 6-62
- gprof(1), 6-12, 6-14
- grep, 2-11
- hardware configuration, 6-3
- hash buckets, 6-52
- hitimed, 6-60
- holidays file, 10-4
- holidays, chart of for United States, 10-4
- host names, 11-3
- hosts file, 11-5
- hosts.equiv, 11-7, 11-13
- icount(1M), 4-5, 6-13, 6-40, 6-51
- inetd daemon, 11-13
- init(1M), C-1
- initdefault, C-4
- inittab file, C-4
- inode table entries, 6-35, 6-47
- inodes, free, 6-13
- interface programs for printers, 8-8
- Internet, 11-1 to 11-22
- interprocess communication activities, 6-29
- interrupt latency, 6-10
- ITICNTPROC, 6-33, 6-73
- ITIMAXK, 6-73
- ITIMAXPROC, 6-33, 6-73
- ITIMAXSYS, 6-33, 6-73
- job accounting, 10-1 to 10-13
- kernel and paging parameters, 6-44
- kernel profiler
 - prfdc, 6-17
 - prfld, 6-17
 - prfpr, 6-17
 - prfsnap, 6-17
 - prfstat, 6-17
- kernel virtual address space, 6-48

INDEX [continued]

labeling the File System

- labelit, 4-8
- labelit(1M), 4-2, 4-8
- lastdate file, 10-6
- line discipline interrupts, 6-60
- line printer administration, 8-1 to 8-13
- link level address, mapping an Internet address to, 11-4
- ln(1), A-4
- Local Area Network (LAN), Ethernet, 11-1 to 11-22
- lock files for LP Spooling, 8-12
- log file for LP Spooling, 8-11
- log files, cleaning up, 4-15
- logical block size, 1-4
- logical block sizes, 4-4, 6-13, 6-50
- login cycle, 7-1
- login(1), 7-1
- loopback connection, 11-5
- lost+found, 4-2, 4-11, 4-15
- lotimed, 6-60
- LP Spooler, starting, C-2
- lp(1), 8-1
- lp.cnfg(1M), 8-1
- lpadmin(1M), 8-2
- lpmove(1M), 8-5
- lpsched(1M), 8-4, 8-9
- lpshut(1M), 8-5
- lpspooler, 4-1
- lpstat(1), 8-1

M204, 6-62

MACHINE, 6-62

mail, 3-20

mailx, 3-20

major and minor numbers, A-2

maximum time slice for user processes, 6-49

maxmem, 6-47

MAXPMEM, 6-45, 6-46

MAXPRBUFS, 6-45, 6-46

MAXRDAHEAD, 6-62

MAXSEPGCNT, 6-63

MAXSLICE, 6-45, 6-49

MAXSWAPLIS, 6-58

MAXTREQ, 6-62

MAXUMEM, 6-45, 6-49

MAXUP, 6-45, 6-49

MBCNT, 6-53, 6-54

MBMAXBLKS, 6-53

MBMAXSZ, 6-53, 6-54

memory

- special device files, A-5

memory dumps, 4-1

memory,

- free, 6-32

- freeing, 6-12

- waiting for, 6-34

message operations, 6-29

messages, 6-66

MINARMEM, 6-58

MINASMEM, 6-58

MINPAGMEM, 6-58

mkfs(1M), 4-2, 4-6

mkfs(1M), DJNTMAXSZ impact on, 6-76

mklost+found, 4-2

mknod(1M), A-4

model directory for LP Spooling, 8-12

model interface programs for printers, 8-8

monacct(1M), 10-3, 10-6

monitoring

- disk usage, 4-14

- file system space, 4-14

Monitoring Directories

- /usr/adm/sulog, 4-16

mount entries, 6-47

mount(1M), 4-2, 4-9

Mounting the File System

- mount, 4-9, 4-10

- umount, 4-9, 4-10

MSGMAP, 6-67

MSGMAX, 6-67

MSGMNB, 6-67

MSGMNL, 6-67

MSGSEG, 6-68

MSGSSZ, 6-68

MSGTQL, 6-68

multi-block transfers, tunable parameters for, 6-53

INDEX [continued]

multiple swap areas, 6-5
multiuser state, C-2

NAUTOUP, 6-55
NBUF128K, 6-50
NBUF16K, 6-50
NBUF1K, 6-50
NBUF2K, 6-50
NBUF32K, 6-50
NBUF4K, 6-50
NBUF64K, 6-50
NBUF8K, 6-50
NCALL, 6-45, 6-46, 6-73
ncheck, 2-11
NCLIST, 6-45, 6-46
networking devices, 7-3
NFILE, 6-35, 6-45, 6-47
NHBUF, 6-50, 6-52
NINODE, 6-35, 6-45, 6-47
nite directory, 10-8
NLOG, 6-65
NMOUNT, 6-45, 6-47
NMUXLINK, 6-64
NODE, 6-61
nodename, setting, C-2
NOFILES, 6-45, 6-49
NPBUF, 6-56
NPROC, 6-35, 6-45, 6-47
NQUEUE, 6-65
NREGION, 6-45, 6-48
NSSINODE, 6-45
NSTREAM, 6-35, 6-64
NSTREVENT, 6-64
NSTRPUSH, 6-64
NTTYDREQ, 6-62
number of open files, 6-49
NUMTIMOD, 6-64

off, C-5
once, C-4
ondemand, C-5
onesecond, 6-59
open file table entries, 6-47
open files, 6-49

pacct, 4-15
pacct file, 10-2
page faults, 6-11, 6-30
paging, 4-3, 6-32
paging activity, 6-6, 6-30, 6-36
paging, excessive, 6-10, 6-11
parallel fsck, 5-15
partition, disk, *see* disk partitions
password aging, 3-3
\$PATH, 3-18
PATH, length of, 6-1
performance tools, 6-16 to 6-42
pgrpsigd, 6-59
PHYSBSIZE, 6-56
PHYSCNT, 6-56
physical I/O buffer, 6-56
physical input/output buffers, 6-56
physical memory, maximum amount to use in
 pages, 6-46
physical reads and writes, 6-22
plock(2), 6-5, 6-12
powerfail, C-5
powerwait, C-5
preallocation of file space, 1-5
prfd, 6-59
PRIBDFLUSH, 6-55, 6-59
PRIHITIMED, 6-60
PRILOTIMED, 6-60
prime time, establishing for accounting, 10-4
printers, 7-3
printers, administration of, 8-1 to 8-13
printing terminals, 3-17
PRIONESEC, 6-59
priorities, realtime, 1-5
PRIPGRPSIG, 6-59
PRIPRFD, 6-59
PRISTREAMS, 6-59
PRITTYD, 6-60
PRIVHAND, 6-60
privileges, administrative, 1-2
process
 active, 6-19
 paging, 6-32
 priorities, 6-19

SHMALL, 6-69
SHMBRK, 6-69
SHMMAX, 6-69
SHMMIN, 6-69
SHMMNI, 6-69
SHMSEG, 6-69
shutacct(1M), 10-2
shutdown, C-2
slice, disk, *see* disk partitions
SPCNT, 6-64
special device files, A-1 to A-8
 and file systems, 4-3
 block and character, A-3
 creating and linking, A-4
 system, A-6
 terminal devices, A-3, A-5
 TTY devices, 7-3
spooler programs, 4-1
SPTMAP, 6-45, 6-48
stderr, 6-49
stdin, 6-49
stdout, 6-49
sticky bit, 6-5, 6-12, 6-13, 6-32
STRCTLSZ, 6-65
"streamhead" structures, 6-35
STRLOFRAC, 6-64
STRMEDFRAC, 6-64
STRMSGSZ, 6-65
stty(1), 3-17, 7-1
stty(1), with printers, 8-8
su, 2-10
sulog, 4-15
sulog file, B-10
sum directory, 10-9
superuser privileges, 1-2
superuser, log of access, B-10
swap area, 4-3
swap areas, 6-5, 6-13, 6-32, 6-36
swap areas, automatic resizing of, 6-61
swap device, 6-13
synchronous file system, 4-9
SYS, 6-62
sysadm
 lineset, 7-7

mklineset, 7-8
modtty, 7-9
ttypgmt, 7-7
sysdump, 6-57
 WHOLEDUMP, 6-57
sysgen menu screens
 Standard Configuration, 9-6, 9-7
sysgen(1M), 1-4, 9-1 to 9-9
sysinit, C-4
SYSRESET, 6-56
systat(1), 6-3, 6-32
system buffer cache, 4-4
system calls, 6-24
system processes, setting priorities of, 6-59
system usage, 6-31

table overflow, 6-10, 6-35
table sizes, 6-35
tacct file
 fixing, 10-10
tail(1), 4-15
tar(1), 5-7
TCP/IP, 11-1 to 11-22, *see* Trouble Analysis
 Guide
telnet(1), 11-12, 11-17
TERM environmental variable, 7-11
TERM variable, 7-2
terminal device activities, 6-37
terminal devices
 special device files, A-5
terminals, printing, 3-17
terminfo directory, 7-11
terminfo(4), 7-2
termio(7), 7-1
text bit, 6-5
time zone, setting for the computer, B-3
timeout table entries, 6-46
timers, realtime, 6-33, 6-73
timex(1), 6-14, 6-39
/tmp, 4-1, 6-13
TMR_PRILOW, 6-73
TMR_UPDLOW, 6-73
tools for analyzing performance, 6-16 to 6-42
TTY management, 7-1 to 7-11

INDEX [continued]

process *[continued]*
 table entries, 6-35, 6-47
 process accounting, 10-1 to 10-13
 process accounting files, 4-15
 .profile, 7-2
 protocols file, 11-11
 prtacct(1M), 10-6
 ps(1), 6-3, 6-19
 pstat(1M), 8-4
 pstatus file for LP Spooling, 8-11
 putbuf, 4-15, 6-48
 PUTBUFSZ, 6-45, 6-48

 queuedefs file, B-7

 r-series commands, 11-7, 11-12, 11-13
 rcp(1), 11-7, 11-8, 11-12, 11-16
 read and write permission, 3-17
 realtime
 setrtusers, 3-9
 realtime privileges, 1-5
 realtime processes, priorities of, 1-5
 region table entries, 6-48
 reject(1M), 8-6
 REL, 6-62
 remsh(1), 11-7, 11-8, 11-12, 11-15
 reports, accounting, 10-11 to 10-13
 requests directory for LP Spooling, 8-12
 resident realtime processes, 6-58
 resident(2), 6-5
 RESIZESWAP, 6-61
 respawn, C-5
 .rhosts, 11-8
 rlogin(1), 11-7, 11-8, 11-12, 11-17
 root file system
 checking and repairing, 5-15
 contents of, B-2
 reorganizing, 4-17
 run queue, 6-31
 run state transitions, files that control, C-1
 runacct(1M), 10-1, 10-3, 10-6

 S5 file system architecture, 1-4

 sadp(1M), 6-20
 SANE, 7-6
 sar, 6-20 to 6-38
 -A, 6-38
 -a, 6-21
 -B, 6-23
 -b, 6-12, 6-22
 -C, 6-25
 -c, 6-24
 -d, 6-10, 6-13, 6-26
 -E, 6-27
 -I, 6-28
 -m, 6-29
 -p, 6-11, 6-30
 -q, 6-31
 -r, 6-11, 6-32
 -T, 6-33
 -u, 6-10, 6-24, 6-34
 -v, 6-10, 6-12, 6-35
 -w, 6-36
 -wu, 6-10
 -y, 6-10, 6-37
 sar b, 6-52
 savedump, 6-57
 SEMAEM, 6-68
 semaphore operations, 6-29
 semaphores, 6-66
 SEMMAP, 6-68
 SEMMNI, 6-68
 SEMMNS, 6-68
 SEMMNU, 6-68
 SEMMSL, 6-68
 SEMOPM, 6-68
 SEMUME, 6-68
 SEMVMX, 6-68
 set-user identification, 2-10
 setpri(1R), 6-19
 setrtusers, 3-9
 setslice(2), 6-49
 setuid bits, checking for, 2-11
 shadow password file, 2-13
 shared memory, 6-66
 shell programming, 6-15

INDEX [continued]

ttyd, 6-60
 ttymgm, 7-7
 TZ variable, B-3

 UID, 2-10
 umask, 3-18
 umount(1M), 4-11
 usage patterns, 6-10, 6-14
 Usage Report, Daily, 10-12
 user's virtual address space, 6-49
 users, moving to a file system, 4-12
 /usr file system
 contents of, B-2
 /usr/adm directory, 10-7
 /usr/adm/*pacct*, 4-15
 /usr/adm/acct/fiscal directory, 10-9
 /usr/adm/acct/nite directory, 10-8
 /usr/adm/acct/sum directory, 10-9
 /usr/adm/pacct file, 10-2
 /usr/adm/putbuf, 4-15
 /usr/adm/sulog, 2-10, 4-15
 /usr/lib/acct directory, 10-1
 /usr/lib/acct/holidays file, 10-4
 /usr/lib/cron/iog, 4-15
 /usr/mail, 4-15
 /usr/spool, 4-1
 /usr/tmp, 4-1, 6-13
 utmp file, 10-2
 UUCP, starting, C-2

 VERSION, 6-62
 vhand, 6-5, 6-58, 6-60
 VHANDR, 6-58
 VHNDFRAC, 6-58
 virtual address space, 6-48
 volcopy, 3-17
 volcopy(1M), 6-8

 wait, C-4
 WAITRMC, 6-62
 wtmp, 4-15
 wtmp file, 10-2
 fixing, 10-10
 wtmpfix(1M), 10-10

MODCOMP, founded in 1970,
is a worldwide supplier of
real-time systems, products,
and services. MODCOMP is an
AEG company.

Corporate Headquarters:
Modular Computer Systems, Inc.
1650 West McNab Road
P.O. Box 6099
Ft. Lauderdale, FL 33309-1088
Tel: (305) 974-1380
Twx: 510-956-9414

International Headquarters:
Modular Computer Services, Inc.
The Business Centre
Molly Millars Lane
Wokingham, Berkshire
RG11 2JQ, UK
Tel: 0734-786808
Fax: 0734-776399

Americas Headquarters:
Modular Computer Systems, Inc.
1650 West McNab Road
P.O. Box 6099
Ft. Lauderdale, FL 33309-1088
Tel: (305) 974-1380
Twx: 510-956-9414

MODCOMP sales and service
offices are located throughout
the world.

Copyright © 1993, Modular Computer Systems, Inc.
MODCOMP and Modular Computer Systems, Inc. are registered trademarks of Modular Computer Systems, Inc.