ConvexOS System Manager's Guide

Document No. 710-001430-207

Tenth Edition, Rev. 1

January, 1990

CONVEX Computer Corporation

Richardson, Texas USA

ConvexOS System Manager's Guide
Order No. DSW-004
Tenth Edition, Revision 1

Printed in the United States of America

ConvexOS
System Manager's Guide

Document Number 710-001430-207

10th Edition, Revision 1

The enclosed *ConvexOS System Manager's Guide* (10th edition, revision 1, document number 710-001430-207) is the current documentation for the latest release of ConvexOS V8.0. Information in this edition supercedes all previous editions of the *ConvexOS System Manager's Guide*. Please replace previous editions, including the 10th edition (document number 710-001430-206) published in December 1989, with the enclosed manual. The ConvexOS V8.0 Release Notes refer to the 10th edition of the *ConvexOS System Manager's Guide*, which the enclosed manual replaces. For a complete explanation of documentation changes for ConvexOS V8.0, please refer to the Revision History page of the 10th edition as well as the 10th edition, revision 1.

We are sorry for the inconvenience, but our policy continues to be one of quality documentation accompanying a quality product. In the rapidly changing world of technology, software often changes more rapidly than the printed word. As in the past, we urge you to suggest changes and enhancements to the *ConvexOS System Manager's Guide* and to report errors or areas of confusion so we can continue to provide the most useful documentation possible for our customers.

IMPORTANT: PLEASE READ FIRST.

Revision Information for
ConvexOS System Manager's Guide

Edition	Document No.	Description
10.1	710-001430-207	Replaces edition 10.0, Document Number 710-001430-206, which was released with ConvexOS and Utilities V8.0 in December 1989. Edition 10.1, released January 1990, reflects late changes to the documentation and supersedes all previous editions of this manual. Includes the following technical changes: • Chapter 2 — changes to setting up *sendmail*; updates to *uucp* examples • Chapter 3 — updated and enhanced swap partition examples • Chapter 5 — removed *sysgen* information obsoleted by ConvexOS V8.0 • Chapter 13 — updated *op.access* examples • Appendix A — made changes to the following sysfiles: — */usr/adm/acct*; new example — */usr/adm/badlogins*; removed — */etc/disktab*; new example — */etc/ttys*; new example
10.0	710-001430-206	Released with ConvexOS and Utilities V8.0; December 1989
9.1	710-001430-205	Released with ConvexOS and Utilities V7.1; March 1989
9.0	710-001430-204	Released with ConvexOS and Utilities V7.0; October 1988
8.0	710-001430-203	Released with ConvexOS and Utilities V6.2; April 1988
7.0	710-001430-202	Released with ConvexOS and Utilities V6.1.1; December 1987
6.0	710-001430-201	Released with ConvexOS and Utilities V6.1, November 1987
5.0	710-000030-000	Released with ConvexOS and Utilities V6.0; April 1987
4.0	710-000330-000	Released with ConvexOS and Utilities V4.0; August 1986
3.0	710-000330-000	Released with ConvexOS and Utilities V3.0; January 1986
2.0	710-000330-001	Released with ConvexOS and Utilities V2.0; July 1985
1.0	710-000330-000	Released with ConvexOS and Utilities V1.0; March 1985

Table of Contents

Appendices

List of Tables

List of Figures

Preface

Purpose and Audience

The *ConvexOS System Manager's Guide* contains information needed to manage and maintain an installed and configured CONVEX supercomputer. This guide describes the following:

- managing and maintaining a CONVEX supercomputer
- general information on daily operating procedures
- commands and utilities used to perform system-management tasks

This guide is not a complete source for system management information. Complete reference information for commands and utilities is in in the *ConvexOS Programmer's Reference.* Procedures for power-up, power-down, and maintenance are described in the *CONVEX Processor Operation Guide.*

This guide addresses users of CONVEX system who perform system management and operations and who have experience

- with the ConvexOS operating system
- as a system manager or operator of a timesharing computer system that supports an average of at least 10 users
- working with peripheral devices such as disk and tape drives, terminals, and line printers
- with timesharing and computer resource use
- with system and data security

Organization

This manual is organized as follows:

- Chapter 1, "Introduction," describes computing environments and responsibilities of the system manager.
- Chapter 2, "Initial System Setup," describes setting up the following:
  - peripheral devices
  - notesfile system
  - online man pages and their indexes
  - line printer system
  - *mail* system
  - *uucp*
  - system-wide login profiles

- Chapter 3, "Kernel Boot-Time Parameters," describes how to set up and customize system performance.
- Chapter 4, "Device Configuration," describes how to add new devices to the system configuration.
- Chapter 5, "Disk Tuning and Performance Management," describes how to set up and manage disk resources for optimal performance.
- Chapter 6, "Maintaining User Accounts," describes how to add, maintain, and delete user accounts.
- Chapter 7, "System Protection and Security," describes how to protect files, create and protect mail files, and secure the system.
- Chapter 8, "Mail System and Communications," describes how to use *mail*, *msgs*, and other communications utilities.
- Chapter 9, "Accounting," describes how to install and maintain the accounting system.
- Chapter 10, "Resource Monitoring and Scheduling," describes how to use utilities that monitor resource use and control operation of the CPU, disks, and tape drives.
- Chapter 11, "Performing Backups," describes how to back up file systems and design backup schedules.
- Chapter 12, "Crash Dumps," describes how to use the *crashdump* utility after a system goes down.
- Chapter 13, "Operator Interface," describes how to establish an interface for system operators who perform privileged operations on the system.
- Appendix A lists and describes system files requiring periodic maintenance.
- Appendix B is the technical paper "Fsck — The UNIX File System Check Program."
- Appendix C describes how to use the *contact* utility to report software or documentation problems.

Documentation Conventions

The following conventions are used in this document:

- Words enclosed in rounded rectangles indicate keyboard keys you press. For example, RETURN refers to the carriage return key. Words separated by a hyphen and enclosed in rounded rectangles indicate two keys you must press simultaneously. For example, CTRL-X indicates you must press the CTRL key while simultaneously pressing the keyboard CTRL-X key.
- The word "enter" in a phrase such as "enter a command" means you type the command and press the carriage return key. The word "type" (for example, "type a line of text") means you do not press the carriage return key.
- Within command sequences

Boldface	characters that must be typed exactly as they appear.
Italics	indicate file names or arguments for which you must substitute actual file names or arguments.
Brackets ([ ])	designate optional entries
Horizontal ellipsis (...)	shows repetition of the preceding item(s).

In the following example, COMMAND must be typed as it appears; *input_file* indicates file name that must be supplied by the user; the horizontal ellipsis in brackets indicates

that additional input file names, separated by commas, can be supplied; and *output_file* indicates an optional file name.

COMMAND *input_file* [, . . .] [*output_file*]

- References to the *ConvexOS Programmer's Reference* appear in the form *cat*(1), where the name of the manual page is followed by its section number enclosed in parentheses.
- `Constant-width font` is used for examples of computer-generated output and FORTRAN and C code.
- A vertical ellipsis shows continuation of a sequence where not all statements in an example are shown.
- The pound sign (#) is the ConvexOS superuser prompt. The percent sign (%) is the standard C shell user prompt. The dollar sign ($) is the standard Bourne shell prompt. Most examples in this guide use the C shell prompt.

Associated Documents

The following documents provide further information about using CONVEX supercomputers:

- *ConvexOS Primer* provides an introduction for users who have not previously used the ConvexOS operating system.
- *ConvexOS Programmer's Reference* is the standard reference for the ConvexOS operating system.
- *CONVEX Networking System Manager's Guide* describes installing, configuring, testing, and debugging the software used by CONVEX local area networks. This guide also describes how to change system files and troubleshoot problems in the network.
- *CONVEX NFS System Manager's Guide* describes installing and using the Network File System (NFS) product.
- *CONVEX CXbatch System Manager's Guide* describes configuring, using, and maintaining CXbatch, an optional product.
- *CONVEX Guide to Software Development* describes programming techniques and using ConvexOS software development tools.
- *ConvexOS Tutorial Papers* is a collection of previously published papers providing instruction in document preparation, programming, text editing, supporting tools and languages, system maintenance, and system implementation.
- *CONVEX Architecture Reference* describes the architecture and instruction set used by CONVEX computer systems.
- *CONVEX Diagnostic Utilities* contains documents describing the functional test diagnostic procedures for the central processor, main memory, peripheral devices, and the SPU self-test programs.
- *CONVEX Guide to Attaching Multibus Peripherals* describes the procedures for attaching peripherals to a Multibus.
- *CONVEX VME Service Kit* describes procedures for attaching and using VME peripherals.
- *CONVEX Guide to Writing Device Drivers* describes the CONVEX software that must be understood before you can write device drivers for the CONVEX supercomputer.
- *CONVEX SPU UNIX Utilities Manual* describes the SPU UNIX operating system and related utilities.
- *CONVEX HSP User's Guide* describes procedures for attaching and using the High-Speed Parallel (HSP) interface.

- *CONVEX Processor Operation Guide (C100 Series, C200 Series)* describes procedures for routine operation of CONVEX supercomputer hardware, including starting up and shutting down the system.
- *CONVEX System Generation Guide* describes procedures for generating a new ConvexOS operating-system image (required, for example, when adding user-written device drivers, creating non-standard swap partitions, installing ConvexOS patch code).

Ordering Documentation

To order CONVEX documentation, complete the CONVEX Documentation and Subscription Service Order Form enclosed in the Documentation Catalog included with this manual.

To receive a specific edition of a manual (other than the current edition), contact the local CONVEX sales office.

Technical Assistance Center

For hardware and software support, call the CONVEX Technical Assistance Center (TAC):

Within the continental United States	1-800-952-0379
From Alaska, Hawaii, and Canada	1-214-497-4379
From all other locations	contact nearest CONVEX office

Reader's Forum

To mail comments to the Documentation Department, please use the form at the end of this manual and list the document page number with your questions and comments.

Chapter 1
Introduction

This chapter outlines the system manager's responsibilities and describes the computing environments from which the system manager works.

Effective system management requires that the system manager or operator do the following:

- administer system files and processes to ensure system and data integrity
- optimize system resources to gain maximum performance
- maintain effective system operation and system security
- service user requests
- perform the following routine tasks:
  - boot and shut down the system
  - add and delete user accounts
  - repartition disks
  - allocate and audit disk space
  - configure devices
  - administer terminal (*tty*) ports
  - perform system dumps (incremental and full backups)
  - restore damaged or lost files
  - monitor system status
  - ensure file protection
  - maintain system logs

1.1 Computing Environments

The ConvexOS system manager performs tasks from within three computing environments: the ConvexOS operating system, the Service Processor Unit (SPU) UNIX operating system, and the *soft front panel*.

1.1.1 ConvexOS Operating System

ConvexOS is a demand-paged, virtual-memory operating system derived from Berkeley UNIX 4.2 BSD. The system consists of

- command interpreters (shells)
- utilities
- file system
- *kernel* that manages system resources, including interrupts, memory management, process control, and I/O

ConvexOS supports the following command-interpreter, or shell, environments:

- C
- Bourne
- COVUE

The C shell, *csh*, is the standard ConvexOS shell.

ConvexOS operating modes are

- single-user
- multi-user

1.1.1.1 Single-User Mode

From within ConvexOS single-user mode, the system manager runs maintenance software, performs file-system checks, and mounts and unmounts file systems. In ConvexOS single-user mode, multi-user daemons and multi-user logins are inactive.

1.1.1.2 Multi-User Mode

From within ConvexOS multi-user mode, the system manager performs periodic system checks, such as monitoring user file space, number of users, and amount of user processing time.

1.1.2 SPU UNIX

The Service Processor Unit (SPU) UNIX operating system is based on AT&T's Version 7 UNIX operating system. Unlike Berkeley UNIX 4.2 BSD, Version 7 is not a virtual-memory system.

From within SPU UNIX, the system manager runs machine diagnostics and checks error logs. The SPU UNIX operating system boots the CPU.

While in diagnostic mode, the SPU shell environment is called the diagnostic shell or *dshell*; normally the active shell is the Bourne shell (*sh*). See the *SPU UNIX Utilities Manual* and the *C120 and C130, C210, C220 Diagnostics Utilities Manuals* for a complete description of SPU UNIX functions and related utilities.

1.1.3 Soft Front Panel

The third computing environment is an interactive access utility called the *soft front panel*, a program stored in erasable programmable read-only memory (EPROM) on the SPU circuit board.

From within the soft front panel, the system manager modifies the settings of the firmware options used to configure the system. In this environment, the system manager can set the mode of operation that determines how the system boots. For more information on the soft front panel, see the *CONVEX Processor Operation Guide (C100 Series, C200 Series)*.

1.2 System Prompts

System prompts indicate which computing environment is currently running. Table 1-1 lists the default system prompts.

Table 1-1: CONVEX System Prompts

Operation State	Prompt
soft front panel	`(fp)>`
SPU UNIX	`(spu)>`
dshell	:
ConvexOS	
login	`login:`
single-user mode	#
multi-user mode	# (as root)
multi-user mode (csh)	% (as user)

1.3 Superuser (root) Privileges

Some system management tasks (e.g., modifying the */etc/passwd* file to add or delete user accounts) require superuser privileges. Superuser privileges are also called root privileges, because when you log in as a superuser, you log in as user name *root*.

1.4 Supporting the User Community

Establishing a good relationship with system users is an important part of system management. For example,

- Inform users of scheduled system downtimes for reconfiguration and preventive maintenance.
- Respond to user requests.
- Install software necessary for users to perform their jobs effectively.
- Respond to user complaints.
- Provide online or hardcopy *how to* training material and provide immediate feedback to user questions.

1.5 Common Problems

System managers often have difficulty quickly diagnosing bugs in system software. For help, consult this guide and any of the associated diagnostic and reference documents listed in the Preface.

Users often have difficulty interacting with text editors and programming tools as well as properly using system software. Two documents that help users solve their own editing and programming problems are the *Text Editor User's Guide* and the *ConvexOS Programmer's Manual* (described in the "Preface").

1.6 Topics Not Covered in the *System Manager's Guide*

Topics not covered in this guide include

- **Installation Procedures** — Installation procedures for optional software are covered in the software distribution package shipped with each optional software product. Installations procedures for software revisions are covered in the *Installation Procedures* shipped with each version of ConvexOS.

- **Capacity Planning** — Tailoring the CONVEX supercomputer to meet customer computing requirements includes planning for future growth or capacity planning. The Technical Assistance Center (TAC) assists with capacity planning because each system manager has a unique environment and set of applications that dictate capacity requirements.

1.7 Reference

For more information on the CONVEX computing environment and starting up and shutting down your CONVEX supercomputer, see the *CONVEX Processor Operation Guide (C100 Series, C200 Series).*

Chapter 2
Setting Up ConvexOS

This chapter describes setting up the system after it is installed, including

- prerequisites
- setting up notesfiles
- setting up manual pages
- setting up line printers
- setting up *mail*
- setting up UUCP, the UNIX-to-UNIX Communication Protocols used to copy files from system to system across telephone lines
- setting up system logging configuration

2.1 Prerequisite Information

This section describes how to modify the */dev* directory, how to choose a site name, how to modify site-specific */etc/rc.local* and */etc/hosts* files, how to build the *sendmail* database, and how to read the error-log file.

2.1.1 Setting Up Peripheral Devices

Devices supported by ConvexOS are implemented in the operating system kernel by device drivers that communicate between the CPU and a Channel Control Unit (CCU). These devices are made accessible by the *mknod* program and are normally in the */dev* directory.

You can add device parameters for

- tapes (*ta*)
- disks (*da, dd, du*)
- communications controllers (*ca*)
- pseudo-terminals (*pty*)
- printers (*pa*)
- striped file systems (*st*)

For example, to add three disk units, another tape unit, three more communications controllers, and one set of pseudo-teletypes for networking, use the following command sequence:

```
# cd /dev
# /dev/MAKEDEV da1 da2 da3 ta1 ca1 ca2 ca3 pty1
```

Ignore *file exists* warning messages when using */dev/MAKEDEV*.

2.1.2 Site Naming

Many programs that run on your system (for example; *mail, wall, who, getty*) require a host name and obtain that host name through system calls or from files located in the */etc* directory.

2.1.2.1 Site-Naming Conventions

The host name must conform to the following conventions:

- unique name; to verify the name you choose is unique, contact

  UUNET
  ATTN: Donnalyn Frey
  P.O. Box 2685
  Fairfax, VA 22031
  703-764-9789

- maximum of seven lowercase characters
- describe your location
- cannot contain hyphens, underscores, or other special characters

2.1.2.2 Adding Site Names to Files

After you select a host name, add it to the following files:

- */etc/rc.local*
- */etc/hosts*

Enter your host name as the first line in */etc/rc.local* using the following format (*my_hostname* is the name of your host):

/bin/hostname *my_hostname*

The value returned by the *gethostname* system call is defined by this line in */etc/rc.local/*. The *getty*, *mail*, *wall*, *uucp*, and *who* programs use this system call so the binary images are site-independent.

If you are not physically connected to another network, add your site name to the */etc/hosts* file, as follows:

```
127.1  my_hostname        localhost
```

The local host name used by *sendmail* for distributing network mail via *uucp* is defined by this line in */etc/hosts*.

If you are physically connected to another network, see the *CONVEX Networking Utilities System Manager's Guide* for information on adding the host name to these files.

After you change */etc/rc.local* and */etc/hosts*, rebuild the *sendmail* database, as follows:

1. Use one of the following methods to set the host name:
   - Reboot the system
   - As the superuser, enter

     # /bin/hostname *my_hostname*

     For *my_hostname*, use the host name in */etc/rc.local*.

2. Kill and restart *sendmail*. Enter

   # ps axc | grep sendmail
   # kill –9 < ***pid*** **>**
   # /usr/lib/sendmail –bz
   # /Usr/lib/sendmail –bd –q1h

2.1.3 Error Logging

The system displays a warning diagnostic on the console when errors occur on peripherals or in the system. These messages are collected on the SPU and written into the system error-log file, */mnt/errlog*, on the SPU disk.

Examine the error-log file regularly. Use the following command sequence, for example, to display the last 50 lines of */mnt/errlog*:

```
# /usr/convex/spu -r /mnt/errlog | tail -50
```

You do not need to be a superuser to run this command.

Contact the Technical Assistance Center if you find errors indicating hardware problems.

2.1.4 Setting Up Local Utilities

Each site typically establishes a set of site-specific utilities and manual pages for those utilities. When a new software release is installed on your system, many standard files and directories (for example; */bin* and the directories that store CONVEX-released manual pages) are overwritten. To ensure that local utilities are not overwritten, place local utilities in the */usr/local/bin* directory, and place locally-generated manual pages in the */usr/man/manl* directory. Place local libraries in the */usr/local/lib* directory.

2.2 Setting Up the Notesfile System

The notesfile system is described in Chapter 8 and in the "Notesfile Reference Manual" in the *ConvexOS Tutorial Papers*. To set up the notesfile system, enter the following sequence of commands:

1. Move to directory */usr/spool/notes/.utilities*.

   ```
   # cd /usr/spool/notes/.utilities
   ```

2. Substitute as user *notes*.

   ```
   # su notes
   ```

3. Create and open notesfile *nfmaint*.

   ```
   % mknf -o nfmaint
   ```

4. Create and open the networked notesfile *nfgripes*.

   ```
   % mknf -on nfgripes
   ```

2.3 Setting Up Online Manual Pages

The *man* command displays the contents of pages from the online *ConvexOS Programmer's Manual*. For example, for information on the *chmod* command, enter

```
% man chmod
```

2.3.1 Formatting Online Manual Pages

Source files for online manual pages are in the directory */usr/man*. This directory holds subdirectories called *manX*, where *X* is an integer from 1 through 8, corresponding to the section of the *ConvexOS Programmer's Manual* in which the manual pages are found. For example, the */usr/man/man1* directory holds the manual pages for section 1 commands (for example, *more*). If you create new manual pages for local utilities, create the directory */usr/man/manl* (where *l* stands for *local*) and place the local manual pages in that directory.

The manual pages in these directories are input files and are not formatted for viewing. The *catman* command creates the preformatted versions of the manual pages from the input files. *catman* also creates the */usr/lib/whatis* database (see the next section). To use *catman*, enter

```
# /etc/catman &
```

The *catman* utility is described in the *catman*(8) manual page.

2.3.2 Setting Up Indexes for Online Manual Pages

The *makewhatis* utility creates the following index files to the manual pages:

- */usr/lib/whatis*
- */usr/lib/man.index*

To create these index files, enter

```
# /usr/lib/makewhatis
```

These files index the manual page files in the */usr/man* subdirectories. After these files are created, use the *man* and *whatis* commands to view summary information from the manual pages.

- The *whatis* command displays the header of a manual page; it is equivalent to *man -f*. For example, to obtain information on the *more* command, enter

  ```
  % whatis more
  ```

  The following line appears on your screen:

  ```
  more, page (see more (1))       - file perusal filter for CRT viewing
  ```

  This line names the commands that give you access to the function called by *more* (*more* and *page*), tells you the manual page to view for further information (*more*(1)), and summarizes the function that *more* performs.

- The *-k* option of the *man* command allows you to specify a keyword and prints a single-line synopsis of each manual page whose table of contents lists that keyword. For example, if you enter

  ```
  % man -k alias
  ```

  the following lines appear on your screen:

  ```
  aliases (see aliases (5))          - aliases file for sendmail
  newaliases (see newaliases (1))    - rebuild the data base for the mail aliases file
  which (see which (1))              - locate a program file including aliases and paths
                                       (csh only)
  ```

Two or more functions can be described on one manual page (and are available on a single hardcopy manual page indexed under one or more possible entries). For example, *more*(1) and *page*(1) perform functions described on the manual page filed under *more*. The

/usr/lib/man.index file indexes these entries so they are available online by calling either *more* or *page*. For example, if you enter **man more** or **man page**, the *more*(1) manual page (which is also the *page*(1) manual page) displays.

2.4 Setting Up the Line Printer System

The line printer system can handle multiple printers, multiple spooling queues, local and remote printers, and printers attached by serial lines. The typical line printer system consists of the following files and commands:

/usr/ucb/lpq	spooling queue examination program
/usr/ucb/lprm	program to delete jobs from a queue
/usr/ucb/lpr	program to enter a job in a printer queue
/etc/printcap	printer configuration and capability database
/etc/lpc	line printer control program
/usr/spool	spooling queue directories line printer daemon that scans spooling queues; called in the */etc/rc* file as distributed

The */etc/printcap* file is a master database that describing line printers directly attached to a machine and printers attached across a network.

To configure line printers, see the *printcap*(5) manual page and the tutorial paper, *4.2BSD Line Printer Spooler Manual.*

2.5 Setting Up the Mail System

The mail system consists of the following files and commands:

/bin/mail	UNIX 32/V mail program; see the *binmail*(1) manual page
/usr/ucb/mail	Berkeley mail program, described in the *mail*(1) manual page
/usr/lib/sendmail	Mail routing program
/usr/lib/sendmail.cf	*sendmail* configuration file
/usr/lib/sendmail.fc	*sendmail* frozen configuration file
/usr/spool/mail	*mail* spooling directory
/usr/lib/aliases	*mail* forwarding information
/usr/ucb/newaliases	Command to rebuild binary forwarding database
/usr/ucb/biff	*mail* notification enabler
/usr/etc/in.comsat	*mail* notification daemon
/usr/etc/in.syslog	Error message logger used by *sendmail*

Instructions for installing the mail system are included in the documents The *mail* command is used to send and receive mail. *mail* provides a user interface, so messages sent or received can be edited. *mail* passes messages to *sendmail* for routing. *sendmail* then uses information in the */usr/lib/sendmail.cf* configuration file to process and forward each piece of mail.

After the mail is delivered, the local mail delivery daemon */usr/etc/in.comsat* is notified; *in.comsat* notifies users who have included a *biff y* command in their *.login* file that mail has arrived.

After installing the mail system, set up the file */usr/lib/aliases*, creating appropriate mail groups. When you create new mail aliases, execute the command */usr/ucb/newaliases* to update the *mail* database. For more information, see the *newaliases*(1) manual page. Directions for using the mail system are in Chapter 8.

2.6 Setting Up a UUCP Connection

This section describes the steps in UUCP installation. Before installing UUCP, read the paper "UUCP Implementation Description" in the *ConvexOS Tutorial Papers*. This paper describes UUCP file formats and conventions.

To connect two UNIX machines with a UUCP network link using modems, one site must have an automatic call unit (ACU) and the other must have a dialup port. Ideally, each site has both.

If you are setting up UUCP over Ethernet, read the *uucico* manual page first.

The *uucp* software is located in the following directories:

- */usr/bin*
- */usr/lib/uucp*
- */usr/spool/uucp*

/usr/bin contains the following user commands:

/usr/bin/uulook	tails syslog (*/usr/spool/uucp/SYSLOG*)
/usr/bin/uucp	File-copy command
/usr/bin/uux	Remote execution command
/usr/bin/uusend	Binary file transfer using *mail*
/usr/bin/uuencode	Binary file encoder (for *uusend*)
/usr/bin/uudecode	Binary file decoder (for *uusend*)
/usr/bin/uulog	Scans session log files
/usr/bin/uusnap	Gives a snap-shot of *uucp* activity
/usr/bin/uupoll	Polls remote system until an answer is received
/usr/bin/uuq	Examines and manipulates UUCP queue

/usr/lib/uucp contains the following operational commands and files:

/usr/lib/uucp/L-devices	List of dialers and hard-wired lines
/usr/lib/uucp/L-dialcodes	Dialcode abbreviations
/usr/lib/uucp/L.cmds	Commands remote sites may execute
/usr/lib/uucp/L.sys	Systems to communicate with, how to connect, and when
/usr/lib/uucp/SEQF	Sequence-numbering control file
/usr/lib/uucp/USERFILE	Remote-site pathname access specifications
/usr/lib/uucp/uuclean	Cleans up garbage files in spool area
/usr/lib/uucp/uucico	*uucp* protocol daemon
/usr/lib/uucp/uuxqt	*uucp* remote execution server

/usr/spool/uucp, a spooling area, includes the following files and directories:

/usr/spool/uucp/STST	Directory for status files
/usr/spool/uucp/LCK	Directory for lock files
/usr/spool/uucp/C.	Directory for command (*C.*) files
/usr/spool/uucp/D.	Directory for data (*D.*) files
/usr/spool/uucp/X.	Directory for command execution (*X.*) files

/usr/spool/uucp/D.machine	Directory for local *D.machine* files (replace *machine* with your site name)
/usr/spool/uucp/D.machineX	Directory for local *X.* files (replace *machine* with your site name)
/usr/spool/uucp/TM..	Directory for temporary (*TM.*) files
/usr/spool/uucp/LOGFILE	Log file of *uucp* activity
/usr/spool/uucp/SYSLOG	Log file of *uucp* file transfers

To install *uucp*, first check */etc/passwd* for a *uucp* entry. Use the *L.sys.example*, *L.cmds.example*, and *USERFILE.example* files as guidelines. Account names must be in format *XXuucp* and the *XX* must be uppercase letters.

If a *uucp* entry does not exist on your system, create a *uucp* account with *nu*. See Chapter 5, "Maintaining User Accounts," for information on using *nu*. Set the path of the home directory to */usr/spool/uucppublic* and the login shell to */usr/lib/uucp/uucico*. Then, enter the following commands to create *uucp* directories:

# cd /usr/lib/uucp	(move to directory */usr/lib/uucp*)
# UUCP_SETUP	(script to create files in */usr/spool/uucp*)

If you have an auto-call unit, create the *L.sys*, *L-dialcodes*, and *L-devices* files. Include in the *L.sys* file the phone numbers and login sequences required to establish a connection with a *uucp* daemon on another machine. Figure 2-1 is a sample *L.sys* file:

Figure 2–1: Sample *L.sys* File

```
sitew   Any    ACU    1200   0123456789 "" \r ogin: Cxuucp word: secret
sitex   Never  Slave   300   null
sitey   Never  Slave   300   null
sitez   Never  Slave  1200   null
```

This sample *L.sys* file has fields for

- site name
- time the machine can be called
- how the host is connected (for example; ACU or direct connection)
- modem speed
- phone number to use in connecting through an ACU
- login sequence

The phone number can contain common abbreviations, defined in the *L-dialcodes* file. Device specification should refer to devices specified in the *L-devices* file. Entering ACU here the *uucp* daemon, *uucico*, to search for any available ACU in *L-devices*. Figure 2-2 is a sample *L-dialcodes* file.

Figure 2–2: Sample *L-dialcodes* File

```
MCI 9%6518300
out 9%
```

Figure 2-3 is a sample *L-devices* file.

Figure 2–3: Sample *L-devices* File

```
ACU  cua0  cua0  1200  vadic
```

cua0 is the name of an entry in the */dev* directory created by *mv* for the terminal line connected to the autodialer. In the example, the autodialer is a VADIC modem operating at 1200 baud.

As *uucp* operates, it creates and removes small files in the directories beneath */usr/spool/uucp*. Sometimes files are left undeleted; use *uuclean* to purge them. The log files can also grow without bound unless periodically reviewed; use *uulog* to review them.

You can automate the use of *uuclean* and *uulog* by making the appropriate entries in *crontab*. See the *cron*(1) and *crontab*(5) manual pages for details.

Also, you must poll other hosts for data transmission requests. Polling should be done twice per day, and can be automated by running a script like the one shown in Figure 2-4 from *crontab*.

Figure 2–4: Script for *uucp* Polling

```
#!/bin/sh

# Poll these hosts periodically. Invoke this script from /usr/lib/crontab
# (usually twice per day).

# Change the names of the hosts listed below for your configuration.

for i in host1 host2 host3
do
        /usr/lib/uucp/uucico -r1 -f -s$i
done

/usr/lib/uucp/uucico -r1 -f
/usr/lib/uucp/uucico -r1 -f
```

See the *uucico*(8C) manual page for more information on polling.

If you are polling only one host, you can add the following line to *crontab*, instead of running the script in the previous figure:

[*min hour dayofmonth month dayofweek*] **/usr/convex/send** *hostname*

min	minute (0–59)
hour	hour (0–23)
dayofmonth	day of the month (1–31)
month	month (1–12)
dayofweek	day of the week (1–7, 1 = Monday
hostname	name of polled host

Indicate empty fields in the command line with an asterisk (*). In the following command line, the *dayofmonth* and *month* fields are empty:

30 06 * * 5 /usr/convex/send *hostname*

2.7 Setting Up System-Wide Login Profiles

Just as each user can set up a *.login* file to perform certain tasks at login time, the system manager can set up a system-wide file called */etc/login* (for C shell) or */etc/profile* (for Bourne shell) to perform tasks for all users when they log in. When users log in, the shells execute commands first from the system files, then from the user's individual files.

For the C shell, files are read in the following order:

1. *.cshrc*
2. */etc/login*
3. *.login*

For the Bourne shell, files are read in the following order:

1. */etc/profile*
2. *.profile*

The system-wide login/profile files eliminate users having to edit their *.login* files. If users want to define their own profiles, their *.login* files take precedence over the */etc/login* file.

Use the */etc/login* profile to

- Create aliases containing pathnames to programs that you want users to use. For example, if you install a new version of Emacs that you want users to test before deleting the old version, make *emacs* a command alias with the pathname of the new Emacs.
- Use a default account other than 0 for billing.
- Print load averages and other useful information at login.
- Set histories for all users.
- Create an alias *passwd* for *yppasswd* on *nfs* systems.

The system-wide logout profile, */etc/logout*, works the same way as the system-wide login file, except shells execute commands first from the user's individual *.logout* file, and then from the */etc/logout* file.

Use the */etc/logout* profile to print a message similar to the message-of-the-day at logout. For instance, to remind users of an event happening the following day, put the message in the */etc/logout* file. Use the *echo* command (for example, **echo** *my_message_here*) or *cat* to print a file containing the messages.

For more information on system-wide login files, see the *csh*(1), *sh*(1), and *login*(1) manual pages.

Chapter 3
Kernel Boot-Time Parameters

To customize your system, ConvexOS has kernel boot-time parameters that can be set when you boot the system. With these parameters, you can optimize the performance and behavior of your system without recompiling a system image. To set a kernel boot-time parameter value, place a command in a special file that processes at boot-time. The parameter value then becomes the default used each time you boot your system. Each kernel boot-time parameter is specific to a system processor (for example; CPU, IOP) and affects only that processor.

3.1 Files for Kernel Boot-Time Parameters

The boot procedure reads boot commands from the */mnt/os/bootcmd* file. This file contains information about how to boot your system, where the root partition resides, and commands to tune some parameters. After *bootcmd* is read, commands are read from the */mnt/os/bootcmd.local* file (if it exists). The commands in */mnt/os/bootcmd.local* take precedence over those in *bootcmd*, offering a means to customize the way your system boots.

The *bootcmd.local* file is not part of the ConvexOS release. You must create it to change the default values for a kernel boot-time parameter.

Note	Do *not* alter the */mnt/os/bootcmd* file. It is subject to change with each release of ConvexOS. Use */mnt/os/bootcmd.local* to set boot-time parameters for your site.

3.2 Setting Kernel Boot-Time Parameters

Kernel boot-time parameters are set using the *tune* command in the SPU file */mnt/os/bootcmd.local*. You can specify one parameter for each use of the *tune* command, with each specification on a separate line, in the following format:

tune *processor parameter* = *value*

processor — processor for which a parameter is specified (for example; **cpu**, **hsp**, **iop**, **viop**).

parameter — name of the parameter you are tuning

value — new parameter value; must be within minimum and maximum values specified for the parameter. Default, minimum, and maximum values are listed in the SPU file */mnt/os/tunables* and in section 3.5. Do *not* edit the file */mnt/os/tunables*.

An example of a line in the */mnt/os/bootcmd.local* file is:

```
tune cpu number_ptys = 32
```

To place comments in the file, start the line with a pound sign (#).

3.3 Adding Kernel Boot-Time Parameters to Your Device Drivers

If new device drivers are added, kernel boot-time parameter for those device drivers can also be added. To add a new parameter called, for example, *new_param*, you must define *new_param* in one of your device driver source files. Assuming a default value of 32 forfInew_param, the definition would be

```
int new_param = 32;
```

All boot-time parameters must be of type *int*. You must set a default value for the parameter. You must also declare the parameter in each source file referencing it, for example

```
extern int new_param;
```

Add a line to the SPU file */mnt/os/tunables* in the format

parameter_name default minimum maximum processor_name

parameter_name	name of the kernel boot-time parameter
default	default value used in the source file in which *variable_name* is defined
minimum	minimum allowable value for the parameter
maximum	maximum allowable value for the parameter
processor	name of the processor to which the parameter applies; valid processor names are *cpu*, *iop*, *hsp*, and *viop*

> **Note** Do *not* change the values listed in the */mnt/os/tunables* file.

3.4 Adding Swap Devices to *bootcmd.local*

To change swap partitions, add new swap devices in the */mnt/os/bootcmd.local* file. Use the *swap on* option in the following format:

swap on *partition*

For example, to change several swap partitions, enter

swap on dd3c, dd4c, dd5c

The *swap on* specifications in the */mnt/os/bootcmd.local* file override those specified in */mnt/os/bootcmd* (usually da0b or dd0b.) The first swap partition specified in */mnt/os/bootcmd.local* becomes the new default swap partition.

3.5 ConvexOS Boot-Time Parameters

The ConvexOS kernel boot-time parameters are listed in the following sections. These include parameters for the CPU and IOP. CONVEX does not support parameters for the VIOP or HSP.

3.5.1 CPU Boot-Time Parameters

auto_nice_factor (default=4, min=0, max=64)

Specifies the degree by which the scheduling priority of a process is reduced; which is the degree by which a process is *reniced.* Setting the value to 0 disables *renicing.* This parameter only affects processes that are not running with the user ID for superuser and that are running at the default priority. (See also **auto_nice_threshold.**)

auto_nice_threshold (default=600, min=1, max=9999999)

Specifies the amount of time (seconds of CPU time) before a process is automatically given a reduced scheduling priority (*niced*). This parameter only affects processes that are not running with the user ID for superuser and that are running at the default priority. (See also **auto_nice_factor** and *getpriority*(2).)

dst_algorithm (default=1, min=0, max=6)

Specifies the daylight-savings rule used for keeping time on your system; valid values are

- 0 = no daylight savings rule
- 1 = the United States
- 2 = Australian-style daylight savings
- 3 = Western Europe
- 4 = Middle Europe
- 5 = Eastern Europe
- 6 = Canada

See also **time_zone** and the manual page for the *date*(1) command for more information on the daylight-savings rule.

enable_unique_core (default=0, off=0, on=1)

Specifies whether each core dump file has a unique name (set to 1) or all core dump files are named *core.* Setting this parameter prevents one core file from being overwritten by another program that dumps core. Each core file will have a unique name in the form of *core.progname.12345,* where *progname* is the name of the program that dumped core, and *12345* is the process ID number of the program when it failed.

erase_pattern (default=0xAAAAAAAA, min=0x80000000, max=0x7FFFFFFF)

Specifies the pattern written over deleted files when the *erase_unlink* parameter is enabled. The default writes binary "1010..." over the deleted files. (See also **erase_unlink** and section 7.4, "Erasing Deleted Files.")

erase_unlink (default=0, off=0, on=1)

Specifies whether deleted files are erased on the disk. When enabled (set to 1), freed disk blocks are overwritten with the pattern specified by the **erase_pattern** parameter. When disabled, erasing does not occur. See also **erase_pattern** and section 7.4, "Erasing Deleted Files."

fp_default_mode_ieee (default=0, off=0 , on=1)

Specifies the system-wide default floating-point mode for programs when no compiler option for floating-point mode is specified. This parameter has no effect on floating-point operations within the kernel. This parameter can be overridden by using the compiler command-line option *-f(format).* The value set with this parameter is conveyed to system utilities and user programs through the SI_IEEE_DEFAULT bit of the options word returned by *getsysinfo.*

If **fp_default_mode_ieee** is set to 1, programs use IEEE floating-point mode if the underlying hardware and software is capable of using it. If the hardware cannot support IEEE floating-point mode, and **fp_default_mode_ieee** is set to 1, the system prints the following error message at

boot time:

```
IEEE floating-point is not available on this system.
Default floating-point mode will be NATIVE.
```

You can override this specification with the *-fn* option at compile time.

If **fp_default_mode_ieee** is set to 0, programs use native floating-point mode. You can override this specification with the *-fi* option at compile time.

gateway (default=0, off=0, on=1)

Enables sending Internet Control Message Protocol (ICMP) errors if both of the following conditions are true:

- The system has a single network interface, or IP forwarding is disabled (**ipforwarding** parameter)
- The received IP packet is not for the system that has **gateway** enabled

If **gateway** is disabled (set to 0) under these circumstances, errors are not sent to the source machine and the packet is dropped. (See also **ipforwarding**, **ipsendredirects**, and **subnetsarelocal**.)

ipforwarding (default=1, off=0, on=1)

Enables Internet protocol (IP) packets to be forwarded. Packets are forwarded when the IP address does not correspond to any of the Internet addresses for the machine's network interfaces. If this parameter is disabled (set to 0), packets can be dropped. (See also **ipsendredirects**, **gateway**, and **subnetsarelocal**.)

ipsendredirects (default=1, off=0, on =1)

Enables sending Internet Control Message Protocol (ICMP) redirects. Redirects inform the sending machine of the direct address of the machine to which packets were forwarded. If this option is disabled (set to 0), redirects are not sent when packets are forwarded. (See also **ipforwarding**, **gateway**, and **subnetsarelocal**.)

log_resume (default=4, min=0, max=100)

Specifies when to resume logging unsuccessful file access. When the percent of free space on the file system of failure log file is above the specified percentage, logging is resumed. When you have created sufficient free space to allow logging to resume, the message "File access logging resumed" prints on the console. (See also **log_suspend**, and *Logging Failed File Access* in Chapter 7.)

log_suspend (default=2, min=0, max=100)

Specifies when to suspend logging unsuccessful file access. When the percentage of free space on the failure log file system drops below the specified percentage, logging is suspended, and the message "File access logging suspended" is written to the system console. You do not enable file logging with this parameter; you enable it with the command **/etc/faillogon**. (See also **log_resume**, and *Logging Failed File Access* in Chapter 7.)

maxusers (default=32, min=8, max=256)

Specifies the maximum number of users on the system at one time. This parameter does not directly restrict the number of users on the system. It is used at boot time to size system tables (for example; process, *inode*, or open file table). If the value is too low, not enough users can be on the system at a time. If the value is too high, memory is wasted. A safe value is the number of tty lines used in your system. If you have a lot of pseudo-teletypes, increase the number. See also **number_ptys** and **number_tty_controllers**.

max_user_processes (default=40, min=4, max=150)

Specifies the maximum number of processes allowed for each user, including batch jobs. Users running as superuser are not included. Tuning this parameter can help control the load average of the system. A user who tries to start a process beyond the maximum allowed receive a warning message.

nfs_portmon (default=0, off=0, on=1)

Enables Network File System (NFS) server-port checking. The default (0) does not require NFS to check the Internet domain source port of the client machine to see if it is a privileged port. To fully enable NFS server-port checking, you must add the *-p* option for that variable in *rpc.mountd*. See the *CONVEX NFS System Manager's Guide* for details.

nstbuf (default=512, min=128, max=8192)

Specifies the number of stripe buffers used for stripe devices.

number_ptys (default=64, min=0, max=256)

Specifies the maximum number of pseudo-teletype devices. Pseudo-teletypes are used by *emacs*, *script*, and some networking programs. See Chapter 4 for details.

number_ta_iop_wndw (default=8, min=8, max=34) Specifies the number of Multibus "windows" the Multibus tape driver will allocate. IOPs contains 256 windows and the tape driver allocates its windows for direct memory transfers to and from main memory. Increasing this parameter reduces the frequency of tape overruns.

number_tty_controllers (default=2, min=0, max=site dependent)

Specifies the maximum number of ACM-001 or ACM-002 tty controllers. Each controller multiplexes 16 tty lines, so a system with 2 controllers can have up to 32 tty lines connected to the system. This parameter must not exceed the number of controllers licensed to you in your ConvexOS licensing agreement.

stripe_devices (default=16, min=4, max=256)

Specifies the number of supported striped file systems. The default value (16) is suitable for most sites. The value of this parameter should not be significantly higher than the number of striped file systems you are using or intend to use in the near future; setting it to an unnecessarily high value wastes kernel memory. See *st*(4) and Chapter 5.

subnetsarelocal (default=1, off=0, on=1)

Considers an Internet address local even if it belongs to another subnet. A different network number means the address is remote. This option is used only during determination of the TCP maximum segment size. TCP uses a small maximum segment size (536 bytes) for remote destinations. For local destinations, TCP uses the maximum transmission unit of the outgoing network interface. See **ipforwarding**, **ipsendredirects**, and **gateway**.

ta_force_EOF_on_close (default=1, off=0, on=1)

Controls the writing of end-of-file (EOF) marks when a magnetic tape unit is closed. When a tape unit is closed, the tape driver sometimes writes end-of-tape marks on the tape at the current file position. For historical reasons, the decision about whether to write tape marks depends in part on the access type (read/write) options specified when the tape unit was opened.

If the unit was opened for read and write access, tape marks are only written if the last I/O operation to the tape was a write. If the unit was opened for write-only access, tape marks are always written when the tape unit is closed.

Turning off **ta_force_EOF_on_close** tells the tape driver to write an EOF mark at close time only if the user's last tape I/O operation was a write.

time_zone (default=360, min=-720, max=720)

Specifies the number of minutes east (negative values) or west (positive values) of Greenwich Mean Time (GMT). See also **dst_algorithm** and the *date*(1) manual page.

tty_pty_size (default=2, min=1, max=12)

Specifies the size of pseudo-terminals (*pty*) tty structures in half-page intervals. *tty* structures are allocated in main memory. The number of *pty tty* structures allocated is defined by the tunable parameter **number_ptys**. Increasing this number can increase the performance of *ptys*.

udpcksum (default=0, off=0, on=1)

Enables checksumming of User Datagram Protocol (UDP) datagrams. UDP checksumming incurs substantial overhead because each transmitted and received UDP datagram is checksummed. Turning the parameter off increases the risk of allowing bad packets farther up in the protocols.

3.5.2 IOP Boot-Time Parameters

accelerate_enable (default=1, off=0, on=1)

Specifies whether to use a hardware enhancement that accelerates data throughput. To use this enhancement, you must have Revision E or later IOP hardware. If you enable this parameter (the default) and do not have the required hardware, the request to use accelerate mode is ignored.

ca_timer_code (default=8, min=0, max=15)

Controls the frequency with which the ACM-001 or ACM-002 terminal controllers interrupt the IOP. The optimal value for this parameter is site-dependent and is affected by

- controller model
- number of active ports
- baud rate(s)
- other peripherals
- general system load

The default setting is sufficient for most system loads. In rare cases, high-speed tty input can cause data-overrun errors. If this happens, increase the frequency of IOP interrupts to minimize overruns. **A bad value for this parameter can severely degrade system performance.**

Contact a TAC representative for more information before tuning this parameter.

tty_iop_size (default=1, min=1, max=12)

Specifies the size of IOP *tty* structures in half-page intervals. *tty* structures are mapped into main memory through Multibus "windows." Each window can map one page of main memory; windows are allocated to map 16 *tty* structures for each Multibus *tty* controller. IOPs contain 256 windows. Increasing this number can reduce the frequency of *tty* overruns.

tty_viop_size (default=2, min=2, max=12)

Specifies the size of VIOP *tty* structures in half-page intervals. *tty* structures are mapped into main memory through VMEbus "windows." Each window can map one page of main memory; windows are allocated to map 16 *tty* structures for each VMEbus *tty* controller. VIOPs contain 1024 windows. Increasing this number can reduce the frequency of *tty* overruns. This number should be a tuned to a multiple of two.

Chapter 4
Device Configuration

This chapter describes how to reconfigure the ConvexOS operating system to add new physical devices. If you are adding new disk drives, see Chapter 5, "Disk Tuning and Performance Management," for information on configuring your disk system for optimal performance.

If you are adding a device that is not supported by CONVEX, refer to the *CONVEX Guide to Writing Device Drivers*. The guide is part of the optional User–Written Device Drivers package and includes instructions on writing and installing device drivers and generating a new operating system.

Adding CONVEX–supported devices is primarily a hardware task. Refer to the documentation for the hardware you are adding, as well as the *CONVEX Guide to Attaching Multibus Peripherals*, the *CONVEX VME Service Kit* if you are adding VME peripherals, and the *CONVEX HSP User's Guide* if you are adding an HSP.

After you physically attach the hardware, use the following procedure to reconfigure the software for new supported devices:

1. Add the new device information to the */ioconfig* file.
2. Run diagnostics, if available.
3. Boot the system.
4. Use */dev/MAKEDEV* to create special files for the new devices.

This chapter describes editing the */ioconfig* file and creating special files for new devices. The boot procedure is described in the *CONVEX Processor Operation Guide*.

4.1 Editing the */ioconfig* File

When you add a device to your system, you must modify operating system files to integrate the new device into ConvexOS. The */ioconfig* file lists the Channel Control Units (CCUs), interfaces, controller boards, and peripheral devices for your system. The boot program reads */ioconfig* to determine the physical connections for your system. The ordinal sequence of controllers listed in */ioconfig* determines the special files (in the */dev* directory) to which the peripherals are mapped.

To edit the */ioconfig* file for your system, first boot to the SPU prompt **(spu)**. See the *CONVEX Processor Operation Guide* for booting information and the *xed*(1) manual page in the *CONVEX SPU UNIX Programmer's Manual* for information on using the *xed* editor.

To edit the */ioconfig* file, enter

(spu) **xed /ioconfig**

Figure 4-1 is a sample */ioconfig* file (line numbers are included for reference).

Figure 4–1: Example */ioconfig* File

```
1     iop 6
2       mbus 0
3         ctlr LAN-001 csr 0x4c0 int 1
4           unit 0 type ex
5         ctlr ACM-001 csr 0x3c0 int 6
6           unit 0 type TTY
7           unit 1 type TTY
  .                   .
  .                   .
  .                   .
21          unit 15 type TTY
22      mbus 1
23        ctlr COV-002 csr 0x4c0 int 1
24          unit 0 type COV-002
25        ctlr ACM-001 csr 0x3c0 int 7
26          unit 0 type TTY
27          unit 1 type TTY
  .                   .
  .                   .
  .                   .
41          unit 15 type TTY
42    hsp 3
43       drvr HEC-001 csr 0x0000
44         chnl 0 type HED-001
45       drvr HEC-001 csr 0x0000
46         chnl 1 type HED-001
  .                   .
  .                   .
  .                   .
59       drvr HEC-001 csr 0x0000
60         chnl 15 type HED-001
61    viop 5
62      vme 0
63        ctlr DKC-204 csr 0x800 int 3
64          unit 0 type DKD-208
65          unit 1 type DKD-208
66        ctlr DKC-203 csr 0xa00 int 4
67          unit 0 type DKD-214
68      vme 1
69        ctlr DKC-204 csr 0x800 int 3
70          unit 0 type DKD-206
71        ctlr DKC-203 csr 0xa00 int 4
72          unit 0 type DKD-214
  .                   .
  .                   .
  .                   .
80    idc 2
81      ipi 0
82        drvr DKC-IP2
83          unit 0 type DKD-502 master
84      ipi 1
85        drvr DKC-IP2
86          unit 0 type DKD-502
```

The */ioconfig* file contains four levels of information for IOP, VIOP, and IDC-type CCUs and three levels of information for HSP-type CCUs. These levels are usually distinguished by indentation level, although indentation is not required.

To add a device to the */ioconfig* file, include the following information:

1. CCU description

   Beginning at the left margin, the CCU description includes the CCU type and slot number. Supported CCU types include *iop*, *viop*, *hsp*, and *idc*. Slot numbers range from 0–7, corresponding to the I/O slot used and depending on the CONVEX processor series. The number of CCUs supported by your system also depends upon the CONVEX processor series:

   C100 series models support 5 CCUs in slots 3–7.

   C200 series models C210 and C220, and model C240 with 1 PBUS, support 4 CCUs in slots 0–3.

   C240 series models with 2 PBUSes support 8 CCUs in slots 0–7.

   Lines 1, 42, 61, and 80 of Figure 4-1 are CCU descriptions.

2. Bus or Interface description

   Information at the first level of indentation includes the type and chassis number of the bus or interface. Supported types are *mbus* (Multibus), *vme* (VMEbus), and *ipi* (Intelligent Peripheral Interface), for IOP, VIOP, and IDC-type CCUs, respectively. (HSP-type CCUs do not have controller chassis.) Numbering begins with zero and is sequential for all buses or interfaces on a CCU.

   IOP-type CCUs support 2 Multibuses, VIOP-type CCUs support 2 VMEbuses, and IDC-type CCUs support 4 IPIs.

   Lines 2, 22, 62, 68, 81, and 84 of Figure 4-1 are Multibus, VMEbus, or IPI descriptions.

3. I/O controller or driver description

   At the second level of indentation, the I/O controller description specifies the controller type or user-supplied driver type. You specify a *csr* address for Multibus, VMEbus, and HSP controllers and an interrupt level for Multibus or VMEbus controllers. (HSP drivers do not have an associated interrupt level. IDC-type drivers do not have an associated *csr* address or interrupt level.)

   For IOP- and VIOP-type CCUs, the format of the I/O controller description is as follows:

   ctlr *type* **csr** *address* **int** *level*

   To add user-written drivers for HSP-type CCUs, use the following format:

   drvr *type* **csr** *address*

   Use the following format for adding IDC-type driver descriptions:

   drvr *type*

   Table 4-1 lists the controller types and their descriptions. The *csr* address and interrupt levels for a controller are provided in the specific controller documentation and in the *CONVEX Guide to Attaching Multibus Peripherals*. All controllers on a bus must have unique addresses and interrupt levels (0–7).

   Lines 3, 5, 23, 25, 43, 45, 59, 63, 66, 69, 71, 82, and 85 of Figure 3–1 are I/O controller or driver descriptions.

4. I/O unit or channel description

   At the third level of indentation, the I/O unit or channel description specifies I/O unit numbers and type names. For IOP-, VIOP-, and IDC-type CCUs, the I/O unit description is in the format

   unit *number* **type** *name* [**master**]

   The keyword **master** applies only to IDC-type CCUs. Use of this keyword is discussed in Chapter 5, "Disk Tuning and Performance Management."

   HSP-type CCUs have associated channels rather than units. The format for specifying HSP channels is:

   chnl *number* **type** *name*

   Unit numbers begin at zero and increase sequentially. The number of units supported by a single controller depends upon the controller type. You can find this information in the documentation for the specific controller. Unit types and their descriptions are listed in Table 4-2.

   Lines 4, 6, 7, 21, 24, 26, 27, 41, 44, 46, 60, 64, 65, 67, 70, 72, 83, and 86 are I/O unit or channel descriptions.

Table 4-1: ConvexOS Controller Types

Controller Type	Description/Syntax
ACM–001, 002	Asynchronous Multibus controller
ACM–201	Asynchronous VMEbus controller
DKC–001, 002	SMD Multibus disk controller
DKC–203	VME ESDI disk controller
DKC–204	VME SMD disk controller
DKC–IP2	IDC IPI-2 disk controller
GPI–001	DR11–W Emulator
HEC–001	HSP logical controller (predefined as echo test driver)
. . .	. . .
HEC–005	User–supplied HSP device controller type 5
LAN–001	Multibus Excelan Ethernet controller
LAN–002	HYPERchannel interface controller
LAN–004	HYPERchannel network interface
LAN–007	VME Excelan Ethernet controller
LAN–202	UltraNet VMEbus controller
MTC–001	CONVEX Multibus half–inch magnetic tape controller
MTC–201	CONVEX VME half–inch magnetic tape controller
PRC–001	Systech MLP–2000 line printer controller
SCI–001	COM-4 communications controller
UDD–001	User device driver debug routines
UDD–002	User device driver debug routines
USC–001	User–supplied controller type 1
. . .	. . .
USC–009	User–supplied controller type 9
VER–001	IKON Versatec plotter controller with differential interface
VER–002	IKON Versatec plotter controller with TTL interface

Table 4-2: ConvexOS Device Unit Specifications

Unit Type	Description
TTY	random terminal type
COV–002	COVUEnet pseudo–unit type
COV–003	COVUEnet pseudo–unit type
DKD–001	Fujitsu M2351A (*eagle*) disk for Multibus
DKD–002	CDC 9766 (*washtub*) disk for Multibus
DKD–005	NEC 500 Mbyte disk for Multibus (with mounting bracket)
DKD–006	NEC 500 Mbyte disk for Multibus
DKD–008	NEC 1 Gbyte disk for Multibus
DKD–206	NEC 500 Mbyte disk for VMEbus
DKD–208	NEC 1 Gbyte disk for VMEbus
DKD–214	335 Mbyte VME Removable Disk Subsystem
DKD–284	Hitachi model 515–78 780 Mbyte
DKD–501	Imprimis Sabre model 2HP–1150 1 Gbyte, 2 head parallel
DKD–502	Imprimis Sabre model 1150 1 Gbyte, single head
GPD–001	Raster Technologies model 180 graphics display unit
HYP–001	HYPERchannel A400 device driver
NET–002	HYPERchannel A400 network interface
MTD–001	STC model 2920 50ips tape drive for Multibus
MTD–002	STC model 1960 125ips tape drive for Multibus
MTD–003	Fujitsu 200ips tape drive for Multibus
MTD–004	STC model 2922 50/100ips tape drive for Multibus
MTD–201	STC model 2922 50ips tape drive for VMEbus
MTD–202	STC model 1960 125ips tape drive for VMEbus
MTD–203	Fujitsu 200ips tape drive for VMEbus
MTD–204	STC model 2922 50/100ips tape drive for VMEbus
PRT–001	Printronix (or compatible) line printer
PLT–001	Versatec plotter, differential interface
PLT–002	Versatec plotter, TTL interface
USD–001	User–specified device type 1 of possible 9 (USD–009)
USD–009	User–specified device type 9
HED–001	HSP device type 1 of possible 5 (HED–005)

4.2 Creating Device Files Using */dev/MAKEDEV*

ConvexOS requires special files (created by the */dev/MAKEDEV* script) in the */dev* directory to make physical devices look like normal files. CONVEX ships files for a standard system; you do not have to execute *MAKEDEV*. However, executing *MAKEDEV* is not harmful even if the files exist.

To create special files for a device, execute MAKEDEV using the appropriate code for the device as shown in Table 4-3.

Table 4-3: Device Types and Codes

DeviceType	Code
Standard devices	std
Line printers	pa
DR11–W Interface	dm
Plotters	pb
Tape drives	ta
Labeled tapes	lt
HYPERchannel	hy
Terminal controllers	ca
Multibus disk drive	da
VMEbus disk drive	dd
IDC disk drive	du
Pseudo terminals	pty
Disk stripes	st
COVUE	covue
Local	*

* local to site (for user–written device drivers)

The syntax for MAKEDEV is

MAKEDEV *code*

where *code* is the listed code for the device. To execute MAKEDEV, you must be in the */dev* directory and logged in as the superuser.

For example, to add a second terminal controller (the first one is numbered *ca0*), enter

```
# cd /dev
# MAKEDEV ca1
```

MAKEDEV creates the appropriate files in the */dev* directory.

4.2.1 Configuring Teletypes

After you add controllers, configure additional terminal lines as described in this section. ConvexOS supports up to 256 teletype lines, depending on licensing agreements.

Note	The order of controllers listed in the */ioconfig* file determines the numbering of controller names and the set of */dev/tty* files used by that controller.

ConvexOS uses *tty[0–9,A–F][0–f]* for terminal names, as shown in Table 4-4. The lines on the first *ca* controller are named */dev/tty00, /dev/tty01, ..., /dev/tty0f.* If a configuration has more than one *ca* interface, successive 16–line terminal groups are named as shown in Table 4-4. The names *tty[G–Z][0–f]* are reserved by ConvexOS.

Table 4-4: Terminal–Naming Conventions

Controller	Terminals
ca0	tty00 through tty0f
ca1	tty10 through tty1f
. . .	. . .
ca9	tty90 through tty9f
ca10	ttyA0 through ttyAf
. . .	. . .
ca15	ttyF0 through ttyFf

To configure terminal lines, edit and add entries in the */etc/ttys* file. A portion of the */etc/ttys* file shipped with ConvexOS is shown in Figure 4-2.

Figure 4–2: Example */ttys* File

```
console   "/etc/getty std.9600"  vt100n  on  secure
tty00     "/etc/getty std.9600"  vt100n  on  secure
tty01     "/etc/getty std.9600"  vt100n  on  secure
tty02     "/etc/getty std.9600"  vt100n  on  secure
tty03     "/etc/getty std.9600"  vt100n  on  secure
tty04     "/etc/getty std.9600"  vt100n  on  secure
tty05     "/etc/getty std.9600"  vt100n  on  secure
tty06     "/etc/getty std.9600"  vt100n  on  secure
tty07     "/etc/getty std.9600"  vt100n  on  secure
tty08     "/etc/getty std.9600"  vt100n  on  secure
tty09     "/etc/getty std.9600"  vt100n  on  secure
tty0a     "/etc/getty std.9600"  vt100n  on  secure
tty0b     "/etc/getty std.9600"  vt100n  on  secure
tty0c     "/etc/getty std.9600"  vt100n  on  secure
tty0d     "/etc/getty std.9600"  vt100n  on  secure
tty0e     "/etc/getty std.9600"  vt100n  on  secure
tty0f     "/etc/getty std.9600"  vt100n  on  secure
tty10     "/etc/getty std.9600"  vt100n  off  secure
tty11     "/etc/getty std.9600"  vt100n  off  secure
  .
  .
  .
tty1f     "/etc/getty std.9600"  vt100n  off  secure
tty20     "/etc/getty std.9600"  vt100n  off  secure
  .
  .
  .
tty2f     "/etc/getty std.9600"  vt100n  off  secure
```

The */etc/ttys* format is as follows:

name command type [**on** | **off**] [**secure**] [**dialup**] [**uucp**] [**window**=*cmnd*] [*access*]

name	terminal name; listed in */dev*
command	command to execute; usually *getty*
type	terminal type; listed in *termcap*
on/off	specifies whether *init* executes *command*
secure	used with *on*, allows root to log in
dialup	specifies whether users are asked for a dialup password when logging in
uucp	allows user ID *uucp* to log in
cmnd	command(s) *init* executes before starting *getty*
access	access control list; specifies users allowed to log in

You can selectively allow or disallow logins by users or groups by specifying an access control list. Users or groups whose names appear in the list are allowed access; preceding a name with "!" denies access to the user or group. Group names are placed between "<" and ">" symbols. Entries in the access control list are read from left to right; be careful of the order in which you list users or groups. If you explicitly give a user or group access, all users and groups must be specified to have access. If there is only a denial list, a user or group can log in as long as that user or group is not one of those explicitly denied access.

Edit */etc/ttys* to reflect your terminal configuration and add entries if your site has more ttys than are included in the file as shipped with ConvexOS.

The *terminal type* definitions are in */etc/gettytab*, which is read by *getty*. To make custom terminal types, consult the *gettytab*(5) manual page before modifying the file.

After editing */etc/ttys*, enter the following command to make the changes take effect:

```
# kill -1 1
```

To disable logins on a terminal line during normal operations, use *off*. To enable logins, use *on*.

Wire dialin lines so carrier is asserted only when the phone line is dialed up. For normal terminals from which modem control is not available, wire back the DTR signal so the carrier always appears present. To be sure, check the modem or terminal documentation.

init tries to reopen the terminal once a minute, printing a warning message on the console at 10 minute intervals, if one of the following is true:

- file (for example, */dev/ttyXY*) not accessible when *init* tries to create a process for it
- device controller missing
- device controller broken
- device controller not listed in *ioconfig*

For *accounting* and *ps*, the last two characters of the *tty* names must be distinct. The naming convention for a dialin line is

tty[*d*][*0–f*]

Dialin lines require dialin passwords. To create the dialin password, add an entry to */etc/passwd* for a user named *dialin*. See Chapter 6, "Maintaining User Accounts," for information on how to set up the *dialin* user and password.

4.2.2 Configuring Pseudo–Terminals

A pseudo–terminal is a pair of character devices implemented as a device driver. The devices are linked in a master/slave relationship. That means everything written on the master device is input to the slave device and everything written on the slave device is input to the master device. Logins are disabled on pseudo–terminals in */etc/ttys*. See the *pty*(4) manual page for details.

Networking programs, the *emacs* editor, and *script* use pseudo–terminals. ConvexOS supports a maximum of 256 pseudo–terminal pairs, with a default of 64. The pseudo–terminal pairs are a master and a slave. *MAKEDEV* creates the pseudo–terminals in pairs. When you edit */etc/ttys*, add only the slave, designated by *ttyp*. The master, *ptyp*, is created when you add an entry for the slave.

The naming convention for pseudo–terminals is

/dev/[*pt*|**tty**[*e–t*][*0–f*]

If the pseudo–terminal files are not on your system, use the following commands to create the first 16 pairs:

```
# cd /dev
# MAKEDEV pty0
```

To create additional groups of 16, use MAKEDEV with *ptyN* as an argument, *N* being a number 0 through 15. Add an entry to */etc/ttys* for each *pty* pair created.

Do not add the *pty* half of the pseudo–terminal pair (*/dev/ptyN*) to */etc/ttys*.

4.2.3 Configuring Line Printers

Each line printer installed with your system must have a device entry. Line printer device entries begin with */dev/lp0*, numbered in increments of 8. For example, the device entry for the second line printer installed is */dev/lp8*. Use the following command sequence to add a device entry for a line printer:

```
# cd /dev
# MAKEDEV pax
```

Check the */etc/printcap* file for an *lp* device entry. *lp0* has a default entry, but you must add add entries for other *lp* devices.

4.2.4 Configuring Tape Drives

For each tape drive installed on your system must have 12 device entries in */dev*. The following entries correspond to tape drive 0 (zero):

- *mt0*
- *mt4*
- *mt8*
- *mt12*
- *mt16*
- *mt20*
- *rmt0*
- *rmt4*
- *rmt8*
- *rmt12*
- *rmt16*
- *rmt20*

To create these entries, enter

```
# cd /dev
# MAKEDEV ta0
```

Chapter 5
Disk Tuning and Performance Management

This chapter describes setting up and managing disk resources for optimum performance, organized as follows:

- Physical resources
- Performance tradeoffs and issues
- Tunable parameters and disk striping
- Case studies
- Disk partitioning procedures

Setting up your disk system means allocating file systems to the available disks. A file system is a collection of files allocated to a portion of the disk. Common ConvexOS file–system names are

- / (root)
- *usr* (system programs and supporting information needed by users)
- *tmp* (temporary files)
- *etc* (system commands)
- *bin* (system commands)
- *dev* (device files)
- *mnt* (user directories and files)

5.1 Physical Resources

Physical resources of your system (for example, the amount of memory and the number of available disks and controllers) affect the performance of the disk system.

5.1.1 Physical Memory and the Disk Cache

Physical memory size has an indirect effect on system performance because of the disk cache. The disk cache is an area of physical memory allocated to buffering data from disk. In general, the larger the disk cache, the better your disk performance. To enlarge the disk cache, increase the total physical memory.

Adding physical memory enlarges the disk cache without taking away memory for user programs. Especially for systems processing large data files, a larger disk cache can greatly increase performance.

5.1.2 Disks and Controllers

The number of disks and disk controllers affects performance as follows:

- Adding resources increases performance.
- More disks and controllers allows more flexibility in distributing disk–system load; that is, disk–system resources are more efficiently used because the processing load is balanced

5.1.2.1 Disk–Naming Conventions

By ocnvention, disk names specify the type of controller to which the disk is attached and the number of the disk. When used in configuring your disk system, disk names often include a disk partition. Possible values for these disk–name components are

1. Controller and device type (*da* and *rda*, *dd* and *rdd*, *du* and *rdu*)
2. Disk number (0–63)
3. Disk partition (a–h)

If a disk is attached to a Multibus controller, the disk name begins with *da* or *rda*. If a disk is attached to a VMEbus controller, the disk name begins with *dd* or *rdd*. If attached to an IDC, the disk name begins with *du* or *rdu*.

For each controller type, the disk name can begin with an *r* (for example, *rda*, *rdd*, *rdu*), but the *r* prefix is not required. Names starting with *r* refer to raw (character) devices, which support unbuffered interfaces for each disk. Names without the *r* prefix refer to block devices, which support buffered interfaces for each disk. The block and raw devices correspond to files in the */dev* directory.

Disks are numbered sequentially from 0 to 63, starting with the first disk listed in the */ioconfig* file. The */ioconfig* file lists the physical devices for your system; the first disk listed in the file is numbered 0. ConvexOS supports a maximum of 32 devices for disk *dd*, 32 for disk *da*, and 32 for *du* without a *sysgen*. To add 64 devices on *dd*, you must use *sysgen*.

The first Multibus disk listed is named *da0*; the next listed is *da1*. The first VMEbus disk listed is named *dd0*; the second is named *dd1*. The first IDC disk is named *du0*; the second is named *du1*.

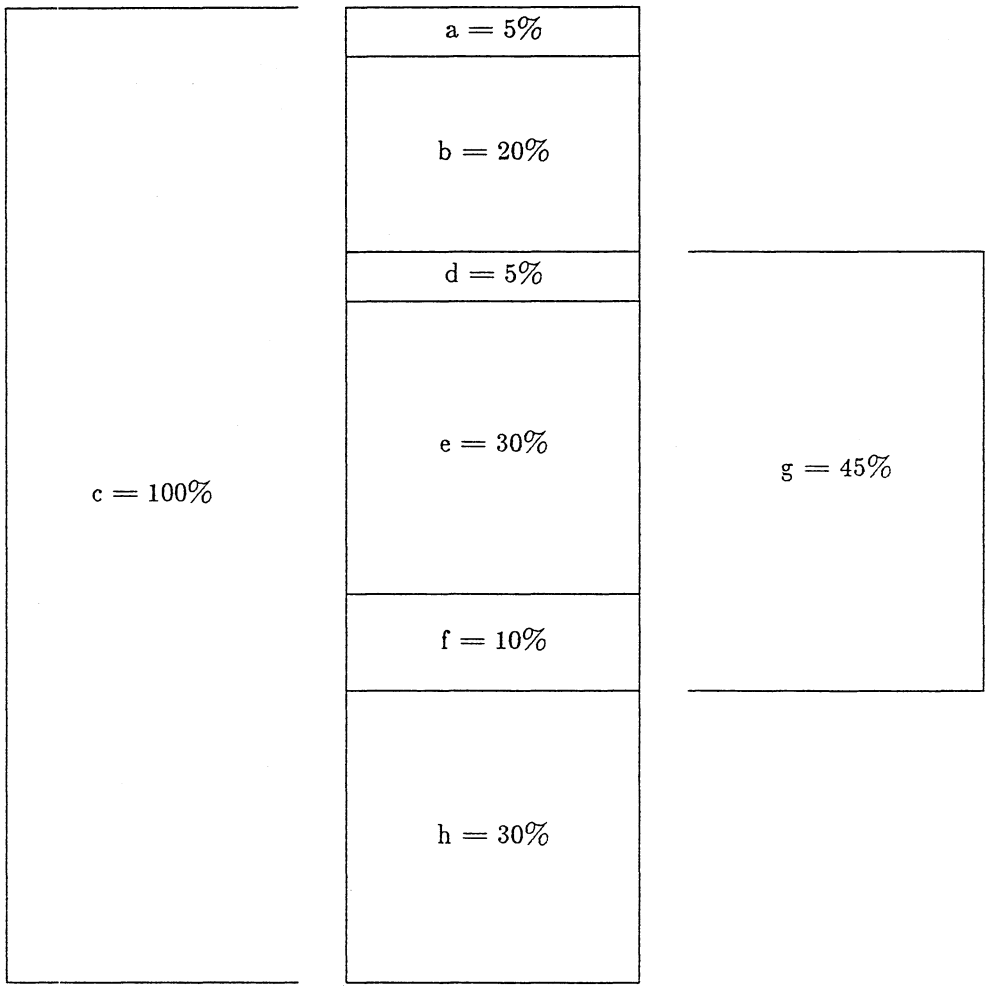
CONVEX disks can be used as a unit (one partition) or divided into four or six partitions. Assign file systems to the specific partitions based on

- ConvexOS conventions
- how much space the file system needs
- performance considerations

The partitions *a*, *b*, *c*, *d*, *e*, *f*, *g*, and *h* are each preassigned percentages (from 5 to 100) of the total formatted disk space (see Figure 5–1). Some partitions overlap other partitions. For example, if you use partition *c*, you cannot use another partition; partition *c* takes 100 of the disk space. If you use partition *g*, you cannot use partitions *d*, *e*, and *f*.

Before assigning a file system to a disk partition, be sure the chosen partition has enough space for the file system. For example, if you assign *mnt* to the *g* partition of a 400 Mbyte disk, the maximum disk space available for *mnt* is 180 Mbytes (45 of 400).

Figure 5–1: Disk–Allocation Percentages per Partition



5.1.2.2 Disk Use Conventions

The *a* partition of the first disk is always the default / (root) file system, which includes the */etc*, */bin*, and */dev* directories.

The *b* partition on the first disk in any multi-disk configuration is, by default, a swap partition. On other disks, the other partitions can also be allocated as swap space.

The *c* partition represents the whole disk.

Disk partitions *d*, *e*, and *f* are allocated only when the *g* partition is not used, because they overlag the *g* partition.

5.2 Performance Tradeoffs and Issues

Achieving good system performance involves compromise; changing one variable affects others. Faster speed can decrease space efficiency. Allocating more resources to a file system affects other file systems. Creating large disk stripes to increase space and speed also increases the risk of data loss.

5.2.1 Speed Versus Space Efficiency

When setting up your disk system, determine how much data can be transferred at once as well as the minimum space used for a single file. When you increase transfer speed, space efficiency is sacrificed. Using disk space more efficiently means slower transfers.

If your users primarily have many small files, limit the minimum space used for a single file to limit wasted disk space. If your users primarily use large files, maximize the amount of data that can be transferred at once to achieve better performance.

5.2.2 Load Balancing

Load balancing means distributing file-system use across available disks and disk controllers. A balanced load is critical in multi-disk configurations when the system is busy. If all disk activity at one time needs the resources of the same disk, throughput is limited to the bandwidth of the disk. When you evenly distribute the most-used file systems across disks, disk performance improves, often doubling throughput.

CONVEX provides two ways to improve load balancing:

- add disks and controllers
- stripe disks so file systems are distributed across multiple disks

Adding disks and controllers increases flexibility and makes load balancing easier. Ideally, systems that configure several Multibuses and several IOPs have each physical disk on a separate Multibus, a separate controller, and preferably on a different IOP. This is also true for systems that configure multiple VMEbuses and corresponding VIOPs.

CONVEX also provides disk striping for splitting a file system across disks. ConvexOS performs I/O more efficiently when striped partitions span disk drives that each have a controller. Combining partitions across different disk drives using the same controller can cause a bottleneck at the controller level. Combining two or more partitions on one disk degrades performance because the disk controller must seek between partitions on the same disk instead of successively reading from alternate disks.

5.2.3 Risk of Data Loss

Disk striping increases the risk of data loss during a hardware failure.

When a striped file system spans several disk drives, the failure of one of the drives renders the striped file system unusable until the hardware failure is corrected.

5.2.4 System Use

Depending on your system configuration and use, the issues of speed, space efficiency, load balance, and data loss can range from unimportant to extremely important. The system user needs determine the best disk configuration.

For example, if you have as many disks and controllers as you want, space efficiency is much less of a concern than speed. If you have a regular program of making backup tape dumps, the minimal data-loss risk may not be very important. With a one-disk system, load balancing is not applicable. With larger systems, load balancing can increase performance.

5.3 Tunable Parameters

With ConvexOS, you can tune parameters to enhance the disk system according to your needs. The following tunable parameters, and recommendations for tuning disk-system variables, are described in this section:

- file-system fragment and block size
- inode count
- swap space
- disk stripes

5.3.1 Fragment and Block Size

ConvexOS file systems are divided into fragments and blocks, which are contiguous segments of data. When creating file systems, you can specify fragment and block size. Depending on the average size of files in a file system, you can adjust the fragment and block size to increase performance.

At least one fragment is allocated for every file of nonzero length. Fragment size specifies the minimum disk space used by a file; a small fragment size prevents wasted disk space.

Block size specifies the maximum amount of data transferred in an operation. Files in large blocks take fewer disk operations and transfer faster, so a large block size yields better performance.

Theoretically a small fragment size and a large block size yield the best performance and greatest space efficiency. A small fragment size, however, mandates a relatively small block size, and a large block size mandates a relatively large fragment size.

The maximum ratio of fragment size to block size is 1:8. Block size can be 4, 8, 16, 32, or 64 Kbytes. Fragment size must be either the same as the block size or 1/8, 1/4, or 1/2 of the block size. Table 5-1 lists possible fragment sizes for the available block sizes.

Table 5-1: Possible Block and Fragment Sizes (in Kbytes)

If block size is	Fragment size can be
4	1/2, 1, 2, or 4
8	1, 2, 4, or 8
16	2, 4, 8, or 16
32	4, 8, 16, or 32
64	8, 16, 32, or 64

The block and fragment sizes you choose for a file system depend primarily on the size of files in the file system. There are typically four major file systems to distribute among available disks:

- / (root)
- */tmp*
- */usr*
- */mnt*

The / (root) file system includes the */etc*, */bin*, and */dev* directories. The */usr* file system includes system programs and other supporting information needed by users. The */mnt* includes user directories and files.

The */tmp* directory is shipped as part of the / (root) file system. The */tmp* directory can become huge, so CONVEX recommends placing it on a separate partition. Putting */tmp* on its own partition lessens the danger of *fsck* errors on / (root) after a crash. In a configuration with several disks, mount */tmp* in partition *a* of the second disk. For performance reasons, do not make */tmp* a symbolic link to another directory or partition. System programs (compilers, editors, the assembler, etc.) create intermediate files in the */tmp* directory; make the file system containing */tmp* large enough to accommodate the temporary increase in file size.

If */tmp* is not on a separate partition, clean out */tmp* nightly or weekly using a *crontab* routine. This routine can automatically delete files created before a specified time. See the *find*(1) and *crontab*(5) manual pages for more information.

The following examples show how disk space requirements for various file system are affected by varying the block and fragment sizes. Systems with limited disk space can select the next smaller block and fragment size in the following tables to preserve as much disk space as possible. The larger the fragment size, the more disk space wasted.

Example 1

/mnt is a file system containing a mix of user files, mostly program source files and executables. Although it means an 8% loss of space, the recommended block size is 16 Kbytes; fragment size 2 Kbytes.

Block Size	Fragment Size	Disk Space Used: */mnt*
4 Kbytes	512 bytes	245 Mbytes
8 Kbytes	1 Kbytes	251 Mbytes
16 Kbytes	2 Kbytes	264 Mbytes*
32 Kbytes	4 Kbytes	294 Mbytes
64 Kbytes	8 Kbytes	362 Mbytes

* recommended size

Example 2

/work1 is a file system consisting almost entirely of small source and data files. Most files are less than 1 Kbyte, so each increase in fragment size results in a large increase in file-system size. The recommended block size of 8 Kbytes and fragment size of 1 Kbyte result in an 11% loss of space, but higher throughput.

Block Size	Fragment Size	Disk Space Used: *work1*
4Kbytes	512	66 Mbytes
8 Kbytes	1 Kbytes	73 Mbytes*
16 Kbytes	2 Kbytes	87 Mbytes
32 Kbytes	4 Kbytes	118 Mbytes
64 Kbytes	8 Kbytes	185 Mbytes

* recommended size

Example 3

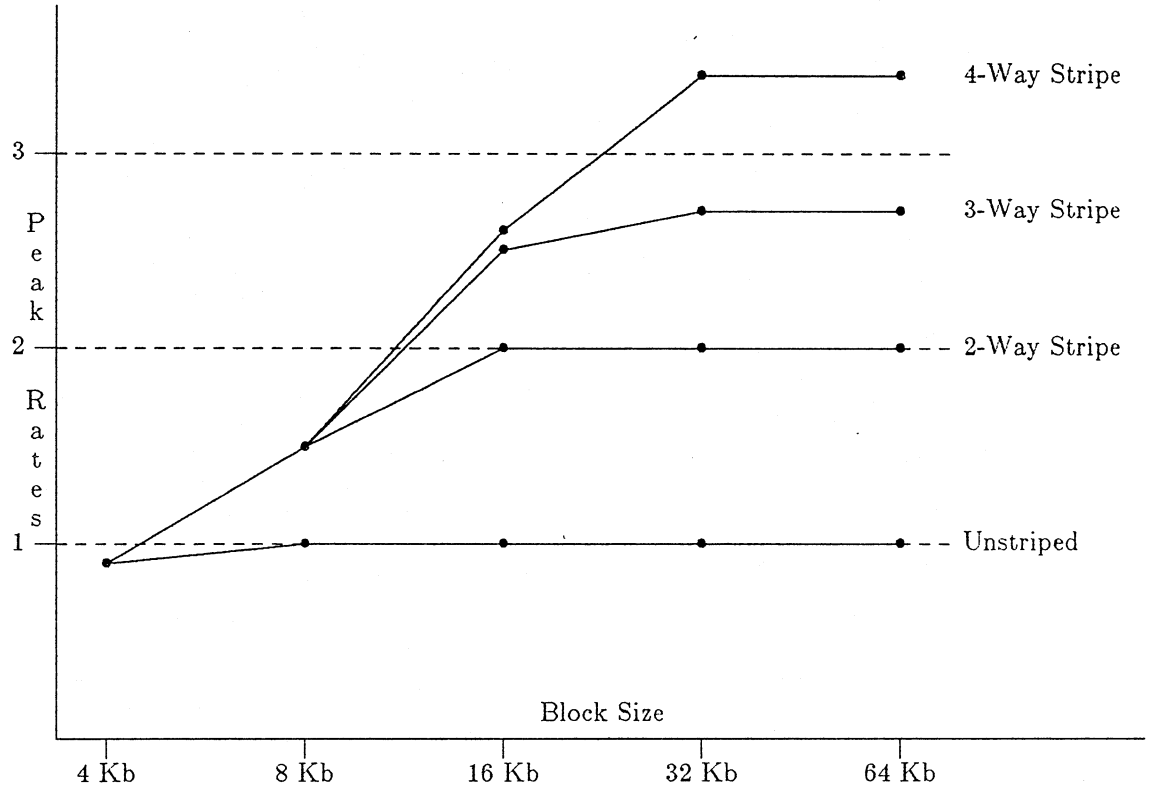
/work2 is a file system consisting primarily of data files greater than 1 Mbyte. Using a block size of 64 Kbytes and a fragment size of 8 Kbytes wastes only 5% disk space and results in higher performance.

Block Size	Fragment Size	Disk Space Used: *work2*
4 Kbytes	512	590 Mbytes
8 Kbytes	1 Kbytes	592 Mbytes
16 Kbytes	2 Kbytes	596 Mbytes
32 Kbytes	4 Kbytes	603 Mbytes
64 Kbytes	8 Kbytes	622 Mbytes*

* recommended size

Figure 5–2 shows the performance tradeoffs of block size and throughput as well as the benefits of disk striping. The performance measurements represent peak rates for large (greater than 1 Mbyte) file transfers for DKD–206 disks (1/2–gigabyte). The performance increases are not obtained with small transfers because of the lack of data to be transferred.

Figure 5–2: Throughput Versus File–System Size



File–system block size affects the CPU time required to read and write data for a file in the buffer cache. Table 5–2 shows the time required for a 1MB read and write using a 64K user buffer at five block sizes (4K through 64K). The pattern characterizes user buffers of other sizes.

Table 5-2: Performance Rates for Buffer Cache Read/Write

Block Size	CPU Time (seconds)			
	C120		C210	
	Write	Read	Write	Read
4K	.86	.47	.23	.16
8K	.47	.30	.12	.08
16K	.21	.11	.07	.04
32K	.16	.07	.05	.03
64K	.11	.05	.04	.02

Write operations take almost twice as long as read operations because disk blocks must be allocated during a write.

5.3.2 *Inode* Count

The structure of ConvexOS file systems requires one *inode* (index node) for each file. Each *inode* uses 128 bytes on the disk; *inode* space must be allocated when the file system is created. A file system can run out of *inode* space if the system parameters are not properly tuned and too many small files are created.

You must have enough *inodes* for all files in a file system; do not waste space by having more than necessary. When you create file systems, *newst* and *newfs* attempt to optimize the *inode* number, but frequently generate too few or too many *inodes* for the intended use of a file system. After creating and mounting a new file system, but before placing files in it, check the *inode* count by entering

```
# df -i
```

Inodes should range from a minimum of one *inode* per 16 Kbytes of storage to a maximum of one *inode* per 4 Kbytes of storage.

The *-I* option of the *newst* command requests that a specified number of *inodes* be created. Specify a multiple of 2048; otherwise, roundoff errors may result.

5.3.3 Swap Space

ConvexOS uses swap space to move data from main memory to disk until the process using the data becomes active again. Swapping can occur on any partition set up in */etc/bootcmd.local*. Swapping space is automatically striped; if more disks are used in configuring swap space, transfers from memory to swap space and back occur more rapidly.

Caution	Do not run *mkfs*, *newfs*, or *newst* across swap partitions.

When configuring swap space, consider the following:

- The minimum amount of disk to be used as swap space depends on the size of the largest set of processes running on a specific system.
- The sum of available real memory plus swap space determines the size of the largest set of processes that can run on your system.
- System memory requires about 3 Mbytes plus 10 percent of physical memory; the rest is available for user programs.
- For configuration, 75% of available memory plus the amount of swap space must be greater than the sum of all process virtual sizes. To run with reasonable throughput, however, most processes should fit in real memory.

Note	Partition *da0b* is the default swap device. Contact the TAC before allocating it to another use.

1. Choose the partitions, making sure that none are in use in use, either individually or as part of a disk stripe. You cannot share partitions between swap and file–system space.

2. Add the new swap entries to */mnt/os/bootcmd.local*.

3. Add the new swap entries to the *fstab* file, for example

   /dev/da1g swap_space swap xx 0 0

4. Boot in single–user mode to test the new kernel. If it succeeds, boot in multi–user mode. If you get errors from *swapon* (for example, */dev/da1g invalid device*), repeat the *sysgen* procedures.

See the *swapon*(8) manual page for more information on adding non–standard swap partitions.

5.3.4 Disk Stripes

A striped file system is a logical disk partition interleaved over several physical disk partitions. In multiple–disk systems, disk stripes allow flexibility in allocating file systems, as well as increasing disk–system performance. For example, in a four–disk system, you can combine two or more partitions across four disks to create a single striped file system. Data in striped file systems is read and written concurrently on all disk drives configured in the striped file system, so I/O processing is spread across all available resources.

Striped disk partitions also allow logical creation of larger file systems. You can create one striped partition holding more data than one physical disk drive can hold. By combining two or more physical disk partitions, you can create a striped disk partition the same size as the sum of the sizes of the conventional disk partitions. The combined size of all partitions in a single file system must be less than two gigabytes.

Disk striping is not a universal solution for disk–performance problems. Bad disk–striping decisions can make disk performance worse. When planning disk stripes, follow these guidelines for best performance:

- Stripes should span two or more disks. If you stripe two or more partitions on the same disk, the disk arm alternates from one partition to another on the same disk between each read. A long seek operation is required on each successive read, instead of infrequently, seriously impairing performance.

- In configurations with both Multibus and VME disks, put the most active file systems on VME disks.

- Partitions should span multiple controllers. If two or more partitions of a disk stripe are on the same disk controller, operations are performed in sequence (instead of overlapping), negating the benefits of disk striping. If you stripe three disk partitions on one disk controller and a fourth on a second disk controller, the best throughput is about one and one/half times better than an unstriped file system.

- Partitions should span Multibuses. If one stripe is on three or more controllers using the same Multibus, the bandwidth of the Multibus limits performance. When three disk controllers try to make simultaneous transfers, there is insufficient bandwidth on a single Multibus, so one or more must delay its transfer. The observed practical throughput for a Multibus connected to an IOP is limited to about 2 Mbytes per second. For the VMEbus, the observed practical limit is greater than 5 Mbytes per

second.

- Stripe together the same size partitions, if possible, for better performance.
- Never stripe the / (root) file system. The root file system must be on the a partition of the first disk.
- Because swap space is pre–allocated in the kernel, never include swap partitions in a disk stripe.

5.4 Case Studies

This section includes three case studies. First is a typical configuration for a single–disk system. Second and third are four–disk striped configurations, both Multibus and VMEbus.

5.4.1 Single–Disk Layout

Table 5–3 shows the layout for a conventional single–disk system.

Table 5-3: Conventional File-System Configuration

Disk	Space	File System	Block Size (Kbytes)	Fragment Size (Kbytes)
da0a	5%	/ (root)	8	1
da0b	20%	/swap	not applicable	not applicable
da0d	5%	/tmp	16	2
da0e	30%	/usr	16	2
da0f	10%	/usr/spool	16	2
da0h	30%	/mnt	8	1

The / (root) file–system default block size is 8 Kbytes with a 1 Kbyte fragment size. Here, it is a compromise between optimizing bandwidth to the disk and retaining enough disk space for files that must be stored in the / (root) file system. Using an 8 Kbyte block size instead of a 4 Kbyte block size reaps substantial performance gains. The 1 Kbyte fragment size is the smallest that can be used with an 8 Kbyte block size.

Because the */tmp* file system stores temporary files and is used by many system utilities, including compilers and editors, a block size of 16 Kbytes is needed.

The */usr* file system consists mostly of executable files, so a larger block size of 16 Kbytes assists total system throughput. The fragment size is set to the minimum 2 Kbytes to minimize wasted disk space for the smaller files such as those in */usr/include*.

/usr/spool stores temporary files for some system utilities, including *mail*, *cron*, and the print utility. The recommended block size and fragment size is 16 Kbytes and 2 Kbytes, the same as for */tmp*.

/mnt consists of user programs and data. With a single–disk system, you want to increase available disk space. The relatively small block size (8 Kbytes) allows a fragment size of 1 Kbyte.

5.4.2 Multibus Disk Striping

This case study system includes four disks, two Multibuses, and a separate controller for each disk. This larger configuration provides more options in creating file systems.

Tables 5–4 and 5–5 show an overview layout of this four–disk system and a more detailed view of the file–system disk stripes and block and fragment sizes.

Table 5-4: Striped Multibus Layout

Partition	da0	da1	da2	da3
a	/	st0	st0	st0
b	swap	swap	swap	swap
c	–	–	–	–
def	–	–	–	–
g	st3	st3	st3	st3
h	st1	st2	st1	st2

Table 5-5: Striped Multibus Layout, File Systems

File System	Disk Space (%)	Block/Fragment Size (Kbytes)	Stripe	Disk Partitions
/ (root)	5	8 / 1	not applicable	da0a
/swap	80	not applicable	not applicable	da0b, da1b, da2b, da3b
/tmp	15	16 / 2	st0	da1a, da2a, da3a
/usr	60	16 / 2	st1	da0h, da2h
/app	180	64 / 8	st3	da0g, da1g, da2g, da3g
/mnt	60	16 / 2	st2	da1h, da3h

The */tmp*, *usr*, and */mnt* stripes all use a 16–Kbyte block size with a 2–Kbyte fragment size. This combination is a good compromise between high disk throughput and minimum waste of disk space. Sites with application needs for high disk throughput can use a larger block size (and larger fragment size); those with high disk space utilization can use a smaller fragment size (and smaller block size).

The *app* stripe (st3) is for local applications that write files larger than 64 Kbytes, so there is no reason not to have maximum block size for this file system. Also, the total size of st3 is greater than 512 Mbytes, requiring a minimum fragment size of 4 Kbytes.

The following *newst* commands created the previously described disk stripes:

```
# newst –b 16k –f 2k /dev/rst0 /dev/rda1a DKD–005 /dev/rda2a DKD–005
  /dev/rda3a DKD–005
# newst –b 16k –f 2k /dev/rst1 /dev/rda0h DKD–005 /dev/rda2h DKD–005
# newst –b 16k –f 2k /dev/rst2 /dev/rda1h DKD–005 /dev/rda3h DKD–005
# newst –b 64k –f 8k /dev/rst3 /dev/rda0g DKD–005 /dev/rda1g DKD–005
  /dev/rda2g DKD–005 /dev/rda3g DKD–005
```

5.4.3 VMEbus Disk Striping

The issues for VMEbus disk striping are almost identical to Multibus disk striping. The major difference is that peak read rate is 40% faster on VMEbus disks than on Multibus disks, due to a read–ahead cache in the VMEbus disk controllers. In configurations with both VMEbus and Multibus disks, putting the most active file systems on VMEbus disks also improves performance.

By convention, the VMEbus disks are named *dd* instead of *da*. The following *newst* commands create equivalent disk stripes on a VMEbus disk:

```
# newst -b 16k -f 2k /dev/rst0 /dev/rdd1a DKD-214 /dev/rdd2a DKD-214
  /dev/rdd3a DKD-214
# newst -b 16k -f 2k /dev/rst1 /dev/rdd0h DKD-214 /dev/rdd2h DKD-214
# newst -b 16k -f 2k /dev/rst2 /dev/rdd1h DKD-214 /dev/rdd3h DKD-214
# newst -b 64k -f 8k /dev/rst3 /dev/rdd0g DKD-214 /dev/rdd1g DKD-214
  /dev/rdd2g DKD-214 /dev/rdd3g DKD-214
```

5.5 Disk–Partitioning Procedures

This section describes the commands for partitioning the disk and provides a step–by–step procedure for configuring your disks.

5.5.1 Getting Started

Before beginning the disk–partitioning procedure, make a diagram of how the disks are to be striped and explicitly designate the partition organization. You need the following information:

- Which file systems go where (Multibus or VMEbus, controller, disk, partition)?
- Which partitions are striped? Which file system is allocated to each disk stripe?
- What is the optimal fragment and block size? What is the optimal *inode* count?
- How much swap space do you need? Can it fit on partition *b*?

Use the following steps to determine answers for these questions:

1. Study the existing disk partitioning.

   Use the *df* (disk free) command to check current free disk space, block sizes, and *inode* count. The ConvexOS I/O system reserves 10 percent of the file–system disk space free to prevent gross fragmentation of the files; that is, the sum of the used kilobytes and available kilobytes reported by *df* does not add up to the amount displayed). Enter

   ```
   # df
   ```

   Used without an argument, *df* reports free space for all mounted file systems. Given a specific mounted file system (for example, *df /dev/st1*), *df* reports free space for that specific file system. With the *-i* option, *df* reports free *inode* space and the number of *inode*s used. To estimate the unusable disk space for a given fragment size, multiply the fragment size by the number of *inode*s used. If you figure the unusable space given several fragment sizes, you can better estimate the ideal fragment size.

   Look at the *fstab* file to be sure selected partitions do not conflict with existing partitions (unless you want to destroy the data on the old partitions).

2. Run */etc/getst* to see if there are existing stripes. Enter

   ```
   # /etc/getst st0
   ```

   to get information on the first stripe. If there are no stripes, you get the message

```
/etc/getst: Can't open stripe device /dev/rst0
```

If your system has multiple disk stripes, you can list the stripes as arguments to *getst*. For example, if your system has two stripes, enter

 # /etc/getst st0 st1

The command example output is similar to that shown in Figure 5-3.

Figure 5–3: Sample *getst* Output

```
/dev/rst0:
   Partition 1:  /dev/rda5a
   Partition 2:  /dev/rda3a
   Section 1:  Starts with logical block 0.  (st_start)
         Uses space on partitions 1 through 2.  (st_ndev)
         Uses 42496 blocks on each partition.  (st_size)
         Uses stripe blocking factor of 16 blocks.  (st_block)
   Total size: 84992 blocks  (st_total_size)
/dev/rst1:
   Partition 1:  /dev/rda5g
   Partition 2:  /dev/rda3g
   Section 1:  Starts with logical block 0.  (st_start)
         Uses space on partitions 1 through 2.  (st_ndev)
         Uses 383360 blocks on each partition.  (st_size)
         Uses stripe blocking factor of 16 blocks.  (st_block)
   Total size: 766720 blocks  (st_total_size)
```

3. Check disk activity.

   Run *syspic* while the system is running a normal load to check load balance. The *-p disk* option displays a picture of the disk statistics instead of the default system statistics. Enter

   # syspic –p disk

   If the middle column in a disk window (*da0–da15*, *da16–da32*, or *st0–st15*) is greater than 500 (ms), that disk is heavily loaded; investigate improving the load balance.

Caution	Repartitioning a disk is similar to a disk restore; it destroys all data on the disk. Make backup tapes of your disks before beginning.

5.5.2 Disk–Partitioning Commands

Several commands are executed to effect partitioning for a conventional or striped file system. Table 5–6 lists the commands, their function, and their manual page.

Table 5-6: Disk-Partitioning Commands

Command	Function	For More Information
/dev/MAKEDEV	Creates corresponding special file in the */dev* directory for every striped device configured; for example, */dev/st1*.	*mknod*(2), *mknod*(8)
/etc/newfs	Creates conventional file systems.	*newfs*(8)
/etc/newst	Creates striped file systems.	*mkfs*(8), *newst*(8), *st*(4), *stripecap*(5), *putst*(8)
/etc/preen	Checks disk integrity.	*fsck*(8), *preen*(8)

5.5.3 Disk-Partitioning Procedure

1. Boot the system.
   Consult the *CONVEX Processor Operation Guide* for detailed booting procedures.

2. Log in as the superuser.
   Superuser permission is required to repartition disks.

3. Shut down to single–user mode.
   Repartition the disk in single–user mode to ensure user–data integrity. To get from multi–user mode to single–user mode, enter

   # /etc/shutdown

4. Back up the disk.

   You can lose disk files through operator error. Executing *newst* or *newfs* on the file systems destroys all data previously stored on the partitions. CONVEX recommends backing up all your disk files with level 0 dumps. To back up */dev/da0a*, enter

   # dump /dev/da0a

   For more information, see Chapter 11, "Performing Backups."

5. *umount* designated file systems.
   Unmount the file systems to be striped. Change to the / (root) directory before attempting to unmount file systems. Use the *-a* option to unmount all file systems.

   # cd /
   # /etc/umount –a

6. Modify */etc/fstab*.
The *fstab* file contains file–system tables that specify disk partitions to be mounted onto file systems. *fsck*, *mount*, *umount*, and *dump* use the *fstab* file. Before editing, make a backup copy of the original *fstab* file. Edit the *fstab* file so the system recognizes the new striped devices.

fstab file entries have the following format:

device_name dir type opts freq passno

device_name — disk partition

dir — name of the directory the file system is to be mounted on

type — type of file system; valid types are

- *4.2bsd* (Berkeley 4.2 UNIX)
- *nfs* (Network File System)
- *swap* (swap partition)
- *ignore* (tells *mount* to not mount this file system)

opts — options:

- *rw* (read/write)
- *ro* (read only)
- *suid* (set uid execution allowed)
- *nosuid* (set uid execution not allowed)
- *quota* (usage limits enforced)
- *noquota* (usage limits not enforced)

freq — frequency for dumping file system to tape; values are

- *0* = no dumps
- *1* = daily dumps (recommended frequency)
- *2* = dumps every two days
- etc.

passno — not used, but must be present. CONVEX recommends the following values:

- *0* = swap space
- *1* = root file system
- *2* = all other file systems

Compare the *fstab* files in Table 5–7. One column includes disk stripes and the other does not. It is helpful to designate striped partitions in the *fstab* file, even though it is not required.

Table 5-7: Sample *fstab* Files

Striped	Not Striped
/dev/da0a / 4.2 rw 1 1 /dev/da0b swap ignore xx 0 0 /dev/da0g (/st_1of2) ignore xx 0 0 /dev/da0h /usr 4.2 rw 1 2 /dev/da1a /tmp 4.2 rw 1 2 /dev/da1b swap swap rw 00 /dev/da1g (/st_2of2) ignore xx 0 0 /dev/da1h /mnt 4.2 rw,quota 1 2 /dev/st0 /design 4.2 rw 1 2	/dev/da0a / 4.2 re 1 1 /dev/da0b swap ignore xx 0 0 /dev/da0g /mnta 4.2 rw 1 2 /dev/da0h /usr 4.2 rw 1 2 /dev/da1a /tmp 4.2 rw 1 2 /dev/da1b swap swap rw 0 0 /dev/da1g /mntb 4.2 rw 1 2 /dev/da1h /mnt 4.2 rw,quota 1 2

To monitor disk space, CONVEX supports a full disk quota system. Disk quotas are set on a file–system basis by editing the *fstab* file. For example, one file system, */mnt*, can have disk quota checking enabled while another, */usr*, does not.

Set up disk quotas as described in Chapter 10.

7. Execute */dev/MAKEDEV*.
   The disk types *st* and *rst* are pseudo disk types that specify disk stripes. */dev/MAKEDEV* creates block and raw devices (files in the */dev* directory) for each disk stripe. For example, if you have two stripes, enter

   ```
   # cd /dev
   # /dev/MAKEDEV st0 st1
   ```

8. Execute *newst* to build the striped partitions and *newfs* to create non–striped partitions. The *newst* utility is similar to *newfs* in many ways, but unlike *newfs*, it is run simultaneously on multiple partitions. *newst* automatically invokes the *putst* utility and updates the *stripecap* file.

   To redirect *newst* and *newfs* output to the screen without modifying partitions, use the *-n* option. This is helpful for checking your work before executing it.

```
# newst –n –b 64k –f 8k /dev/rst1 /dev/rda0g DKD–001 /dev/rda1g
  DKD–001
```

The *-b* option selects the block size and the *-f* option selects the fragment size.

The invocation of *newfs* and *newst* depends on the selected system configuration. See the *newst*(8), *newfs*(8), and *mkfs*(8) manual pages to determine the commands and options to use.

Use the following *newst* command to create a new CONVEX striped file system.

```
# newst –b 64k –f 8k /dev/rst1 /dev/rda0g DKD–001 /dev/rda1g
  DKD–001
```

The */etc/newst* command syntax takes the *rst* device identifier as an argument. You should use the raw–device name instead of the block–device name. If the stripe already exists, you receive a message asking if you want to overwrite it.

9. Create a top–level directory for the new file system.

Each newly created file system (device) must have a matching directory to which these devices are mounted. For example, if */design* is to be a new file system, execute the following:

```
# cd /
# mkdir /design
# chmod 775 /design
```

10. Mount all file systems.
File systems must be mounted before they can be accessed. Mounting places the contents of the specified disk partition into the top–level file system directory. Use the *mount* command with the *–a* option to mount all disk partitions:

```
# /etc/mount –a
```

11. Check mounted file systems.
To check the file systems you just mounted, enter

```
# df –i
```

The output tells you the disk partition, used and free *inodes*, the block and fragment size, and the name of the file system.

```
File system   iused   ifree   %iused   blks/frags   Mounted on
/dev/da0a       614    5530     10%       16k/2k     /
/dev/st3      10446   28466     27%       16k/2k     /hw
/dev/st2       4580    9756     32%       16k/2k     /sw
/dev/st1       5977    8359     42%       16k/2k     /fonts
/dev/da1e      9912   29000     25%        8k/1k     /mnt1
/dev/da2d       266    3830      6%       64k/8k     /tmp
```

Access permissions on the directory should be at least *rwxr-xr-x*. If they are not, enter

```
# /etc/chmod 755 /design
```

You can also use *chgrp* to set the appropriate group permissions.

12. Start */etc/update*, which flushes the disk buffers at 30 second intervals. Restoring dump tapes makes major changes to the file system; run *update* first to minimize the chances that a system crash during the restoration process might seriously corrupt a disk. *update* starts automatically when you boot the system multi–user; the system is in single–user mode but during a disk repartition, so *update* must be started manually, as follows:

```
# /etc/update
```

13. Restore files backed up on tape. See Chapter 11, "Performing Backups."

14. Run *preen*.
Use the following command sequence to unmount all file systems and run *preen* to check disk integrity:

```
# cd /
# /etc/umount –a
# /etc/preen
```

If *preen* tells you to run *fsck* manually, contact the CONVEX Technical Assistance Center (TAC). The *fsck* utility tests the internal structure of a file system, ensuring that each *inode* has the appropriate number of disk blocks assigned, the reference count matches the appropriate number of pathnames, the list of free disk blocks is accurate, and the header information is accurate.

15. Return to multi–user mode.
If no errors are reported, boot to multi–user mode by holding down the Control key and pressing the D key (CTRL-D).

This brings up the system header information and the **login:** prompt. Update your dump scripts to recognize the new partitions.

Chapter 6
Maintaining User Accounts

This section describes a set of interactive utilities for adding, maintaining, and deleting user accounts, and essential system-management functions.

6.1 User Accounts

Each user on the system has a user account with an associated user name, password, and user ID. Information about this account and its associated components is maintained in the */etc/passwd* file. User accounts are set up on a case-by-case basis, according to the procedure outlined in section 6.4.

Each user also belongs to a primary group and can belong to as many as eight groups. A group is one of three classes of users (that is; user, group, and other) to which file access can be extended. (See Chapter 7, "System Protection and Security," for an explanation of file access and access protections.) File access can be extended to the members of a group to which the file's owner belongs. As a result, group members can share files with group members without granting file-access privileges to the entire user community. This ability allows projects to be organized on a group basis.

6.2 Superuser Accounts

A superuser is a user with a numeric user ID of 0 (zero). A superuser account bypasses all system security measures. The superuser has full read, write, and execute (if at least one bit is set) privileges for all files in the system, regardless of file-access privileges assigned to the file. Although these accounts are useful to system management staff, they are a security and safety risk and must be handled with caution.

Superuser privileges can be extended in two ways. In the simplest case, any user with a user ID of 0 (zero) has superuser privileges. Although this method has its advantages, reserving the zero user ID exclusively for root is highly recommended. A safer method is to require users to assume superuser privileges with the *su* command. *su* requires the root password and writes a record to the operator's console each time someone attempts to gain superuser privileges.

6.3 Password Restrictions

Password restrictions are used to increase security at your site. These optional restrictions are a good way to ensure users participate in password security by selecting secure passwords and by changing those passwords regularly. (See Chapter 7, "System Protection and Security," for more information on password security.) The two types of password restrictions are

- typing restrictions
- password aging

Select one or both of these restrictions and make the selection separately for each individual user, yielding varying degrees of security for the users at your site.

With typing restrictions, a password must

- contain at least six characters
- contain at least two alphabetic characters and one numeric or special character
- not be the user's login name or any rotated permutation of same
- differ from the user's previous password by a minimum of three characters

Password aging tracks how long a user has had a particular password and, at the end of a specified time, forces the user to change passwords. The following are requirements for password aging:

- A password must remain effective (unchanged) for a minimum number of weeks. This means users cannot change back to their original password immediately after being forced to select a new one.

- A password remains valid for a maximum number of weeks. When the password is no longer valid, the user is prompted to set a new password.

- Temporary passwords (valid for one login) and other special passwords are possible using special "age" codes. This way, you can be assured that guest users are only users for the period you intend.

6.3.1 Installing Password Restrictions

To install password restrictions, you must first decide on typing restrictions, aging restrictions, or both.

To install the restrictions, generate a file called */etc/pwrestrict*, according to the the level of restrictions required. The *nu* utility (see section 6.4) creates the entry for new users. */etc/pwrestrict* contains an entry for each line in the */etc/passwd* file. To generate */etc/pwrestrict*, use *genrest*. The *genrest* utility scans the lines in */etc/passwd*, pulls the required fields from each line, and creates entries in the */etc/pwrestrict* file.

For example, start with the */etc/passwd* file shown in Figure 6-1.

Figure 6–1: Example */etc/passwd* File

```
root:EtUg79IEEqmw.:0:10:Superuser:/:/bin/csh
daemon:4b7Cm12RO1Sq.:1:1:Our friend, the daemon:/:/bin/csh
lindholm:kajFukvcsjb56:100:60::/mnt/lindholm:/bin/csh
garza:jkasjIuana89:101:60::/mnt/garza:/bin/csh
mcbroom:aoTO8gfkajhi:102:60::/mcbroom:/bin/csh
johnson:lgjh*7hfTRlfkk:103:60::/lp/johnson:/bin/csh
esmark:Lkasjjhiah:104:60::/lp/esmark:/bin/csh
horwitz:klajlaYIB02lkf8:105:60::/lp/horwitz:/bin/csh
wu:hKL/REb8o64oM:106:60::/doc:/bin/csh
```

If an */etc/pwrestrict* file does not exist, you can create one from the */etc/passwd* file by executing the command

```
# genrest -t -m4 -M8
```

genrest command to generate the restrictions file

–t enable typing restrictions for every user

–m4 specifies the minimum number of weeks a password must remain in effect before it can be changed. In this case, it is 4 weeks. The default for this option is 1 week.

–M8 specifies the maximum number of weeks a password can remain in effect before the user must change passwords. In this case, 8 weeks is specified. The default for this option is 52 weeks.

The command example yields the */etc/pwrestrict* file shown in Figure 6-2.

Figure 6–2: Example */etc/pwrestrict* File

```
# genrest -t -m4 -M8
root:EtUg79IEEqmw.:4,8,Oct 8 1988:Y:0
daemon:4b7Cm12RO1Sq.:4,8,Oct 8 1988:Y:1
lindholm:kajFukvcsjb56:4,8,Oct 8 1988:Y:100
garza:jkasjIuana89:4,8,Oct 8 1988:Y:101
mcbroom:aoTO8gfkajhi:4,8,Oct 8 1988:Y:102
esmark:lgjh*7hfTRlfkk:4,8,Oct 8 1988:Y:103
horwitz:Lkasjjhiah:4,8,Oct 8 1988:Y:104
geary:EKils/hfua978f:4,8,Oct 8 1988:Y:105
doc:klajlaYIBO2lkf8:4,8,Oct 8 1988::106
```

In the */etc/pwrestrict* file, the five fields correspond to

1. login name
2. encrypted password
3. password age
4. password typing (where *Y* is enabled and *N* is disabled)
5. user ID

The password age can be specified in any form acceptable to *calendar*. For example, the notation in the first line of Figure 6-2

- *4,8,Oct 8 1988* means the password changed on October 8, 1988
- *4* means the password must remain unchanged for 4 weeks
- *8* means the password remains for a maximum of 8 weeks

Each user now is subject to password restrictions. You do not have to treat all users equally; you can relax or tighten restrictions for particular users. You can change the password for root more often. Section 6.6 describes how to make these changes using *vipw*.

If a *pwrestrict* file exists and you attempt to execute the *genrest* command, you receive the error message

```
genrest: /etc/pwrestrict already exists
```

To add restrictions for all users and the file exists, remove the */etc/pwrestrict* file, then execute *genrest*. To selectively add restrictions, use *vipw* to edit the */etc/pwrestrict* and */etc/passwd* files. (See section 6.6, "Changing Password Files," for more information.) Although the password file is a text file and can be edited with any text editor, use *vipw* because it has a locking protocol and ensures that the */etc/passwd* and */etc/pwrestrict* files remain compatible (have the same entries). Section 6.6, "Changing Password Files," describes *vipw*.

For NFS sites, the */etc/pwrestrict* file can be installed as a Yellow Pages (*yp*) map on the network file server. The *ypmake* utility creates and installs the maps *pwrestrictby.name* and *pwrestrictby.uid*. See the *ConvexOS Network File System System Manager's Guide* for details.

6.3.2 Maintaining Password and Restriction Files

The following tasks are performed regularly in maintaining the password files:

- adding a new user to the system (see section 6.4)
- changing the account information for a user (see section 6.6)
- removing a user from the system (see section 6.7)

The same functions are applied simultaneously to the *companion* files */etc/passwd* and */etc/pwrestrict*. Use *vipw* to change these files. Editing the files separately can lead to disparity. The */etc/passwd* file takes precedence over the */etc/pwrestrict* file, so if the files differ, you may not have the restrictions you intended in place.

6.4 Adding a New User

The utility to add new users to the system, *nu*, can be used interactively or in batch mode. The *nu* utility

- adds a new entry to the */etc/passwd* file each time a user account is created (the *passwd*(5) manual page describes the fields password lines must contain)
- creates a home directory for the user
- copies start-up files into the user's home directory

After you create the initial user accounts, no further maintenance on them is required unless you want to alter the restrictions. Each user can change

- the *finger* listing of their office number, work phone, and home phone with the *chfn* command
- the login shell by using *chsh*, but only to those shells listed in */etc/shells*
- their password by using *passwd*

To change the password file, use *vipw*, *chfn*, *chsh*, and *passwd*. See section 6.8., "Some Useful Utilities."

6.4.1 Using *nu* Interactively

In interactive mode, *nu* prompts for all information about a user, including password-restriction information. The format for the *nu* command in interactive mode is

nu [**-n** *defaults_file*] *login_name*

defaults_file is an optional file specifying default values for the user account; *login_name* is the name (lower case; of the user for whom you are creating the account.

Note	The login name cannot have more than eight alphanumeric characters (a–z, 0–9). All characters must be lower case. Do not use underscore, hyphen, or punctuation characters in the login name. Invalid characters in a login name cause trouble with several ConvexOS utilities; for example, *sendmail*.

When *nu* prompts for information, it uses a set of default values embedded in the code. You can establish an alternate set of defaults in a defaults file which is either the file */etc/nurc* or an alternate file specified with the *-n* flag. If */etc/nurc* exists, *nu* uses it automatically (you do not need the *-n* flag). If you specify an alternate defaults file with the *-n* flag, *nu* ignores */etc/nurc* and uses the alternate file instead. The format of the defaults file is

fieldname:value

fieldname field for which information is specified

value constant assigned to a *fieldname*

Fields that can be contained in */etc/nurc* (or the alternate defaults file) and their defaults are listed in Tables 6-1 and 6-2. Table 6-2 lists fields requiring Boolean operators; entering the *fieldname* in the file turns the default value on.

Table 6-1: Fields and Defaults for *nu* Defaults File

Tag	Default Value	Description
uid	(see Table 6-2)	user ID
gid	EMPTY	group ID
group	staff	group name (used only if *gid* field is not given)
directory	*/mnt*	path under which home directory is placed
protection	0755	home directory permission
shell	*/bin/csh*	login shell
password	login name	user's initial password
username	username	user's full name (for the *finger* command)
office	office	user's office (for the *finger* command)
extension	extension	user's work phone extension
homephone	homephone	user's home phone number
minwks	1	minimum number of weeks for password aging
maxwks	52	maximum number of weeks for password aging
diskquota	6000, 8000, 1200, 1500	*blksoft, blkhard, inodesoft, inodehard* quotas
skeleton	*/usr/skel*	directory containing files to copy into home directory
homedir	*<directory>/<login>*	home directory to create for this user

Table 6-2: Boolean Fields and Defaults for *nu* Defaults File

Tag	Default Value	Description
typed	OFF	set password typing restrictions
aged	OFF	set password aging restrictions
quota	OFF	initialize quotas for home directory file system
nouidfile	OFF	ignore the contents of */etc/uidcount*

The default *uid* is the value specified in the */etc/uidcount* file unless the *nouidfile* Boolean field is set. If *nouidfile* is set, the default for the *uid* field is the number one greater than the highest *uid* already in the */etc/passwd* file, to a maximum of one less than the largest unsigned integer (4294967295).

The *skeleton* field specifies the directory in which templates for start-up files (*.cshrc, .login, .logout, .exrc,* and *.project*) are stored. *nu* copies these files to the user's new home directory.

In Figure 6-3, file-system quotas and typing restrictions are turned on by listing the field names in the file.

Figure 6–3: Example *nu* Defaults File

```
shell: /bin/csh
group: swtst
directory: /swtst
protection: 0755
typed
quota
diskquota: 3000, 4096, 1000, 1500
```

For more information on */etc/nurc*, refer to the *nu*(8) and *nurc*(5) manual pages in the *ConvexOS Programmer's Manual.* Figure 6-4 is a sample session of *nu* used interactively. You must be the superuser to run *nu.* Defaults for each field display in square brackets. To use the default, press RETURN.

Figure 6–4: Using *nu* Interactively

```
# nu mcbroom
Login name: mcbroom
User id [3012]:
Group id [staff]: lp
Home directory [/doc/mcbroom]:
Home directory protection [0755]
Login shell [/bin/csh]:
Parent share group [staff]:
Number of shares [100]:
Is this a new scheduling group [N]:

Changing the user information...
Default values are printed inside of of '[]'.
To accept the default, type <return>.
To have a blank entry, type the word 'none'.

Name [username]: Sharon McBroom
Room number (Exs: 18A or 17B) [office]: 134
Office Phone (Ex: 223) [extension]: 490
Home Phone (Ex: 6610379) [homephone]: 555-1212

Password to be typed [N]: Y
Password subject to aging [N]: Y
   Enter the minimum period for the password [1 week]: 4
   Enter the maximum period for the password [52 weeks]: 8

Entering the user password...
New password:
Retype new password:

Creating the home directory...
#
```

In Figure 6-4, user *mcbroom* is added to the system with both typing and aging restrictions. The *nu* utility writes the entry

```
mcbroom:GjUhdskY:301:60:Sharon McBroom,134,,490,844-4444:/mnt/mcbroom:/bin/csh
```

to */etc/passwd* and the entry

```
mcbroom:GjUhdskY:4,8,Oct 8 1988:Y:301
```

to */etc/pwrestrict.*

6.4.2 Using *nu* in Batch Mode

The format for using *nu* in batch mode is

nu -f *batchfile* [**-n** [*defaults_file*]]

batchfile	information for new users; one user per line
defaults_file	file specifying default values for all user accounts

If the *-n* flag is used, *nu* uses the defaults file */etc/nurc* unless an alternate file is specified. If both a defaults file and a batch file are specified (either by using the *-n* flag alone or by using it to specify an alternate file), the defaults file overrides any defaults specified in the *nu* code itself before processing of the users in the batch file begins. Defaults specified in the batch file override defaults set in the defaults file.

The format of the batch file is

fieldname=value[: . . .]

Although this format is different than the defaults file, you can define all of the fields listed in Tables 5-1 and 5-2. An example of a line in *batchfile* is:

```
login=tgarza:username=Thea Garza:office=107A:extension=555:typed
```

You *must* specify the *login* field for each user in the batch file.

6.4.3 Additional File Changes for a New User

After you add a new user, you can make the following changes:

- Customize the start-up files copied by *nu* from */usr/skel* (or from an alternate directory you specified)
- Add aliases to the */usr/lib/aliases* file for mailing purposes. This file includes mailing lists and alternate names (for example, both *jim* and *bashful* are legal mail aliases for user Jim Smith).

6.5 Adding Group Memberships

A user's primary group is specified in the */etc/passwd* file. Membership in additional groups is specified in the */etc/group* file. Figure 6-5 shows an example of this file.

Figure 6–5: Example */etc/group* File

```
operator:*:5:root,sam
notes:*:13:*
games:*:20:*
uucp:*:40:*
docn:*:51:lutz,tuttle,mcbroom
diag:*:52:zelle,marciniak,garza
mgmt:*:57:horwitz,esmark,lindholm
quality:*:59:
```

Each line contained in the group file is a group entry consisting of the following fields:

Group name — The group name is chosen by the system manager. It is an arbitrary name and must be from one to eight characters (from the set [a-zA-Z0-9-_]) long. The name should be easy to use and descriptive of the group.

Group password — The password field is not supported. (There is no direct way to specify or change the password; for further explanation, refer to the *group*(5) manual page.)

Group ID	The numerical group ID is the system's internal representation for the group. It has a range of 0 to 4294967295. When assigning group IDs, assign numbers in sequence.
Group members	users in a group; include only users who need access to more than one group. (The group IDs listed in the */etc/passwd* file already specify one group, making it unnecessary to duplicate that information in the group file.)

In the */etc/group* file, list only group memberships not listed in the */etc/passwd* file.

When a user creates a file, the new file inherits the group ownership of the directory in which the new file resides; the user's own groups have no bearing on the group of the newly created file. To change the group to which a file belongs, use the *chgrp* command.

6.6 Changing Password Files

After you use *genrest* to create the */etc/pwrestrict* file for your site, you can make changes to that file. For instance, you can add tighter restrictions to some accounts or remove all restrictions from some accounts. Use *vipw* to make these changes.

vipw simultaneously edits the */etc/passwd* and */etc/pwrestrict* files. It sets the appropriate locks so only one person can alter these files at a given time.

To prevent separate editing of the two companion password files, *vipw* combines both files into one for the editing session. It displays a single line for each user containing both restriction and password information. When you have finished editing the file, *vipw* splits the information and updates the original two files.

If differences exist between entries in the two files, *vipw* notes them at the end of the temporary file you are editing. After you determine the cause, you can delete them or add them to the appropriate file (using *vipw*).

The *vi* editor is invoked for *vipw* unless you specified an alternate editor in your EDITOR environment variable (in your *.login*). To use *vipw*, invoke *vipw* as shown in Figure 6-6 and edit the text file *vtmp*. When you exit the editor, the */etc/passwd* and */etc/pwrestrict* files are updated at the same time.

Figure 6–6: *vipw* Example

```
# vipw
root:EtUg79IEEqmw.:0:10:Superuser:/:/bin/csh
test:.HupMP3p8SCwA:16:49:TEST User:/mnt/test:/bin/csh
mcbroom:wt.ZCGPC1tLOE:80:159:Sharon McBroom:/mnt/mcbroom:/bin/csh:1,4,Oct 9 1986:Y
garza:KJtopagar/DZE:293:60:Thea Garza:/lp/garza:/bin/csh:0,0,:N
esmark:SOUvU1eJmG8tc:395:55:Bruce Esmark:/mnt/esmark:/bin/csh:1,0,Sep 1 1986:Y
horwitz:EykHd.02BNA1.:469:60:Barbara Horwitz:/lp/horwitz:/bin/csh
mjohnsto:NfP.3I8Q97WUw:473:60:Marilyn Johnstone:/lp/mjohnsto:/bin/csh::Y
lindholm:8fVQtKU1qM2s:561:60:Peter Lindholm:/mnt/lindholm:/bin/csh
```

In the preceding file, you can see restrictions imposed for each user toward the end of each line (after the user's shell is named). User *mcbroom* has a minimum password age of one week, a maximum of four weeks, the password changed on October 9, and password typing restrictions are turned on (Y). User *garza* has a temporary password in effect (0,0), which means she must change her password the next time she logs in. She does not have the password typing restriction

(N). User *mjohnsto* has no aging in effect (: :), but has typing turned on. To put aging into effect for *mjohnsto*, edit the file and insert the numbers you want for minimum and maximum between the last two colons.

6.7 Deleting User Accounts

Use the following steps to delete user accounts:

1. Change the user password to something untypeable; for example, ** (two asterisks).
2. Archive the user directory on tape.
3. Remove the user's home directory and mail file (*/usr/spool/mail/*`username`).
4. Remove the user name from global mail aliases (*/usr/lib/aliases*), groups (*/etc/group*), and special accounting files. If the user is moving to another system, add a new alias to the */usr/lib/aliases* file to forward their mail.
5. Use *edquota* to set the user's limits to zero. See section 10.2.2.6, "Quota System Administration."

Use the *find* command to locate files belonging to a deleted user name.

An alternative to step 1 is to change the user's password and specify the restriction to make the password changeable only by the superuser. This keeps the user's account intact, but secures it from the user by allowing only the superuser to know the new password.

If disk space is not a critical issue, you can only do step 1. When a user leaves, the person taking over the user's work needs the files.

6.8 Useful Utilities

Utilities described in this section simplify the tasks of adding and maintaining user accounts. Read the information on these utilities in the *ConvexOS Programmer's Reference*.

6.8.1 *su* Utility

Use *su* to change your user ID (UID) to that of another user or to root. Under normal circumstances, you are required to supply the user's password to assume their user ID. If you invoke *su* as a superuser, however, you are not required to submit a password.

If you use *su* for root login, you can check the audit trail of user logins to determine the specific user who invoked it. Using the audit trail is less time-consuming than tracing a user who logs in directly as the superuser. See the *last*(1) manual page.

6.8.2 *whoami* Utility

whoami prints your user name. Use *whoami*, for example, after repeatedly using *su* to change UIDs.

6.8.3 *groups* Utility

groups tells you the groups to which you or another user belong. This information is essential for removing references from the */etc/passwd* and */etc/group* files when you are deleting accounts. Knowing the groups to which a user belongs is also useful when you need to edit a file; if the user belongs to a different group, you may have to log in as root to edit the file. Enter

% **groups**

to find out the groups to which you belong. To determine the another user's group, add a user name as an argument.

6.8.4 *finger* Utility

finger (or *f*) lists a login name, terminal name, idle time, login time, phone number, and other interesting facts for each user currently logged in. To display this list, enter

% **finger**

If you add a user name as an argument, a display similar to the one shown here is printed, but will also work for users not currrently logged in.

```
Login name: mcbroom                    In real life: Sharon McBroom
Office: 151 port C, x225               Home phone: 555-1212
Directory: /mnt/mcbroom                Shell: /bin/csh
On since Feb 14 16:02:25 on ttyic
```

Two more fields, *plan* and *project* can also be used. Write a line in a *.project* file and one or more lines in a *.plan* file in the new user's top-level directory. The *.plan* and *.project* file contents display with the *finger* display.

6.8.5 *chfn* Utility

chfn changes the user information referenced by *finger* and other programs. To use it, enter

% **chfn** *username*

The superuser can change any user's *finger* information. You are prompted for a *real life* name, office room number, office phone number, and home phone number. Enter a carriage return to leave the same or type **none** to enter a blank field. Check your work with *finger* when you are done.

6.8.6 *chsh* Utility

The default shell for most environments is the C shell located in */bin/csh*. To use the Bourne shell (*/bin/sh*), use *chsh* to change your default login shell. You can also use *chsh* to change back to the C shell. Acceptable shells are listed in the */etc/shells*(5) manual page.

6.8.7 *passwd* Utility

Use *passwd* to change passwords. The program prompts you for the old password and then for the new one. You are prompted for the new password twice to verify that you typed it correctly.

You also receive an error message if you do not correctly follow the typing restrictions in place at your site or if you attempt to change your password before the designated amount of time in the restrictions file. These error messages are self-explanatory; try again with a password that complies with the typing restrictions.

If your system has password aging, your password expires at set intervals, forcing you to select a new password. When you type your user name to log in, you are prompted to change your password before being allowed to log in.

Only the superuser can change another user's password. To change a user's password, enter

% **passwd** *username*

and enter the new password.

Chapter 7
System Protection and Security

The system manager is responsible for establishing a consistent security policy to govern password protection, user access, remote access, protection of sensitive information, and protection of system files and utilities for a system or network of systems.

A multi-user operating system allows multiple methods of access for users and is potentially accessible from active terminals or a terminal and modem. A security policy is necessary not only to protect against unauthorized users but to guard against user errors that can result in file or data corruption.

This chapter describes ConvexOS features for establishing system protection and security procedures, including

- protecting system and user files
- logging failed file access
- erasing deleted files
- logging failed login attempts
- password security
- *uucp* security
- file encryption

These procedures, alone or in combination, cannot guarantee system integrity; that responsibility lies with the users. The system manager can develop a clear, consistent, easily-implemented policy and procedures as well as regularly monitoring the system for security breaches or inconsistencies.

7.1 Protecting Files

Each file and directory in the ConvexOS system has attributes that specify who can access it and the degree of access allowed.

7.1.1 File Access

The ConvexOS environment has the following types of file access:

- *read* access specifies whether users can read a file
- *write* access specifies whether users can modify the contents of a file
- *execute* access determines whether users can execute a file (program)

Corresponding levels of protection exist for directories, as follows:

- *read* access enables the user to list the directory's contents
- *write* access allows the user to alter the contents of a directory (create files, add files, delete files, rename files), regardless of the type of access assigned to these files
- *execute* access allows users to search the directory and its subdirectories or use the directory in the user's pathname

Permission to read, write, or execute each file or directory can be extended to one or more of the following classes of users:

- *user* — owner of the file
- *group* — members of the group to whom the file belongs (typically, the group to whom the owner of the file belongs; group assignments are defined in the */etc/passwd* and */etc/groups* files)
- *others* — all users of the system

The system uses a set of three 3-bit fields to represent file-access permissions extended to the user, group, and others. To view these permissions for a file, use the *ls* command with the *-l* (*long*) argument as in the following example for the file *newsort*:

```
% ls -l newsort
-rwxr-xr--  1 mcbroom     13329 Jul 12 14:52 newsort
%
```

In this example, the owner of the file (*mcbroom*) has read, write, and execute access; members of the group to which the file belongs have read and execute access; other users only have read access (which means they could copy the file).

Different types of files require different file access permissions, for example

- Object files must be executable; permission to execute these files must be extended to anyone needing to use them.
- Text files (for example, files created by an editor) usually do not have execute access so cannot be executed, even by the owner of the file. If the text files are shell scripts, they must be both readable and executable to be invoked; their file-access permissions can be changed using the *chmod* command (see section 7.2.3).
- The loader (*ld*), which links object files and libraries, creates files all users can execute, depending on their *umask*.

Protection can be specified for user files as well as administrative and system files. System files are standard files owned by special user names; for example *uucp* or *notes*. Use the following guidelines to protect system files:

- Extend write access for password and group files only to superuser. Allow everyone read access to the password file, because many general-purpose utilities (for example, *ls*) use it.
- Do not extend write privileges to ordinary users for system directories (including /, the root directory) except the */tmp* directory, to which all users should have read, write, and execute access; this includes group permissions.

Note	Under *nfs*, users have access to files they own, regardless of underlying permissions.

7.1.2 File-Protection Bits

File-access permissions can be described symbolically or by using file-protection bits expressed in octal.

Using the symbolic method, file-access levels and the classes of users to whom they are assigned are described by an alphabetic character, as follows:

- file-access levels are *r* (read), *w* (write), and *x* (execute)
- user class is *u* (user), *g* (group), *o* (other), or *a* (all; user, group, and other)

Using file-protection bits, an octal number is assigned to each access level and the sum of the bits describes the set of access levels, for example

- octal 4 = read access
- octal 2 = write access
- octal 1 = execute access

Summed, for example:

- octal 7 = read, write, and execute access
- octal 6 = read and write access

A set of three octal digits describes file-access levels (mode) for each file or directory. For example, mode 644 assigns read and write access to the user (file owner) and read access to the group and others.

7.1.3 Changing File Access

Users can change access to files they own using the *chmod* command. Using symbolic expression of file access, the format is

chmod [**ugoa**][+|-][**rwx**] *filename*

To add execution access to a file for all users, enter

% **chmod x** *filename*

The *a* is assumed; **a+x** is the same as **x**. Using file-protections bits, the format is

chmod *mode filename*

mode is the 3-digit octal mode specification. For example,

The following example assigns read and write access to all users:

% **chmod 666** *filename*

7.1.4 Default File Access

Users can set default file-access permission with the *umask* variable. The value of *umask* is a set of three octal digits representing file-access permissions assigned to user, group, and other. *umask* is a mask because each digit is subtracted from the system-wide default file-access levels. The system-wide defaults are 777 for executable files and 666 for text files.

Common values for *umask* are

002	all access to user and group; read and execute access to others (read access for text files)
022	all access to user; read and execute access (read access for text files) to group and others
027	all access to user; read to group; no access to others

To determine the current value of *umask*, execute the command without arguments, as follows:

```
% umask
000
%
```

umask is set in the *.login* file in the user's home directory.

7.2 Logging Failed File Access

For additional system security, ConvexOS provides a facility for logging file-access attempts that fail because of insufficient file-access permissions. With this facility, you can track unauthorized attempts to access protected files or directories.

7.2.1 Enabling and Disabling File-Access Logging

To enable file-access logging, enter (as the superuser)

```
# /etc/faillogon /usr/adm/failure_log
```

where */usr/adm/failure_log* is the name of the log file where failed access attempts are recorded. The file must exist and typically is */usr/adm/failure_log*.

To disable logging, use the same command with no argument. See the *failure_log*(5) manual page for a description of the log format.

Enabling this utility increases system overhead and degrades performance.

Two boot-time tunable parameters dictate when logging is suspended and resumed:

- *log_suspend*
- *log_resume*

The *log_suspend* parameter specifies the percent of free disk space on the file system that must be available to continue logging. When the free space falls below the number specified by *log_suspend*, logging is suspended and a message is written to the system console and to the error log.

The *log_resume* parameter specifies the percent of free space on the log file system necessary to resume logging. Use *cron* to clean the files periodically to ensure continuous logging. See Chapter 3, "Tuning ConvexOS" for details on tuning these parameters.

7.2.2 Failure Log File

The */etc/faillogon* command records the major and minor device numbers and the *inode* number for the file that a user unsuccessfully attempts to access. The */usr/adm/faillogpr* program translates this information into a full pathname and displays a human-readable report of the failed file-access attempts.

To print the formatted log file to the screen, enter the following command as the superuser:

```
# /usr/adm/faillogpr /usr/adm/log_filename
```

faillogpr displays information on each failed file access in the following format:

time uid (uid) errno mode command filename

time	date and time of attempted access
uid	real user ID of user who attempted access
uid	effective user ID of user who attempted access (if different from the real user ID)
errno	mnemonic of error
mode	file-access mode (read, write, execute) of the file
command	first word of command name used in attempt
filename	file to which user attempted access

Figure 7-1 is a sample output from the failure log.

Figure 7–1: Sample Output from Failure Log

```
Tue Apr 14 16:08:29 CDT 1987 mcbroom RW EPERM csh /etc/rc
Tue Apr 14 16:12:29 CDT 1987 garza X EPERM csh /etc/rc
```

The character strings are written with control characters translated to backslash sequences to distinguish the names of the file and command. The format is like that used in C character strings. Space characters are translated to the sequence *\040* so *awk* can treat the filename as a single field.

Filenames generated by *faillogpr* are the correct filenames at the time *faillogpr* is run, not when the the failed access occurred.

The following system calls can generate log messages when they fail:

access	*chmod*	*execve*	*lstat*	*open*	*stat*	*truncate*
acct	*chown*	*exportfs*	*mkdir*	*quotacl*	*statfs*	*unlink*
bind	*connect*	*faillog*	*mknod*	*relink*	*swapon*	*unmount*
chdir	*creat*	*link*	*mount*	*rename*	*symlink*	*utimes*

An attempt to generate a core file also generates a log message.

7.2.3 Logging Failed File Access With *cron*

You can put the */etc/faillogon* command (as shown in section 7.3.1) into */etc/rc.local* and place an entry in */.crontab* to run a shell script that executes *failogpr* at least daily, renames old log files, and creates a new log file. Figure 7-2 is a sample shell script for maintaining the log file.

Figure 7–2: Shell Script for Log File Maintenance

```
#!/bin/sh
# Bourne shell script to run periodically from cron; script formats failed file
# access log and starts new log. You must run script as superuser.
# $tmp is name of file to which failure log is moved so logging continues
# without missing files. $tmp must be on same file system as
#     /usr/adm/failure_log
# so mv command renames rather than copying.
tmp="/usr/adm/fl.$$"

# $permanent is name of file in which permanent formatted dopy of log is kept.
# Name format is /usr/adm/faillog/faillog.<year>.<month>.<day>.hh:mm:ss
set 'date'
permanent="/usr/adm/faillog/faillog.$6.$2.$3.$4"

# Move open log to $tmp while still logging to it
mv /usr/adm/failure_log $tmp

# Create new log file
touch /usr/adm/failure_log

# Switch logging from old to new file
/etc/faillogon /usr/adm/failure_log

# Format log, resolving dev/inumber into pathnames. Keep as long as needed.
/usr/adm/faillogpr $tmp > $permanent

# Done with unformatted log.
rm $tmp
```

For more information, see the *failure_log*(5) and *faillog*(2) manual pages.

7.3 Creating and Protecting Mail Files

ConvexOS automatically creates, maintains, and deletes mail files. The system creates a mailbox when the first piece of mail is sent to a user. When a reader deletes all the mail, the mailbox is removed. Refer to the *mail*(1) manual page for mail system documentation.

The mail system is secured as follows:

- Only the superuser can write files to the mail directory.
- Users can only read mailboxes with appropriate permissions. The default is mailboxes readable only by their owners.
- Individual pieces of mail are created as root-owned files in the directory */usr/spool/mqueue*. File ownership transfers to the recipient when the mail is delivered and placed into the directory */usr/spool/mail*.

7.4 Erasing Deleted Files

The ability to "erase" deleted files is another security feature. When enabled, the file-erasing facility overwrites free blocks with a value specified by the system manager; blocks from deleted files retrieved directly from the raw disk are not readable.

To enable or disable file erasing, set the kernel boot-time parameter option *erase_unlink*. Setting this parameter to 1 enables file erasing; setting it to 0 (zero) disables file erasing.

Another boot-time parameter, *erase_pattern*, specifies the 32-bit pattern that overwrites the deleted files. The default for this parameter writes the binary pattern "101010..." over the deleted files. See Chapter 3 of this guide for details on setting these parameters.

You can also initialize a file system with the erase pattern written over all unused blocks. The *newfs* and *newst* utilities can be specified with the optional argument *-E pattern*, where *pattern* is the erase pattern passed to *mkfs*. When *mkfs* gets a *-E pattern* argument before any of its other arguments, it overwrites the partition with the pattern before creating the file system.

Note	Enabling this utility increases system overhead and can degrade performance.

7.5 System Security

Providing adequate security for your system requires

- controlling physical access to the system
- using passwords effectively
- securing access by remote users

7.5.1 Physical Security

Part of protecting your system is inhibiting physical access to the system by nonusers. One way to do this is using security systems that restrict entry into the building or the room containing the system and backup media.

Physical security, however, requires an alert and security-conscious user community. Users should, for example, always log out from the system when leaving their terminals.

7.5.2 Password Protection

If an unauthorized user gains access to a terminal line connected to the system, only the lack of a valid login name and password bars entry to the system. Unfortunately, login names are easily deduced because they are almost always the user's first name, last name, or initials. Consequently, ConvexOS demands a password before it grants access to the system.

The primary characteristic of a secure password (that is, one that resists detection well) is that it is not obvious or easily derived. Observe the following guidelines (at minimum) when selecting passwords:

- Do not use passwords based on family names, maiden names, initials, phone numbers, or social security numbers.
- Alter passwords derived from a dictionary by a slight misspelling or by mixing one or more numerals with the characters.

- Choose passwords that can be typed easily (or users will be likely to select other passwords).
- Change passwords frequently, especially for root.

Secure passwords include words from other languages, fantastic words, words based on acronyms, and strings of mixed case, numerals, or characters. You can improve the more transparent passwords by typing them on a row of keys different from the home row.

Chapter 6 of this guide describes the following options available to the system manager to foster password security:

- typing restrictions, which require the user to select a password that
  - — has at least six characters
  - — has at least two alphabetic characters and at least one numeric or special character
  - — is not the user's login name nor a permutation of it
  - — differs from the user's previous password
- password aging, which forces the user to change passwords after a specified time

Passwords are recorded in the */etc/passwd* file in a one-way encrypted form. Carefully maintained file-access permissions are important on this file; if the file is writable, users can access superuser permissions and, subsequently, all files and processes on the system. Several programs use the */etc/passwd* file to map user IDs to user names, so all system users must have read permission.

You can delete a user's encrypted password from the */etc/passwd* file if that user is not required to specify a password, but it is *not* recommended. For more information, see "Password Security: A Case History," by Robert Morris and Ken Thompson in the *ConvexOS Tutorial Papers*.

7.5.3 Logging Failed Login Attempts

You can further protect your system by logging failed login attempts. Create file named */usr/adm/badlogins*. Then, each time a user attempts to log in and fails, a line like the following is added to */usr/adm/badlogins* file:

```
Tue Apr 14 16:08:29 CDT 1987 tty37 mcbroom
```

Tue Apr 14 16:08:29 CDT 1987	date and time of attempted login
tty37	terminal used for the login
mcbroom	name used in the failed login

If the write to the log file */usr/adm/badlogins* fails, the message is written to the console.

7.5.4 *uucp* Security

The *uucp* program copies files from system to system across phone lines. Because it is designed to provide access to your system by remote users, it is a security risk.

The easy way to avoid breakins through the phone lines is to eliminate the phone lines. That, however, restricts users to working on-site and foregoes access to the worldwide Usenet. (Usenet is a subset of the *uucp* network, which connects thousands of UNIX machines over dialup phone lines, over which news items are exchanged worldwide.)

The best approach to *uucp* security requires assigning regular *uucp* callers their own *uucp* passwords. The passwords also establish an audit trail to be used if the system is compromised.

Use the following procedures to protect the *uucp* program:

- Files located in the directory */usr/lib/uucp* should be owned by the *uucp* administrator and have permissions that secure them from access by unauthorized users. For example,
  - the file */usr/lib/uucp/L.sys*, where phone numbers and login information for remote systems are stored, should be owned by the *uucp* administrator and have mode 0400 protection.
  - the files */usr/bin/uucp*, */usr/bin/uux*, */usr/lib/uucico*, */usr/bin/uulog*, */usr/lib/uuclean* should be owned by the *uucp* administrator, have the *setuid* bit set, and have only execute permission

  When your system software is initially loaded, these files have the proper permissions. Do not alter these permissions when you make changes to files in this directory.

- Require remote users, like local users, to sign on with a login name and a password.

- You can also require a callback protocol for remote users by specifying the callback option in */usr/lib/USERFILE*. In this file, you can require, on a user-by-user basis, that a call-in is told that callback is requested; the local system then terminates the connection and immediately calls back the remote host.

- Require remote users to provide the name of the system they are calling from. The system checks this name against a list of authorized remote sites before allowing the user to log in.

- After logging in, remote users are permitted to use only a few programs. This access is established by permissions established in the following files:
  - */usr/lib/USERFILE*

    In this file, specify the pathname(s) of directories accessible to users. You can establish this on the basis of individual users or remote hosts. The default *USERFILE* restricts access to the public *uucp* directory */usr/spool/uucppublic*.

  - */usr/lib/L.cmds*

    This file lists commands permitted for remote execution and the default search path. Filenames supplied as arguments to these commands are checked against the list of accessible directories in *USERFILE*; do not include commands in *L.cmds* (especially *sh* and *csh*) that accept local filenames or you open the system to breaches of security.

When configured carefully, the *uucp* program is difficult to compromise.

In addition to the *uucp* program, you can set up a *dialin* password for your system. *CONVEX recommends changing the dialin password monthly.* Use the following steps to set up a *dialin* password:

1. Edit the password file (using *vipw*) to create an entry for *dialin*. The *dialin* account does not require a valid shell or a separate home directory.

```
# /etc/vipw
"passwd" 118 lines, 9230 characters
:a
dialin::15:31::/tmp:/bin/false
:wq
"passwd" 119 lines, 9261 characters
```

2. Set a password for the dialin account. This is the extra password users must enter to access their accounts through a dialin port.

```
# passwd dialin
New password:
Retype new password:
```

3. Change the name of the terminal device connected to the modem. The name changes from *ttyXX* to *ttydX* in the */dev* directory.

```
# mv /dev/tty00 /dev/ttyd0
```

4. Edit */etc/ttys* to reflect the port name change. You must also change the port type as necessary to reflect the speed(s) of the modem (see "*tty* Configuration" in Chapter 5).

```
# ex /etc/ttys
"/etc/ttys" 81 lines, 649 characters
:/tty00/
12tty00
:s/2tty00/3ttyd0/
13ttyd0
:wq
"/etc/ttys" 81 lines, 649 characters
```

5. Edit the */etc/ttytype* file to change the name of the dialin port.

```
# ex /etc/ttytype
"/etc/ttytype" 82 lines, 1069 characters
:/tty00/
vt100n tty00
:s/tty00/ttyd0/
vt100n ttyd0
:wq
"/etc/ttytype" 82 lines, 1069 characters
```

6. Add the word *dialin* to the */etc/ftpusers* file. If you do not make this entry, a user can use *ftp* to gain non-privileged access to a system on a network even if they do not have access to an account on that system.

7. Use the *on* command (with no arguments) to signal the *init* program of the change.

```
# on
```

7.6 File Encryption

To encrypt files, use the *crypt* utility. (*crypt* is not available for international releases of ConvexOS.) *crypt* encrypts and decrypts the contents of a file based on a password; the password is the key that selects a particular transformation for encryption.

The format for encrypting a file is

crypt < *plain_file* > *encrypted_file*

crypt prompts for a password and inhibits printing to the terminal while the password is entered. The following example illustrates the password prompt:

```
$ crypt <test.data> test.hidden
Enter key:
$
```

After encrypting the file, remove the original (plain text) file from the system.

The format for decrypting a file is

crypt *<encrypted_file> plain_file*

The encryption algorithm is difficult, but not impossible, to break. Because passwords and password security are the most vulnerable aspect of *crypt*, make passwords longer than three letters (at least) and do not store them on the system. For more information, refer to the *crypt*(3) manual page in the *ConvexOS Programmers' Reference*.

7.7 General Comments About Security

Do not put sensitive material (for example; payroll data, confidential memos, strategic information, classified data) online without encrypting it.

To prevent misuse of the system by legitimate users, consistently use the file-protection methods discussed in this chapter. To protect files

- add *umask* to the user's *.cshrc* file to restrict directory access to a group
- use *chmod* to restrict access to existing files and *umask* to restrict access to files to be created
- Restrict access to superuser password.
- In a general-purpose timesharing system like ConvexOS, users must protect their passwords and files and log out before leaving a terminal unattended.

  The autologout option for *csh* automatically logs out users whose shells have been idle for a specified time. To enable this option, include the following line in */etc/login* or the *.cshrc* file of each user:

  set autologout = *#min*

  #min is the number of minutes the shell can be idle before automatic logout.

Chapter 8
Mail System and Communications

The ConvexOS communication tools are used to move text to user mailboxes, terminals, notesfiles, or *msgs* base. The *mail*, network mail, *talk*, and *write* tools are primarily used for one-to-one communication. The *msgs*, */etc/motd* (for messages of the day), *wall*, and *rwall* utilities are for communicating with groups of people.

msgs, */etc/motd*, and *wall* are for system-wide announcements. Use */etc/motd* and *wall* to communicate important messages to users and use *msgs* to communicate announcements. Messages sent with *wall* are transmitted *immediately* to all logged in users.

The *mail* system, used primarily for personal communication, can also be used for announcements. The notesfile is a forum for discussion and for announcements. Use *talk* or *write* for one-to-one real-time communication.

The *contact* utility is an interactive program for submitting a problem report to the CONVEX Technical Assistance Center (TAC).

The following sections contain basic instructions for using these utilities. See the *ConvexOS Primer* for more information on the *mail*, *msgs*, and *talk* utilities.

8.1 *mail* Utility

mail is a mail-processing system used for sending letters and memos to individuals or groups. Though easy to use, *mail* can be difficult to install.

Site tailoring is not limited to changing system files; users can modify their interface to the system by specifying options in a *.mailrc* file placed in the top-level directory. See the *mail*(1) manual page for details. Lower-level tailoring can be controlled by a *.mailcf* file in the home directory, interpreted by *sendmail*(8) as a user-specific configuration file.

To send mail to one or more other people, type

% **mail** *username* [*usernames...*]

You are prompted to fill in a "subject" line, which should describe the contents of the memo. Press the carriage return (RETURN), then type the text of the letter. To end a letter, type a period (".") and (RETURN) at the beginning of a line, or use (CTRL-D).

You can send mail to *msgs* (described next). Mail to *msgs* is queued in the directory */usr/msgs* to be accessed by all system users when they invoke *msgs*. To send mail to msgs, enter

% **mail msgs**

If you type *mail* with no arguments, a list of mail you've received displays, for example

```
% mail
"/usr/spool/mail/shemp": 3 messages
  1 garza  Mon Sept 16 14:53  28/1070  "Please call Sharon at home"
  2 mcbroom  Tues Sept 17 16:07  17/1457  "Re: weekly summary"
  3 esmark  Tues Sept 17 16:10  47/1081  "art work is ready"
&
```

The ampersand (&) is the mail command prompt. Type a question mark (?) at the prompt to display a list of mail system commands. If you type **1**, **2**, or **3**, the corresponding message displays. To delete a mail message you've just read, type **d**. To delete any message on the list, type **d #** (**#** = number of the mail message).

To reply to mail you've read, type either **R** or an **r**. **R** sends a reply *only* to the originator of your mail; **r** sends a reply to the originator *and* all recipients of the message. End your response with a period typed alone at the beginning of a line or your end-of-file character (CTRL-D).

To exit the mail command, type *q*.

The *ConvexOS Primer* and the *mail*(1) manual page contain more information about use of the system. Also see the *mailaddr*(7) manual page for a description of the address conventions used to route mail across the ARPANET and UUCP networks.

8.2 *msgs* Utility

msgs is the utility used to broadcast and read system-wide messages. Messages are transmitted to the message system by mailing letters to the *msgs* mail alias. Mail aliases are located in */usr/lib/aliases*.

To read messages, type

```
% msgs
```

If a message is queued in */usr/msgs*, the headline displays. For example:

```
% msgs
 Message 958:
From mcbroom Tue Nov  8 14:43:32 1988
Subject: company meeting this afternoon
(3 lines) Display this message? (y)es, (n)o, (q)uit, (h)elp [y]:
```

Type **y** to read the message, **n** to skip over it, and **q** to quit *msgs*. Other responses and *msgs* options are described in the *msgs*(1) manual page.

8.3 Notesfile System

The notesfile system is a computer bulletin-board system used to coordinate group discussions, individual mail, automatic problem reporting, and error logging. For a complete description of the notesfile system, read "The Notesfile Reference Manual" in the *ConvexOS Tutorial Papers*.

You can use notesfiles as follows:

- Set up local notesfiles for your site. The notesfiles can be used for discussions or for problem reporting. You can also set up notesfiles for individual users, which is a convenient way to manage mail.
- Set up a phone link to the USENET network. USENET comprises hundreds of bulletin-board notesfiles transmitted across phone lines among several thousand computers.

Your CONVEX Technical Assistance Center representative can help you set up local notesfiles.

8.4 */etc/motd* File

The */etc/motd* file contains *messages of the day* for system-wide distribution. To add new messages, edit */etc/motd* or replace it with a new version. New messages display automatically as each user logs in. Users can read messages added after their last login by using *more* or *less* to look at the */etc/motd* file.

/etc/motd displays to a user only on the first login after */etc/motd* has changed.

8.5 *wall* Utility

The *wall* utility writes messages to all logged-in system users. Messages sent via *wall* are routed immediately, rather than queued. Messages sent with *wall* can warn users of pending system shutdowns, request logouts, etc. Use *wall* according to the following procedure:

1. Log in as the superuser (superuser accounts are described in Chapter 2).
2. Enter

   # wall
3. Type your message (possibly multiple lines), followed by RETURN.
4. Press CTRL-D.
5. Log out of the superuser account.

For example, to send a message asking users to remove files from a loaded disk, enter

```
# wall /mnt full. Please delete unneeded files.
CTRL-D
#
```

Refer to the *wall*(1) manual page for additional information.

8.6 *write/talk* Utilities

You can use *write* to copy lines from your terminal to the terminal of another user on your local host. *talk* does the same thing over the network.

To write to another user, enter

% **write** *username*

user is the login ID of the person to whom you wish to write.

To talk to a user on the network, but not on your local host, enter

% **talk** *username@host*

user is the login ID of the person to whom you want to talk. When you enter the command, the following message displays on the other user's terminal:

```
Message from yoursystem!yourname  yourttyname
```

When the message displays, the recipient writes back, using **write** or *talk* with the caller's name. The response establishes communication, which continues until one user sends an interrupt. Read the *talk*(1) and *write*(1) manual pages for additional information.

8.7 *contact* Utility

The *contact* utility is an interactive program for submitting a problem report to the CONVEX Technical Assistance Center (TAC). *contact* is the recommended way to report problems or bugs, as it makes machine-readable text available to the TAC. A sample *contact* session is shown in Appendix C. For more information about the *contact* utility, enter

```
% info contact
```

Chapter 9
Accounting

The accounting system collects data on users, groups, and activities for process terminations, connect time, tape use, printer use, and disk space. This chapter describes

- an overview of the accounting system
- types of data you can collect with the accounting system
- methods for summarizing and archiving accounting date
- system and library calls for accounting
- security considerations
- installation
- error messages of the *bill* accounting

9.1 Overview

The accounting system works with the ConvexOS accounting facilities to track user activities. You can determine, for example, how much disk space, line printer use, or connect time a user consumes; how CPU time is split between users and overhead; or the programs using the most CPU cycles. This information is helpful for planning system use and effectively managing system resources.

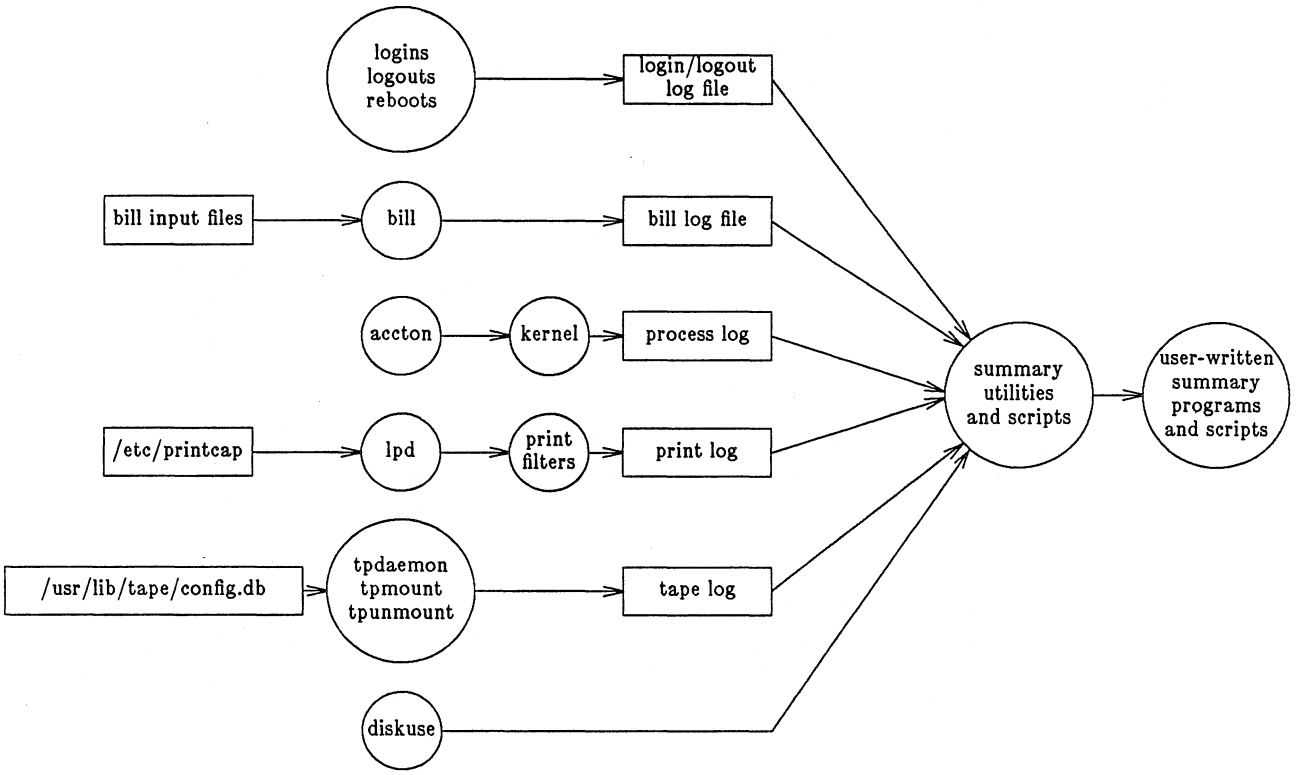
For government contractors and others who need strict accounting of project costs, users can bill resource use to specified projects. The accounting system then tracks CPU time, batch queue use, connect time, magnetic tape operations, and line printer use for each project.

Users of the accounting system belong to groups, and the tasks they do are called activities. An account is a group paired with an activity. A user can change accounts using the *bill* command.

For example, for in-house accounting, you can specify that user *garza* belongs to group *software* and does activities like *design*, *develop*, *document*, and *test*. You can also classify users by project.

Figure 9-1 is an overview of the accounting system and how the files and utilities work together.

Figure 9–1: Accounting System Overview



You must create files specifying user, group, and activity relationships. The accounting system is designed for tailoring your system to your site.

9.2 Using the *bill* Command

The *bill* command enables users to change billing accounts or determine their current billing account. *bill* displays the user's current account when it is executed without any arguments. At login, a user automatically begins billing to a default account that is the group specified in */etc/passwd*, with an activity ID of 0. It is a convention to name this activity something like "overhead" or "miscellaneous" because it is part of the default login account. Users can put a *bill* command in their *.login* files to select a different default account. You can (as system manager) put a *bill* command in the *.login* file of a user who is not cognizant of the accounting system or one who always bills to the same account.

Specifying one argument to the *bill* command changes the activity, leaving the group unchanged. For example, if user *watson* is billing to account *software design* and wants to change to *software document*, the command

```
% bill document
```

changes activity to *document*; the group remains *software*. You also change accounts by entering

a group and activity as arguments to *bill*. For example, a user writing a paper describing work with DNA issues the following:

```
% bill dna_discovery document
```

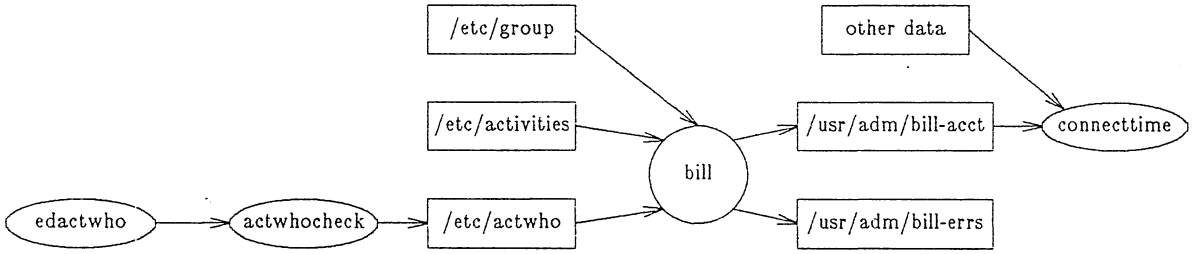
to change group to *dna_discovery* and activity to *document*.

When a process terminates, ConvexOS writes a log entry to the */usr/adm/acct* file. When a process forks a child process, the current billing account is passed to any child processes generated. The current billing account for a process is stored in the kernel *proc* structure, which is resident in memory.

You can also set up the accounting system so *bill* keeps a log of failed bill attempts. For example, if a user attempts to change accounts to a group and activity to which they do not have access, *bill* writes an ASCII record containing the current time, user's ID, and requested group ID, and activity ID to */usr/adm/bill-errs*. You must create an empty *bill-errs* file in the */usr/adm* directory; otherwise *bill* does not keep a log of failed attempts.

In addition to getting information from */etc/passwd*, *bill* gets information from the files */etc/activities*, */etc/actwho*, and */etc/group*, shown in Figure 9-2. As system manager, you must create these files and specify names and IDs for groups and activities. To speed *bill* access time, list entries in these files so that the most-often-used groups and activities are first. You can also write a utility that dynamically reorders these files based on current use trends as determined from */usr/adm/wtmp* and */usr/adm/bill-acct*.

Figure 9–2: *bill* Input and Output



9.2.1 */etc/group* File

Each group must have an ID, which is a positive number between 0 and 32767, specified in the */etc/group* file in the following format:

groupname:*:*groupID*: *username*, *username* ...

The */etc/group* file has general read permission and maps numerical group IDs to group names. The group ID is used for accounting and file protection. A user can belong to up to eight groups. Figure 9-3 shows a sample */etc/group* file.

Figure 9–3: Sample *letc/group* File

```
operator:*:5:root,sam
notes:*:13:*
games:*:20:*
uucp:*:40:*
docn:*:51:diane,thea,garza,adobe
diag:*:52:mcbroom,thompson
mgmt:*:57:bush,quayle
quality:*:59:barbara,jordan
```

9.2.1.1 */etc/activities*

The */etc/activities* file associates activity names with an activity ID that is a signed number between –2,147,483,648 and 2,147,483,647. The format of the entries is as follows:

activityname:activityID

Set up an activity 0 (zero) called, for example, "overhead" or "miscellaneous" to be used as a default activity. A sample */etc/activities* file is shown in Figure 9-4.

Figure 9–4: Sample */etc/activities* File

```
overhead:0
design:100
develop:200
integration:300
document:400
tooldev:500
test:600
```

9.2.1.2 */etc/actwho*

The */etc/actwho* file contains entries that link accounts with users. The *bill* command checks the */etc/actwho* file to determine whether a user is allowed to bill the account before changing the user's account. The wildcard character " * " can be used to specify group, activity, or user names. The wildcard character " ? " can be used when specifying group and activity names. The format of the entries is as follows:

groupname.activityname:username, username, ...

Figure 9-5 is a sample */etc/actwho* file.

Figure 9–5: Sample *letc/actwho* File

```
*.overhead:*
software.test:ginac,wikelius,noden
software.document:mcbroom,tuttle,greathouse,lutz
hardware.develop:geist,forsythe,gardner
project1.design:blakemore,ledger,daisy,cfarmer
project1.develop:mckaig,christiansen,streepy
project2.*:cash,brandel,johnson
project?.document:johnstone,ellis,gonzalez
```

This sample shows four groups: *software*, *hardware*, *project1*, and *project2*. The asterisk (*) in the first line specifies all groups, the asterisk in the 7th line specifies all activities, and "project?" in the last line specifies either *project1* or *project2*. The asterisk and question mark (?) can be used in any field of the */ect/actwho* file. This sample specifies four different activities. The "overhead" activity can be paired with any group for the default account and can be accessed by any user. Other activities are test, document, develop, and design.

The information in */etc/actwho* is used to keep track of accounting data. To modify the list of users who have access to the accounts, use *edactwho*, which invokes the editor (either the editor specified by your EDITOR environment variable or *vi*) on the*/etc/actwho* file. When editing is complete, *edactwho* uses *actwhocheck* to validate the modified file before replacing the old file.

9.2.1.3 */etc/accton*

The */etc/accton* command turns on the logging of accounting information. The following command tells the accounting system to append accounting information to the */usr/adm/acct* file:

```
# accton /usr/adm/acct
```

To deactivate logging, use *accton* without arguments, for example:

```
# accton
```

When disk space in the */usr/adm* directory exceeds 98% capacity, logging terminates automatically. When available disk space drops to 96%, use */etc/accton* to restart logging.

9.3 Types of Accounting Data

Most accounting data is collected in logs in the */usr/adm* directory. Types of data collected include

* process terminations
* login time
* successful executions of *bill*
* unsuccessful executions of *bill*
* tape allocations and deallocations
* tape error messages
* printer use
* printer use errors

Many log files contain time stamps, which are large integer values, obtained from *time*. Time stamps can be converted to a readable form using *idtoname*, which is explained in section 9.4, "Summarizing and Archiving Data."

9.3.1 Process Terminations

Process terminations are logged in */usr/adm/acct*, a binary data file. The *sa* and *lastcomm* utilities translate the information into ASCII text.

9.3.2 Connect Time

Connect time data comes from the files */usr/adm/wtmp* and */usr/adm/bill-acct*. The *connecttime* utility creates an ASCII file with data including when, on which device, and by whom logins, logouts, executions, and terminations of *bill* occur. The *awk* script *connecttime.awk* converts *connecttime* output to a format that can be processed by standard *awk* scripts.

9.3.3 Tape Use

The tape utilities *tpmount* and *tpunmount* write entries to */usr/adm/tp-acct*. Entries include the name of the tape drive and time of mounting/unmounting. Figure 9-6 is a sample */usr/adm/tp-acct* file.

Figure 9–6: Sample */usr/adm/tp-acct* File

```
allocated     507507897   /dev/rmt16   311   49   0   mt:0
deallocated   507507923                311   49   0   mt:0
allocated     507568225   /dev/rmt20     0   58   0   mt:1
deallocated   507569265                0   58   0   mt:1
allocated     507572068   /dev/rmt20     0   58   0   mt:1
deallocated   507573815                0   58   0   mt:1
```

From left to right, the columns specify

* whether tape drive was allocated or deallocated
* time of mounting or unmounting
* name of mounted drive
* user ID
* group ID
* activity ID
* name of the tape

The *awk* script *tape.awk* converts data in */usr/adm/tp-acct* to a format that can be processed by standard *awk* scripts.

9.3.4 Printer Use

When a user prints a job, the line printer daemon calls a printer filter that places an entry in */usr/adm/lpd-acct*. Figure 9-7 is a sample *lpd-acct* file.

Figure 9–7: Sample Printer Log

```
507536910    2   293    60     0   lp   mcbroom     muse
507536942    2   293   158   520   lp   garza       artemis
507537107    2     0     0     0   lp   root        convexe
507574106   16   268    55     0   lp   horwitz     arioch
507574193   17   268    55     0   lp   gonzalez    muse
507574613   14   268    55     0   lp   johnstone   muse
507623663    2     0     0     0   lp   root        convexe
507623675    3     0     0     0   lp   root        convexe
```

From left to right, the columns specify

* time
* pages printed
* user ID
* group ID
* activity ID
* device
* user name
* machine name

Columns 3 through 7 are determined by the machine from which the job was submitted.

pac and the general summary scripts summarize this data.

9.3.5 Disk Space

The *du* and *diskuse* utilities track disk space use. The following command returns the amount of disk space used by *my directory*:

% **du** *mydirectory*

With the *-a* option, *du* returns an entry for each file; the *-s* option returns the total for the directory.

The *diskuse* utility computes total disk space use in kilobytes for users and groups. Figure 9-8 shows sample output from *diskuse*.

Figure 9–8: Sample *diskuse* Output

```
:::::users
    731 root
  23666 horwitz
    349 garza
  12501 mcbroom
:::::groups
  53875 software
     23 documentation
```

To track disk space use, add an entry to the *daily, weekly,* or *monthly* shell scripts to run *diskuse* and the *diskmerge.awk* script periodically. *diskmerge.awk* merges *diskuse* output from different time periods to create a summary of disk use over time. Figure 9-9 is a sample *weekly* accounting

script to run *diskmerge.awk* weekly. See the section "Summarizing Disk Use" for more information on *diskmerge.awk*.

Figure 9–9: Sample *diskmerge.awk* Entry to Weekly Script

```
mv total oldtotal          # save old totals
diskuse -d / > thisweek    # compute current use for all directories in /

# add on to previous totals
cat oldtotal thisweek | awk -f /usr/adm/sumscripts/diskmerge.awk > total
```

9.4 Summarizing and Archiving Data

ConvexOS provides summary utilities, data-specific *awk* scripts, and general *awk* scripts to process accounting data. The summary utilities include

* *connecttime* for processing information from logins, logouts, and *bill* executions
* *lastcomm* and *sa* for summarizing process termination data
* *pac* for summarizing printer use data
* *du* for providing disk use totals for directory structures
* *diskuse* for providing disk use totals for users and groups

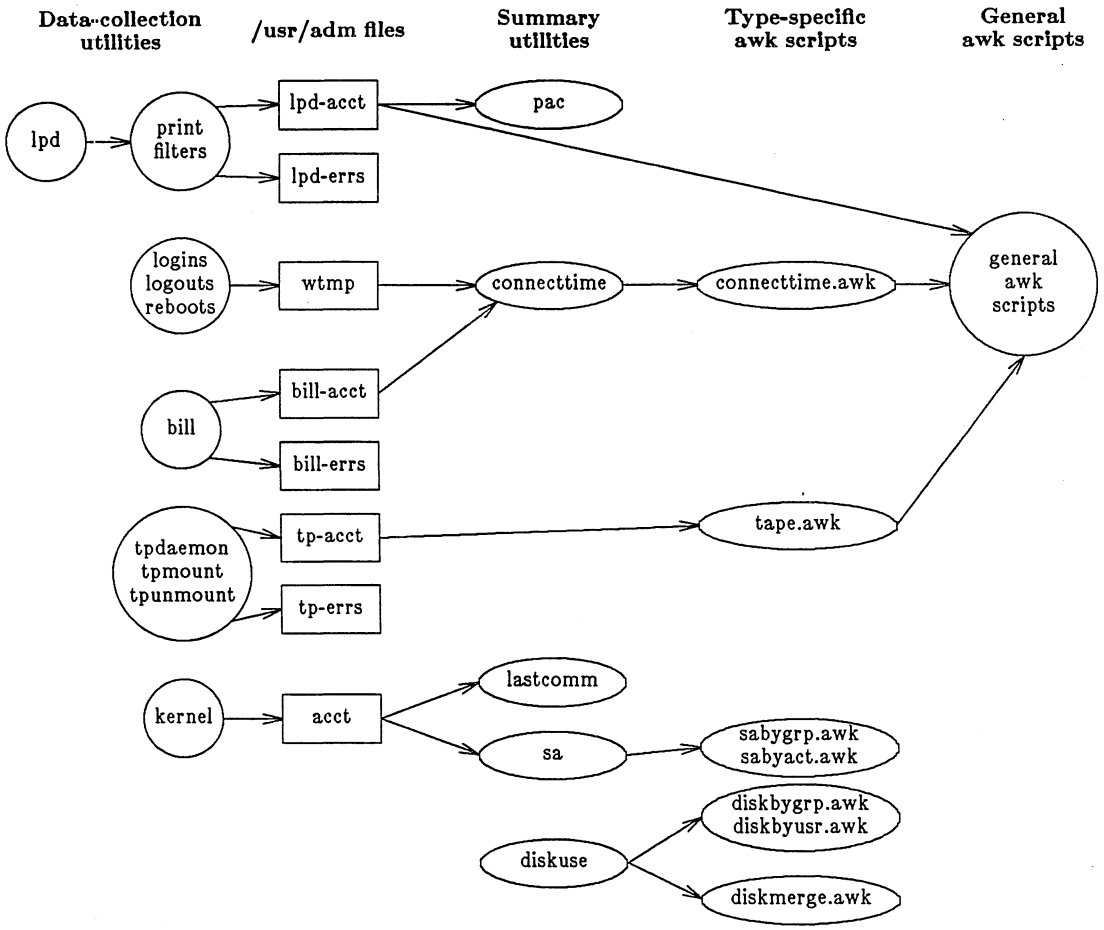
Data-specific *awk* scripts summarize specific types of accounting data. These include

* *connecttime.awk* for summarizing connect time data
* *tape.awk* for summarizing tape use data
* *sabygrp.awk* and *sabyact.awk* scripts for summarizing *sa* output
* *diskbygrp.awk* and *diskbyusr.awk* disk use summary scripts
* *diskmerge.awk*

Figure 9-10 shows the relationship of accounting files, their summary programs, and general summary *awk* scripts.

Four *awk* scripts that summarize by activity, group, user, or account reside in the */usr/adm/sumscripts* directory. Use of these generalized scripts is described toward the end of this section, along with use of the *idtoname* and *sort* utilities.

Figure 9–10: Accounting Data Collection and Summarization



9.4.1 Summarizing */usr/adm/acct* Information

The *sa* and *lastcomm* utilities run summaries of the */usr/adm/acct* file. Output from *sa* is saved in files, then run through *sabyact.awk* and *sabygrp.awk*, which summarize process termination data by activity. *sa* creates */usr/adm/acct* summaries, which are written to */usr/adm/savacct* and */usr/adm/usracct*, to summarize process accounting and user/group/activity accounting data. By default, *sa* reads information from the summary files. Use the to *-i* option to suppress the default. Use *-e* to echo records (in ASCII text) exactly as they appear in the accounting file. Use *-g* to summarize the information in */usr/adm/acct* by account. Figures 9-11 and 9-12 show the output from the following commands:

```
% /etc/sa -e /usr/adm/acct
```

and

```
% /etc/sa -g /usr/adm/acct
```

Figure 9–11: Sample *sa -e* Output

```
sh       514207800  0.05      __        0    0    0   0.74s
date     514207801  0.01   console      0    0    0   0.34s
echo     514207801  0.02   console      0    0    0   0.13s
echo     514207801  0.03   console      0    0    0   0.06s
sh*n     514207801  0.10   console      0    0    0   0.83s
sh       514207800  0.04      __        0    0    0   1.12s
oslog    514207801  0.92      __        0    0    0   1.87s
echo     514207802  0.03      __        0    0    0   0.17s
sh       514207800  0.06      __        0    0    0   2.25s
stty     514207805  0.03    ttyp0     263   51    0   0.16s
msgs     514207805  0.09    ttyp0     263   51    0   0.44s
tset     514207805  0.10    ttyp0     263   51    0   0.20s
biff     514207806  0.04    ttyp0     263   51    0   0.13s
csh*n    514207812  0.00    ttyp0     263   51    0   0.04s
clear    514207841  0.07    ttyp0     263   51    0   0.24s
csh      514207802  1.49    ttyp0     263   51    0  38.83s
stty     514208362  0.04    ttyp0     263   51    0   0.16s
msgs     514208362  0.08    ttyp0     263   51    0   0.36s
tset     514208362  0.09    ttyp0     263   51    0   0.15s
biff     514208363  0.04    ttyp0     263   51    0   0.09s
csh*n    514208376  0.00    ttyp0     263   51    0   0.17s
```

From left to right, the columns specify

* command executed
* starting time
* total CPU time used
* tty from which command was executed, or "_ _" for daemon
* user ID
* group ID
* activity ID
* elapsed time
* memory use (in kilobyte seconds)
* total disk I/O (total number of disk blocks read into or written from the buffer cache for execution of the command)
* system CPU time

The difference (subtraction) between system CPU time (column 11) and total CPU time (column 3) is user CPU time.

Figure 9–12: Sample *sa -g* Output

```
swtst      admin        26670    235.17cpu   282202tio
lp         runtimes     14015    162.55cpu   153842tio
staff      marv           682      9.16cpu     6777tio
staff      vc           49196    361.05cpu   234357tio
docn       misc           529     18.74cpu     9200tio
proj1      os              11      0.10cpu      264tio
swtst      fc_test      84015   2854.56cpu   934627tio
hardware   peripheral     120      1.11cpu     1383tio
```

From left to right, the columns in Figure 9-12 specify

* group name
* activity name
* commands executed
* total CPU time
* number of disk blocks read into or written from the buffer cache for execution of the command (total I/O operations)
* memory used, in kilobyte * seconds

Use the following commands to pipe *sa -g* output through *sa awk* scripts:

% **/etc/sa -g | awk -f /usr/adm/sumscripts/sabyact.awk**

% **/etc/sa -g | awk -f /usr/adm/sumscripts/sabygrp.awk.**

Figure 9-13 shows output from *sabyact.awk*. Figure 9-14 shows output from *sabygrp.awk*. The first column in each figure lists the activity name and group name.

Figure 9–13: Sample *sabyact.awk* Output

```
admin       26670    235.17cpu   282202tio
runtimes    14015    162.55cpu   153842tio
merv          682      9.16cpu     6777tio
vc          49196    361.05cpu   234357tio
misc          529     18.74cpu     9200tio
os             11      0.10cpu      264tio
fc_test     84015   2854.56cpu   934627tio
```

Figure 9–14: Sample *sabygrp.awk* Output

```
swtst      110685   3089.73cpu   1216829tio
lp          14015    162.55cpu    153842tio
staff       49878    370.21cpu    241134tio
docn          529     18.74cpu      9200tio
proj1          11      0.10cpu       264tio
hardware      120      1.11cpu      1383tio
```

lastcomm reads the process log file (*/usr/adm/acct*) and displays its contents in an ASCII format. Commands display in order of most recent to least recent, and information displayed includes

* command
* modes of operation, for example, "S" for a user in superuser mode (optional)
* user name
* tty line
* CPU time used
* starting date and time of execution

Figure 9-15 shows sample *lastcomm* output. Refer to the *lastcomm*(1) manual page for further information.

Figure 9–15: Sample *lastcomm* Output

```
cron       F   root     __       0.30 secs
uuxqt          uucp     ttyu0    0.47 secs
uucico     S   uucp     ttyu0    1.63 secs
telnet         smith    tty02    1.23 secs
sh             root     __       0.17 secs
ruptime        root     __       0.79 secs
date           root     __       0.44 secs
cron       F   root     __       0.82 secs
csh        S   jones    __       0.35 secs
rcp            jones    __       0.19 secs
sendmail   F   root     __       0.24 secs
                                  __
```

The 4.0 versions of *lastcomm* and *sa* are not backward compatible with pre-4.0 release data files from */usr/adm/acct*. Truncate */usr/adm/acct* before installing the 4.0 accounting system.

9.4.2 Summarizing Connect Time

connecttime uses log files generated by */usr/adm/bill-acct*) and */usr/adm/wtmp*) to collect the following data:

* whether the connection started or terminated
* starting or ending time
* tty line
* user ID
* group ID
* activity ID
* remote system

connecttime can also generate connect time information for a specific time. Figure 9-16 shows output from the following command:

% **/etc/connecttime**

Figure 9–16: Sample *connecttime* Output

```
started      513987521   console   125   159   17000
terminated   513987521   console
started      513986436   tty06     125   159   17000
terminated   513986436   tty06
started      513983248   tty01     268    55   26000
terminated   513983248   tty01
started      513982776   tty07     268    55   26000
terminated   513982776   tty07
started      513980757   tty04     268    55   26000
terminated   513980757   tty04
started      513980515   tty03     293    60   32000
terminated   513980515   tty03
started      513976196   tty02     268    55   26000
terminated   513976196   tty02
started      513976153   tty01     268    55   26000
terminated   513976153   tty01
started      513972879   tty07      84    55   13000
terminated   513972879   tty07
started      513972237   tty06     125   159   17000
terminated   513972237   tty06
terminated   513990093   ttyp3
terminated   513988387   ttyp2
started      513988309   ttyp2     293    60       0   (convexs-ex)
terminated   513988264   ttyp2
started      513988203   ttyp3       0    10       0   (convex-ex)
started      513987988   ttyp2     293    60       0   (convexs-ex)
terminated   513987983   ttyp2
started      513987900   ttyp2     293    60       0   (convexs-ex)
terminated   513987813   ttyp2
terminated   513987544   console
started      513987491   ttyp2     293    60       0   (convexs-ex)
started      513987491   console   125   159       0
terminated   513987239   console
```

The most recent entry appears first. The *awk* script *connecttime.awk* summarizes this data. The command

% /etc/connecttime | awk -f /usr/adm/sumscripts/connecttime.awk

produces a summary similar to Figure 9-17. From left to right, the sample shows

- starting time, measured in seconds elapsed since 00:00:00 GMT, January 1, 1970. See the *time*(3C) manual page.
- amount of time connected (in seconds)
- user ID (—1 if user is not in /etc/passwd)
- group ID (—1 if user is not in /etc/passwd)
- activity ID (—1 if user is not in /etc/passwd)
- tty line
- remote machine from which you are logged in

Figure 9–17: Sample *connecttime.awk* Output

```
513017497   13774    80   159      0   ttyp1    (convexs-ex)
513017451   13569   125   159  17000   tty07
513017447       4   125   159      0   tty07
513015204    1548    77    67      0   tty06
513015126    2961   199    56      0   ttyp0    (convex1-ex)
513013804     673    77    67      0   tty05
513013752     260   286    60      0   ttyp2    (convexs-ex)
513013382    2573    80   159      0   ttyp1    (convexs-ex)
513013097     637    77    67      0   tty04
513013030    1481   353    67      0   ttyp0    (convexs-ex)
513013027    1377   424   159  16000   tty03
513013025       2   424   159      0   tty03
513013000    3096   125   159  17000   tty02
513012997       3   125   159      0   tty02
513012917      39     0    10      0   ttyp0    (toto2-ex)
513010486      42   124    67      0   console
513010297      12    -1    -1     -1   ttyp1    (client4)
513009368    1163     0    10      0   ttyp0    (toto2-ex)
513008707     116   263    51      0   ttyp0    (convex-ex)
513007992    1164   424   159  16000   tty07
513007986       6   424   159      0   tty07
513007686    1012   263    51      0   ttyp0    (convex-ex)
513007256      11   406    53      0   ttyp0    (convexs-ex)
513006971    2184    77    67      0   tty06
```

9.4.3 Summarizing Tape Use

The *awk* script *tape.awk* script summarizes the data in */usr/adm/tp-acct*. The command

% **awk -f /usr/adm/sumscripts/tape.awk /usr/adm/tp-acct**

produces a summary similar to Figure 9-18.

Figure 9–18: Sample *tape.awk* Output

```
512763700     8   0   60   60100   unit0
512768993   100   0   60   60100   unit0
512770017     2   0   60   60100   unit1
```

From left to right, the columns specify

- starting time, measured in seconds elapsed since 00:00:00 GMT, January 1, 1970. See the *time*(3C) manual page.
- amount of time the tape drive was allocated (in seconds)
- user ID
- group ID
- activity ID
- logical name of tape drive

9.4.4 Summarizing Printer Use

The general summary scripts are the preferred method of summarizing printer use data. Use of the general summary scripts is described in section 9.4.6. The *pac*(8) manual page also summarizes the printer log, */usr/adm/lpd-acct*.

The following command produces a summary of */usr/adm/lpd-acct* similar to the output shown in Figure 9-19:

```
% /etc/pac
```

Figure 9–19: Sample *pac* Output

USER	PAGES	RUNS	COST($)
mcbroom	23	8	0.46
garza	11	2	0.22
esmark	11	2	0.22
horwitz	5	2	0.10
lindholm	12	4	0.24
davis	5	1	0.10
riley	253	34	5.06
hosack	306	38	6.12
davis	13	3	0.26
root	149	69	2.98
haight	8	4	0.16
berman	119	8	2.38
vonda	7	2	0.14
owen	31	4	0.62
thompson	80	7	1.60
murphy	4	1	0.08
matzner	47	3	0.94
TOTAL	1084	192	21.68

pac output is usually sorted alphabetically by user. Specifying the *-c* option sorts the output by cost. The *-r* option reverses the usual sorting order. The cost is primarily arbitrary. See the *pac*(8) manual page for information on changing the cost equation.

With the *-P* option you can specify a printer other than the default printer. The *-s* option automatically summarizes the accounting log and places the summary in */usr/adm/lpd-acct_sum*.

9.4.5 Summarizing Disk Use

You can summarize output from *diskuse* using the disk use *awk* scripts. *diskbygrp.awk* and *diskbyusr.awk* generate disk use totals by group and by user. There is not an *awk* script that summarizes disk use data by activity because *diskuse* does not collect data by activity. *diskmerge.awk* can be used to keep track of disk use over time by adding current use data to old data.

diskmerge.awk merges two versions of *diskuse* output to create totals for each user and group. To override directory protections, log in as superuser to run *diskuse*.

The following command produces output shown in Figure 9-20:

```
# /etc/diskuse | awk -f /usr/adm/sumscripts/diskbygrp.awk
```

Figure 9–20: Sample *diskbygrp.awk* Output

```
  44   zero
   8   mktg
 520   tac
  68   docn
```

The first column lists total disk space (in kilobytes) used by the group listed in the second column.

The following command produces output shown in Figure 9-21.

```
# /etc/diskuse | awk -f /usr/adm/sumscripts/diskbyusr.awk
```

Figure 9–21: Sample *diskbyusr.awk* Output

```
1739   root
   9   cleburne
23636  granbury
14276  tintop
  349  glenrose
 2093  tolar
 7944  cresson
 3638  godley
 2031  acton
   56  harmony
12925  kaufmann
    5  hico
```

The first column is disk space (in kilobytes); column two is user name.

The following command produces output shown Figure 9-22.

```
# /etc/diskuse | cat - diskuse.out.old | awk -f /usr/adm/sumscripts/diskmerge.awk
```

Figure 9–22: Sample *diskmerge.awk* Output

```
:::::users
     88 root
     96 mcbroom
     48 garza
    768 nancy
     16 esmark
     80 horwitz
     48 lindholm
:::::groups
     88 zero
     16 mktg
   1040 test
```

9.4.6 General Summary Scripts

The following *awk* scripts summarize accounting data:

* *genbyact.awk* (by activity)
* *genbygrp.awk* (by group)
* *genbygrpact.awk* (by group and activity pair)
* *genbyusr.awk* (by user)

The scripts can be copied and modified to do other types of summarizing and used used on connect time, printer, and tape accounting data. Special *awk* scripts summarize the */usr/adm/acct* file and *diskuse* output.

Syntax for general *awk* scripts is

% **awk -f genby{grp, act, grpact, usr}.awk** *general-format-logfile*

general-format-logfile can be one of the following outputs:

* */usr/adm/lpd-acct*
* *tape.awk*
* *connecttime.awk*

The *daily*, *weekly*, and *monthly* shell scripts in */usr/adm* create summaries on a time basis. ConvexOS can be set up to run the scripts automatically. The scripts summarize accounting data for users (CPU/memory use) and processes (CPU/memory use) and can be modified to meet your specific needs.

Accounting reports are generated by *cron*, which invokes a program called */usr/adm/accounting* every hour on the half hour. */usr/adm/accounting* program checks to see if it is time to do a daily, weekly, or monthly accounting report. If so, it runs the appropriate scripts (which contain print commands for hardcopy and also record their output in */usr/adm*.

The source for */usr/adm/accounting* is in */usr/adm/accounting.c*. You can create another version that behaves differently. For example, change */usr/adm/accounting.c* to consider a work week as Monday through Sunday instead of Sunday through Saturday.

The *sort* and *idtoname* utilities are helpful in summarizing accounting information. You can pipe output from general summary scripts through *sort* then through *idtoname* to get summarized, sorted, readable results. Figure 9-23 shows the output from running the *awk* script *genbygrpact.awk* on the accounting log file */usr/adm/lpd-acct*.

Figure 9–23: Sample *genbygrpact.awk* Output

```
  0      0    64.000
 55      0    29.000
159      0     4.000
159  10200     6.000
159  10600    44.000
159  17000    92.000
 58      0    13.000
159  29000   295.000
159  10000    17.000
 67      0     2.000
```

The first column contains the group ID, the second column is the activity ID, and the third column shows printer use. To get the output shown in Figure 9-24, pipe the output through *sort*,

as follows:

```
% awk -f /usr/adm/sumscripts/genbygrpact.awk /usr/adm/lpd-acct | sort +2nr
```

2 is the number of fields to skip; *n* is the type of data in the field being sorted (numeric); *r* specifies sorting in descending order. This command sorts output from *genbygrpact.awk* in reverse order based on the third field in each line (pages printed).

Figure 9–24: Sample *sort* Output

```
159   29000   295.000
159   17000    92.000
  0       0    64.000
159   10600    44.000
 55       0    29.000
159   10000    17.000
 58       0    13.000
159   10200     6.000
159       0     4.000
 67       0     2.000
```

To decipher the group and activity IDs, use *idtoname*, which takes an ID (group, activity, user, or timestamp) and converts it to group name, activity name, user name, or date. Add **idtoname -gae** to the previous command to pipe the output in Figure 9-24 through *idtoname*.

The *g* option of *idtoname* specifies that the first field is a group ID, the *a* specifies the second field is an activity ID, and the *e* tells *idtoname* to echo the third field. Figure 9-25 shows the output.

Figure 9–25: Sample *idtoname* Output

```
swtst      tool_dev   295.000
swtst      fc_test     92.000
zero       misc        64.000
hwdoc      misc        29.000
swdoc      admin       17.000
overhead   misc        13.000
swtst      misc         4.000
os         misc         2.000
```

9.4.7 Archiving Accounting Data

The *lpd-acct, tp-acct, acct, bill-acct,* and *wtmp* files are stored in */usr/adm*. If these log files do not exist in */usr/adm*, create them as zero-length files. CONVEX recommends writing a shell script that periodically saves and truncates these files. You can set up subdirectories in */usr/adm* for these and any other */usr/adm* files that are archived.

To change the name of the printer accounting file (for example, you have two printers and want one to use a different file for accounting), change the object of the *af* parameter in */etc/printcap*.

9.5 System and Library Calls

The system calls *pgetregid, psetregid, setaid, getaid,* and *getrusage* are helpful in managing the accounting system. You must be a superuser to execute *psetregid* and *setaid.*

getrusage returns information describing the resources used by the current process or its terminated child processes. *getrusage* reports resident memory sizes in units of 1/100 of a Mbyte/second.

Use *pgetregid* to determine the real and effective group ID for a process. *psetregid* sets the real and effective group ID for a process.

Use *setaid* to set the activity ID for a process. An activity ID can be any 32-bit value except –1. *getaid* returns the activity ID of a process.

"accounting'*getacwent*" The library calls *getactent, getacwent, getgrent,* and *getpwent* are helpful in managing the accounting system. These calls can be used by user programs that sort or summarize accounting files. *getacwent* returns a pointer to a structure containing the broken-out fields of a line in the group-activity (*/etc/actwho*) access control file. *getactent* returns a pointer to an object in the */etc/activities* file. *getgrent* does the same for the */etc/group* file, and *getpwent* does the same for the */etc/passwd* file.

9.6 Security Considerations

The *bill* command does a *setuid* to superuser to provide security for the */usr/adm/wtmp, /usr/adm/bill-acct,* and */usr/adm/bill-errs* files. The *bill* command checks to see if a user is permitted to *bill* an account before changing group and user IDs, so you can control user access to files with the group protection bits.

Sensitive files could be owned by a phantom user. Only the superuser could access the files or know of their existence.

9.7 Troubleshooting

The following list of *bill* error messages are listed in order of common occurrence:

usage: bill [[*group*] **activity**]
The *bill* command invocation is incorrect.

bill: billing denied
The user tried to bill to an account and did not have access to that account as specified in */etc/actwho.*

bill: *group name* **is not a group**
The user tried to bill to an account, and specified a group name that does not appear in */etc/group.*

bill: *activity name* **is not an activity**
The user tried to bill to an account with an activity name that does not appear in */etc/activities.*

bill: missing entry for activity ID # in /etc/activities
No entry for activity ID # in */etc/activities.*

bill: could not open /usr/adm/bill-acct
The */usr/adm/bill-acct* file cannot be read or does not exist.

bill: could not find user *user name* **in /etc/passwd**
A user's entry has been deleted from */etc/passwd* since the user logged in.

bill: could not find group ID *group ID* **in /etc/group**
The current group has been deleted from */etc/group* since the user logged in.

bill: no group with gid *group ID*
A user tried to bill to an account specifying only an activity name, and the entry for the user's current group has been deleted from */etc/group* since the user logged in.

bill: could not write /usr/adm/bill-acct
bill cannot write to the */usr/adm/bill-acct* file after it has been successfully opened.

bill: could not open /usr/adm/bill-errs
A user tried to bill to an account they did not have access to (as specified in */etc/actwho*) and */usr/adm/bill-errs* exists but cannot be opened.

bill: could not write /usr/adm/bill-errs
A user tried to bill to an account they could not access (as specified in */etc/actwho*) and */usr/adm/bill-errs* could not be written to.

Chapter 10
Resource Monitoring and Scheduling

This chapter describes the utilities that monitor resource use and control operation of system resources. Section 10.2 describes programs used to monitor CPU, disk, printer, and *uucp* use. Section 10.3 describes scheduling the use of the CPU, disks, and tapes.

10.1 Resource Monitoring

Resource monitoring tools provide you with information on system performance. With the set of tools provided with ConvexOS, you can monitor the use and performance of the CPU, disk and tape drives, printers, the *uucp* software, and the error-monitoring software.

10.1.1 Monitoring CPU Use

Use the following utilities to monitor CPU use:

- *uptime*
- *ruptime*
- *pstat*
- *vmstat*
- *syspic*
- *ps*

These utilities are described in the following sections and in the *ConvexOS Programmer's Reference*.

10.1.1.1 *uptime* Utility

The *uptime* utility lists the date, number of users, and load average of the system. The load averages represent the exponentially weighted average of the number of jobs waiting in the run queue over the last 1, 5, and 15 minutes. The following message, for example, tells you that the system has been up for 4 days, 11 hours, and 27 minutes; that 60 users are logged in; and that the load averages were 4.18, 4.40, and 4.27.

```
5:12pm  up 4 days, 11:27,  60 users,  load average: 4.18, 4.40, 4.27
```

A load average above 7 or 8 slows editing and other interactive work.

10.1.1.2 *ruptime* Utility

The *ruptime* utility lists the date, number of users, and load average for all machines on your network. *ruptime* provides the same information as *uptime* and in the same format, but provides the information for multiple machines. Response time for *ruptime* is less than that for *uptime*. Figure 10–1 is a sample *ruptime* output.

Figure 10–1: Sample *ruptime* Output

```
muse     up    4+02:24,  25   users, load  1.04, 1.01, 1.01
convexe  up      20:59,      0      users, load  0.02, 0.02,
convex   up    1+04:42,  13   users, load  2.02, 2.12, 2.09
convex1  up    3+11:44,  27   users, load  1.01, 1.01, 1.01
convex2  up       2:30,      10     users, load  1.08, 1.13,
convex3  up    1+04:28,   5   user,  load  0.01, 0.07, 0.15
convex4  up      18:46,      29     users, load  1.08, 1.03,
eclipse  up    2+07:33,   4   users, load  0.00, 0.00, 0.00
marv     up    1+05:26,   1   user,  load  0.00, 0.11, 0.16
sunspot  up    3+08:38,   0   users, load  0.07, 0.00, 0.00
toto2    down     9:06
```

10.1.1.3 *pstat* Utility

The *pstat* utility provides statistics on I/O operations and processes, specifically system tables (for example; *inode* and text tables). *pstat* is an effective, but complex, problem-solving tool. See the *pstat*(8) manual page in the *ConvexOS Programmer's Manual* for more information.

10.1.1.4 *vmstat* Utility

The *vmstat* utility displays statistics you can use to monitor system activity, including job distribution, virtual memory load, paging and swapping activity, and disk and CPU utilization. If you invoke *vmstat* without options, the system displays a short summary of virtual memory activity since the last system boot. For example

```
procs      memory                  page  faults  cpu
r b w   avm   fre  re at  pi  po  fr  de  sr  in  sy  cs vs us sy id
1 0 0  4992 24904   0 3   5   0   0   0   0 130 165  58  0 75 22  4
```

The fields displayed by *vmstat* are defined in Table 10–1.

Table 10-1: *vmstat* Field Definitions

Mnemonic	Definition
r	Number of processes in run queue
b	Number of processes blocked for resources (for example; I/O, paging)
w	Number of processes runnable or short sleeper (< 20 secs) but swapped
avm	Active virtual pages
fre	Number of pages on free list
re	Page reclaims (simulating reference bits)
at	Page reclaims from free pool
pi	Pages paged in
po	Pages paged out
fr	Pages freed per second
de	Anticipated short-term memory shortfall
sr	Pages scanned by clock algorithm, per second
in	Device interrupts per second
sy	System calls per second
cs	CPU context switch rate (switches per sec)
vs	Vector register switch rate (switches per sec)
us	User time for normal- and low-priority processes
sy	System time
id	CPU idle

When you run *vmstat*, you should find few blocked (*b*) jobs, and the user CPU utilization (*us*) should be greater than 60%. If the system is busy, however, the count of active jobs can be large (*r*), and several of these jobs can be blocked (*b*). The paging daemon is active whenever available free physical memory is low (when the paging daemon is active, the *sr* field displays a nonzero value).

To access system–generated statistics, enter

```
% vmstat 1
```

Figure 10–2 is a sample output.

Figure 10–2: Sample *vmstat* Output

```
procs      memory                    page  faults  cpu
r b w   avm    fre  re at  pi  po  fr  de  sr  in  sy  cs vs us sy id
2 0 0  4852  24904   0  3   5   0   0   0   0 130 165  58  0 75 22  4
2 0 0  4852  24904   0  1   0   0   0   0   0 122 200  27  0 99  1  0
2 0 0  4852  24904   0  0   0   0   0   0   0 118 163  24  0 98  2  0
2 0 0  4852  24904   0  0   0   0   0   0   0 116 135  23  0 97  3  0
1 0 0  5232  24852   0  0   0   0   0   0   0 113 115  21  0 98  2  0
1 0 0  5232  24852   0  0   0   0   0   0   0 110  95  19  0 98  2  0
1 0 0  5232  24852   0  0   0   0   0   0   0 108  79  18  0 98  2  0
```

In the statistics generated by *vmstat*, abnormal job distributions often signify system imbalances. If, for example, processes are blocked (*b*), the disk subsystem is overloaded or imbalanced. System overhead increases to 60–70% if devices or open teletype lines begin *ringing* (sending spurious interrupts) or if user programs do high-speed nonbuffered input/output. Use *syspic* to detect the *ringing* port; enter

```
% syspic -p tty -i 3
```

Other reasons for high system overhead include

- excessive context switching (*cs*)
- interrupt activity (*in*)
- system call activity (*sy*)

If the system is heavily loaded or has little memory available, the system can be forced to swap. If the output from *vmstat* shows the columns *pi* and *fr* at nonzero, paging is starting to occur. If *w* shows nonzero, the system is starting to swap, which significantly reduces system performance. Another sign of swapping is pauses when interactive jobs (for example, editors) swap out. To avoid swapping, consider administratively limiting system load if you expect to remain in a memory-poor environment.

10.1.1.5 *syspic* Utility

The *syspic* utility displays statistics on system processes, memory, paging, faults, network use, and the latest *uptime* statistics. It is an interactive monitor that reports system performance information in windows, periodically updating the statistics and highlighting numbers that indicate abnormal system loading. *syspic* is one of the best tools available for system monitoring. See the *syspic*(8) manual page for a complete description of this utility.

10.1.1.6 *ps* Utility

The *ps* utility gives basic information about running processes. Typically, *ps* is used to check the status of your own processes. Most options print user name(s), process IDs, and the time used by the processes. You can specify one of a large set of options, depending on the type of output you want to receive. For example, entering

```
% ps —u
```

prints the following output:

```
USER       PID    %CPU    %MEM     SZ    RSS   TT    STAT     TIME   COMMAND
mcbroom  15715     0.0     3.6     64    272   p0    I       10:58   rlogin muse
mcbroom  16716     0.0     4.4    112    336   p1    I       11:29   make
mcbroom  18086     0.0     0.3    484    120   d0    R        0:00   ps -u
```

This display shows that PID 18086 (the *ps -u* command) has consumed 0.0% of available CPU cycles and 0.3 of the real memory space available. The virtual size (SZ) of the process is 484 Kbytes; real size (RSS) is 120 Kbytes. The process is controlled by terminal (TT) d0, and it is runnable, as the *R* option shows. The process has used less than 1 second of CPU time.

To display a list of all user processes, enter

```
% ps —a
```

A typical output is shown in Figure 10–3.

Figure 10–3: Example *ps* Output

```
 PID TT STAT   TIME COMMAND
2719 co  I    0:00 /bin/csh 2daily.incr
3131 co  I    0:16 dump 5uGf /dev/rmt21 /usr
3542 co  D    0:24 dump 5uGf /dev/rmt21 /usr
4017 01  R    0:01 a.out
4007 04  S    0:00 /bin/csh compilemodule FM900 f -O0
4021 07  R    0:01 ps a
2324 08  S    0:13 vi back.c
3997 0f  S    0:00 mail
3909 13  I    0:00 vi getsysinfo.2
2439 15  I    0:00 .s
2440 15  I    0:00 sh -c csh
2441 15  I    0:08 -sh (csh)
3964 16  S    0:01 emacs 06.s
3565 19  I    0:00 mail
3951 1a  I    0:00 mail opstaff
3988 1a  I    0:00 /usr/ucb/vi /tmp/Re03951
2908 22  T    0:00 vi diff.out
2941 22  T    0:02 vi Makefile
3368 22  I    0:00 mcs update sys/iop_ons.c
3402 22  I    0:00 sh -c vi /tmp/Ugh003368
3404 22  S    0:07 vi /tmp/Ugh003368
3949 23  I    0:00 mail
```

With this command, you can determine the process ID of a job or see how the system is being used. *ps aux* tells you the percentages of CPU and memory each process is using; necessary information if the system is being used heavily or if you suspect the presence of a *runaway* process. If *ps* shows *getty* running with nonsense for arguments, you probably have an open, noisy tty line and are wasting CPU time responding to nonexistent interrupts. If this happens, turn off the offending port with the *off* command.

See the *ps*(1) manual page for more information on the options available.

10.1.2 Monitoring Disk Use

The *df* and *du* utilities monitor disk use. The *df* utility reports the amount of free disk space available on a specific file system or in the file system in which a specified file is contained. To check the amount of free disk space in the */dmaster* partition, for example, enter

```
% df /dmaster
```

Output is similar to the following:

```
File system          Kbytes    used    avail capacity  Mounted on
/dev/st6             163423   141277   5803    96%     /dmaster
```

Entries in the *used* and *available* columns are 10% less than the number shown in the *Kbytes* column. ConvexOS attempts to keep 10% of the disk unallocated to prevent disk fragmentation.

du summarizes disk use by printing the number of kilobytes contained in all files and, recursively, the number of files or directories within each specified directory or filename. For example, entering

```
% du
```

might list the following output for the directory from which the command is executed:

```
106     ./tmp
82      ./adm
52      ./reports
46      ./bin
18      ./projects/test
14      ./projects/notes
150     ./projects/management
66      ./mail
534     .
```

The *df*(1) and *du*(1) manual pages contain more information on these utilities. Section 9.4.5 describes tracking and summarizing disk use.

10.1.3 Monitoring Printer Use

Utilities for monitoring printer use include

- *lpq*
- *lpd*
- *lpc*

These utilities are described in the following sections and in the *ConvexOS Programmer's Manual*. Section 9.4.4 describes tracking and summarizing printer use.

10.1.3.1 *lpq* Utility

The *lpq* utility reports the status of files sent to the line printer. Entering

```
% lpq
```

reports the status of jobs in the default print queue, for example:

```
Rank    Owner   Job  Files                    Total Size
active  mcbroom 276  (standard input)         12189 bytes
1st     horwitz 199  (standard input)         32904 bytes
2nd     garza   200  (standard input)         60963 bytes
```

Options are available to generate reports on particular jobs, jobs sent by particular users, or jobs sent to printers other than the default printer. In each case, *lpq* reports the user's name, the job's rank in the queue, the job identifier, the names of files constituting the job, and the total size of the job in bytes.

The *—P* option checks the status of jobs sent to different printers. Type the printer alias (for example, *lpq —Pip* for laser printer queue, *lpq —Plp* for line printer queue) directly after the *—P* option). The printer aliases must be the same ones used in */etc/printcap*. Use *lprm* to remove individual jobs from the print queue. See the *lpq*(1) manual page for more information.

10.1.3.2 *lpd* Utility

lpd, the line printer spooler, does the following:

- processes requests to print queued files
- transfers files to the spooling area
- displays the queue
- removes files from the queue

Although *lpd* is usually transparent, understanding how *lpd* spooling works in ConvexOS is important for making repairs if the printer hangs or if another problem arises.

Files sent to a printer, or to *uucp*, *notes*, or *mail* are first queued in the */usr/spool* directory. The files are queued in a dedicated subdirectory (for example, files sent to the line printer are queued in the subdirectory */usr/spool/lpd*; files sent to *notes* are queued in the subdirectory */usr/spool/notes*). A program (called a *daemon*) starts up and processes the files in the queue. *lpd*, for example, starts when *lpr* is issued. Other daemons — *uucp*, *at* — are started at regular intervals by the *cron* program.

To watch *lpd* work, look in */usr/spool* for a file called *lpd.lock* and for the *lpd* subdirectory. *lpd* contains the *.seq*, *lock*, and *status* files. As the superuser, you can use *ll* to display files in the directory; otherwise use *ll -a*. The *.seq* file contains the process ID (PID) of the next job to be queued. The *lock* file contains two lines: the PID of the daemon assigned to the line printer and the job ID of the job currently printing (or the last one printed). The *status* file contains a line describing the status of the printer (for example, active and printing, needs paper, etc.). This line is written into *status* by a daemon that communicates with the line printer device driver. *lpq* reads *status* to determine the status of the line printer queue.

lpd.lock is a lock file used by the master printer spooler, which coordinates all printer requests and allocates jobs among printers attached to the system. *lpd.lock* contains the PID of *lpd*, so the master spooler can start it on request.

Use the following procedure to restart a "hung" printer.

1. Use *cd* to change directories to */usr/spool/<device>pd*. For example, for line printers, enter

   # cd /usr/spool/lpd

2. **Use** *cat* to view the lock file associated with the device.

   # cat lock

3. Determine the name of the file in the queue and copy it. This step is optional; if you do not copy the file, you will lose the job in the queue.

4. Use *kill -9* to kill the job with the PID found in the lock file. For example, if the PID is 591, enter

   # kill -9 591

5. To restart *lpc*, enter

   # /etc/lpc restart *queuename*

   queuename is the name of a printer queue (for example, *ip* or *lp*).

10.1.3.3 *lpc* Utility

lpc controls operation of the line printer system, as follows; More specifically,

- disables or enables printers
- disables or enables a printer spooling queue
- rearranges jobs in a spooling queue
- determines the status of printers, their associated spooling queues and daemons

Use the following procedure to stop and restart a spooling daemon:

```
# /etc/lpc
lpc > abort lp
lpc > start lp
lpc > quit
```

If *lpc* is executed without arguments, it returns a prompt; *lpc* cannot be used to control a particular printer unless an entry exists for that printer in */etc/printcap*. See the *lpc*(8) manual page for more information. Some *lpc* commands can only be used by the superuser.

10.1.4 Monitoring *uucp* Use

The following utilities track *uucp* traffic:

- *uusnap*
- *uuclean*
- *uulook*

These utilities are described in the following sections and in the *ConvexOS Programmer's Manual*. Also see the *uucp*(1) manual page for information on *uucp*.

10.1.4.1 *uusnap* Utility

The *uusnap* utility generates a table showing the status of pending *uucp* activity. To use *uusnap*, enter

```
% uusnap
```

Typical output is shown in Figure 10–4.

Figure 10–4: Example *uusnap* Output

```
smu      1 Cmd    ---   ---     TALKING
allegra  1 Cmd    ---   ---
ctvax    1 Cmd    ---   ---     DIAL FAILED  Retry time 53 mins
convex    ---     ---   ---     TIMEOUT  Retry time reached
cua1      ---     ---   ---     LOCKED
infoswx   ---     ---   ---     LOCKED
```

The columns describe (from left to right)

- system (or resource) name
- number of outgoing commands
- number of incoming or outgoing data files
- number of commands queued for local execution
- status message

The LOCKED status shown in column 5 means the resource is currently being used and unavailable for acquisition by another process. The status message can include the time remaining before *uucp* can retry the call and the number of times *uucp* tried unsuccessfully to reach the site. *cua* units can also be locked by processes running *tip* and *cu*.

10.1.4.2 *uuclean* Utility

The *uuclean* utility removes old files from the */usr/spool/uucp* subdirectories. Files that are *cleaned out* are usually the result of interrupted *uucp* queuing or transmission procedures. *uuclean*, started by *cron*, runs a series of scripts that delete all files older than a specified number of hours. Figure 10–5 is a typical script run by *cron*.

Figure 10–5: Example *uuclean* Commands

```
/usr/lib/uucp/uuclean -d/usr/spool/uucp/D.convex -n169 -pD.convex
/usr/lib/uucp/uuclean -d/usr/spool/uucp/C. -n169 -pC.
/usr/lib/uucp/uuclean -d/usr/spool/uucp/D. -n169 -pD.
/usr/lib/uucp/uuclean -d/usr/spool/uucp/D.convexX -n169 -pD.convexX
/usr/lib/uucp/uuclean -d/usr/spool/uucp/TM. -n169 -pTM.
/usr/lib/uucp/uuclean -d/usr/spool/uucp/X. -n169 -pX.
```

Each line in the script begins with an invocation of *uuclean*. The second field in each line is the name of the subdirectory to be cleaned. The third field tells *uuclean* to delete files older than the number of hours specified (in each case here, files older than 169 hours are deleted). The last field tells *uuclean* to look for files whose prefix matches the specified string.

You can invoke a number of options with *uuclean*. See the *uuclean*(8) manual page for details.

10.1.4.3 *uulook* Utility

The *uulook* utility, which is the *uucp* traffic log printer, displays the end of the *uucp* log and then monitors it. New entries are printed on arrival. *uulook* displays new entries until you exit by typing the interrupt character (usually CTRL-C).

To use *uulook*, enter

```
% uulook
```

Typical output is shown in Figure 10–6.

Figure 10–6: Example *uulook* Output

```
uucp infoswx (2/15-12:14-29856) SUCCEEDED (call to infoswx )
uucp infoswx (2/15-12:14-29856) OK (startup)
uucp infoswx (2/15-12:14-29856) OK (conversation complete)
uucp mostek1 (2/15-12:19-29957) FAILED (LOGIN)
uucp mostek1 (2/15-12:19-29957) FAILED (call to mostek1 )
CVuucp convexs (2/15-12:29-29285) OK (startup)
CVuucp convexs (2/15-12:29-29285) OK (conversation complete)
doe rice (2/15-12:33-689) XQT QUE'D (rmail rtm@harvard.ARPA )
uucp rice (2/15-12:33-700) LOCKED (call to rice )
uucp rice (2/15-12:33-664) SUCCEEDED (call to rice )
uucp rice (2/15-12:34-664) OK (startup)
doe rice (2/15-12:34-664) REQUEST (S D.convexBNEL2 D.convexBNEL2 doe)
doe rice (2/15-12:34-664) REQUEST (S D.convexXNEL0 X.convexXNEL0 doe)
doe rice (2/15-12:34-664) OK (conversation complete)
uucp rice (2/15-12:41-980) FAILED (call to rice )
uucp rice (2/15-12:41-985) SUCCEEDED (call to rice )
uucp rice (2/15-12:41-985) OK (startup)
doe rice (2/15-12:41-985) REQUEST (S D.convexBNEj2 D.convexBNEj2 doe)
doe rice (2/15-12:42-985) REQUEST (S D.convexXNEj0 X.convexXNEj0 doe)
doe rice (2/15-12:42-985) OK (conversation complete)
```

From left to right, the entries area

- name of the user calling *uucp*
- name of the system called
- time and date of the call and the associated process ID
- call status
- status detail (S = send)

10.1.4.4 Error Monitoring

uucp errors are logged in the following files:

- */usr/spool/uucp/ERRLOG* contains entries that describe major protocol errors. *ERRLOG* is usually empty.
- */usr/spool/uucp/LOGFILE*
- */usr/spool/uucp/SYSLOG* logs data received and transmitted, for example

```
notes convex (2/12-10:12) (477072730) received data 71 bytes 3 secs
notes convex (2/12-19:33) (477106388) received data 433 bytes 5 secs
```

Statistics programs summarize the *uucp* error log files, but the error messages can usually be safely ignored.

10.2 Resource Control

With the resource control (or scheduling) tools, you can change run or queue priorities or issue instructions to peripheral devices. ConvexOS has tools for scheduling jobs, changing execution priorities, ensuring disk consistency, and copying data from tapes.

10.2.1 CPU Scheduling

Use the following utilities for job scheduling and job execution:

- *nice*
- *renice*
- *at*
- *cron*

These utilities are described in the following sections and in the *ConvexOS Programmer's Manual.*

10.2.1.1 *nice* Utility

The *nice* utility executes programs at nonstandard priority. *nice* accepts a numerical argument that specifies the priority. The standard priority is zero (0). High numbers mean lower priorities. You can lower your priority by using increasingly *higher* numbers (up to 64).

Use *nice* for large jobs or jobs that are not urgent, as follows:

```
% nice +3 make &
```

Lowering the priority to 3 or 4 makes jobs run much slower. Jobs with priority set at 64 run only in idle time. Only the superuser can increase the priority of a job.

10.2.1.2 *renice* Utility

The *renice* utility changes the priority of a running process. Jobs with negative values run at higher priority. For example, to increase the standard priority of a job, enter

```
# renice -4 process_id
```

The *process_id* argument is a process identification number obtained as output from *ps*. A job with priority set to –10 takes a larger share of available CPU time. Using very negative priorities can make a process run at such a high priority that it cannot be killed by another process.

10.2.1.3 *at* Utility

The *at* utility executes programs at a specified future time (specified in standard or military time) and day. You can use *at* interactively or create a file containing the commands to execute; use *chmod* to make the file executable. Then enter *at*, specifying the dates and times you want the job run. For example, to run a big program at 3:00 a.m., enter

```
% at 3am foo.big
```

To run the same program Friday at 3:00 a.m., enter

```
% at 3am fr foo.big
```

Refer to the *at*(1) manual page for information on more options.

10.2.1.4 *cron* Utility

The *cron* utility executes commands at user-specified dates and times in the future. Commands

in the *.crontab* file are executed automatically and can run once or several times at regular intervals. For example, you can use *cron* to

- truncate growing files on a regular basis
- search for and delete out-of-date messages
- remove editor checkpoint files
- print system statistics
- start up *uucp* for regular calls

To use *cron*, specify event pairs of the form (*time, command*), where *time* is the specified execution time and *command* is the utility to be executed. The *time* field has five parts:

- minutes
- hours
- days of the month
- months of the year
- days of the week

Each subfield in *time* can be a single entry or a list of entries.

cron is available to all users. Each user can specify a list of events by creating a *.crontab* file in their home directory. The root directory *crontrab* file is */usr/lib/crontab*.

If you are running *nfs* on your system, with home directories linked to a remotely mounted file system, the *.crontab* files are the same for all machines sharing that file system. The *.crontab* file is executed on each machine that shares that file system. To avoid this, add a test for hostname in the *.crontab* file so the instruction is executed only on the selected machine.

An example of how a user might specify a set of events is shown in the following *.crontab* file:

```
0 1    * * *     find /lp/aloha/.mh -name ",*" -exec rm {}
0 9-17 * * 1-5   msgs -f
```

On the first line, the *find* command is run at minute 0 of hour 1 (1 a.m.) every day, specified by the asterisk (*) in the *days* fields. On the second line, *msgs* is run at minute 0 of hours 9 through 17 (9 a.m. until 5 p.m.), Monday through Friday, every week of the month.

To process user events, the *cron* daemon scans the event pairs from a *.crontab* file. If the current time matches the time specified for a command in the *.crontab* file (the correct minute, hour, day of the month, etc.), the command is executed. More than one event can be specified for any *time*.

cron keeps a chronological list of event pairs, with the next event to become current is at the head of the list.

The event list is updated once an hour; use *tellcron* to manually update event list entries.

Use *tellcron* to update the event list before the scheduled update. To change the time a command runs, update *.crontab* and use *tellcron* to effect that change immediately. If you do not use *tellcron*, the changes are effected at the next hourly update.

Other features of *cron* include

- **Error messages**
  cron produces a variety of messages about syntax errors, file-access errors, and illegal use errors. Messages are logged in */usr/adm/cron.log*. Commands are not executed if they generate an error message.

- **Input and Output**
  cron events can read input directly from the *.crontab* event list file. The output from executed commands is directed to *dev/null* unless redirected.

- **Environments**
  By creating a *.cronrc* file in their home directory, users can customize the environment for commands (for example, specify an alternate SHELL or PATH run by *cron*).

10.2.2 Disk Control Utilities

Retrieving a word from disk is much slower than retrieving one from memory. Disk retrieval is often slow enough to limit the overall system performance. With ConvexOS, the problem of disk access is dealt with by using *disk caches*, which are segments of memory that remember blocks recently read or written. These blocks are stored in a disk cache because often a block stored in the cache is a block needed to complete the processing of the next instruction or set of instructions. In these cases, disk access and seek time are eliminated, which decreases the time time needed to execute the task.

One drawback to their use is that the disk state is partitioned between the disk and disk caches. Information that should be on the disk is in memory in a disk cache. For the system to work effectively, the contents of the disk caches must be frequently copied to disk. The *sync* utility copies the contents of the disk cache to the disk, forcing the disk to be "consistent." The *update* utility runs *sync* a few times a minute.

If the system crashes in the interval between *syncs*, the set of disks are, to varying degrees, inconsistent. For this reason, you must run *fsck* (file-system check) as part of each system startup. The *preen* utility runs *fsck* in parallel over normally mounted disks. Appendix C contains instructions for using *fsck*.

10.2.2.1 Quota System for Disk Use

Disk space is a limited resource. The disk-quota system provides a mechanism to control use of disk space by users. Quotas can be set for each user on any or all file systems. The quota system warns users when they exceed their allotted limit but allows extra space for current work. A user who remains over quota longer than a specified time-limit gets a fatal over-quota condition.

The quota system is an optional part of the operating system that can be included when the system is configured.

10.2.2.2 User's View of Disk Quotas

The *quota* command provides information to a user about disk quotas when the system administrator has imposed quotas for that user; it also reports on current use by a user. Two limitations can be imposed on a user; a quota can be set for the amount of disk space a user can fill, or a limit can be imposed on the number of files (*inodes*) they can own. Usually, both limitations are imposed.

The disk-quota system has soft limits and hard limits. The soft limit is the number of 1K blocks (or files) the user is not expected to exceed. Each time a user exceeds the soft limit, the system warns them and sets a time limit. If the user is exceeding quota when time limit expires, the soft limit is treated as if the hard limit has been reached, and no more resources are allocated to the user. The only way to reset this condition is to reduce use below quota. The system administrator can set the over-quota time limits for each file system restricted by quotas.

The hard limit cannot be exceeded; when use reaches the hard limit, requests for space or attempts to create a file fail with an EDQUOT error. The first time this occurs, a message is written to the user terminal. Only one message is written each time the hard limit is reached.

If you are modifying a file owned by root or another user, you do not see the EDQUOT error message. You cannot extend the file past the limit, regardless, and the hard-quota time-limit countdown begins for the owner of the file. Space occupied must be reduced below the limit.

Users modifying a file owned by someone else are subject to the quota limit of the owner of that file. This applies to users extending files owned by root, so the system manager must decide whether to enforce soft and hard quotas for the root partition. Setting the root quota limit to zero allows all users with write permissions to root files to write until the partition is filled, regardless of personal quota limits. Setting root quota limits to some values sets a maximum number of blocks that can be assigned to root on the partition. This allows users with write permission to root files to write only to the maximum number of blocks allocated to root on the partition. In this case, only users logged in as root can exceed the quota limit.

10.2.2.3 Reaching the Quota Limit

To recover from exceeding a hard-quota limit, abort the process in progress on the file system that has reached its limit, remove enough files to bring the limit back below quota, and restart the program.

If the process in progress is an editor, the write attempt that generated the *quota exceeded* message probably truncated the file in the editor. Aborting the editor means losing recent changes as well as some or all of the file. Use one of the following methods to avoid losing the changes:

- In Emacs, use *csh* to suspend the editor session while you remove files. For example, from the editor, press CTRL-Z to return to *csh* to remove files and get below the quota limit. To continue the editor session, enter **fg** (foreground) to the shell prompt to restart the suspended editor. (Different editors have different commands to suspend an editing session — refer to the editor documentation.

- Write the file to another file system (for example, to a file on */tmp*) where the quota has not been exceeded. Correct the quota problem and move the file back to the file system where it belongs.

10.2.2.4 Implementation Information

Disk-quota use and limits are stored in the file *quotas* on the root of the file system where the quotas are imposed. Do not change the file name; several user-level utilities depend on it.

The data in *quotas* is an array of structures, indexed by *uid*, with one structure for each user on the system (regardless of whether the user has a quota on this file system). If *uid* space is sparse, the file can contain holes that would be lost by copying. Therefore, avoid copying the file.

The *quotactl* system call informs the system of the existence of the *quotas* file. Each time a file is

opened, it is paired with its quota information. This information is usually retained in core for one of the following reasons:

- the user who owns the file is running a process
- other files owned by the same user are open
- a file was recently accessed

If the information is not in memory, the quota file is read to retrieve it.

The quota system is informed when a block is allocated or released and when an *inode* is allocated or freed. The quota system can object to any allocation.

The quota code uses a small percentage of the system cpu time consumed in writing a new block to disk.

10.2.2.5 Setting Up the Quota System

Use the following procedure to set up the disk-quota system:

1. Determine which file systems require quotas. Usually, only file systems containing users' home directories or other user files need quotas. You can, however, include */usr*. If possible, */tmp* should not have a quota.

2. Plan the disk-quota system before you begin the allocation

3. Allocate available space according to your plan.

4. The *edquota* command sets quota limits for specific users, singly or several at a time. The *-p* option duplicates the quotas of a prototypical user for a list of users, which is useful for giving multiple users the same limits. The *-t* option edits the over-quota time-limit for specific users. You can impose any combination of hard and soft limits for each user. A limit set to zero is disabled; disabling all limits for a user disables the entire quota for that user. See the *edquota*(8) manual page. You can also use *edquota* as a non-interactive command, for example

   /usr/etc/edquota -b soft -B hard -i soft -I hard *filenames usernames*

5. Before turning quotas on, check the quotas file with the command

   % **/usr/etc/quotacheck -a**

   When that line is present, all file systems in */etc/fstab* marked read-write with quotas have their quotas turned on, for example

   % **/dev/da1e /usr/labserver 4.2 rw,quota 1 3**

6. After quotas are set and ready to operate, use *quotaon* enforce quotas set with *edquota*. *quotaon* enables disk quotas for one or more file systems; the specified file systems must be mounted at the time. To turn on quotas at boot time, include the following line in the */etc/rc.local* file:

   /usr/etc/quotaon —a

   For *nfs*, add *quotaon* to */etc/rc.local* after */usr/etc/quotacheck —a* on the *nfs* server machine.

The quota system is invoked at boot time as illustrated here. See the *quotaon*(8) and *fstab*(5) manual pages for details.

The *quotaoff* command disables quotas; see the *quotaon*(8) manual page. When a file system is about to be unmounted by *umount*, the *umount* system call disables quotas just before unmounting the specified file system; this guarantees that the quota system is not disabled if the unmount fails because the file system is busy.

10.2.2.6 Quota System Administration

All quota administration commands must be run on mounted file systems. The commands work when the quota system is disabled.

The superuser can use *quota* to examine the use and quotas for a specific user and *repquota* to check all users on a file system.

The system administrator can increase or decrease a user's quota limits. A user who exceeds these limits can ask the system administrator to increase them.

When you remove a user from a system, use *edquota* to set the removed user's limits to zero. Then, when the user's login is removed, there is no entry for it in the *quotas* file. See section 6.7, "Deleting User Accounts."

10.2.2.7 Using Quotas With *nfs*

Quotas on remotely mounted file systems work much the same as quotas on locally mounted file systems. The difference is that no warnings are printed when the user goes over the soft limit. *nfs* clients should use *quota* to find out the state of their quota allocation. Hard-limit errors are treated like other hard errors in the *nfs* — for example, exceeding disk capacity. When you exceed the hard limit, you get return code errors, not system error messages. You can see *EDQUOT* error return codes from the system calls *write*, *fsync*, and *close*. Check these return codes to avoid disaster. For more information on *nfs*, see the *Network File System System Manager's Guide*.

10.2.2.8 Tape Archiving Utilities

Several utilities move data to or from magnetic tape. You can use *dump* and *restore*, described in Chapter 11, or the following:

tar — copies directory structures and files to tape. Many options exist; refer to the *tar*(1) manual page for more information.

dd — reads some "foreign" tapes (for example, tapes from IBM or DEC). The *dd* utility converts blocking information, translates EBCDIC to ASCII, and converts variable-length records to fixed-length records.

Chapter 11
Performing Backups

Data stored on disks can be lost due to hardware or software problems with the disk, damage to equipment, or accidental deletion of files by a user. Making backup tapes of the information on the disk on a regular basis ensures against loss of data. It also allows you to recover files previously deleted in standard "clean-up" operations and to recover earlier versions of files.

The backup process copies files from disks to backup tapes (also known as dump tapes). The backup utilities used to create these tapes are *dump*, *rdump* (remote dump over Ethernet), and *xdump*. These utilities copy a complete or partial file system to tape. Individual files or an entire file system can be extracted from these tapes with the *restore* or *rrestore* (remote restore over Ethernet) utilities.

The backup utilities support two types of backups (dumps)

- A full dump (level 0) backs up an entire file system
- Incremental dumps (levels 1–9) back up all files changed since the last full dump or since the last incremental dump of any lower level, whichever is most recent

Design the backup schedule for your site carefully, taking into consideration the amount of time required to perform a backup, the time when backups can be performed, and the cost of losing the data on your file systems.

The remainder of this chapter describes

- Designing backup schedules for your site
- Creating backup tapes using *dump*, *xdump*, and *rdump*
- Sample backup scripts
- Restoring files from backup tapes using *restore* and *rrestore*

11.1 Backup Schedules

CONVEX recommends a backup of all file systems each working day. For example, assuming a moderate amount (for example, twenty percent) of the data in a file system changes each week, a full backup should be performed each week and an incremental backup each working day. If a file system has only small, infrequent changes (for example, the root file system on many systems), it does not require a full backup each week, but still requires daily incremental backups.

The backup utilities require the user to assign a level (0–9) to each file-system backup.

A level–0 backup specifies a full dump of an entire file system. Level 1–9 backups are incremental dumps. An incremental dump backs up all files changed since the previous dump of a lower level. For example, a level–3 dump backs up all files changed since the most recent dump of level 0, 1, or 2. A dump of level *n* does *not* back up files changed since the last dump of level *n*, but backs up files changed since the last dump of any level lower than *n*. For example, if you perform a full dump (level–0) followed by level–5 incremental dumps on each of the two following days, the second level–5 dump is an almost perfect superset of the first level–5 dump, because both incremental dumps back up all files changed since the full dump. (The second level–5 dump may not be a perfect superset of the first, because files deleted after the first level–5 dump are not copied by the second level–5 dump.) The specific level of an incremental dump (1–9) has no meaning except in relation to the level chosen for a previous incremental dump.

When designing a backup schedule for your system, consider the following:

- sequence of full and incremental dumps in the backup schedule
- distribution of file-system backups (full dumps) throughout the week

11.1.1 Backup Sequence for a File System

For each file system at your site, establish a sequence of full and incremental dumps on a weekly cycle. Because backing up and restoring a file system is a time-consuming process, the sequence you select has an impact on the amount of time required to perform backups and restore files from backup tapes.

The following example illustrate the principle of scheduling full and incremental dumps for a single file system:

11.1.1.1 Recommended Backup Schedule Example

Use the same level of incremental dump each time. The following schedule is for a full dump performed weekly and incremental dumps on intervening business days (that is, four incremental dumps each week):

Monday	level 0 (full dump): backs up all files in the file system
Tuesday	level 5: backs up all files changed since Monday's full dump
Wednesday	level 5: backs up all files changed since Monday's full dump
Thursday	level 5: backs up all files changed since Monday's full dump
Friday	level 5: backs up all files changed since Monday's full dump

Restoring files requires two sets of dump tapes: tapes for the full dump plus tapes for the most recent incremental dump. Although the time required to perform incremental dumps can be slightly longer than the time required in the first example, the time required to restore files is significantly less.

11.1.2 */etc/dumpdates* File

If you specify the *u* key when you invoke a dump utility (see section 11.2.1), it records the following information in the */etc/dumpdates* file:

- name of the file system dumped
- level of the dump
- date of the dump

For each file system, */etc/dumpdates* keeps only one entry for each level of dump. An example of an */etc/dumpdates* file for three file systems is illustrated in Figure 11–1.

Figure 11–1: */etc/dumpdates* File

```
/dev/rda0a      0 Tue Apr 19 19:16:34 1988
/dev/rda0d      0 Tue Apr 19 18:53:19 1988
/dev/rda0d      5 Tue Apr 19 19:43:35 1988
/dev/rda0e      0 Thu Apr 14 17:24:21 1988
/dev/rda0e      5 Tue Apr 19 19:39:21 1988
```

Because an incremental dump of level *n* backs up all files that have changed since the last date recorded in the */etc/dumpdates* file for the previous dump of any level less than *n*, the *u* key should always be used for regularly scheduled backups. Do not edit the */etc/dumpdates* file: changing the information in this file alters the backup schedule.

11.1.3 Scheduling File-System Backups

Schedules of full and incremental dumps for several file systems can be designed to equally distribute the time spent taking dumps throughout the backup cycle (typically a weekly cycle).

For example, each day you might perform full dumps of a few file systems and incremental dumps of all file systems, scheduling the full dumps so all file systems are backed up once each week. Assuming a system with six file systems

- */dev/rda0a*
- */dev/rst3*
- */dev/rst2*
- */dev/rst1*
- */dev/rda2d*
- */dev/rda2e*

the schedule is as follows:

Monday	full dumps of */dev/rda0a* and */dev/rst3*; incremental dumps of all file systems
Tuesday	full dump of */dev/rst2*; incremental dumps of all file systems
Wednesday	full dump of */dev/rst1*; incremental dumps of all file systems
Thursday	full dump of */dev/rda2d*; incremental dumps of all file systems
Friday	full dump of */dev/rda2e*; incremental dumps of all file systems

This schedule runs incremental dumps on *all* file systems each day, including incremental dumps on those file systems completely dumped on the same day. This schedule provides a uniform load on the person taking the dumps. Conditions at your site may require you to distribute the backup schedule differently.

The example dump scripts in section 11.3 are designed for this type of backup schedule.

11.1.4 Archiving Backup Tapes

In the schedule presented in section 11.1.3, as many as 20 backup tapes might be required to complete the full dumps and incremental dumps for all file systems, depending on the size of the dumps. To ensure data integrity, these dump tapes must be saved for a period of time so backup tapes are available for more than one week past.

For example, you might archive backup tapes according to the following schedule.

- Use three sets of backup tapes in a cycle to ensure backups for the present week and two weeks past.
- At the end of each month, place one set of tapes in the archive for an entire quarter.
- At the end of each quarter, place one set of tapes in the archive for an entire year.
- At the end of each year, place one set of tapes in a permanent archive.

When creating a set of dump tapes, record at least the following information on the label:

- date of the dump

- file system(s) dumped
- operator who performed the dump
- *inode* numbers of the files on the tape; when the dump utilities execute, they report the first *inode* number for each tape
- sequence number of tape, if more than one tape is used for a dump

11.2 Backup Utilities: *dump*, *xdump*, and *rdump*

ConvexOS has the following utilities for dumping a file system to tape:

- *dump*
- *xdump*
- *rdump*

dump and *xdump* are almost identical in function, although *xdump* runs from two to 10 times faster than *dump* by using shared memory, asynchronous I/O, and faster disk-reading algorithms. With *xdump*, files copied to tape might be corrupted if the file system is in use while *xdump* is running. However, running an incremental dump on a file system immediately after a full dump (as shown in section 11.1.3 and the example scripts in section 11.3) reduces this risk.

Unlike *dump*, *xdump* usually aborts if either a tape write or a disk read error occurs. Tape write errors are rare, and aborting on a disk read error actually prevents *xdump* from producing a tape that cannot be restored.

rdump is part of the ConvexOS Networking Utilities and is an optional product. It is a version of the *dump* utility, which allows dumps over Ethernet. To use *rdump*, use the *f* key with an argument (see section 11.2.1 for an explanation of this switch and an example of its use).

Note	In the remainder of this section, reference to the *dump* utility also applies to *rdump* and *xdump*.

The dump utilities examine directories within a file system to select the files to be dumped. If a system is active (for example; running jobs, timesharing, or batch), files can change while *dump* is executing and the files copied to tape incorrectly. The files can then be restored with bad data and without an error reported. To ensure accurate backups, run the system in single-user mode during backups. Even when using *xdump*, backups can take up to four hours a day on a system with many disks and few sites can afford that much downtime. If a file is incorrectly copied to tape because the file is being changed when *dump* is running, taking an incremental dump immediately after the full dump saves the file (unless it is being changed again). The chance of completely losing a file is slight.

11.2.1 Invoking *dump*

The syntax for the command invoking the *dump* utility is

/etc/dump *key*[. . .] [*argument* [. . .]] *filesystem*

key specifies options to the *dump* utility. If more than one key is specified, they can appear in any order.

argument is a specification associated with a key. The arguments must appear in the same order as their associated keys.

filesystem is the name of the raw disk partition holding the file system being backed up (for example, */dev/rda0a* for the root file system).

The *xdump* and *rdump* utilities are invoked in the same way, except */etc/xdump* or */etc/rdump* is specified instead of */etc/dump*.

0–9 dump level — level *n* backs up all files changed since the last date recorded in the file */etc/dumpdates* for the previous dump of any level less than *n*. The default is level 0 (full dump).

b (blocks) — specifies the blocking factor (named in *argument*) *dump* uses. This key takes a decimal argument. The default is 10. The *G* key sets the blocking factor to 100 (recommended).

f (file) — specifies the device (named in *argument*) receiving the output of the *dump* program. Because the default is */dev/rmt8* (a 1600 bpi device), use this key or the *G* key to specify a 6250 bpi device. The following example takes a full dump of the / (root) file system:

```
# /etc/dump 0f /dev/rmt16 /dev/rda0a
```

When using *rdump*, use the *f* key to specify the system to which the dump is sent, as follows:

```
# /etc/rdump machine:device filesystem
```

For example

```
# /etc/rdump 0f convex:/dev/rmt16 /dev/rda0a
```

convex is the machine name of the system to which the dump is sent.

n (notify) — notifies all users in the group *operator* in the */etc/group* file when *dump* needs attention. The following example takes a full dump of the / (root) file system and notifies all operators when *dump* requires attention:

```
# /etc/dump 0nf /dev/rmt16 /dev/rda0a
```

s (size) — specifies the size (named in *argument*) of the dump tape in feet. *dump* assumes the dump tape is 2300 feet long. When this size or the size specified by the *s* key is reached, *dump* requests that a new tape be mounted. If a tape is shorter than the assumed length, and the dump is longer than the tape, a tape error occurs because the *dump* program does *not* use the end-of-tape marker. This can be a problem when combining several dumps on one tape.

u (update) — writes the date of the beginning of the dump in the file */etc/dumpdates* if the dump completes successfully. Because this file is used to track dates and levels of dumps for each file system, this key should always be used for scheduled backups. The following example takes a full dump of the / (root) file system and records the date in the */etc/dumpdates* file:

```
# /etc/dump 0fu /dev/rmt16 /dev/rda0a
```

G (Giant) — specifies a 6250 bpi device with a blocking factor of 100 (*/dev/rmt16)* to receive the output of the *dump* utility. This key is recommended for all uses of *dump* and *xdump* and is used in all examples in this section. The following example takes a full dump of the / (root) file system to a 6250 bpi device with a blocking factor of 100:

```
# /etc/dump 0nGfu /dev/rmt16 /dev/rda0a
```

In this example, */dev/rmt16* is implied by the *G* key. To avoid errors, specify the device name explicitly.

Other keys are described in the manual pages for *dump*(8), *rdump*(8), and *xdump*(8).

11.2.2 *dump* Operation

The *dump* utility requires operator intervention for

- end of tape
- end of dump
- error conditions; for example: tape write error, tape open error, disk read error

When intervention is required, *dump* prompts the operator for information in a form requiring a "yes" or "no" response.

dump prints messages to the system console as it executes. If you specify the *n* key when you invoke *dump*, all users in the group *operator* in the */etc/group* file are notified. These messages include estimates of the number of blocks to write, the number of tapes required, the time to completion, and the time to tape change. *dump* also reports the first *inode* number on each tape in a multi-tape dump. Record these numbers on the tape label with other information retrieving the correct tape to restore files.

11.2.3 Combining Dumps on a Single Tape

Normally, *dump* copies files to a tape, exits, and the mounted tape rewinds. The procedure can easily require a dozen tapes for incremental dumps, even on a small system. At sites using a systematic dump schedule, incremental dumps are not large. Combining several incremental dumps on a tape saves operator time by reducing the number of tape mounts and tapes required to complete incremental dumps.

If you are combining several dumps on the same tape, specify a non-rewinding tape drive as the output device. For example, to copy an incremental dump of the / (root) file system onto a non-rewinding tape drive, enter

```
# dump 5Guf /dev/rmt20 /dev/rda0a
```

When the dump completes, the tape is positioned for the next dump.

On final dump on a tape, include the *n* key to notify operators to change tapes and specify the rewinding version of the tape drive; for example:

```
# dump 5Guf /dev/rmt16 /dev/rda0a
```

If your combined dumps exceed the length of a single tape, the *dump* program aborts when the end-of-tape (EOT) marker is sensed. If you know how much tape remains at the beginning of each incremental dump, you could specify this to the dump program using the *s* switch. Unfortunately, previous executions of the *dump* program do not give you this information. If you run past the end of a tape on a well-planned set of incremental dumps, do a full dump on the file system that had more changes than you expected.

11.3 Sample Backup Scripts

The scripts listed in Figure 11–2 are sample scripts designed to equally distribute the time spent taking dumps throughout the backup cycle, in this case, a weekly cycle. The sample scripts assume a system with six file systems: */dev/rda0a*, */dev/rst3*, */dev/rst2*, */dev/rst1*, */dev/rda2d*, and */dev/rda2e*. Each day, a full dump is taken on one or two file systems, and an incremental dump is taken of all file systems.

Figure 11–2: Backup Schedule: Sample Scripts

```
Name          Script

monday         # fulls on /dev/rda0a, /dev/rst3
              /etc/dump 0nGfu /dev/rmt16 /dev/rda0a
              /etc/dump 0nGfu /dev/rmt16 /dev/rst3
              daily.incr

tuesday        # fulls on /dev/rst2
              /etc/dump 0nGfu /dev/rmt16 /dev/rst2
              daily.incr

wednesday      # fulls on /dev/rst1
              /etc/dump 0nGfu /dev/rmt16 /dev/rst1
              daily.incr

thursday       # fulls on /dev/rda2d
              /etc/dump 0nGfu /dev/rmt16 /dev/rda2d
              daily.incr

friday         # fulls on /dev/rda2d
              /etc/dump 0nGfu /dev/rmt16 /dev/rda2e
              daily.incr

daily.incr     # Two tapes, three filesystems/tape.  Note devices and 'n' flags:
              # tape 1
              /etc/dump 5Gfu /dev/rmt20 /dev/rda0a
              /etc/dump 5Gfu /dev/rmt20 /dev/rst3
              /etc/dump 5nGfu /dev/rmt16 /dev/rst2
              # tape 2
              /etc/dump 5Gfu /dev/rmt20 /dev/rst1
              /etc/dump 5Gfu /dev/rmt20 /dev/rda2d
              /etc/dump 5nGfu /dev/rmt16 /dev/rda2e
```

11.4 Restoring Files

Use following ConvexOS utilities for restoring files from tape:

- *restore*
- *rrestore*

rrestore is part of the ConvexOS Networking Utilities and is an optional product. It is a version of the *restore* utility for restoring over Ethernet. To use *rrestore*, use the *f* key with an argument (see section 11.4.1 for an explanation of this switch and an example of its use).

> **Note** In the remainder of this section, reference to the *restore* utility also applies to *rrestore*.

restore and *rrestore* copy entire file systems, directories, or selected files from a tape to disk and place them in the location from which they were dumped or in a different location for verification

before moving them to their original location.

The *restore* utility places restored files in the current working directory of *restore*. Unlike *dump*, the directories should be mounted when moving files from tape to disk.

11.4.1 Invoking *restore*

The syntax for the command to invoke *restore* is similar to *dump* and *xdump*:

/etc/restore *key*[. . .] [*argument* [. . .]] [*file* [. . .]]

key specifies options to the *restore* utility. If more than one key is specified, they can appear in any order.

argument specification associated with a key. The arguments must appear in the same order as their associated keys.

file name of the file or directory (set of files) to be restored.

The *rrestore* utility is invoked in the same way, except */etc/rrestore* is specified instead of */etc/restore*.

f (file) — specifies the input device or file (named in *argument*) to restore. When using *rrestore*, use the *f* key to specify the remote system from which the restore is received by using the format

 # /etc/rrestore f *machine:device*

For example:

 # /etc/rrestore f convex:/dev/rmt16

convex is the machine name of the remote system.

i (interactive) allows interactive restoration of files from a dump tape. *restore* reads in directory information from the tape and provides a shell-like interface that allows the user to move around the directory tree, selecting files to be restored. Commands available in interactive mode are listed in section 11.4.2.

r (restore) causes *restore* to restore a full dump tape onto a clear file system or to restore an incremental dump tape after restoring a full dump tape. This requires creation of an empty file system matching the file system dumped in size and scope before restoring the tape. This key is rarely used; do not attempt to restore with this key unless advised to do so by the Technical Assistance Center.

s (skip) causes *restore* to skip to a specified file on the tape before doing the restore. The format for this command is

 # /etc/restore s *n*

n is the number of the file on the tape. Tape files are numbered starting from 1 for the first file.

t (titles) causes *restore* to list the specified filenames (named in *argument*) if they are on the tape. If no filenames are specified, *restore* lists the entire contents of a tape. The following example is used to list the filenames on a dump tape (for a multi-tape dump, mount the first tape of the dump):

 # restore Gt

v (verbose) causes *restore* to print the name of each file it restores, along with the file length and change dates, to the screen. The following example causes *restore* to list the entire contents of a tape and print additional information about each file (for a multi-tape dump, mount the first tape of the dump):

```
# restore Gvt
```

x (extract) causes *restore* to extract the named files from the tape. If no files are named, the entire tape is restored.

G (Giant) specifies a 6250-bpi device with a blocking factor of 100 (*/dev/rmt16)* as the device from which the file are restored. This key is recommended for all uses of *dump* and *xdump* and, therefore, *restore.*

Other keys are described in the manual pages for *restore*(8) and *rrestore*(8).

11.4.2 Using *restore*

The *dump* program stores files on the tape with full pathnames in which the name of the mounted file system is removed. For example, dumps on the */mnt* file system store the filename */mnt/smith/progs/a.c* as *./smith/progs/a.c.* More complex mount-point names can cause some confusion. If a file system is mounted on */usr/external,* the dump of file */usr/external/smith/mailbox* stores the name *./smith/mailbox.*

To restore a file to its original location, you can

- Execute *restore* in the directory that is the mount point from which the original files were dumped
- Execute *restore* in an alternate location (for example, */tmp*) to verify files before they overwrite contents of the original directory

Restore a set of files by specifying the filenames (or directory name) on the command line that invokes *restore* as in the following example.

```
# cd /mnt
# /etc/restore Gx ./smith/file1 ./smith/file2 ./smith/file3
```

Full pathnames (of the form *./directory/filename*) are required. When you invoke *restore,* it asks you to mount the last tape in the series of dump tapes so *restore* can search for *file1, file2,* and *file3.* To determine the *inode* numbers contained in *file1, file2,* and *file3,* enter

```
# /etc/restore Gt
```

With this information, you can mount only tapes with the required *inode* numbers. Mount the highest-numbered tape first and continue in order.

If you dump many incrementals on one tape, you might have to skip the first dump file on the tape. For example, you can skip two incremental dump file to read the third dump file on */dev/rmt17*

```
# mt -f /dev/rmt17 fsf 2
```

You can also skip files on a tape using the *s* key to a restore utility. The following example skips to the third file on the tape:

```
# /etc/restore Gts 3
```

When *restore* is complete, the following prompt displays on the screen:

```
restore: Change mode of . [yn]?
```

This prompt asks the operator if *restore* should change the file protections of the working directory in which *restore* was invoked. Because the *restore* utility restored only the contents of this directory and not the directory itself, the answer is **no**. Answering **yes** alters high-level file-system protections and prevent users from accessing files they own.

11.4.3 Using *restore* Interactively

If the *i* key is specified on the command line to invoke *restore*, *restore* reads directory information from the tape and provides a shell-like interface that allows the user to move around the directory tree, selecting files to be restored.

add [*arg*] Adds the current directory or specified directory to the list of files to be extracted. If a directory is specified, then it and all of its descendants are added to the extraction list (unless the *h* key is specified on the command line). File names on the extraction list start with an asterisk (*) when listed by *ls*.

cd arg Changes the current working directory to the specified argument.

delete [*arg*] Deletes the current directory or specified directory from the list of files to be extracted. If a directory is specified, then it and all of its descendants are deleted from the extraction list (unless the *h* key is specified on the command line). The most expedient way to extract most of the files from a directory is to add the directory to the extraction list and then delete unnecessary files.

extract Extracts files on the extraction list from the dump tape. *restore* asks which volume to mount. The fastest way to extract a few files is to start with the last volume and work towards the first volume.

help Lists a summary of available commands.

ls [*arg*] Lists the current or specified directory. Entries in directories are appended with a /. Entries marked for extraction start with an asterisk (*). If the *v* (verbose) key is set, the *inode* number of each entry is listed.

pwd Prints the full pathname of the current working directory.

quit Exits *restore* immediately, even if the extraction list is not empty.

verbose Toggles the sense of the *v* (verbose) key. When set, the *v* key causes the *ls* command to list the *inode* numbers of all entries and causes *restore* to print information about each file as it is extracted.

Figure 11–3 illustrates an interactive *restore* session.

Figure 11–3: Sample Session: Interactive Use of *restore*

```
# restore iGf /tmp/a
restore > ls
 .history   core    .  dev/      etc/      lib/      vmunix
restore > cd etc
restore > ls
 Opasswd        mtab          remote        ttys
 group          passwd        rmtab         utmp
 motd           passwd.o.chfn   syslog.pid
restore > pwd
/etc
restore > add passwd
restore > add remote
restore > ls
 Opasswd        mtab         *remote        ttys
 group         *passwd        rmtab         utmp
 motd           passwd.o.chfn   syslog.pid
restore > delete remote
restore > ls
 Opasswd        mtab          remote        ttys
 group         *passwd        rmtab         utmp
 motd           passwd.o.chfn   syslog.pid
restore > extract
restore > extract
You have not read any tapes yet.
Unless you know which volume your file(s) are on you should start
with the last volume and work towards the first.
Specify next volume #: 1
set owner/mode for '.'? [yn] n
restore > quit
```

Chapter 12
Crash Dumps

When a computer system crashes, record the state of the system at the time of the crash and send the information to the CONVEX Technical Assistance Center (TAC) where TAC personnel can determine the cause of the crash. The system's state information, called a *crash dump*, is saved on tape using the *crashdump* utility.

crashdump is executed from the SPU of the system that is down. A crash dump can be sent to a tape drive on the local machine or over an Ethernet connection to a remote machine.

Call the TAC before executing *crashdump*. Some crashes can only be effectively analyzed if special procedures are followed; TAC representatives can inform you of procedures that may be required and can give you information to prevent future crashes.

Caution	If you do not take a crash dump before rebooting the system that crashed, you cannot use the crashdump to determine the cause of the crash or to reduce the risk of future crashes from the same problem.

12.1 Crash Dump Methods

There are three ways to take a crash dump, each described in a separate section of this chapter. The method you use depends on the configuration of your system.

- If your system has a local tape drive, send the crash dump to the local tape drive (section 12.3).
- If your system does not have a local tape drive but has an mt-format (SPU) cartridge tape drive, send the crash dump to the cartridge drive (section 12.4).
- If your system does not have a local tape drive or an mt-format cartridge tape drive but is connected to another CONVEX system over Ethernet using Multibus Ethernet controllers, send the crash dump to a tape drive on the remote system (section 12.5).

Crash dumps cannot be sent over Ethernet using VMEbus Ethernet controllers.

12.2 Restarting a Crash Dump

Most crash dump errors occur before *crashdump* begins writing to tape; for example, remote system down or tape drive not online. These errors do not terminate *crashdump*. The *crashdump* utility reports the error and supplies a prompt to which you respond when the error is corrected and you are ready to continue.

If an error occurs after *crashdump* starts writing to tape, *crashdump* terminates. If you restart *crashdump* after it aborts during execution; you will lose valuable data.

12.3 Crash Dump to a Local Tape Drive

If your system has a local tape drive, send the crash dump to the local tape drive, which takes approximately 25 minutes per 160 megabytes. Complete the following steps:

1. Mount a 2400-ft, 6250-bpi magnetic tape on the tape drive.

2. Put the tape drive online.

3. Change to the */mnt/os* directory on the SPU; enter

   (spu)> **cd /mnt/os**

4. End execution of all programs other than the standard shell program run by SPU UNIX. Enter the following command at the 1SPU prompt on the system console:

   (spu)> **osclean**
   (spu)> **cpureg >> /mnt/errlog**

5. To execute *crashdump*, enter

   (spu)> **crashdump**

   a. You are prompted to enter a comment to be included with the crash dump tape(s). Enter the machine name, the condition of the machine at crash time, the last time the machine crashed, and anything else that might be helpful to the TAC.

   b. If you are dumping more than 128 megabytes of physical memory, you are prompted when it is necessary to mount a new tape.

   After the crash dump begins, status messages display on the SPU console to tell you how the crash dump is progressing.

6. When the crash dump is complete, label the tapes and send them to the TAC for evaluation. Use the label to ensure that the tapes are easily identifiable when they arrive at the TAC; record the problem report number and, if more than one tape was required, the number of the tape.

   Before packaging the tape to return to the TAC, use a strap, foam block, or similar device to hold the loose end of the tape in place. This reduces the risk of damage to the tape during shipment.

7. When the crash dump is complete, reboot by typing

   (spu)> **/etc/reboot**

   Rebooting the SPU ensures that lingering processes are cleaned up.

12.4 Crash Dump to a Cartridge Tape Drive

If your system does not have a local tape drive, but does have an mt-format (SPU) cartridge tape drive, send the crash dump to the cartridge drive. Complete the following steps:

1. Place a cartridge tape in the tape drive.

2. Ensure that the cartridge tape drive is online.

3. Change to the */mnt/os* directory on the SPU by entering

   (spu)> **cd /mnt/os**
   (spu)> **mt rew**

4. End execution of all programs other than the standard shell program run by SPU UNIX by entering the following command at the SPU prompt on the system console:

   (spu)> **osclean**
   (spu)> **cpureg >> /mnt/errlog**

5. Execute *crashdump*

   (spu)> **crashdump -s**

   a. You are prompted to enter a comment that will be included with the crash dump tape(s). Enter the machine name, the condition of the machine at crash time, the last time the machine crashed, and anything else that might be helpful to the TAC.

   b. If you are dumping more than 128 megabytes of physical memory, more than one tape is required. *crashdump* prompts you to mount a new tape.

   After the crash dump begins, status messages display on the SPU console to tell you how the crash dump is progressing.

6. When the crash dump is complete, label the tapes and send them to the TAC for evaluation. Use the label to ensure that the tapes are easily identifiable when they arrive at the TAC; record the problem report number and, if more than one tape was required, the number of the tape.

   Before packaging the tape to return to the TAC, use a strap, foam block, or similar device to hold the loose end of the tape in place. This reduces the risk of damage to the tape during shipment.

7. When the crash dump is complete, reboot by typing

   (spu)> **/etc/reboot**

   Rebooting the SPU ensures that lingering processes are cleaned up.

12.5 Crash Dump Over Ethernet

If your system does not have a local tape drive or an mt-format (SPU) cartridge tape drive, but is connected to another CONVEX system over Ethernet using Multibus Ethernet controllers, send the crash dump over the network to a remote tape drive. If you have only VMEbus Ethernet controllers, you cannot take a crash dump over Ethernet.

Note	If the crash dump requires more than one tape, do *not* wait more than an hour between mounting tapes. When a crash dump is sent to a remote system, the crash dump is received on that system by the *in.crashreceive* program. If an hour passes in which *in.crashreceive* does not receive a packet, the program writes an error message to the log file */usr/adm/crashreceivelog* and terminates.

Complete the following steps to perform the crash dump:

1. The system with the local tape drive (that is, the system receiving the crash dump) must be on the same physical network as the system that is down. You *cannot* send a crash dump through an intermediate gateway machine.

2. Ensure that the system receiving the crash dump is running ConvexOS Networking Utilities.

3. Add the following line to the */etc/services* file of the receiving system if it is not already present:

   crashdump 670/udp

4. Add the following line to the */etc/servers* file of the system receiving the crash dump if it is not already present:

   crashdump udp /usr/etc/in.crashreceive

5. If you want to use *tpalloc* on the system receiving the crash dump, do so before you mount the tape, and invoke *tpdealloc* when the crash dump is finished.

6. Be sure the tape drive of the system receiving the crash dump is online.

7. Mount a 2400-foot, 6250-bpi magnetic tape on the machine receiving the crash dump.

8. Restart the *inetd* on the receiving system, or reboot the machine, which automatically restarts *inetd*. See the *inetd*(8C) manual page.

9. On the system that is down, change to the */mnt/os* directory on the SPU by entering

   (spu)> **cd /mnt/os**

10. On the system that is down, terminate execution of all programs other than the standard shell program run by SPU UNIX by entering the following command at the SPU prompt on the system console:

    (spu)> **osclean**
    (spu)> **cpureg >> /mnt/errlog**

11. Execute the *crashdump* utility by entering

    (spu)> **crashdump**

    a. You are prompted to enter a comment that will be included with the crash dump tape(s). Enter the machine name, the condition of the machine at crash time, the last time the machine crashed, and anything else that might be helpful to the TAC.

    b. *crashdump* prompts you for the Internet addresses of the source and destination systems. The addresses for both systems should be listed in the */etc/hosts* file of the destination system. An Internet address must be specified to *crashdump* as a four-component address; for example, 100.0.0.5.

    c. *crashdump* prompts you for the name of the tape drive to use on the remote machine. Enter the name of the tape drive at the prompt.

    d. If you are dumping more than 128 megabytes of physical memory, *crashdump* prompts you when it is necessary to mount a new tape.

After the crash dump begins, you receive status messages on the SPU console to tell you how the crash dump is progressing.

12. When the crash dump is complete, label the tape(s) and send them to the TAC for evaluation. Use the label to ensure that the tapes are easily identifiable when they arrive at the TAC; record the problem report number and, if more than one tape is required, the number of the tape.

    Before packaging the tape to return to the TAC, use a strap, foam block, or similar device to hold the loose end of the tape in place. This reduces the risk of damage to the tape during shipment.

13. When the crash dump is complete, reboot by typing

    (spu)> **/etc/reboot**

    Rebooting the SPU ensures that lingering processes are cleaned up.

Chapter 13
Operator Interface

The need for a system operator or group of operators varies from site to site. At many sites, operators are required to perform tasks such as dumps, restores, tape handling, printer control, and system shutdowns. Within ConvexOS, the commands and utilities required to perform these tasks require the user to have superuser privileges. However, granting superuser privileges to a user gives that user access to every command and file in the system.

With *op*, the system manager can grant an operator class of users access to superuser commands without granting those users superuser privileges. The system manager can establish an operator interface that has controlled access to superuser commands.

This chapter describes

- an overview of the operator interface
- how to set up the *op.access* file that establishes access privileges
- how operators use the *op* command
- logging *op* access information
- security considerations

13.1 Overview

ConvexOS distinguishes two classes of users: users and superusers. Using the operator interface, the system manager can establish an operator class. The system manager can grant these operators access to a proscribed set of commands that require superuser privileges without granting broad superuser privileges to the users in that class.

The system manager can customize the operator interface to site requirements by restricting:

- who may be an operator
- what commands the operator may use
- what arguments the operator may pass to the command

The operator interface is formed by the *op.access* file, which sets restrictions, and the *op* utility that is used to invoke the restricted commands. The system manager may restrict an operator class to an individual user, several individual users, a group of users (as defined in the */etc/group* file), or a combination of individuals and groups.

The system manager can also restrict the specific commands that may be executed by users in an operator class and the arguments that may be passed to these commands. These commands may include custom scripts or programs that are designed for your site. The system manager can define more than one operator class, each with access to a different set of commands.

The system manager implements the operator interface by defining the restrictions on operators and commands in the */etc/op.access* file. In this file, the system manager

- establishes an operator function (for example, *weekly*)
- defines who may use it (for example, members of the group *opers*)
- defines the command and arguments that the operator function calls (for example, *weekly* could call the command sequence */etc/dump 0Gun /mnt*)

To perform operator functions, a user is a member of an operator class uses the *op* command to call an operator function. For example, a user who is a member of the "opers" group may invoke the *weekly* function by entering the following command at the system prompt:

```
% op weekly
```

This is equivalent to entering the following command, except that execution of the following command requires superuser privileges:

```
# /etc/dump 0Gun /mnt
```

13.2 Creating the *op.access* File

The *op.access* file contains the rule list for operators and the commands to which they have access. To take advantage of the flexibility and system security *op* offers, the restrictions established in this file should accurately reflect the tasks users are required to perform. Therefore, before creating the *op.access* file, carefully consider the users who will have operator privileges and the commands to which they will be granted access.

13.2.1 Planning the Operator Interface

You can assign access to privileged commands to users without granting superuser status to those users. Consequently, there is the risk of creating a security breach if the assignments and restrictions are not made carefully. (See section 13.6 for more information on security considerations.)

Before creating the *op.access* file, make a list of the tasks that require superuser privileges and that must be delegated to users who should not have superuser privileges. Decide which users are required to perform which tasks. If several users must perform the same set of tasks, you may want to define them as a group in the */etc/group* file. The format for this file is described in Appendix A, "System Files" and in the *group*(1) manual page of the *ConvexOS Programmer's Reference*. For example, you can create a group called *opers* or you can create several operator groups; for example, *batchopers* or *dumpopers*.

Next, identify commands necessary to accomplish the tasks. You can assign these commands to the appropriate users or groups in the rule list of the *op.access* file. Finally, you can place restrictions on access to the assigned commands by specifying the set of options and arguments that can be passed to those commands by the *op* utility.

13.2.2 Characteristics of the *op.access* File

After you have planned the operator interface, create the *op.access* file.

The full pathname for the *op.access* file is */etc/op.access*. For security reasons, this file should be owned by root with mode 0400. All users except superusers should be prevented from reading or writing this file.

13.2.3 Format of the *op.access* File

Create a separate entry in the *op.access* file for each function established. Each entry in the file describes one function, how it is to be performed, and who can perform it. Separate entries by a blank line for legibility. Lines beginning with a pound sign (#) are comments, and everything from the pound sign to the next newline is ignored by *op*. If a line begins with a tab or spaces, it is considered part of the previous line.

The syntax of the *op.access* file includes characters that carry special meaning as field separators or clarifiers; these cannot be used in a string that is an entry field unless they are enclosed in double quotation marks. The special characters are

- comma (,)
- semicolon (;)
- equal sign (=)
- dollar sign ($)
- pound sign (#)
- characters used in regular expressions (. , [, *); see section 13.2.5.

Entry fields in *op.access* are separated by spaces or tabs. Each entry in *op.access* has the form

mnemonic command [*arg* [. . .]]; [*option* [. . .]]

mnemonic unique name for an operator function

- required field
- name must be alphanumeric
- name cannot contain spaces
- name cannot contain tabs
- name can contain an underscore
- name cannot be more than 100 characters long
- name must begin in column 1

If there is more than one entry for a *mnemonic*, the first entry is used.

command full pathname of the executable command, utility, or script run by *op* when the mnemonic is entered with the *op* command. This field is required and must immediately follow *mnemonic*, separated only by spaces or tabs. For example, following is an entry in the *op.access* file

```
disco    /etc/opbin/start_disco;
```

where *disco* is the mnemonic, and */etc/opbin/start_disco* is the pathname of the script. This operator function is invoked by entering

% **op disco**

arg literal or variable argument passed to *command*.

- Literal arguments are specified directly; for example, specific command arguments such as *0Gun* or files such as */dev/rmt20* are passed to the *dump* command. For example:

  ```
  weekly    /etc/dump 0Gun /mnt;
  ```

  where *weekly* is the mnemonic that invokes the *dump* utility for the */mnt* file system.

- Variable arguments are specified here as *$1*, *$2*, *$3* . . . *$n*, where *n* is a positive integer from 1 to 63. These are described more fully in section 13.2.4, "Setting Options." *$1* refers to the first argument given on the *op* command line (immediately following the mnemonic). For example:

  ```
  weekly    /etc/dump 0Gun $1;
  ```

  where *$1* is the name of the file system to be dumped. To invoke this function, an operator enters

  % **op weekly** *filesystem*

where *filesystem* is the name of the file system to be dumped; for example, */mnt*. This is the same as entering the following superuser command:

 # /etc/dump 0Gun *filesystem*

where *filesystem* is the name of the file system to be dumped.

Multiple *args* must be separated by spaces or tabs. If *arg* in the *op.access* entry is $*, there can be any number of trailing arguments on the *op* command line, including none.

option optional field used to restrict who can use *command* or how it is invoked. The format for an option is

keyword=value[, . . .]

where *keyword* is the name of the option, and *value* is the value for the *keyword*. If more than one *option* is used, they must be separated by spaces or tabs. Section 13.2.4, "Setting Options," defines the types of options that can be set.

13.2.4 Setting Options

The *option* field of an entry in the *op.access* file sets restrictions on use of the command for that entry. You can place restrictions on

- who can use the command
- arguments passed to the command
- special variables set for the command; for example, environment variables

The format for *option* is

keyword=value[, . . .]

where *keyword* is the name of the option, and *value* is the value for the *keyword*. Do not use spaces or tabs in an element of the *value* string unless the string is enclosed in double quotation marks.

13.2.4.1 Restricting Access

The following keywords restrict who can use an *op* function:

groups defines the groups whose members may execute the restricted function. The value for this keyword is a group name or list of group names. Members of a named group are defined in the file */etc/group*. If no value is specified for **groups**, the default list of groups is used (see section 13.2.4.4). If default groups are not assigned, no groups have access to the function. For example:

```
weekly   /etc/dump 0Gun /mnt; groups=opers
```

allows only members of the "opers" group to use the *weekly* function.

users defines the users who may execute the restricted function. If no value is specified for **users**, the default list of users is used (see section 13.2.4.4), If there are no default users assigned, no user has access to this function. Access permissions are not checked for the superuser, so a user with superuser privileges can execute the function even if not explicitly listed here. For example

```
weekly    /etc/dump 0Gun /mnt; groups=opers users=mcbroom,garza
```

allows only members of the *opers* class and users mcbroom and garza to use the *weekly* function.

You can redefine the default users or groups for the *op.access* file; see section 13.2.4.4, "Default Options."

13.2.4.2 Restricting Command Arguments

The following keywords restrict arguments passed to the *command* of an entry in the *op.access* file:

$n — where *n* is a positive integer (but not zero), defines the *n*th variable argument specified in the argument list on the *op* command line. The value can be one or more of the following, separated by commas:

- a literal value
- a regular expression (see section 13.2.5, "Expressions")

Values must be separated by commas.

This *keyword* defines values allowed for a variable argument specified in the *arg* field of the *op.access* entry (*$1*, *$2*, ... *$n*). If a variable argument is specified in an *arg* field but not described by an *option* field, any non-null value is permitted. For example

```
reboot    /etc/shutdown -r $1 $2; $1=now,+[0-9]*,[0-9]*:[0-9]*
```

In this example, only the following expressions can be passed to the reboot command for the first argument (*$1*):

- now
- +00 through +99
- 00:00 through 99:99

Any value can be passed for the second argument (*$2*).

If a variable argument is specified in an *arg* field for a function in the *op.access* file, the user must supply a corresponding argument on the *op* command line when invoking the function. However, there is a way to establish an argument as optional: you can define one of the possible values in its *option* value field as `""`. If the optional argument is not the last argument specified in the *op.access* entry, `""` must be explicitly given on the *op* command line to indicate which argument is omitted.

$* — places restrictions on trailing arguments specified as $* in the *arg* field of the *op.access* entry. If an argument (*$1*, *$2*, ... *$n*) given on the *op* command line corresponds to the $* variable argument, and that variable argument has restrictions placed on it in the *options* section of *op.access*, trailing arguments are checked for validity against those restrictions.

13.2.4.3 Setting Special Variables

The following keywords set special variables for the function called on the *op* command line:

chroot changes the root directory path to the pathname specified as the *value*. (See the *chroot*(2) manual page in the *CONVEX Programmer's Reference* for more information.) When you change the root directory pathname, full pathnames begin with a new root directory pathname as a prefix, including the *command* specified in the *op.access* entry. The default value is the current root directory pathname.

dir changes the working directory to the pathname specified as the *value*. The default value is the current working directory.

gid sets the *gid* to the value specified. The value can be a numeric group ID or a group name. The default is to leave the *gid* unchanged.

uid sets the *uid* to the value specified. The value can be a numeric user ID or can be a login name. The default value is *root*.

umask sets the file creation *umask* to the octal value specified. The default value is 022.

$var where *var* is the name of an environment variable, sets the specified environment variable to the specified *value*. An item is assumed to be an environment variable if it is an alphanumeric string that begins with a dollar sign ($). The specified environment variable is set to the stated value before the command is executed.

The following example illustrates setting special variables:

```
disco    /etc/opbin/start_disco; uid=disco gid=proj dir=/scratch
       umask=027 groups=geo,disco users=snoopy,woodstck $SHELL=/bin/shell
```

You can redefine default options for the *op.access* file; see section 13.2.4.4 , "Default Options."

13.2.4.4 Default Options

To redefine the default value for an option, set the value of that option in a special default line at the beginning of the file. The default line must be the first non-comment line of the file.

The format for the default line is

DEFAULT *keyword=value*[, . . .] [*keyword=value*[, . . .]]

where *keyword* is the name of the option, and *value* can be a single value or a list of values separated by commas. The *keyword* must be one of the following:

- **chroot**
- **dir**
- **gid**
- **groups**
- **uid**
- **umask**
- **users**

The default values for options set in the special default line can be overridden by an entry in the *op.access* file. For any entry, if you set an option set on the default line, the default setting is ignored for that entry and the values specified for that option are used. To set an option including default values, list the default values as well as any additional values in that entry. Values specified for an option in an entry replace the values set for that option in the default line; they do *not* add to them.

For example, if you have defined

```
DEFAULT groups=opers
```

and you want the "opers" group and *test* group to have access to the mnemonic *weekly*, you must specifically define access for both groups as in the following example:

```
weekly    /etc/dump OGun /mnt; groups=opers,test
```

In this example, you must include the "opers" group in the value list for the *groups* option set for the mnemonic *weekly*; when the *groups* option is set, the default line for groups is ignored.

To deny the *opers* group access to the mnemonic *weekly*, set the option for that mnemonic to null. For example

```
weekly    /etc/dump OGun /mnt;  groups=
```

13.2.5 Expressions

Because *op.access* is not parsed by the C shell, rules for regular expressions differ from those on the command line or in shell scripts. In *op.access*, regular expressions used in strings follow the rules used by *ed*. See the *ed*(1) manual page in the *CONVEX Programmer's Reference* for more information. For example:

- if a period, decimal point, asterisk (*), or bracket ([) is part of a value, the value must be enclosed in double quotation marks; for example, "3.14." If not, *op* interprets the period as a regular expression that matches any character.
- the characters ".*" do not indicate a match of any string; you must use "..*" to match any string.
- a question mark (?) is not a special character

13.2.6 Example *op.access* File

Figure 13–1 is a sample *op.access* file showing the file syntax. The sample is used in the following sections to describe how the *op* command is used.

Figure 13–1: Example *op.access* File

```
#
# define the site defaults to use; we want users in the 'operators' group to be able to execute
#   almost everything; it is easier to put it here than on every line...
#
DEFAULT         groups=operators
#
# filesystem backups
weekly /etc/dump 0Gun $1; users=mcbroom,garza $1=/,/usr,/mnt,/project
daily  /etc/dump 5Gun $1; users=mcbroom,garza $1=/,/usr,/mnt,/project
#
# taking the system down
#   $1 shows a good use of regular expressions; $2 can be anything
shutdown        /etc/shutdown -h $1 $2; $1=now,+[0-9]*,[0-9]*:[0-9]*
reboot          /etc/shutdown -r $1 $2; $1=now,+[0-9]*,[0-9]*:[0-9]*
#
# kill all batch procs so system can be restarted
#   (want to override default groups setting given at top, allowing none)
killbatch       /etc/opbin/kill_batch_procs; groups= users=mcbroom,garza
startbatch      /etc/opbin/start_batch;
#
# start up disco daemon
disco           /etc/opbin/start_disco; uid=disco gid=proj dir=/scratch
                        umask=027 groups=swdoc,disco users=esmark,lindholm
                        $USER=disco $SHELL=/bin/shell
#
# let certain people mount and unmount the removable drive
rdsmount        /etc/mount $1 $2; groups=operators,swdev,disco dir=/
        users=mcbroom,garza $1=/dev/dd0.   $2=/.*
rdsumount       /etc/umount $1; groups=operators,swdev,disco dir=/
        users=mcbroom,garza $1=/dev/dd0.
```

13.3 Using the *op* Command

The operator uses the *op* command to invoke functions defined in the *op.access* file, providing that the operator has been granted access to the functions by the restrictions established in that file. Each person who uses *op* should have the pathname */etc* defined for the $PATH environment variable.

The format of the *op* command is

op [**-h**] *mnemonic* [*arg* [. . .]]

where

mnemonic — operator function name defined in the *op.access* file. Each function calls an executable command, utility, or script.

-h — option that invokes the help facility. See section 13.4 for more information on the help option.

arg argument passed to the command called by the function

If an argument entered on the *op* command line does not match a valid *option*, an error message is printed on the user's terminal and logged. (For more information on access logging, see section 13.5.)

The following examples of the *op* command use the example *op.access* file in Figure 13–1.

Example 1 **% op weekly /mnt**

The *op.access* entry

```
weekly /etc/dump OGun $1; users=mcbroom,garza $1=/,/usr,/mnt,/project
```

permits users mcbroom or garza to backup the / (root), */usr*, */mnt*, or */project* file systems.

Example 2 **% op reboot 17:30 "We have to fix our network"**

The *op.access* entry

```
reboot    /etc/shutdown -r $1 $2; $1=now,+[0-9]*,[0-9]*:[0-9]*
```

permits any member of the *operators* group (*operators* is defined as the default group on the first line of the example file) to shutdown the system. The first argument (*$1*) requires a value defined by the entry; the second argument (*$2*) can be any value because it is not defined in the option field.

Example 3 **% op rdsmount /dev/dd0c /mnt/me/mystuff**

The *op.access* entry

```
rdsmount    /etc/mount $1 $2; groups=operators,swdev,disco dir=/
          users=bob,steve $1=/dev/dd0.    $2=/.*
```

permits defined users and groups to mount removable drives on any *dd0* partition. In this example, the user mounts the directory */mnt/me/mystuff* on the c partition.

Example 4 **% op tape disable unit1**

The *op.access* entry

```
tape    /etc/tpc $1 $2; groups=tapeopers,operators users=sysmgr
      $1=enable,disable,stop,restart $2=all, unit[01]
```

permits defined groups and users to enable, disable, stop, or restart all tape units or only the following tape units: unit0, unit1.

13.4 The *op* Help Option

The help option to the *op* command can be used

- by the user to determine functions available to the user or to determine the arguments available for a function
- by the system manager to check the *op.access* file for syntax errors

13.4.1 Operator Help Facility

If a user enters

% **op -h**

op lists the functions to which that user has access. Using the example in Figure 13–1, user mcbroom would receive the following output:

```
weekly
daily
```

If user *horwitz* entered the same command, *op* would generate the message

```
horwitz is not allowed to execute any mnemonics
```

If a user enters

% **op -h** *mnemonic*

where *mnemonic* is a function in the *op.access* file, *op* lists the arguments required for the function (assuming the user who entered the command has access to the function). For example, if the *op.access* file contains the following entry:

```
weekly /etc/dump 0Gun $1; users=mcbroom,garza $1=/,/usr,/mnt,/project
```

and user garza entered the following command

% **op -h weekly**

op would generate the following output:

```
Help for mnemonic "weekly":
   op weekly <arg1>
where
   <arg1>   is "/" OR "usr" OR "/mnt" OR "/project"
```

13.4.2 Checking the *op.access* file

The superuser can run all functions defined in the *op.access* file; access privileges are not checked for the superuser. If a user with superuser privileges uses the help option (*-h*) to the *op* command, *op* reports all functions in the file, parsing each function and checking it for correct syntax.

After you create the *op.access* file, and each time you modify it, check the file using the help option before telling users the new functions are available for use.

To check the *op.access* file, log in as the superuser and enter

op -h

If the syntax of the *op.access* file is correct, *op* lists each mnemonic in the file.

13.5 Logging Access Information

Each attempted execution of the *op* command (successful or not) is logged using *syslog*. Logged information includes

- who executed the *op* command

- when the execution was attempted
- which mnemonic was executed
- which arguments were used
- invalid variables
- invalid users

op logs four classes of information, each mapped to a separate *syslog* level. (*syslog* encodes priorities in defined levels; for more information, see the *syslog*(3) manual page.) The *op* message classes and corresponding *syslog* levels are listed in Table 13–1.

Table 13-1: *op* Message Classes

Message Class	*syslog* Level
Standard use of *op*	INFO
Syntax errors	NOTICE
Unauthorized user	WARNING
exec failure	ERR

The following messages are examples of *op* access information messages:

```
Sep 13 10:25:22 localhost: 11128 op: [mcbroom] too many arguments given for \
mnemonic "mnemonic1"
Sep 13 10:25:31 localhost: 11148 op: [garza]  invalid argument for $1 ("arg") \
of mnemonic "mnemonic5"
Sep 13 10:25:33 localhost: 11150 op: [mcbroom] run with argv= mnemonic6 /mnt
Sep 13 10:55:23 localhost: 12048 op: [mcbroom] run with argv= -h
Sep 13 10:55:25 localhost: 12054 op: [mcbroom] run with argv= -h mnemonic2
Sep 13 12:31:23 localhost: 13408 op: [horwitz] tried to execute "7" (not authorized)
```

All *op* log messages are encoded to the *syslog* facility LOG_AUTH.

syslog directs the *op* messages into a set of log files defined in the file */etc/syslog.conf*. For example, you can direct messages about standard use (*syslog* level INFO) to a file called */usr/adm/log/oplog* and direct all messages about unauthorized users (*syslog* level WARNING) to the console. To establish logging procedures for *op*, see the *syslog*(3) and *syslogd*(8) manual pages.

13.6 Security

The *op* utility allows limited operator access to functions requiring superuser privileges, which creates system security tasks for the system manager. If the tool is not well-constructed, there is opportunity for a security breach. To eliminate this possibility, create the *op.access* file with the following considerations:

- Characteristics of the *op.access* file

  Root owns *op.access* with access mode 0400. Only the superuser can read it. This means an operator can only view information (using the *op* help option) for which the operator has permission.

- Access logging

  Use *syslog* to log all attempts to execute *op*; types of access are differentiated (standard use, syntax errors, unauthorized user) and can be directed to different locations (log

files, system console). Establish logging to ensure that WARNING messages receive immediate attention. These messages can be directed to the system console.

- Operator functions

  Security of an operator function is only as good as the command or utility it calls. For example, do not establish an operator function that calls an interactive utility: interactive utilities prevent limits on arguments. Also, many interactive utilities can invoke an interactive shell that inherits root privileges from *op*. Do not establish *setuid* root functions that call shell scripts because shell scripts inherit root privileges from *op*.

- Arguments

  Be careful in your choice of arguments to commands listed in the *op.access* file. For example, if you establish the *dump* command without restricting arguments, a careless operator can destroy file systems by omitting crucial arguments.

- User access

  Update the list of users and groups defined in *op.access*. Delete obsolete users from this file as you would from the */etc/passwd* file.

A

System Files

This appendix describes system files requiring periodic maintenance. These files, with the exception of */mnt/errlog* on the SPU disk, reside on CPU disks. System files for optional CONVEX products (for example, CONVEX Networking Utilities and CXBatch) are described in their product documentation.

The following information is provided for each file:

- file format
- file description and maintenance instructions
- related programs
- examples
- caveats and bugs

The *termcap*(3X) manual page contains a detailed explanation of termcap-format files. Refer to this manual page as you read the descriptions of */etc/disktab*, */etc/gettytab*, */etc/printcap*, */etc/remote*, and */etc/stripecap*.

The files included in this appendix are listed in alphabetical order by filename, rather than full pathname. For cross-reference, these files are listed below in alphabetical order by full pathname.

- /etc/activities
- /etc/actwho
- /etc/disktab
- /etc/fstab
- /etc/gettytab
- /etc/group
- /etc/hosts
- /etc/motd
- /etc/nologin
- /etc/nurc
- /etc/op.access
- /etc/passwd
- /etc/phones
- /etc/printcap
- /etc/pwrestrict
- /etc/rc.local
- /etc/remote
- /etc/stripecap
- /etc/syslog.conf
- /etc/ttys
- /usr/adm/acct
- /usr/adm/batch-acct
- /usr/adm/bill-acct
- /usr/adm/diskuse
- /usr/adm/failure_log
- /usr/adm/log/batchlog
- /usr/adm/lpd-acct
- /usr/adm/shutdownlog
- /usr/adm/stat
- /usr/adm/tp-acct
- /usr/adm/wtmp
- /usr/lib/aliases
- /usr/lib/crontab
- /usr/lib/uucp/L-devices
- /usr/lib/uucp/L-dialcodes
- /usr/lib/uucp/L.cmds
- /usr/lib/uucp/L.sys
- /usr/lib/uucp/USERFILE
- /usr/spool/uucp/ERRLOG
- /usr/spool/uucp/LOGFILE

Name

/usr/spool/uucp/ERRLOG — *uucp* error log

Format

ASCII single-line entries

Description

The */usr/spool/uucp/ERRLOG* file keeps track of problems encountered by the *uucp* subsystem. It grows slowly and can be ignored unless a major error causes it to expand significantly.

Example

```
ASSERT ERROR (uux)  pid: 3999  (1/26-0:29)
                CAN'T OPEN D.convexBKow2 (0)
ASSERT ERROR (uucico)  pid: 274  (7/12-16:27)
                PKXSTART ret  (-1)
```

This example shows typical errors. Errors usually result from system crashes; files in the process of being sent to or received from remote sites at the time of a system crash are sometimes lost.

Programs

uucp, uux, uucico, uuxqt

Name

/usr/lib/uucp/L-devices — *uucp* dialout device descriptions

Format

%s %s %s %s %s\n

Description

This file lists the automatic call units (dialout modems) and direct-connect lines for the *uucico* utility. Each line in the file describes one direct-connect line. Some lines describe lines previously described but with differing characteristics (e.g., baud rate). The field descriptions are as follows:

1. ACU (automatic call unit) or DIR (direct-connect line)
2. device in the */dev* directory
3. ignored
4. baud rate
5. modem type

Example

```
ACU cua1 cua1 300 vadic
ACU cua1 cua1 1200 vadic
DIR tty2a 0 9600 direct
```

Programs

uucico(8C)

Caveats

The *cu*(1C) and *tip*(1C) utilities also use the devices.

Name — */usr/lib/uucp/L-dialcodes* — common dialcodes used by *uucp* dialers

Format — %s %s\n

Description — This file maps dialing prefixes in the */usr/lib/uucp/L.sys* file into dialable sequences.

Example

```
MCI  6514321,654321
```

In this example of an entry in the *L-dialcodes* file

- first seven numbers are the local MCI number
- comma tells the system to wait for a dial tone
- last six numbers are the MCI access code

With the system entries set up in this manner, a user may dial an MCI number by specifying "MCI" and the desired number (e.g., MCI2145551212).

Programs — *uucico*(8C)

Bugs — Some long-distance services (like Rolm) require trailing digits for accounting; this facility does not solve that problem.

See Also — */usr/lib/uucp/L.sys*

Name

/usr/lib/uucp/L.cmds — list of valid remote commands

Format

%s\n

Description

This file specifies the following:

- PATH containing valid remote *uux* commands
- names of valid commands

Example

```
PATH=/usr/local/bin:/bin:/usr/bin
rmail
nfrcv
nfxmit
ruusend
uncompress
uusend
```

Programs

uux(1C)

Caveats

Choose the commands you include carefully. Giving remote sites access to commands can create security risks.

Name

/usr/lib/uucp/L.sys — dialing sequences for *uucp* for remote hosts

Format

%s %s %s %s %s [^\n]\n

Description

This file tells *uucico* how to dial remote hosts. Each line in the file describes one way to access a remote host. Each entry must have at least five fields:

1. name of remote host; use more lines if there is more than one way to dial the host
2. times to call the host; for example Any, None, and hybrids (Any1700-2400)
3. ACU (automatic call unit) or DIR (direct connect line)
4. baud rate
5. telephone number for ACU lines (modified by L-dialcodes) or device name (for direct connects)
6. describe (respectively) what the remote system should send to you and what you should send to the remote system. Waiting for a null ("") is the same as waiting for nothing at all. The "waiting for" phrase is a hyphen-separated list of phrases and items to send if the phrase is not received within a time-out period (see example).

Use the following procedure to add a new *uucp* site:

1. Replicate a working site's entry.
2. Change the site name.
3. Change the calling times (if necessary).
4. Change the baud rate (if necessary).
5. Change the phone number.

Use this procedure when you are calling another site. If the site is calling you, give the site a unique login name and password; otherwise, there is no audit trail for security.

Example

```
convex1 Any ACU 1200 5551212 "" \r login:-BREAK-login: XYuucp
    Password: intruder password: thief
```

The example is on two lines for convenience; in the file, the information is on one line. A remote machine can reply in either uppercase or lowercase (e.g., *Login*, *login*).

Programs

uucico(8C)

Caveats

To debug entries in *L.sys*, use

```
rm -f /usr/spool/uucp/STST.sitename
/usr/lib/uucp/uucico -f -r1 -ssitename -x4
```

This starts a conversation with site *sitename* and displays the conversation on your terminal as the conversation proceeds.

To use debugging, your *uid* must be 0, 1, 2, or 3 (root or daemon).

When a new *uucp* site contacts you, put its site name in the *L.sys* file, whether or not you intend to call it. Each entry in the file must have at least five fields. The following example is an entry for a system you do not plan to call:

```
systemname None DIR 9600 null
```

Name */usr/spool/uucp/LOGFILE* — log of *uucp* activity

Format %s %s (%d/%d-%d:%d-%d) [^(]([^)])]\n

Description

The */usr/spool/uucp/LOGFILE* file notes completions of *uucp* functions; for example, connects, queues, file copies, and executes. The */usr/spool/uucp/STST.site* file contains status information; for example, information about recent failed calls and calls in progress. The following is a partial list of *LOGFILE* entries:

ACCESS (denied)
: In a file-access operation, a file's path prefix might be bad, or the file might be protected from read or write. The */usr/lib/uucp/USERFILE* file lists valid pathname prefixes. Some remote systems give this message if your system is not known to their system's *L.sys* file.

ACU LINE CLOSE (fail)
: The close on the ACU failed; this rarely occurs.

BAD READ (-)
: The login sequence did not get what it expected. Check *L.sys* to make sure the login sequence is correct. Use the *-x4* debug option of *uucico* or *send* to see what the other system is really saying.

CANNOT CALL (*system status*)
: An *STST.system* file exists and contains the status of the last call. If the status is not TALKING or CONVERSATION, you can remove this file. Status is left when a system cannot be reached. This prevents calls from being placed too often.

CANNOT DETERMINE (*system name*)
: All heuristics for learning the local system's name failed.

CAN'T LOCK (*resource name*)
: The specified resource could not be locked. If this happens repeatedly, *carefully* remove a lockfile by hand.

CAN'T OPEN (*device or file name*)
: The device or file enclosed in parentheses was not available for opening when it should have been. If this happens repeatedly, be sure the file or device exists and that permissions on the file or parent directory are set correctly.

CAUGHT (*signal name*)
: The specifed signal hit *uucp*. After cleanup, *uucp* exited.

COPY (failed)
: The file copy did not complete. It will be retried later.

DEBUG (enabled)
: Self-explanatory.

DENIED (*can't open*)
: *uucp* tried to create a temporary file in a *TM.* directory in */usr/spool/uucp* but failed. When this happens, it means the file system is full, so the the directory is also full.

DONE (*work here*)
Someone requested *uucp* to copy a file from the home system to the home system. It did so without dialing anywhere.

FAILED (ACU READ)
After successfully opening a modem for input/output, a read returned unsuccessfully. This happens when the modem hangs up or someone disconnects the cable.

FAILED (can't create TM)
Cannot create temporary file for transfer.

FAILED (can't read DATA)
A file was to be sent via *uucico*, but the file could not be read after all the *setuid* mechanisms. This happens when permissions in */usr/spool/uucp* are incorrect or *uucp*, *uucico*, and other *UUCP* programs are not in agreement about the correct *uid*.

FAILED (connection refused)
An attempted *uucp* connection over Ethernet failed because the remote system does not have the *uucp* service defined in the /etc/inetd.conf database.

FAILED (dialup ACU write)
Bad return status from write to auto-dialer.

FAILED (dialup line open)
Auto-dialer did not return connection established. Possible reasons: no dial tone (the modem might be in talk mode), line busy, no answer (system down, wrong number).

FAILED (login)
The login sequence was unable to succeed during login at the remote site. If this happens repeatedly, use the *-x4* debug option of *uucico* or *send* to see what the remote site is really sending (and possibly expecting).

FAILED (conversation complete)
The call has been abnormally terminated.

FAILED (imsg 1) & FAILED (imsg 2)
The *uucico* read routines returned an EOF when least expected. This rarely occurs.

FAILED (to call system)
No connection to the call system could be made.

FAILED (startup)
No common protocol could be found.

HANDSHAKE FAILED (-)
The two hosts could not agree on whether the connection should proceed, or one host explicitly said *no*. This happens when more than one call is in progress between the two hosts or when someone has a LCK file set up for the host dialing in.

LOCKED (*system*)
A *LCK.system* file exists in */usr/spool/uucp* indicating that a call is in progress to this system. If *uucp* died without removing this file or the

system crashed and */etc/rc* did not remove it, you can remove it. Do *not* remove it unless you are certain that *uucp* and other dialer-type programs are not running.

LOST LINE (login)
A read error occurred during login. This could be caused by the loss of the line when the remote system hangs up or by lack of response from the remote system.

MAIL FAIL (*username*)
Mail to someone failed and was returned to sender.

NO CALL (*max # of recalls*)
uucico gave up trying to call a site after too many failures. Remove *STST.sitename* file to try again.

NO CALL (retry time not reached)
No call was attempted because *STST.system* shows the last attempt failed and the retry time has not been reached. If you are certain that the conditions causing the failure have been rectified, remove *STST.system* and try again.

NO (*device name*)
Cannot find *L-devices* device needed to make call (or all devices are currently locked).

NO WORK (*site name*)
Nothing is available to send to the listed site.

OK (conversation complete)
The call completed normally.

OK (startup)
A protocol has been agreed upon.

PATIENCE (–)
uucico is rereading a message.

PERMISSION (denied)
Cannot access remote site.

PREVIOUS (bad sequence)
Sequencing information between the two systems is wrong.

QUE'D (*filename*)
A file is queued to be transmitted.

REQUEST (*filename*)
A file is to be transmitted.

REQUESTED (*filename*)
A file is to be received.

REQUIRED (callback)
For security reasons, your system must call back the other system. This is a standard informative message.

SUCCEEDED (call to system)
A call has been placed, although login is not complete.

TABLE OVERFLOW (-)
uucico tried to open too many files. This rarely occurs.

TIMEOUT (dialup DN write)
Unable to write to auto-dialer. Check to make sure that the modem is asserting the DCD signal.

TIMEOUT (-)
An alarm expired and it is too late now. This can happen when one or both systems have high loads. If that is the case, try again when the loads are lower.

WRONG TIME TO CALL (*system name*)
Calls to this system can be made only during certain times (listed in */usr/lib/uucp/L.sys*). Generally, *cron*, rather than user demand, starts long distance calls.

XQT QUE'D (-)
A remote execution has been queued for transmission.

XUUCP DENIED (-)
It is not legal to execute the specified program remotely. Fix the */usr/lib/uucp/L.cmds* file on the remote system if you think you should be able to execute this command.

Example

The following example shows the format of the entries:

```
uucp ihnp4 (10/20-16:19-4619) OK (conversation complete)
INuucp smu (10/20-16:20-4794) SUCCEEDED (call to smu)
INuucp smu (10/20-16:21-4794) OK (startup)
```

Programs

uucico(8C), *uuxqt*(8C)

Caveat

The *LOGFILE* grows without boundaries at a site using *uucp*. To save space, regularly discard all but the last few thousand lines.

Name

/usr/spool/uucp/SYSLOG — *uucp* file transfer log

Format

%s %s (%d/%d-%d:%d) (%d) %s data %d bytes %d secs\n

Description

The */usr/spool/uucp/SYSLOG* file logs completed *uucp* file transfers, listing the following:

- name of user (usually a daemon) requesting the transfer
- system name
- date and time
- time in seconds since 1970
- sent or received status of file
- number of bytes
- time required to complete the transfer

Example

```
daemon mazama (10/1-0:51) (496993902) sent data 671 bytes 5 secs
uucp allegra (10/1-0:53) (496994048) received data 204 bytes 4 secs
daemon mazama (10/2-0:51) (496993907) sent data 62 bytes 0 secs
```

Programs

Summary programs have not been released for this file.

Caveats

The sum of the transfer times is not the long-distance call time. The long-distance call time, which is higher because of setup and takedown overhead, is not included in the transfer times.

The time to send a file can be short (0 seconds) because the teletype output system is buffered.

Name

/usr/lib/uucp/USERFILE — list of valid pathname prefixes for *uucp*

Format

%s,%s [^\n]\n

Description

The USERFILE specifies which file systems are accessible to local users and to remote systems using UUCP by describing the pathname prefixes that are valid for login IDs and systems. The USERFILE can be used to specify "automatic callback."

Each line of the USERFILE describes one user-machine pair. A comma (but no space) separates the user name and machine name. If either the user or machine entry is omitted, all users or machines are valid for the pair. If both user and machine entries are omitted, the entry matches all entries not previously matched.

If included, the character "c" (separated by spaces from the user-machine pair and paths) specifies callback. Callback causes the current conversation to end quickly and the current slave to call back the current master.

The final set of fields on each line is a set of pathname prefixes. For security reasons, one pathname of the set must match that of each file before it is transferred.

Example

```
,convex /
, /usr/spool/uucppublic
```

This example is a prototype USERFILE. Substitute your site's name for *convex*. Anyone on *convex* can send files away from the machine; explicitly: % *uucp file remote!path*. Machines calling in can move files to and from */usr/spool/uucppublic*.

Programs

uucico(8C)

Caveats

If you change USERFILE from what is shipped with your system, you open your machine to possible security violations.

Bugs

uucp has bugs that allow security violations.

Name */usr/adm/acct* — execution accounting file

Format Binary

Description The *acct* system call makes entries in an accounting file (*/usr/include/sys/acct.h*) for each process that terminates. The format of the account record has changed for versions of CONVEX UNIX greater than V3.0. The new record contains larger fields to provide more precision and/or greater range. The new and old account records as defined by the include file are as follows:

```
/*      $CHeader: acct.h 1.7 88/08/31 09:52:58 $    */
/*      Copyright 1984 Convex Computer Corp.        */

/*
 * Accounting structure
 */

struct acct
{
        char    ac_comm[10];        /* Accounting command name */
        struct timeval ac_utime;    /* Accounting user time */
        struct timeval ac_stime;    /* Accounting system time */
        struct timeval ac_etime;    /* Accounting elapsed time */
        time_t ac_btime;            /* Beginning time */
        u_short         ac_uid;                 /* (real) user ID */
        u_short         ac_gid;                 /* (real) group ID */
        long    ac_aid;                 /* activity ID */
        long long       ac_kbsec;       /* memory usage integral */
        long    ac_io;                  /* number of disk IO blocks */
        dev_t   ac_tty;                 /* control typewriter */
        char    ac_flag;                /* Accounting flag */
        char    ac_unused[5];           /* future expansion */
        float   ac_avconcur;            /* Average concurrency level */
};

#define         AFORK   0001            /* has executed fork, but no exec */
#define         ASU     0002            /* used super-user privileges */
#define         ACOMPAT         0004            /* used compatibility mode */
#define         ACORE   0010            /* dumped core */
#define         AXSIG   0020            /* killed by a signal */
#define AFIXSCH 0040            /* used fixed scheduling */

#ifdef KERNEL
struct acct     acctbuf;
struct vnode    *acctp;
#endif
```

If the process calls *execve*, the first 10 characters of the filename appear in *ac_comm*. The accounting flag contains bits that show whether *execve* was accomplished and whether the process ever had superuser privileges.

Programs *sa*(8)

Caveat

The */usr/adm/acct* file grows without boundary unless you use *sa* to compress it. Use *cron* to run the accounting scripts to compress the file.

See Also

acct(2), *execve*(2), *sumscripts*(8)

Name

/etc/activities — activity list

Format

ASCII single-line entries

Description

The */etc/activities* file is a list of activities that defines correspondences between activity names and activity IDs. For each activity, a single line contains the activity name, followed by a colon and the activity ID. To save time, sort the activity list, placing activities most often referenced at the beginning of the file.

Activity names cannot contain colons, and it is recommended that they not contain spaces (to not confuse *awk* scripts).

On login, activity ID is set to 0 (zero). For this reason, the activity with ID 0 should be a catchall activity and should have an entry to this effect in the activities list. Typical names for this activity are "overhead" and "miscellaneous." An activity ID must fit in 32 bits and must fall in the range -2,147,483,648 to 2,147,483,647. There is a one-to-one mapping between activity names and activity IDs, so each activity name is associated with one activity ID and vice versa.

When used with the activity ID offset in CXbatch, activity IDs provide a means of keeping track of batch jobs. When a process is executed from a batch queue, the activity ID of the process is set to the activity ID of the user who submitted the job plus the number specified (if any) by the queue's activity ID offset. A suggested system is to number all activity IDs in increments of 10. Each batch queue would then have an activity ID offset from 1 to 9, with 0 reserved for jobs that were not submitted using the batch queues.

Example

```
misc:0
vctest:100
ostest:110
integration:500
tooldev:520
```

See Also

bill(1), *idtoname*(1), *setaid*(2), *acct*(5), *actwho*(5), *qmgr*(8)

Name

/etc/actwho — group-activity access control file

Format

ASCII single-line entries

Description

The *actwho* file is used by the *bill*(1) command to determine who can bill to which group-activity combination. Each entry in the *actwho* file is of the form

group[.[*activity*]]:*users*

group a group listed in the */etc/group* file

activity an activity listed in the */etc/activities* file. In the absence of an *activity* field, the activity with activity ID 0 is assumed. This activity is usually named something like "overhead" or "miscellaneous."

users comma-separated list of user names allowed to bill to the group-activity combination given in the initial part of the entry

group, *activity*, or *users* can consist of the single character * (asterisk), which stands for "all." *group* or *activity* can contain the ? character, which matches any character in that character position. With creative naming of activities, this capability can be used to allow certain users to bill to classes of activities. For example, suppose all activity names related to documentation were of the form "3820-xxx-7589", where *x* is some character that varies with the activity. Then an entry in */etc/actwho* such as "*.3820-???-7589:user, ..." would list users allowed to bill to documentation.

To reduce search time, */etc/actwho* should be sorted, with group-activity combinations referenced most often placed at the beginning of the file. Use the *edactwho* utility to edit */etc/actwho*; this utility assures correct syntax and coordinates file locking with the *bill* command.

Example

Users *malone*, *erving*, and *shapira* can bill to the group-activity combination of group *nba* and any activity.

```
nba.*:malone, erving, shapira
```

All users can bill to the group-activity combination of group *sdev* and any activity starting with "017-" and ending in any four characters.

```
sdev.017????:*
```

See Also

bill(1), *getacwent*(3), *activities*(5), *group*(5), *edactwho*(8)

Name

/usr/lib/aliases — *sendmail*(8) aliases file

Format

%s: %[^\n]\n

Description

This file describes user ID aliases used by */usr/lib/sendmail*. Each line is of the form

```
name: name_1, name_2, name_3, ...
```

where *name* is the alias for the list of *name_n*. Lines beginning with white space are continuation lines. Lines beginning with # are comments.

Aliasing occurs only on local names. Loops cannot occur, because messages are not sent to a person more than once. After aliasing is done, messages are forwarded to the list of users defined in each recipient's *.forward* file.

/usr/lib/aliases contains only raw data. *newaliases*(1) puts the aliasing information into a binary format in the files */usr/lib/aliases.dir* and */usr/lib/aliases.pag*. To make the change take effect, execute *newaliases* each time you change the aliases file.

Example

The following example shows three aliases:

```
joans: jones
smithnotes: "|/usr/spool/notes/.utilities/nfmail smithnotes"
forum: garza, esmark, horwitz, chiarelli, kleinfeld, riley, keiko
```

The first alias corrects people who misspell user jones' name. The second forwards mail sent to "smithnotes" to the *smithnotes* notesfile. The third is a mailing list alias that forwards mail to many users.

Programs

sendmail(8)

Bugs

Because of restrictions in *dbm*(3X), an alias cannot contain more than about 1000 bytes of information. For longer aliases, use a dummy name as the last name in the alias; this method is called "chaining." The dummy name is a continuation alias.

See Also

newaliases(1), *dbm*(3X)

Name

/usr/adm/bill-acct — bill accounting file

Format

Binary data

Description

The */usr/adm/bill-acct* file is a record of successfully executed *bill* commands. The *bill.h* include file defines the structure written twice each time a user changes billing accounts from a login shell. The first record corresponds to the group-activity combination that is terminating; the second corresponds to the new combination. The *bill.h* include file is shown in the following illustration:

```
/*  $CHeader: bill.h 0.0 86/01/28 18:37:17 $  */
/*  Copyright 1984 Convex Computer Corp.  */
struct billlog {
  short  bi_uid;       /* user id */
  short  bi_gid;       /* new/old group id */
  int  bi_aid;       /* new/old activity id */
  long  bi_time;     /* time as returned by time() */
  char  bi_tty[8];     /* tty line, as in <sys/acct.h> */
  unsigned short  bi_flags;  /* see BI_xx #defines */
  char  bi_unused[4];     /* reserved for future use */
};

#define  BILLLOG   "/usr/adm/bill-acct"  /* log file */
#define  BI_START  0x0001       /* 1 if starting, 0 if ending */
```

See Also

bill(1), *utmp*(5), *connecttime*(8), *sumscripts*(8)

Name

/usr/lib/crontab — schedule of programs run by the *cron* utility

Format

%d %d %d %d %d [^\n]\n

Description

The *crontab* file consists of lines of six fields each, separated by spaces or tabs. Each line lists a command and a time to execute the command. The first five fields are integer patterns specifying the following:

1. minute (0–59)
2. hour (0–23)
3. day of month (1–31)
4. month of year (1–12)
5. day of week (1–7, 1 = Monday)

Each pattern can contain one of the following:

- a number in the range listed

  two numbers separated by a minus for a range inclusive
- list of numbers separated by commas for any of the numbers
- an asterisk (*) for all legal values

The sixth field is a string executed by the Bourne shell at the specified times. The shell translates percent characters in this field as newline characters. The shell executes only the first line (up to a % or end-of-line). The other lines available to the command as standard input.

The *cron* program examines */usr/lib/crontab* every minute.

Example

In the following example, *cron* invokes a command daily at 5 a.m. The command examines files in */usr/preserve* and deletes files more than one week old. Every hour (at 35 minutes past the hour), *cron* feeds the *runbuglist* script to a Bourne shell.

```
00 05 * * * find /usr/preserve -mtime +7 -a -exec rm -f {} ;
35 * * * * su root < /usr/local/lib/buglist/runbuglist
```

Programs

cron(8)

Caveat

Scripts invoked by *cron* must include full pathnames for programs except */bin* and */usr/bin*.

The search path for programs invoked by *cron* is usually more restricted than the search path invoked by most users; for instance, it often lacks */usr/local/bin* and */usr/convex*.

Name

/etc/disktab — disk description table

Format

same as *termcap* file

Description

Each entry in */etc/disktab* describes the partition layout for one type of disk.

Capabilities

Refer to *termcap*(5) for a description of the file layout.

Name	Type	Description
ty		type of disk
se		number of bytes per sector
ns		number of sectors per track
nt		number of tracks per cylinder
nc		number of cylinders per disk
rm		disk speed (revolutions per minute)
p[a-h]		partition sizes (sectors)
b[a-h]		partition block sizes (bytes)
f[a-h]		partition fragment sizes (bytes)

Example

```
dkd-001|dkd001|DKD-001|DKD001|eagle|Eagle|Fujitsu Eagle (45 sectors):
        :ty=winchester:se#512:ns#45:nt#20:nc#842:rm#3900
        :pa#37800:ba#8192:fa#1024:
        :pb#151200:bb#8192:fb#1024:
        :pc#756000:bc#65536:fc#8192:
        :pd#37800:bd#8192:fd#1024:
        :pe#227700:be#8192:fe#1024:
        :pf#75600:bf#8192:ff#1024:
        :pg#341100:bg#8192:fg#1024:
        :ph#225900:bh#8192:fh#1024:
```

The example specifies data for a disk with the following names:

- *dkd-001*
- *dkd001*
- *DKD-001*
- *eagle*
- *Eagle*
- *Fujitsu Eagle (45 sectors)*

Fujitsu Eagle (45 sectors) is included as a descriptor. The disk type is *winchester*; the other type option is *removable*. The disk has 45 sectors per track and 20 tracks per cylinder for each of 842 cylinders. The disk spins at 3900 rpm. The *a* partition is 37,800 sectors (19,353,600 bytes) long. Major blocks are 8,192 bytes long; minor blocks are 1,024 bytes long. The example also specifies 7 other partitions. Block sizes are not specified for the *b* or *c* partitions.

Programs

The *newfs*(8) and *newst*(8) utilities determine physical characteristics for given disks by looking in */etc/disktab*.

Caveats

Choose the layout of file systems carefully to avoid problems mounting file systems that share cylinders.

Do not change */etc/disktab* except when adding a new style of disk to the system.

Name

*/usr/adm/diskuse/** — disk summaries for each user

Format

Each user has a file with a one-line description, for example:

```
%d %s [^\n]\n
```

Description

Each file in the */usr/adm/diskuse* directory corresponds to a user ID on the system and contains a series of lines. Each line in the file contains

- use in blocks
- directory for which use is specified
- date at which use is specified

Certain summary programs use this information to inform users of their excessive disk use.

Example

The following entry is from the file */usr/adm/diskuse/smith*:

```
181 /mnt/smith  Fri Mar 18 05:26:47 CDT 1988
```

The */mnt/smith* directory had 181 kilobytes on Friday, March 18, 1988.

Programs

The following local programs can create and process data from the */usr/adm/diskuse* files:

- *diskspace* — calculate usage
- *diskmail* — inform users of their disk use
- *quotaon* — turn file-system quotas on
- *quotaoff* — turn file-system quotas off
- *quotacheck* — file-system quota consistency checker
- *repquota* — summarize quotas for a file system
- *edquota* — edit user quotas
- *quota* — display disk use and limits, manipulate disk quotas

Name

/usr/adm/failure_log — log of file access failures

Format

Binary

Description

The */usr/adm/failure_log* file is a log of all file accesses that failed because of insufficient permissions. To print the formatted log file to the screen, enter

 # /usr/adm/faillogpr

Example

Each record in the file is printed one one line in the following format:

```
Tue Apr 14 16:08:29 CDT 1987 carson(rivers) RW EPERM csh /etc/rc
```

Tue	day of failed access
Apr	month of failed access
16:08:29	time (central daylight time) of failed access
carson	real user ID of accessor
rivers	effective user ID of accessor (if different from real user ID)
EPERM	error code
RW	access code (read, write, execute) of the file
csh	command being executed
/etc/rc	file user tried to access

Programs

faillogpr(8)

Bugs

The failure log does not contain enough information to uniquely identify a file. For example, if the file in question is deleted after the failed access occurs, it is impossible to determine where that file was in the directory structure.

See Also

faillog(2), *faillogon*(8), *faillogpr*(8)

Name

/etc/fstab — static information about file systems

Format

ASCII: %s %s %s %s %d %d \n

Description

Each single-line entry in the ASCII */etc/fstab* file describes the following fields:

1. block (not raw) physical device described by the entry. For NFS partitions, this field is

   `machine`:*/full_path_name*

2. logical name upon which the device is normally mounted

3. type of partition entry for this device:

4.2	local disk
nfs	network file system
swap	swap partition
ignore	comment line

4. type of access for this device. The following options are valid on 4.2 and *nfs* file systems:

rw	read/write (default)
ro	read only
suid	set *uid* execution allowed (default)
nosuid	set *uid* not allowed
quota	usage limits enforced
noquota	usage limits not enforced (default)
hide	ignore this entry during a **mount -a** command

   The following options are available only on *nfs* systems:

bg	if first attempt fails, retry in the background
fg	retry in foreground
retry=#	set number of failure retries to #
rsize=#	set read buffer size to # bytes
wsize=#	set write buffer size to # bytes
timeo=#	set *nfs* timeout to tenths of a second
retrans=#	set number of *nfs* retransmissions to #
port=#	set server IP port number to #
soft	return error if server does not respond
hard	retry request until server responds

5. obsolete entry sometimes used to specify the dump frequency (in days)
6. pass number for parallel dumps (and for running *fsck*)

The ASCII file contains an entry for each logical file system. Blanks or tabs separate the fields on each line. Comment lines start with #. Add an entry to this file each time you add a disk partition to the system. No program writes to */etc/fstab*.

Example

```
/dev/da0d /mnt 4.2 rw 1 3
```

In this example, **mount -a** puts the */mnt* file system on partition */dev/da0d* for read and write access. No program currently examines the fourth field. The *preen*(8) disk-checking program ignores the final "3," but *fsck*(8) checks this file system concurrently with all other file systems with "3" as their pass number.

Programs

df(1), *fsck*(8), *preen*(8), *swapon*(8), *mount*(8), *umount*(8)

Caveat

Mounting overlapping file systems causes loss of data on both partitions.

To access records from */etc/fstab*, use *getmntent*(3).

Order records in */etc/fstab* so *mount* and *umount* can process them sequentially. For example, if */master* and */master/slave* are partitions, you must include the */master* entry in *fstab* before the entry for */master/slave*.

Disk partitions specified as *swap*, *ignore*, and *nfs* do not have meaningful dump frequencies or pass numbers and are not checked by *fsck*(8).

You can include information on striped disk partitions in */etc/fstab*.

See Also

/usr/include/fstab.h, *stripecap*(5), *fstab*(5)

Name

/etc/gettytab — terminal configuration file

Format

Same as *termcap*

Description

The *gettytab* file is a *termcap*-style file that describes terminal lines. The initial terminal login process *getty*(8) reads the *gettytab* file each time it starts, allowing dynamic reconfiguration of terminal characteristics. Each entry in the database describes a class of terminals.

Capabilities

Refer to *termcap*(5) for a description of the file layout.

The **Default** column lists defaults obtained if no entry is specified and the special default table has no entry.

Name	Type	Default	Description
ap	bool	false	terminal uses any parity
bd	num	0	backspace delay
bk	str	0377	alternate end of line character (input break)
cb	bool	false	use CRT backspace mode
cd	num	0	carriage-return delay
ce	bool	false	use CRT erase algorithm
ck	bool	false	use CRT kill algorithm
cl	str	NULL	screen clear sequence
co	bool	false	console—add \n after login prompt
ds	str	^Y	delayed suspend character
ec	bool	false	leave echo off
ep	bool	false	terminal uses even parity
er	str	^?	erase character
et	str	^D	end-of-file (EOF) character
ev	str	NULL	initial environment
f0	num	unused	tty mode flags to write messages
f1	num	unused	tty mode flags to read login name
f2	num	unused	tty mode flags to leave terminal as
fd	num	0	form-feed (vertical motion) delay
fl	str	^O	output flush character
hc	bool	false	do *not* hang up line on last close
he	str	NULL	host name editing string
hn	str	host name	host name
ht	bool	false	terminal has real tabs
ig	bool	false	ignore garbage characters in login name
im	str	NULL	initial (banner) message
in	str	^C	interrupt character
is	num	unused	input speed
kl	str	^U	kill character
lc	bool	false	terminal has lowercase
lm	str	login:	login prompt
ln	str	^V	"literal next" character
lo	str	*/bin/login*	program to execute when name obtained
nd	num	0	newline (line-feed) delay
nl	bool	false	terminal has (or might have) a newline character
nx	str	default	next table (for auto-speed selection)
op	bool	false	terminal uses odd parity
os	num	unused	output speed
pc	str	\0	pad character
pe	bool	false	use printer (hard copy) erase algorithm

pf	num	0	delay between first prompt and following flush (seconds)
ps	bool	false	line connected to a MICOM port selector
qu	str	^\	quit character
rp	str	^R	line retype character
rw	bool	false	do *not* use raw for input, use cbreak
sp	num	unused	line speed (input and output)
su	str	^Z	suspend character
tc	str	none	table continuation
to	num	0	time-out (seconds)
tt	str	NULL	terminal type (for environment)
ub	bool	false	do unbuffered output (of prompts, etc.)
uc	bool	false	terminal is known uppercase only
we	str	^W	word erase character
xc	bool	false	do *not* echo control characters as ^X
xf	str	^S	XOFF (stop output) character
xn	str	^Q	XON (start output) character

Example

```
default:\
    :ap:fd#1000:im=\r\n\r\nCONVEX UNIX, RELEASE V6.0 (%h)\r\n\r\
\r\n\r:sp#1200:2|std.9600|9600-baud:\
    :sp#9600:
```

ap terminal uses any parity (default entry)

fd form-feed delay is a full second

im initial banner message substitutes the name of the system for *%h*

sp default speed is 1200 baud

As this example shows, only a few of the available options are typically used.

The "2" entry in *sp* changes nothing but the terminal speed. Additional entries (3, 4, etc.) also change the baud rate.

Caveats

If no line speed is specified, speed is not altered from the line speed in use when *getty* is entered. Specifying an input or output speed overrides line speed for the stated direction only. CONVEX does not support split baud rates.

Terminal modes are derived from the Boolean flags specified (*f0*, *f1*, *f2*). You can use these modes for message output, login name input, and to be sure no one changes *tty* modes. If the derivation should prove inadequate, any (or all) of these three can be overridden with the *f0*, *f1*, or *f2* numeric specification. These specifications identify (usually in octal, with a leading 0) the exact values of the flags. Local (new *tty*) flags are set in the top 16 bits of this 32-bit value.

When it receives a null character, *getty* restarts, using the table pointed to by *nx*. Null characters are presumed to show a line break. If no *nx* entry exists, *getty* reuses its original table.

You can precede the *cl* screen clear string with a (decimal) number indicating the number of milliseconds of delay required to clear the screen (*termcap* style). You can simulate this delay repeatedly using the pad character (*pc*).

To print the host name, include the character sequence *%h* in the initial message (*im*) and the login message (*lm*). *%%* prints a single % character. The host name is normally obtained from the system, but you can use the *hn* table entry to set it.

You can edit the host name with *he*.

The *he* string is a sequence of characters — each character that is neither @ nor # is copied into the final host name. A @ in the *he* string causes one character from the real host name to be copied to the final host name. A # in the *he* string causes the next character of the real host name to be skipped. Surplus @ and # characters are ignored.

When *getty* executes the login process given in the *lo* string (usually */bin/login*), it sets the environment to include the terminal type, as shown by the *tt* string. The *ev* string can be used to enter additional data into the environment. It is a list of comma-separated strings, with each string having the form *name=value*.

If a nonzero time-out is specified with *to*, *getty* exits within the indicated number of seconds, either having received a login name and passed control to *login*, or having received an alarm signal and exited. This may be useful for hanging up dial-in lines.

Output from *getty* is even parity unless *op* is specified. Specify *op* with *ap* to allow any parity on input, but generate odd parity output. This only applies while *getty* is being run; terminal driver limitations prevent a more complete implementation. *getty* does not check parity of input characters in *RAW* mode.

Bugs

Some administrators change the default special characters, so it is wise to always specify (at least) the erase, kill, and interrupt characters in the *default* table. The characters # or ^H in a login name are treated as erase characters; @ is treated as a kill character.

The delay implementation is not flexible, and some of the delay algorithms are not used. The terminal driver should support useful delay settings.

Because *login*(1) resets the environment, there is no point setting it in *gettytab*.

Programs

getty(8)

See Also

termcap(5)

Name

/etc/group — group file

Format

%s:%s:%d:[^\n]\n

Description

This ASCII file contains the following information for each group:

- group name
- encrypted password (OBSOLETE)
- numerical group ID
- comma-separated list of users allowed in the group

Each line describes one group; fields are separated by colons.

The file maps group names (the first field) to numerical group identification numbers (third field) and vice versa. The *ls*(1), for example, uses this file.

Example

```
daemon:*:1:smith,jones
```

Group *daemon* (with * in the obsolete password field) is also known as group ID "1". Users *smith* and *jones* are in this group in addition to the group specified for them in */etc/passwd*.

Programs

Several programs use the */etc/group* file.

Caveats

Users become members of groups in the following ways:

- having the group number listed in the */etc/passwd* file
- having their user name listed in the */etc/group* file

See Also

/etc/passwd

bill(1), *setgroups*(2), *initgroups*(3X), *crypt*(3), *passwd*(1), *passwd*(5), *getgrent*(3), *getgrgid*(3), *getgrname*(3), *setgrent*(3), *endgrent*(3), *getgroups*(2)

Name

/etc/hosts — host name database

Format

%s [^\n]\n

Description

The *hosts* file contains information on known hosts on the DARPA Internet. For each host, a single line containing the following information must be present:

- official host name
- Internet address (in 4-octet format)
- aliases by which the host is known

Blanks or tabs separate items. Comments start with #; characters from # to the end of the line are not interpreted by routines searching the file. The */etc/hosts* file is created from the official host database maintained at the Network Information Control Center (NIC). Make local changes to add unofficial aliases or unknown hosts (for example, your own local host name).

Network addresses are specified in the conventional "." notation using the *inet_addr* routine from the Internet address manipulation library, *inet*(3N). Host names can contain any printable character except a field delimiter, newline, or comment character.

Example

The following example shows the local portion of the */etc/hosts* database. The lines following it in */etc/hosts* come from NIC distribution tables or from the CONVEX distribution.

```
# Berkeley Host Database
127.1  localhost
#  Convex 10Mb/s local Ethernet
100.0.1.0x82  convexs-ex   convexs sw
100.0.0.1     convexe-ex
100.0.0.6     convex1-ex   convex1
```

The *127.1 localhost* line defines a host address (for this host) used for testing, loopbacks, and some daemons.

The local hosts (100.*) are members of the local Ethernet. The "100" prefix choice is arbitrary, but choices should be compatible with existing host numbers.

When an unknown name is requested, the *arp*(4) name-resolution protocol broadcasts packets requesting identification.

Programs

ftp(1), *rcp*(1), *rlogin*(1), *sendmail*(8)

Bugs

Use a name server instead of a static file. A binary indexed file format should be available for fast access.

See Also

gethostent(3N), *arp*(4)

CONVEX Network File System System Manager's Guide

Name

/usr/adm/lpd-acct — printer accounting file

Format

%d%d%s%s%s\n

Description

Printer accounting is done by the print filters. There can be more than one printer accounting file, as determined by */etc/printcap*. However, most ConvexOS filters use */usr/adm/lpd-acct*. For consistency, these filters use the same log routine. Accounting data in this format can be summarized by *awk* scripts as described in *sumscripts*(8).

Example

```
#include <stdio.h>

log(acctfile, pages, userinfo, logname, host)
char   *acctfile;
int    pages;
char   *userinfo;
char   *logname;
char   *host;
{
       FILE   *fptr;

       if (acctfile == NULL)/* missing from filter argument list */
              return;
       if ((fptr = fopen (acctfile, "a")) == NULL)
              return;
       fprintf (fptr, "%10d %6d %s %8s %8s\n",
                           time (0), pages, userinfo, logname, host);
       fclose (fptr);
}
```

The arguments *acctfile*, *logname*, and *host* are generated as described by the "Berkeley 4.2BSD Line Printer Spooling Manual" in the *CONVEX Tutorial Papers*. The *pages* argument is computed by the filter. In ConvexOS, if the *pu* capability (propagate user information) is present in a printer's */etc/printcap* entry, *lpd* passes the command-line arguments *-u userinfo* to the printer's filter(s). The *userinfo* portion can then be passed to the log routine.

See Also

bill(1), *printcap*(5), *lpd*(8), *pac*(8), *sumscripts*(8)

Name

/etc/motd — the message of the day

Format

ASCII text

Description

The *login* program displays */etc/motd* when a user who has not seen the message logs on. The file only displays one time for each user login.

Use */etc/motd* to send messages of general interest to all users.

Example

The following message is an example of an */etc/motd* file entry:

```
9/10/82: The system will be down all day Saturday for updates.
```

Programs

login(1)

See Also

/etc/nologin

Name

/etc/nologin — inhibit logins and print message

Format

ASCII text

Description

If an */etc/nologin* file exists, *login*(1) does the following:

- denies login to everyone except root
- prints contents of */etc/nologin* file as the reason for login denial

Example

The following message is an example */etc/nologin* file:

```
System unavailable -- operator performing special file backup
operations.  Try again at 17:00 hours.
```

Programs

login(1), *shutdown*(8)

Name

/etc/nurc — *nu* defaults database

Format

%s:%s

Description

The */etc/nurc* file contains default constants for the *nu*(8) utility. The *nu* utility adds new users to the system; *nu* uses the constants defined in */etc/nurc* if no other values are provided and if an alternate default file is not specified in place of */etc/nurc*.

Lines in the *nurc* file have the form

```
tag: value
```

tag is the parameter to modify and *value* is the default constant for the parameter. The tags and their default values are as follows:

Name	Default	Description
uid	(see below)	user ID
gid	EMPTY	group ID
group	staff	group name (used if no gid field)
directory	/mnt	path for home directory
protection	0755	home directory protection
shell	/bin/csh	login shell
password	(see below)	new user's initial password
username	username	user's full name (for *finger*)
office	office	user's office (for *finger*)
extension	extension	user's work extension
homephone	homephone	user's home phone number
minwks	1	minimum # weeks for password aging
maxwks	52	maximum # of weeks for password aging
diskquota	6000,8000,1200,1500	*blksoft, blkhard, inodesoft, inodehard* quotas
skeleton	/usr/skel	directory of files to copy to home dir
homedir	<*directory*>/<*login*>	home dir to create for user
typed	OFF	will user have password typing restrictions?
aged	OFF	will user have password aging restrictions?
quota	OFF	will home dir filesystem have quotas initialized?
nouidfile	OFF	ignore contents of */etc/uidcount*?

For more information on these parameters, see the *nu*(8) man page.

Example

The following is an example *nurc* file:

```
directory: /mnt
protection: 0700
shell: /bin/csh
group: swtst
username:
typed
quota
diskquota: 3000, 4096, 1000, 1500
```

In the example, new user's directories are created in the */mnt* directory, which is readable, writable, and executable only by the user (*protection: 0700*). The default group is *swtst*. Typing restrictions are activated (although password aging restrictions are not), and the home directory file system has quotas initialized.

See Also *chfn*(1), *finger*(1), *nurc*(5), *passwd*(5), *nu*(8)

Name

/etc/op.access — *op* (operator) access file

Format

%s %s [%s [%s]]; [%s [%s]]

Description

The *op.access* file describes the functions a user can perform by invoking the *op* utility, how these functions are performed, and who is allowed to perform them. Because *op.access* is used to grant command access to commands that require superuser privileges, this file should be owned by root with mode 0400 (read only).

Each entry in the *op.access* file describes a single function. Lines beginning with a pound sign (#) are comments. If a line begins with a tab or spaces, it is considered part of the previous line. The entries describe the following:

- name or mnemonic of the operator function
- command line (full pathname) to execute to perform this function
- *uid* to set (root by default)
- *gid* to set (not changed, by default)
- directory to *chdir* (not changed, by default)
- root directory to set with *chroot* (not changed, by default)
- *umask* to set (022 by default)
- list of groups allowed to execute the function (none by default)
- list of users allowed to execute the function (superuser only, by default)
- range of valid arguments for the command (any value per variable argument by default)
- environment variable settings (none by default)

Entry fields in *op.access* are of the form

mnemonic command [*arg*[...]]; [*option*[...]]

mnemonic — unique identifier for the operator function

command — full pathname of command executed when the *mnemonic* is called with *op*

arg — literal or variable arguments needed by *command*

option — option or set of options restricting who can use *command* or how *command* is invoked. The format for *option* is

keyword=*value*[, ...]

keyword is the name of the option; *value* is the value assigned to the option.

Example

```
# define the site defaults; we want the people in 'operators' group to be
# able to execute almost everything; easier here than on every line.
DEFAULT     groups=operators
# filesystem backups
weekly      /etc/dump 0Gun $1; users=snow,white $1=/,/usr,/mnt,/project
daily       /etc/dump 5Gun $1; users=snow,white $1=/,/usr,/mnt,/project
#
# taking system down; $1 is good use of regular expressions
shutdown    /etc/shutdown -h $1 $2; $1=now,+[0-9]*,[0-9]*:[0-9]*
reboot      /etc/shutdown -r $1 $2; $1=now,+[0-9]*,[0-9]*:[0-9]*
#
# kill all batch procs so system can be restarted
# (want to override default groups setting given at top, allowing none)
killbatch   /usr/convex/qmgr shutdown 0; groups= users=bashful,happy
startbatch  /usr/lib/nqs/nqsdaemon;
#
# start disco daemon
disco       /etc/opbin/start_disco; uid=disco gid=proj dir=/scratch
            umask=027 groups=geo,disco users=snoopy,woodstck
            $USER=disco $SHELL=/bin/shell
#
# let particular people mount and unmount the removable drive
rdsmount    /etc/mount $1 $2; groups=operators,swdev,disco dir=/
            users=bob,steve $1=/dev/dd0.$2=/.*
rdsumount   /etc/umount $1; groups=operators,swdev,disco dir=/
            users=bob,steve $1=/dev/dd0.
```

From this example, users with operator privileges could use the following commands:

```
op weekly /mnt
op reboot 17:30 "We have to fix our network."
op disco
op rdsmount /dev/dd0c /mnt/me/mystuff
op mounted 3 8688
```

Programs

op(8)

Caveat

For security reasons, this file is shipped as read-only by root (mode 0400); do not change these permissions.

See Also

regex(3)

Name

/etc/passwd — password file

Format

ASCII: %s:%s:%s:%s:%s:%s:%s\n

Description

Each single-line entry in the ASCII */etc/passwd* file maps a sign-on name to the following seven fields:

1. name
2. encrypted password
3. numerical user identification number
4. group identification number
5. full user name, office, extension, and home phone number
6. user's home directory
7. user's default command interpreter

The ASCII file contains an entry for each user on the system. Colons separate the fields on each line. If the password field is empty, no password is required. A null command interpreter field defaults to the */bin/sh* interpreter.

The *nu*(1) program appends new users to the */etc/passwd* file. The following programs modify the user information when necessary:

- *chsh*(1) changes the user's shell
- *chfn*(1) changes the user's identification field
- *passwd*(1) changes the user's password

Example

```
smith:2sadfNQDlxsMg:11:19:Rob &,22A,228,3823133:/mnt/smith:/bin/csh
```

In this example, Rob Smith has the sign-on name *smith* (first field). The second field is his encrypted password. His user identification number is 11; his group ID is 19. His full name is Rob Smith (the & causes the user name to be substituted appropriately); his office is 22A with office phone 228 and home phone 382-3133. His home directory is */mnt/smith*, and he uses the standard C shell command interpreter.

The fifth field (user identification) has four comma-separated subfields.

Programs

ls(1), *finger*(1), *passwd*(1), *nu*(1), *chfn*(1), *chsh*(1), *vipw*(8), *login*(1)

Caveats

System users are generally permitted read privileges to */etc/passwd* because the passwords are encrypted. Write permission should be reserved for the superuser. Read permission is necessary because programs use the */etc/passwd* file to map user IDs to user names.

The password file can be edited with a text editor, but it is important to ensure that only one person at a time edits the file. The editing programs *chfn*(1), *chsh*(1), *passwd*(1), and *vipw*(8) are preferred because they perform the required file locking. Be aware that writing out an ill-formatted *passwd* file can create problems with programs that use the ill-formatted lines and may introduce security holes.

See Also

/etc/group

bill(1), *crypt*(3), *group*(5), *getpwent*(3), *getpwuid*(3), *getpwnam*(3), *setpwent*(3), *endpwent*(3)

Name

/etc/phones — remote host phone number database

Description

The file */etc/phones* contains the system-wide private phone numbers for the *tip*(1C) program. This file is normally unreadable, and so may contain privileged information.

Format

%s %[^\n]\n

Each line contains a system name followed by a phone number. The system name is one of the names defined in the *remote*(5) file; the phone number is constructed from [0123456789-=*%K]. The =, K, and * characters cause the auto-call units to pause and wait for a second dial tone (when going through an exchange). VADIC modems use the K character.

Only one phone number per line is permitted. If more than one line in the file contains the same system name, *tip*(1C) attempts to dial each one in turn until a connection is made.

Example

```
orpheus    9K7654321
```

Programs

tip(1C)

Name

/etc/printcap — printer capability database

Format

Same as *termcap*.

Description

printcap is a *termcap*-style file that describes your system's set of line printers. The spooling system reads the *printcap* file each time a file is to be printed, so you can add and delete printers dynamically. Each entry in the file describes one printer.

Refer to the "4.2BSD Line Printer Spooler Manual" in the *CONVEX UNIX Tutorial Papers* for instructions on setting up an entry for a particular printer.

Capabilities

Refer to *termcap*(5) for a description of the file layout.

Name	Type	Default	Description
af	str	NULL	name of accounting file
br	num	none	if *lp* is a tty, set baud rate (*ioctl*)
cf	str	NULL	cifplot data filter
df	str	NULL	TEX data filter (DVI format)
fc	num	0	if *lp* is a tty, clear flag bits (*sgtty.h*)
ff	str	\f	string to send for form feed
fo	bool	false	print a form feed when device is open
fs	num	0	like *fc* but set flag bits
gf	str	NULL	graph data filter (*plot* format)
ic	bool	false	driver supports (nonstandard) *ioctl* to indent printout
if	str	NULL	name of text filter that does accounting
lf	str	/dev/console	error logging filename
lo	str	lock	name of lock file
lp	str	fI/dev/lp	device name to open for output
mx	num	1000	max file size (in BUFSIZ blocks), 0= nolimit
nd	str	NULL	next dir for list of queues (N/A)
nf	str	NULL	*ditroff* data filter (device-independent *troff*)
of	str	NULL	name of output filtering program
pl	num	66	page length (in lines)
pu	bool	false	propogate user information to filters
pw	num	132	page width (in characters)
px	num	0	page width in pixels (horizontal)
py	num	0	page length in pixels (vertical)
rf	str	NULL	filter for printing FORTRAN style text files
rm	str	NULL	machine name for remote printer
rp	str	lp	remote printer name argument
rs	bool	false	restrict remote users to those with local accts
rw	bool	false	open printer device for reading/writing
sb	bool	false	short banner (one line only)
sc	bool	false	suppress multiple copies
sd	str	/usr/spool/lpd	spool directory
sf	bool	false	suppress form feeds
sh	bool	false	suppress printing of burst page header
st	str	status	status filename
tf	str	NULL	*troff* data filter (cat phototypesetter)
tr	str	NULL	trailer string to print when queue empties
vf	str	NULL	raster image filter
xc	num	0	if *lp* is a *tty*, clear local mode bits (*tty*)
xs	num	0	like *xc* but set bits

The *CONVEX UNIX Programmer's Manual* describes these capabilities.

Example

```
fast|Fast Imagen serial printer:\
        :af=/usr/adm/imagen:\:br#19200:\
        :df=/usr/local/lib/idvi:fc#0177777:ff=\00:fo=0:\
        :fs#040:gf=/usr/local/lib/igraph:if=/usr/local/lib/ipf:\
        :lf=/usr/spool/nipd/ilog:lo=lock:lp=/dev/imagen2:\
        :mx=#10000:nf=/usr/local/lib/idimp:pl#60:pw#80:\
        :px#2016:py#2624:rf=/usr/local/lib/ifort:rw:\
        :sb=CONVEX Computer Corporation:sc:sd=/usr/spool/nipd2:\
        :sh:st=/usr/spool/nipd2/pstatus:vf=/usr/local/lib/imp:
```

fast	printer name stands for "fast Imagen printer"
af	accounting information goes to the */usr/adm/imagen* file
br	baud rate is 19200
df	text data filter */usr/local/lib/idvi* preprocesses the data before it is printed when you use the *lpr* command and specify the TEX option
fc	all *sgtty.h* style bits are cleared on startup
ff	form-feed strings are nulls
fo	no form feed is sent when the device is opened
fs	flag bits 040 (octal) are set on startup
gf	graphics filter */usr/local/lib/igraph* preprocesses the data before it is printed when you use the *plot*(3X) command
if	accounting data is processed by filter */usr/local/lib/ipf*
lf	errors are logged to the */usr/spool/nipd/ilog* file
lo	lock file is *lock* (default)
lp	printer name is */dev/imagen2*
mx	maximum file size is 10,000 blocks
nf	*ditroff* output goes through the */usr/local/lib/idimp* filter
pl	page length is 60 lines
pw	page width (line length) is 80 characters
px	page width (horizontal dimension) is 2016 pixels
py	page height (vertical dimension) is 2624 pixels
rf	FORTRAN listings go through the */usr/local/lib/ifort* filter
rw	line opens in read/write mode
sb	short banner is "CONVEX Computer Corporation"
sc	multiple copies suppressed
sd	spool directory is */usr/spool/nipd2*
sh	burst page headers suppressed
st	status messages appear in */usr/spool/nipd2/pstatus*
vf	raster images go through the */usr/local/lib/imp* filter

Programs *lpc*(8), *lpd*(8), *pac*(8), *lpr*(1), *lpq*(1), *lprm*(1)

Caveat Be sure to specify # or = carefully, depending on whether a parameter has a numerical or string value.

See Also *lpd-acct*(5), *termcap*(5), *lpd*(8)

"4.2BSD Line Printer Spooler Manual" in the *CONVEX UNIX Tutorial Papers*

Name

/etc/pwrestrict — password restrictions file

Format

ASCII: %s:%s:%s:%s:%s\n

Description

Each single-line entry in */etc/pwrestrict* maps a sign-on name to the following fields:

1. name
2. encrypted password
3. password aging information
4. password type flag (Y or N)
5. numerical user identification number

The *pwrestrict* file contains an entry for each system user. Colons separate the fields on each line. If the aging information field is empty, password aging is not enforced.

The *nu*(1) program appends new users to the *pwrestrict* file. *passwd*(1) modifies the user's password and the password aging information.

Example

```
smith:ENEaEinahyIAo:1,2,Oct 13 1986:Y:469
```

In this example, the sign-on name is *smith* (first field). The second field is the encrypted password. The password age field denotes that the password was last changed on Oct 13 1986; the user cannot change the password for one week (from the given date) and is forced to change the password after Oct 27 1986 (2 weeks after the last-changed date). The fourth field is a flag (Y or N); if the flag is N, the password can be composed of any sequence of characters. If the flag is Y, the user's password must follow the guidelines for passwords described in *passwd*(1). The user identification for *smith* is 469 (the last field).

Programs

passwd(1), *pwage*(1), *nu(8)*, *vipw*(8), *login*(1)

Caveats

Users are permitted read privileges to */etc/pwrestrict* because the passwords are encrypted. Reserve write permission for the superuser. Read permission is necessary for programs that use the */etc/pwrestrict* file to map user IDs to user names.

The *pwrestrict* file can be edited with a text editor, but only one person should edit the file at a time. The editing programs *passwd*(1), *vipw*(8), and *nu*(8) are preferred because they perform the required file locking. Writing an ill-formatted *pwrestrict* file can cause problems with programs that use the ill-formatted lines. If inconsistencies are found between the */etc/passwd* file and the */etc/pwrestrict* file, information from the password file is used; the restriction information is ignored.

See Also

/etc/passwd, */etc/group*

crypt(3), *group*(5), *getpwrestent*(3), *getpwrestuid*(3), *getpwrestnam*(3), *setpwrestent*(3), *endpwrestent*(3)

Name

/etc/rc.local — system-specific startup information

Format

Shell script

Description

The system startup routine processes the */etc/rc.local* file after mounting the file systems but before starting the daemons. The *rc.local* file contains shell commands that perform the following tasks:

- set the host name
- set up networks
- check quotas
- save core dumps
- start local daemons
- remove temporary files after crashes

Example

```
/bin/hostname convex
/etc/ifconfig ex0 convex-ex up arp -trailers   >/dev/console
echo -n 'check quotas: '                        >/dev/console
      /usr/etc/quotacheck -a
echo 'done.'                                   >/dev/console
/usr/etc/quotaon -a
echo -n 'local daemons:'                        >/dev/console
if [ -f /etc/routed ]; then
      /etc/routed & echo -n ' routed'           >/dev/console
fi
if [ -f /etc/rwhod ]; then
      /etc/rwhod & echo -n ' rwhod'             >/dev/console
fi
if [ -f /usr/local/bin/stat ]; then
      /usr/local/bin/stat&
      echo -n ' statistics'                     >/dev/console
fi
if [ -f /usr/lib/nqs/nqsdaemon ]; then
      /usr/lib/nqs/nqsdaemon                    >/dev/console
      echo -n ' cxbatch'                        >/dev/console
fi
echo '.'                                       >/dev/console
rm -f /tmp/Emacs* /tmp/queue?/*
rm -f/usr/spool/notes/.locks/*
/usr/convex/spu -r /mnt/os/vmunix > /vmunix
/usr/convex/spu -r /mnt/errlog > /errlog.back
```

See Also

Descriptions of the daemons, editor temporary files, and administrative support programs.

Name

/etc/remote — remote host description file

Format

termcap style

Description

Systems known by *tip*(1C), and their attributes, are stored in the *termcap*-style file */etc/remote*. Each line in the file describes a remote host.

The first entry is the name of the host system; vertical bars separate multiple names for a single system.

The *tip* program and the *cu* interface to *tip* use entries named "tip*" and "cu*" for defaults as follows. When *tip* is invoked with only a phone number, it looks for an entry of the form "tip300", where "300" is the baud rate with which the connection is to be made. When you use the *cu* interface, it looks for entries of the form "cu300".

Capabilities

Capabilities are either strings (str), numbers (num), or Boolean flags (bool). A string capability is specified by *capability=value*, for example, *dv=/dev/harris*. A numeric capability is specified by *capability#value*, for example, *xa#99*. A Boolean capability is specified (as true) by listing the capability.

Refer to *termcap*(5) for a description of the file layout.

Name	Type	Default	Description
at	str		auto call unit type
br	num	300	baud rate
cm	str		initial connection message to be sent to remote host.
cu	str	(dv)	Call unit if making a phone call
di	str		Disconnect message sent to host on user request
du	bool		host is on a dialup line
dv	str		UNIX device(s) to open to establish a connection
el	str	NULL	Characters marking an end-of-line
fs	str	BUFSIZ	Frame size for transfers
hd	bool		The host uses half-duplex communication (do local echo)
ie	str	NULL	Input end-of-file marks
oe	str	NULL	Output end-of-file string
pa	str	even	parity used when sending data to the host
pn	str		Telephone number(s) for the host
tc	(str)		Continue capability at *string* in named description

Example

```
UNIX-1200:\
        :dv=/dev/cua0:el=^D^U^C^S^Q^O@:du:at=ventel:ie=#$%:oe=^D:\
        :br#1200:
arpavax|ax:\
        :pn=7654321%:tc=UNIX-1200:
```

The *UNIX-1200* device (*dv*) uses */dev/cua0*, which connects to a ventel modem (*at*) on a dialup (*du*) line. Any of the entire line (*el*) list characters (^D, ^U, ^C, ^S, ^Q, ^O) cause the entire line and the character to be sent to the remote host. The connect program sends an output end-of-file (*oe*) string ^D after transferring a file. The characters #, $, and % signify end-of-input-file when preceded by \n (shell prompts). The *arpavax* name specifies a phone number (*pn*) to dial and then the standard *UNIX-1200* characteristics.

Programs

tip(1C)

Caveats

If you use *dv* to refer to a terminal line, *tip* tries to open it for exclusive use.

The ~ (tilde) escapes are only recognized by *tip* after an *el* character or a carriage return.

You can set parity to even, odd, none, zero (always set bit 8 to zero), or 1 (always set bit 8 to 1).

If the phone number (*pn*) field contains an @ sign, *tip* searches the */etc/phones* file for a list of telephone numbers.

Bugs

The *ie* facility for specifying input EOF does not work with all files.

See Also

phones(5)

Name

/usr/adm/shutdownlog — list of voluntary system shutdowns

Format

ASCII

Description

Each line in the shutdown log notes the time your machine was shut down, by whom, and the reason (if specified in the *shutdown* invocation).

Example

```
19:15  Sat Oct 12, 1985.  Halted.
13:25  Mon Oct 14, 1985.  Shutdown: for 20 min (by convex!root)
13:25  Mon Oct 14, 1985.  Halted.
```

In this example, the machine was down twice in three days.

Programs

shutdown(8).

Caveat

The *shutdownlog* can become quite large, but is important to keep.

Name */usr/adm/stat/** — system statistics

Format Binary

Description Each file in the */usr/adm/stat* directory corresponds to a date. The statistics gathered by the */etc/stat program for that date reside in the corresponding stat* file. The */etc/stat* program awakens approximately every 30 seconds and samples system use, disk activity, load average, number of users signed on, etc, then writes the data to the current file in the */usr/adm/stat* directory. The format of the dates is *mm.dd.yy* (lower months and days are single digit). Use the *seestat*(8) program to examine the contents of the *stat* files.

cron usually runs *seestat* early in the morning.

Programs *seestat*(8)

Caveat The *stat* files use a lot of disk space; archive them monthly.

Name

/etc/stripecap — striped disk partition description database

Format

Same as *termcap*.

Description

The */etc/stripecap* file is a database that describes striped disk partitions. The *putst*(8) and *newst*(8) utilities use the *stripecap*.

Striped disk partitions are logical disk partitions spanning several physical disk partitions. Striped disk partitions exploit performance improvements made possible by the parallel operation of the several disk arms. UNIX file systems can be mounted on striped disk partitions just as with conventional disk partitions.

Striped partitions are described in the */etc/stripecap* file by a *termcap*-style descriptor. This descriptor contains information on the physical disk partitions that constitute the striped partition and on the layout of the logical sections of the stripes.

The *newst*(8) and *putst*(8) utilities assume each stripe partition has a name in the format *stX*, where *X* is a numerical digit corresponding to the minor device number of the stripe disk pseudo device */dev/rstX*.

Capabilities

Refer to *termcap*(5) for a description of the file layout.

Name	Type	Description
np	num	number of partitions constituting the stripe file system
M?	num	major device number of partition ?
m?	num	minor device number of partition ?
D?	num	number of devices (partitions) in section ?
B?	num	"stripe blocksize" (interleave factor) of section ?
S?	num	number of blocks in section ? contributed by each partition

Example

To update the */etc/stripecap* file, use the following command sequence:

```
/etc/newst /dev/rst6 da4a dkd-001 da4h dkd-001 da4g dkd-001
```

The output is the following *stripecap* entry:

```
st6:\                                  Name of this stripe device.
   :np#3:\                             Number of partitions included (three).
   :M0#5:m0#38:\                       Device numbers of three partitions
   :M1#5:m1#39:\                         sorted from largest size to
   :M2#5:m2#32:\                           smallest size.
   :D0#3:B0#37800:S0#128:\             First section uses all three partitions.
   :D1#2:B1#188100:S1#128:\            Second section uses first two partitions.
   :D2#1:B2#115200:S2#128:\            Third section uses first partition.
```

To continue entries onto multiple lines, specify \ as the last character of a line. You can include empty fields (adjacent colons) for readability (for example, between the last field on a line and the first field on the next).

All fields have names that are two-character codes. For most field names, the second character is uniquely chosen from the set [0-9a-zA-Z] in an ascending sequence. For example, the minor device numbers of the first five disk partitions are *m0*, *m1*, *m2*, *m3*, and *m4*.

The name of this stripe device is *st6*, which means that */dev/st6* is the name on which to mount the file system. It includes three partitions on device number pairs 5,38 and 5,39 and 5,32. The order of these is important: 5,38 must be the largest partition; 5,32 must be the smallest. The stripe system interleaves all three partitions until the third one's space is exhausted. It then uses the first two partitions, and finally only the first partition (assuming unequal sizes of partitions, which is not necessarily the case).

In this example, the first section uses parts of all three partitions (for the first 37,800 sectors). The *S?* parameter directs the striping system to take the first 128 sectors from 5,38; the next 128 sectors from 5,39; then the next 128 sectors from 5,32. The fourth set of 128 sectors comes from 5,38 (again), and the cycle continues until 37,800 sectors have been used from each partition.

The second section repeats the sequence by using 128-sector chunks from only the first and second partitions. The third section uses only the remaining space on the first partition.

Programs *newst*(8), *putst*(8), *st*(4)

Name

/etc/syslog.conf — *syslog* configuration file

Format

facility.level; send_message_here

Description

Entries in the */etc/syslog.conf* file determine the types of messages to be logged by *syslogd*(8) and where to log the messages. Each entry has a selector that determines the messge priorities and an action. The *facility* and *level* must be separated by a dot and each entry must end with a semicolon. The facilities available are as follows:

kern	messages generated by the kernel
user	messages generated by the user processes
mail	messages generated by the mail system
daemon	messages generated by system daemons
auth	messages generated by the authorization system
lpr	messages generated by the lineprinter spooling system
local 0-7	reserved for local use

More than one facility can be selected, separated by commas. Use an asterisk (*) to indicate all facilities.

The levels available are as follows:

emerg	panic condition; normally to all users
alert	condition to be corrected immediately; e.g., corrupt database
crit	critical conditions; e.g., hard-device errors
err	errors
warning	warning messages
notice	conditions that are not errors but require condition
info	informational messages
debug	information useful for debugging a program

The following format alternatives can be used for the message destination (that is, where the message is to be logged):

- absolute pathname
- hostname preceded by @ sign
- list of usernames, separated by commas (users receive messages if they are logged on)
- asterisk (*) to send message to all logged-in users

Caveat

If you modify the default *syslog.conf* file, you must kill and restart *syslogd* after making the changes.

Example

```
*.err;kern.debug;auth.notice                    /dev/console
kern.debug;daemon,auth.notice;*.err;mail.crit    /usr/adm/messages
lpr.debug                                       /usr/adm/lpd-errs
mail.debug                                      /usr/spool/mqueue/syslog
*.alert;kern.err;daemon.err                     operator
*.emerg                                         *
```

Programs

syslogd(8), *logger*(1)

Name

/usr/adm/tp-acct — tape allocation accounting file

Format

%s%d[%s]%d%d%d%s\n

Description

The */usr/adm/tp-acct* file contains a record of tape allocations and deallocations performed by *tpalloc* and *tpdealloc*. Each successful allocation or deallocation writes a line to */usr/adm/tp-acct* containing the following fields:

1. event — *allocated* or *deallocated*
2. time — time event occured, as returned by *time*(3)
3. device — physical device allocated (present only if *event* is 'allocated')
4. uid — user ID of allocator at time of allocation
5. gid — group ID of allocator at time of allocation
6. aid — activity ID of allocator at time of allocation
7. allocation file — tape allocation name as given in */etc/tapecap*

See Also

tpalloc(1), *tapecap*(5), *sumscripts*(8)

Name

/etc/ttys — terminal initialization data

Format

%s [^\n]\n

Description

init(8) reads the *ttys* file to determine the terminal files for which to create getty processes that enable users to login. Each file is on a separate line in the *ttys* file. Fields are separated by tabs or spaces; if a field contains more than one word, the words must be enclosed in double quotation marks. Blank lines and comments can appear anywhere in the file; comments are delimited by the # symbol and a new line.

The first field is the entry in */dev* for the terminal.

The second field is the command, usually *getty*, to execute for the line. *getty* performs baud-rate recognition, reads the login name, and calls login, among other tasks. The second field can be a command, for example, to start up a window system terminal emulator or another daemon process; *getty* is the most commonly used, but the field can contain any command to be executed.

The third field is the terminal type normally connected to the line, as listed in *termcap*(5).

The remaining fields set flags in the *ty_status* entry, see *getttyent*(3), or specify a window system process to be maintained by *init*(8) for the terminal line. The 'on' and 'off' options specify whether *init* is to execute the command in the second field. The 'secure' option, used with 'on', allows root to login on the line. You can specify the 'dialup' or 'uucp' option after 'secure'. The 'dialup' option requires all users to enter a dialup password when logging on to that tty. The 'uucp' option allows a user with UID uucp to log in to that tty. Ttys designated as 'dialup' do not have to be named in the ttyd? format. Do not use quotation marks in the flag fields. The 'window=' option is followed by a quoted command string that *init* executes before starting *getty*. If a line ends in a comment, the comment is included in the *ty_comment* field of the *ttyent* structure.

Example

```
console "/etc/getty std.1200"  vt100      on secure  .
tty00   "/etc/getty d1200"     vt100      on dialup  # 555-1234
ttyh0   "/etc/getty std.9600"  hp2621-nl  on         # 254MC
ttyh1   "/etc/getty std.9600"  plugboard  on         # Sharon's office
ttyp0   none                   network
ttyp1   none                   network    off
ttyv0   "/usr/new/xterm -L :0" vs100      on         window="/usr/new/Xvs100 0"
tty01   "/etc/getty std.9600"  vt100      on secure  esmark <engr> # eng tty
tty02   "/etc/getty std.9600"  vt100      on secure  mcbroom !<engr> # mkt tty
```

The first line permits root login on the console at 1200 baud. The second line allows dialup at 1200 baud without root login and requires a dialup password. The third and fourth lines allow logins at 9600 baud with terminal types of hp2621-nl and plugboard. The fifth and sixth lines are examples of network pseudo–ttys, on which *getty* is not enabled. The seventh line is a terminal emulator and window system startup entry. The eighth line gives user esmark and members of the engr group access to tty00. The ninth line gives user mcbroom and all users except those in the engr group access to tty01.

Programs *init*(8)

---

q.DF **Caveats** *getlogin*(3) and the */etc/utmp* file rely on the order of */etc/ttys*. Changing the order (even inserting new entries in the middle of the file) invalidates some file entries (for example, */etc/utmp*), causing *getlogin* to return erroneous results. Adding entries to the end of the file does not cause these problems.

If ConvexOS is in multi-user mode when you change */etc/ttys* (for example, turn a teletype on or off or change a baud rate), you must use the **kill -1 1** signal to inform *init*.

See Also */etc/utmp*

gettytab(5), *getty*(8), *login*(1), *on*(1), *off*(1), *getlogin*(3), *init*(8)

Name

/usr/adm/wtmp — login records

Format

Binary

Description

The */usr/adm/wtmp* file records all logins and logouts. A null user name indicates a logout on the associated terminal (it is still easy to pair logins and logouts using the terminal name as the key). The terminal name ~ indicates that the system was rebooted at the indicated time.

The accounting program *ac*(8) summarizes */usr/adm/wtmp* and other accounting files.

Programs

last(1), *login*(1), *init*(8), *who*(1), *ac*(8), *telnetd*(8), *rlogind*(8)

Caveats

The */usr/adm/wtmp* file is not created by a program; if it is removed, login accounting ceases.

The file */usr/adm/wtmp* has no size limits. To split the file, divide it at a *utmp* structure boundary. Another method is to rename */usr/adm/wtmp* to */usr/adm/wtmp.old* occasionally and copy */dev/null* to */usr/adm/wtmp*.

See Also

/usr/include/utmp.h

utmp(5)

B

UNIX File System Check Program

This appendix consists of the technical paper, "Fsck — The UNIX File System Check Program," by Marshall McKusick and T.J. Kowalski. Refer to the *fsck*(8) manual page for a briefer explanation of the program.

Fsck – The UNIX† File System Check Program

Revised July 28, 1983

Marshall Kirk McKusick

Computer Systems Research Group
Computer Science Division
Department of Electrical Engineering and Computer Science
University of California, Berkeley
Berkeley, CA 94720

T. J. Kowalski

Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

This document reflects the use of *fsck* with the 4.2BSD file system organization. This is a revision of the original paper written by T. J. Kowalski.

File System Check Program (*fsck*) is an interactive file system check and repair program. *Fsck* uses the redundant structural information in the UNIX file system to perform several consistency checks. If an inconsistency is detected, it is reported to the operator, who may elect to fix or ignore each inconsistency. These inconsistencies result from the permanent interruption of the file system updates, which are performed every time a file is modified. Unless there has been a hardware failure, *fsck* is able to repair corrupted file systems using procedures based upon the order in which UNIX honors these file system update requests.

The purpose of this document is to describe the normal updating of the file system, to discuss the possible causes of file system corruption, and to present the corrective actions implemented by *fsck*. Both the program and the interaction between the program and the operator are described.

---

†UNIX is a trademark of Bell Laboratories.

This work was done under grants from the National Science Foundation under grant MCS80-05144, and the Defense Advance Research Projects Agency (DoD) under Arpa Order No. 4031 monitored by Naval Electronic System Command under Contract No. N00039-82-C-0235.

TABLE OF CONTENTS

1. Introduction

This document reflects the use of *fsck* with the 4.2BSD file system organization. This is a revision of the original paper written by T. J. Kowalski.

When a UNIX operating system is brought up, a consistency check of the file systems should always be performed. This precautionary measure helps to insure a reliable environment for file storage on disk. If an inconsistency is discovered, corrective action must be taken. *Fsck* runs in two modes. Normally it is run non-interactively by the system after a normal boot. When running in this mode, it will only make changes to the file system that are known to always be correct. If an unexpected inconsistency is found *fsck* will exit with a non-zero exit status, leaving the system running single-user. Typically the operator then runs *fsck* interactively. When running in this mode, each problem is listed followed by a suggested corrective action. The operator must decide whether or not the suggested correction should be made.

The purpose of this memo is to dispel the mystique surrounding file system inconsistencies. It first describes the updating of the file system (the calm before the storm) and then describes file system corruption (the storm). Finally, the set of deterministic corrective actions used by *fsck* (the Coast Guard to the rescue) is presented.

2. Overview of the file system

The file system is discussed in detail in [Mckusick83]; this section gives a brief overview.

2.1. Superblock

A file system is described by its *super-block*. The super-block is built when the file system is created (*newfs*(8)) and never changes. The super-block contains the basic parameters of the file system, such as the number of data blocks it contains and a count of the maximum number of files. Because the super-block contains critical data, *newfs* replicates it to protect against catastrophic loss. The *default super block* always resides at a fixed offset from the beginning of the file system's disk partition. The *redundant super blocks* are not referenced unless a head crash or other hard disk error causes the default super-block to be unusable. The redundant blocks are sprinkled throughout the disk partition.

Within the file system are files. Certain files are distinguished as directories and contain collections of pointers to files that may themselves be directories. Every file has a descriptor associated with it called an *inode*. The inode contains information describing ownership of the file, time stamps indicating modification and access times for the file, and an array of indices pointing to the data blocks for the file. In this section, we assume that the first 12 blocks of the file are directly referenced by values stored in the inode structure itself†. The inode structure may also contain references to indirect blocks containing further data block indices. In a file system with a 4096 byte block size, a singly indirect block contains 1024 further block addresses, a doubly indirect block contains 1024 addresses of further single indirect blocks, and a triply indirect block contains 1024 addresses of further doubly indirect blocks.

In order to create files with up to 2↑32 bytes, using only two levels of indirection, the minimum size of a file system block is 4096 bytes. The size of file system blocks can be any power of two greater than or equal to 4096. The block size of the file system is maintained in the super-block, so it is possible for file systems of different block sizes to be accessible simultaneously on the same system. The block size must be decided when *newfs* creates the file system; the block size cannot be subsequently changed without rebuilding the file system.

2.2. Summary information

Associated with the super block is non replicated *summary information*. The summary information changes as the file system is modified. The summary information contains the number of blocks, fragments, inodes and directories in the file system.

2.3. Cylinder groups

The file system partitions the disk into one or more areas called *cylinder groups*. A cylinder group is comprised of one or more consecutive cylinders on a disk. Each cylinder group includes inode slots for files, a *block map* describing available blocks in the cylinder group, and summary information describing the usage of data blocks within the cylinder group. A fixed number of inodes is allocated for each cylinder group when the file system is created. The current policy is to allocate one inode for each 2048 bytes of disk space; this is expected to be far more inodes than will ever be needed.

All the cylinder group bookkeeping information could be placed at the beginning of each cylinder group. However if this approach were used, all the redundant information would be on the top platter. A single hardware failure that destroyed the top platter could cause the loss of all copies of the redundant super-blocks. Thus the cylinder group bookkeeping information begins at a floating offset from the beginning of the cylinder group. The offset for the $i+1$st cylinder group is about one track further from the beginning of the cylinder group than it was for the ith cylinder group. In this way, the redundant information spirals down into the pack; any single track, cylinder, or platter can be lost without losing all copies of the super-blocks. Except

†The actual number may vary from system to system, but is usually in the range 5-13.

for the first cylinder group, the space between the beginning of the cylinder group and the beginning of the cylinder group information stores data.

2.4. Fragments

To avoid waste in storing small files, the file system space allocator divides a single file system block into one or more *fragments*. The fragmentation of the file system is specified when the file system is created; each file system block can be optionally broken into 2, 4, or 8 addressable fragments. The lower bound on the size of these fragments is constrained by the disk sector size; typically 512 bytes is the lower bound on fragment size. The block map associated with each cylinder group records the space availability at the fragment level. Aligned fragments are examined to determine block availability.

On a file system with a block size of 4096 bytes and a fragment size of 1024 bytes, a file is represented by zero or more 4096 byte blocks of data, and possibly a single fragmented block. If a file system block must be fragmented to obtain space for a small amount of data, the remainder of the block is made available for allocation to other files. For example, consider an 11000 byte file stored on a 4096/1024 byte file system. This file uses two full size blocks and a 3072 byte fragment. If no fragments with at least 3072 bytes are available when the file is created, a full size block is split yielding the necessary 3072 byte fragment and an unused 1024 byte fragment. This remaining fragment can be allocated to another file, as needed.

2.5. Updates to the file system

Every working day hundreds of files are created, modified, and removed. Every time a file is modified, the operating system performs a series of file system updates. These updates, when written on disk, yield a consistent file system. The file system stages all modifications of critical information; modification can either be completed or cleanly backed out after a crash. Knowing the information that is first written to the file system, deterministic procedures can be developed to repair a corrupted file system. To understand this process, the order that the update requests were being honored must first be understood.

When a user program does an operation to change the file system, such as a *write*, the data to be written is copied into an internal *in-core* buffer in the kernel. Normally, the disk update is handled asynchronously; the user process is allowed to proceed even though the data has not yet been written to the disk. The data, along with the inode information reflecting the change, is eventually written out to disk. The real disk write may not happen until long after the *write* system call has returned. Thus at any given time, the file system, as it resides on the disk, lags the state of the file system represented by the in-core information.

The disk information is updated to reflect the in-core information when the buffer is required for another use, when a *sync*(2) is done (at 30 second intervals) by */etc/update*(8), or by manual operator intervention with the *sync*(8) command. If the system is halted without writing out the in-core information, the file system on the disk will be in an inconsistent state.

If all updates are done asynchronously, several serious inconsistencies can arise. One inconsistency is that a block may be claimed by two inodes. Such an inconsistency can occur when the system is halted before the pointer to the block in the old inode has been cleared in the copy of the old inode on the disk, and after the pointer to the block in the new inode has been written out to the copy of the new inode on the disk. Here, there is no deterministic method for deciding which inode should really claim the block. A similar problem can arise with a multiply claimed inode.

The problem with asynchronous inode updates can be avoided by doing all inode deallocations synchronously. Consequently, inodes and indirect blocks are written to the disk synchronously (*i.e.* the process blocks until the information is really written to disk) when they are being deallocated. Similarly inodes are kept consistent by synchronously deleting, adding, or changing directory entries.

3. Fixing corrupted file systems

A file system can become corrupted in several ways. The most common of these ways are improper shutdown procedures and hardware failures.

File systems may become corrupted during an *unclean halt.* This happens when proper shutdown procedures are not observed, physically write-protecting a mounted file system, or a mounted file system is taken off-line. The most common operator procedural failure is forgetting to *sync* the system before halting the CPU.

File systems may become further corrupted if proper startup procedures are not observed, e.g., not checking a file system for inconsistencies, and not repairing inconsistencies. Allowing a corrupted file system to be used (and, thus, to be modified further) can be disastrous.

Any piece of hardware can fail at any time. Failures can be as subtle as a bad block on a disk pack, or as blatant as a non-functional disk-controller.

3.1. Detecting and correcting corruption

Normally *fsck* is run non-interactively. In this mode it will only fix corruptions that are expected to occur from an unclean halt. These actions are a proper subset of the actions that *fsck* will take when it is running interactively. Throughout this paper we assume that *fsck* is being run interactively, and all possible errors can be encountered. When an inconsistency is discovered in this mode, *fsck* reports the inconsistency for the operator to chose a corrective action.

A quiescent‡ file system may be checked for structural integrity by performing consistency checks on the redundant data intrinsic to a file system. The redundant data is either read from the file system, or computed from other known values. The file system **must** be in a quiescent state when *fsck* is run, since *fsck* is a multi-pass program.

In the following sections, we discuss methods to discover inconsistencies and possible corrective actions for the cylinder group blocks, the inodes, the indirect blocks, and the data blocks containing directory entries.

3.2. Super-block checking

The most commonly corrupted item in a file system is the summary information associated with the super-block. The summary information is prone to corruption because it is modified with every change to the file system's blocks or inodes, and is usually corrupted after an unclean halt.

The super-block is checked for inconsistencies involving file-system size, number of inodes, free-block count, and the free-inode count. 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 file-system size and layout information are the most critical pieces of information for *fsck.* While there is no way to actually check these sizes, since they are statically determined by *newfs, fsck* can check that these sizes are within reasonable bounds. All other file system checks require that these sizes be correct. If *fsck* detects corruption in the static parameters of the default super-block, *fsck* requests the operator to specify the location of an alternate super-block.

3.3. Free block checking

Fsck checks that all the blocks marked as free in the cylinder group block maps are not claimed by any files. When all the blocks have been initially accounted for, *fsck* checks that the number of free blocks 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 block allocation maps, *fsck* will rebuild them, based on the list it has computed of allocated blocks.

‡ I.e., unmounted and not being written on.

The summary information associated with the super-block counts the total number of free blocks within the file system. *Fsck* compares this count to the number of free blocks it found within the file system. If the two counts do not agree, then *fsck* replaces the incorrect count in the summary information by the actual free-block count.

The summary information counts the total number of free inodes within the file system. *Fsck* compares this count to the number of free inodes it found within the file system. If the two counts do not agree, then *fsck* replaces the incorrect count in the summary information by the actual free-inode count.

3.4. Checking the inode state

An individual inode is not as likely to be corrupted as the allocation information. However, because of the great number of active inodes, a few of the inodes are usually corrupted.

The list of inodes in the file system is checked sequentially starting with inode 2 (inode 0 marks unused inodes; inode 1 is saved for future generations) and progressing through the last inode in the file system. The state of each inode is checked for inconsistencies involving format and type, link count, duplicate blocks, bad blocks, and inode size.

Each inode contains a mode word. This mode word describes the type and state of the inode. Inodes must be one of six types: regular inode, directory inode, symbolic link inode, special block inode, special character inode, or socket inode. Inodes may be found in one of three allocation states: unallocated, allocated, and neither unallocated nor allocated. This last state suggests an incorrectly formated inode. An inode can get in this state if bad data is written into the inode list. The only possible corrective action is for *fsck* is to clear the inode.

3.5. Inode links

Each inode counts the total number of directory entries linked to the inode. *Fsck* verifies the link count of each inode by starting at the root of the file system, and descending through the directory structure. The actual link count for each inode is calculated during the descent.

If the stored link count is non-zero and the actual link count is zero, then no directory entry appears for the inode. If this happens, *fsck* will place the disconnected file in the *lost+found* directory. If the stored and actual link counts are non-zero and unequal, a directory entry may have been added or removed without the inode being updated. If this happens, *fsck* replaces the incorrect stored link count by the actual link count.

Each inode contains a list, or pointers to lists (indirect blocks), of all the blocks claimed by the inode. Since indirect blocks are owned by an inode, inconsistencies in indirect blocks directly affect the inode that owns it.

Fsck compares each block number claimed by an inode against a list of already allocated blocks. If another inode already claims a block number, then the block number is added to a list of *duplicate blocks*. Otherwise, the list of allocated blocks is updated to include the block number.

If there are any duplicate blocks, *fsck* will perform a partial second pass over the inode list to find the inode of the duplicated block. The second pass is needed, since without examining the files associated with these inodes for correct content, not enough information is available to determine which inode is corrupted and should be cleared. If this condition does arise (only hardware failure will cause it), then the inode with the earliest modify time is usually incorrect, and should be cleared. If this happens, *fsck* prompts the operator to clear both inodes. The operator must decide which one should be kept and which one should be cleared.

Fsck checks the range of each block number claimed by an inode. If the block number is lower than the first data block in the file system, or greater than the last data block, then the block number is a *bad block number*. Many bad blocks in an inode are usually caused by an indirect block that was not written to the file system, a condition which can only occur if there has been a hardware failure. If an inode contains bad block numbers, *fsck* prompts the operator

to clear it.

3.6. Inode data size

Each inode contains a count of the number of data blocks that it contains. The number of actual data blocks is the sum of the allocated data blocks and the indirect blocks. *Fsck* computes the actual number of data blocks and compares that block count against the actual number of blocks the inode claims. If an inode contains an incorrect count *fsck* prompts the operator to fix it.

Each inode contains a thirty-two bit size field. The size is the number of data bytes in the file associated with the inode. The consistency of the byte size field is roughly checked by computing from the size field the maximum number of blocks that should be associated with the inode, and comparing that expected block count against the actual number of blocks the inode claims.

3.7. Checking the data associated with an inode

An inode can directly or indirectly reference three kinds of data blocks. All referenced blocks must be the same kind. The three types of data blocks are: plain data blocks, symbolic link data blocks, and directory data blocks. Plain data blocks contain the information stored in a file; symbolic link data blocks contain the path name stored in a link. Directory data blocks contain directory entries. *Fsck* can only check the validity of directory data blocks.

Each directory data block is checked for several types of inconsistencies. These inconsistencies include directory inode numbers pointing to unallocated inodes, directory inode numbers that are greater than the number of inodes in the file system, incorrect directory inode numbers for "." and "..", and directories that are not attached to the file system. If the inode number in a directory data block references an unallocated inode, then *fsck* will remove that directory entry. Again, this condition can only arise when there has been a hardware failure.

If a directory entry inode number references outside the inode list, then *fsck* will remove that directory entry. This condition occurs if bad data is written into a directory data block.

The directory inode number entry for "." must be the first entry in the directory data block. The inode number for "." must reference itself; e.g., it must equal the inode number for the directory data block. The directory inode number entry for ".." must be the second entry in the directory data block. Its value must equal the inode number for the parent of the directory entry (or the inode number of the directory data block if the directory is the root directory). If the directory inode numbers are incorrect, *fsck* will replace them with the correct values.

3.8. File system connectivity

Fsck checks the general connectivity of the file system. If directories are not linked into the file system, then *fsck* links the directory back into the file system in the *lost+found* directory. This condition only occurs when there has been a hardware failure.

Acknowledgements

I thank Bill Joy, Sam Leffler, Robert Elz and Dennis Ritchie for their suggestions and help in implementing the new file system. Thanks also to Robert Henry for his editorial input to get this document together. Finally we thank our sponsors, the National Science Foundation under grant MCS80-05144, and the Defense Advance Research Projects Agency (DoD) under Arpa Order No. 4031 monitored by Naval Electronic System Command under Contract No. N00039-82-C-0235. (Kirk McKusick, July 1983)

I would like to thank Larry A. Wehr for advice that lead to the first version of *fsck* and Rick B. Brandt for adapting *fsck* to UNIX/TS. (T. Kowalski, July 1979)

References

[Dolotta78] Dolotta, T. A., and Olsson, S. B. eds., *UNIX User's Manual, Edition 1.1* (January 1978).

[Joy83] Joy, W., Cooper, E., Fabry, R., Leffler, S., McKusick, M., and Mosher, D. *4.2BSD System Manual*, University of California at Berkeley, Computer Systems Research Group Technical Report #4, 1982.

[McKusick83] McKusick, M., Joy, W., Leffler, S., and Fabry, R. *A Fast File System for UNIX*, University of California at Berkeley, Computer Systems Research Group Technical Report #7, 1982.

[Ritchie78] Ritchie, D. M., and Thompson, K., The UNIX Time-Sharing System, *The Bell System Technical Journal* **57**, 6 (July-August 1978, Part 2), pp. 1905-29.

[Thompson78] Thompson, K., UNIX Implementation, *The Bell System Technical Journal* **57**, 6 (July-August 1978, Part 2), pp. 1931-46.

4. Appendix A – Fsck Error Conditions

4.1. Conventions

Fsck is a multi-pass file system check program. Each file system pass invokes a different Phase of the *fsck* program. After the initial setup, *fsck* performs successive Phases over each file system, checking blocks and sizes, path-names, connectivity, reference counts, and the map of free blocks, (possibly rebuilding it), and performs some cleanup.

Normally *fsck* is run non-interactively to *preen* the file systems after an unclean halt. While preen'ing a file system, it will only fix corruptions that are expected to occur from an unclean halt. These actions are a proper subset of the actions that *fsck* will take when it is running interactively. Throughout this appendix many errors have several options that the operator can take. When an inconsistency is detected, *fsck* reports the error condition to the operator. If a response is required, *fsck* prints a prompt message and waits for a response. When preen'ing most errors are fatal. For those that are expected, the response taken is noted. This appendix explains the meaning of each error condition, the possible responses, and the related error conditions.

The error conditions are organized by the *Phase* of the *fsck* program in which they can occur. The error conditions that may occur in more than one Phase will be discussed in initialization.

4.2. Initialization

Before a file system check can be performed, certain tables have to be set up and certain files opened. This section concerns itself with the opening of files and the initialization of tables. This section lists error conditions resulting from command line options, memory requests, opening of files, status of files, file system size checks, and creation of the scratch file. All of the initialization errors are fatal when the file system is being preen'ed.

_C_ option?
_C_ is not a legal option to *fsck*; legal options are –b, –y, –n, and –p. *Fsck* terminates on this error condition. See the *fsck*(8) manual entry for further detail.

cannot alloc NNN bytes for blockmap
cannot alloc NNN bytes for freemap
cannot alloc NNN bytes for statemap
cannot alloc NNN bytes for lncntp
Fsck's request for memory for its virtual memory tables failed. This should never happen. *Fsck* terminates on this error condition. See a guru.

Can't open checklist file: _F_
The file system checklist file _F_ (usually */etc/fstab*) can not be opened for reading. *Fsck* terminates on this error condition. Check access modes of _F_.

Can't stat root
Fsck's request for statistics about the root directory "/" failed. This should never happen. *Fsck* terminates on this error condition. See a guru.

Can't stat _F_
Can't make sense out of name _F_
Fsck's request for statistics about the file system _F_ failed. When running manually, it ignores this file system and continues checking the next file system given. Check access modes of _F_.

Can't open _F_
Fsck's request attempt to open the file system _F_ failed. When running manually, it ignores this

file system and continues checking the next file system given. Check access modes of *F*.

F: (NO WRITE)
Either the –n flag was specified or *fsck*'s attempt to open the file system *F* for writing failed. When running manually, all the diagnostics are printed out, but no modifications are attempted to fix them.

file is not a block or character device; OK
You have given *fsck* a regular file name by mistake. Check the type of the file specified.

Possible responses to the OK prompt are:

YES Ignore this error condition.

NO ignore this file system and continues checking the next file system given.

One of the following messages will appear:
MAGIC NUMBER WRONG
NCG OUT OF RANGE
CPG OUT OF RANGE
NCYL DOES NOT JIVE WITH NCG*CPG
SIZE PREPOSTEROUSLY LARGE
TRASHED VALUES IN SUPER BLOCK

and will be followed by the message:
F: **BAD SUPER BLOCK:** *B*
USE -b OPTION TO FSCK TO SPECIFY LOCATION OF AN ALTERNATE SUPER-BLOCK TO SUPPLY NEEDED INFORMATION; SEE fsck(8).
The super block has been corrupted. An alternative super block must be selected from among those listed by *newfs* (8) when the file system was created. For file systems with a blocksize less than 32K, specifying –b 32 is a good first choice.

INTERNAL INCONSISTENCY: *M*
Fsck's has had an internal panic, whose message is specified as *M*. This should never happen. See a guru.

CAN NOT SEEK: BLK *B* **(CONTINUE)**
Fsck's request for moving to a specified block number *B* in the file system failed. This should never happen. See a guru.

Possible responses to the CONTINUE prompt are:

YES attempt to continue to run the file system check. Often, however the problem will persist. This error condition will not allow a complete check of the file system. A second run of *fsck* should be made to re-check this file system. If the block was part of the virtual memory buffer cache, *fsck* will terminate with the message "Fatal I/O error".

NO terminate the program.

CAN NOT READ: BLK *B* **(CONTINUE)**
Fsck's request for reading a specified block number *B* in the file system failed. This should never happen. See a guru.

Possible responses to the CONTINUE prompt are:

YES attempt to continue to run the file system check. Often, however, the problem will persist. This error condition will not allow a complete check of the file system. A second run of *fsck* should be made to re-check this file system. If the block was part of the virtual memory buffer cache, *fsck* will terminate with the message "Fatal I/O error".

NO terminate the program.

CAN NOT WRITE: BLK *B* (CONTINUE)
Fsck's request for writing a specified block number *B* in the file system failed. The disk is write-protected. See a guru.

Possible responses to the CONTINUE prompt are:

YES attempt to continue to run the file system check. Often, however, the problem will persist. This error condition will not allow a complete check of the file system. A second run of *fsck* should be made to re-check this file system. If the block was part of the virtual memory buffer cache, *fsck* will terminate with the message "Fatal I/O error".

NO terminate the program.

4.3. Phase 1 – Check Blocks and Sizes

This phase concerns itself with the inode list. This section lists error conditions resulting from checking inode types, setting up the zero-link-count table, examining inode block numbers for bad or duplicate blocks, checking inode size, and checking inode format. All errors in this phase except **INCORRECT BLOCK COUNT** are fatal if the file system is being preen'ed,

CG *C*: BAD MAGIC NUMBER The magic number of cylinder group *C* is wrong. This usually indicates that the cylinder group maps have been destroyed. When running manually the cylinder group is marked as needing to be reconstructed.

UNKNOWN FILE TYPE I=*I* (CLEAR) The mode word of the inode *I* indicates that the inode is not a special block inode, special character inode, socket inode, regular inode, symbolic link, or directory inode.

Possible responses to the CLEAR prompt are:

YES de-allocate inode *I* by zeroing its contents. This will always invoke the UNALLOCATED error condition in Phase 2 for each directory entry pointing to this inode.

NO ignore this error condition.

LINK COUNT TABLE OVERFLOW (CONTINUE)
An internal table for *fsck* containing allocated inodes with a link count of zero has no more room. Recompile *fsck* with a larger value of MAXLNCNT.

Possible responses to the CONTINUE prompt are:

YES continue with the program. This error condition will not allow a complete check of the file system. A second run of *fsck* should be made to re-check this file system. If another allocated inode with a zero link count is found, this error condition is repeated.

NO terminate the program.

B **BAD I=***I*
Inode *I* contains block number *B* 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. This error condition may invoke the **EXCESSIVE BAD BLKS** error condition in Phase 1 if inode *I* has too many block numbers outside the file system range. This error condition will always invoke the **BAD/DUP** error condition in Phase 2 and Phase 4.

EXCESSIVE BAD BLKS I=*I* (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 last block in the file system associated with inode *I*.

Possible responses to the CONTINUE prompt are:

YES ignore the rest of the blocks in this inode and continue checking with the next inode in the file system. This error condition will not allow a complete check of the file system. A second run of *fsck* should be made to re-check this file system.

NO terminate the program.

B* DUP I=*I
Inode *I* contains block number *B* which is already claimed by another inode. This error condition may invoke the **EXCESSIVE DUP BLKS** error condition in Phase 1 if inode *I* has too many block numbers claimed by other inodes. This error condition will always invoke Phase 1b and the **BAD/DUP** error condition in Phase 2 and Phase 4.

EXCESSIVE DUP BLKS I=*I* (CONTINUE)
There is more than a tolerable number (usually 10) of blocks claimed by other inodes.

Possible responses to the CONTINUE prompt are:

YES ignore the rest of the blocks in this inode and continue checking with the next inode in the file system. This error condition will not allow a complete check of the file system. A second run of *fsck* should be made to re-check this file system.

NO terminate the program.

DUP TABLE OVERFLOW (CONTINUE)
An internal table in *fsck* containing duplicate block numbers has no more room. Recompile *fsck* with a larger value of DUPTBLSIZE.

Possible responses to the CONTINUE prompt are:

YES continue with the program. This error condition will not allow a complete check of the file system. A second run of *fsck* should be made to re-check this file system. If another duplicate block is found, this error condition will repeat.

NO terminate the program.

PARTIALLY ALLOCATED INODE I=*I* (CLEAR)
Inode *I* is neither allocated nor unallocated.

Possible responses to the CLEAR prompt are:

YES de-allocate inode *I* by zeroing its contents.

NO ignore this error condition.

INCORRECT BLOCK COUNT I=*I* (*X* should be *Y*) (CORRECT)
The block count for inode *I* is *X* blocks, but should be *Y* blocks. When preen'ing the count is corrected.

Possible responses to the CORRECT prompt are:

YES replace the block count of inode *I* with *Y*.

NO ignore this error condition.

4.4. Phase 1B: Rescan for More Dups

When a duplicate block is found in the file system, the file system is rescanned to find the inode which previously claimed that block. This section lists the error condition when the duplicate block is found.

B **DUP I=***I*

Inode *I* contains block number *B* that is already claimed by another inode. This error condition will always invoke the **BAD/DUP** error condition in Phase 2. You can determine which inodes have overlapping blocks by examining this error condition and the DUP error condition in Phase 1.

4.5. Phase 2 – Check Pathnames

This phase concerns itself with removing directory entries pointing to error conditioned inodes from Phase 1 and Phase 1b. This section lists error conditions resulting from root inode mode and status, directory inode pointers in range, and directory entries pointing to bad inodes. All errors in this phase are fatal if the file system is being preen'ed.

ROOT INODE UNALLOCATED. TERMINATING.

The root inode (usually inode number 2) has no allocate mode bits. This should never happen. The program will terminate.

NAME TOO LONG *F*

An excessively long path name has been found. This is usually indicative of loops in the file system name space. This can occur if the super user has made circular links to directories. The offending links must be removed (by a guru).

ROOT INODE NOT DIRECTORY (FIX)

The root inode (usually inode number 2) is not directory inode type.

Possible responses to the FIX prompt are:

YES replace the root inode's type to be a directory. If the root inode's data blocks are not directory blocks, a VERY large number of error conditions will be produced.

NO terminate the program.

DUPS/BAD IN ROOT INODE (CONTINUE)

Phase 1 or Phase 1b have found duplicate blocks or bad blocks in the root inode (usually inode number 2) for the file system.

Possible responses to the CONTINUE prompt are:

YES ignore the **DUPS/BAD** error condition in the root inode and attempt to continue to run the file system check. If the root inode is not correct, then this may result in a large number of other error conditions.

NO terminate the program.

I OUT OF RANGE I=*I* **NAME=***F* **(REMOVE)**

A directory entry *F* has an inode number *I* which is greater than the end of the inode list.

Possible responses to the REMOVE prompt are:

YES the directory entry *F* is removed.

NO ignore this error condition.

UNALLOCATED I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F* (REMOVE)
A directory entry *F* has a directory inode *I* without allocate mode bits. The owner *O*, mode *M*, size *S*, modify time *T*, and directory name *F* are printed.

Possible responses to the REMOVE prompt are:

YES the directory entry *F* is removed.

NO ignore this error condition.

UNALLOCATED I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* FILE=*F* (REMOVE)
A directory entry *F* has an inode *I* without allocate mode bits. The owner *O*, mode *M*, size *S*, modify time *T*, and file name *F* are printed.

Possible responses to the REMOVE prompt are:

YES the directory entry *F* is removed.

NO ignore this error condition.

DUP/BAD I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F* (REMOVE)
Phase 1 or Phase 1b have found duplicate blocks or bad blocks associated with directory entry *F*, directory inode *I*. The owner *O*, mode *M*, size *S*, modify time *T*, and directory name *F* are printed.

Possible responses to the REMOVE prompt are:

YES the directory entry *F* is removed.

NO ignore this error condition.

DUP/BAD I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* FILE=*F* (REMOVE)
Phase 1 or Phase 1b have found duplicate blocks or bad blocks associated with directory entry *F*, inode *I*. The owner *O*, mode *M*, size *S*, modify time *T*, and file name *F* are printed.

Possible responses to the REMOVE prompt are:

YES the directory entry *F* is removed.

NO ignore this error condition.

ZERO LENGTH DIRECTORY I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F* (REMOVE)
A directory entry *F* has a size *S* that is zero. The owner *O*, mode *M*, size *S*, modify time *T*, and directory name *F* are printed.

Possible responses to the REMOVE prompt are:

YES the directory entry *F* is removed; this will always invoke the BAD/DUP error condition in Phase 4.

NO ignore this error condition.

DIRECTORY TOO SHORT I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F* (FIX)
A directory *F* has been found whose size *S* is less than the minimum size directory. The owner *O*, mode *M*, size *S*, modify time *T*, and directory name *F* are printed.

Possible responses to the FIX prompt are:

YES increase the size of the directory to the minimum directory size.

NO ignore this directory.

DIRECTORY CORRUPTED I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F* (SALVAGE)
A directory with an inconsistent internal state has been found.

Possible responses to the FIX prompt are:

YES throw away all entries up to the next 512-byte boundary. This rather drastic action can throw away up to 42 entries, and should be taken only after other recovery efforts have failed.

NO Skip up to the next 512-byte boundary and resume reading, but do not modify the directory.

BAD INODE NUMBER FOR '.' I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F* (FIX)
A directory *I* has been found whose inode number for '.' does does not equal *I*.

Possible responses to the FIX prompt are:

YES change the inode number for '.' to be equal to *I*.

NO leave the inode number for '.' unchanged.

MISSING '.' I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F* (FIX)
A directory *I* has been found whose first entry is unallocated.

Possible responses to the FIX prompt are:

YES make an entry for '.' with inode number equal to *I*.

NO leave the directory unchanged.

MISSING '.' I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F*
CANNOT FIX, FIRST ENTRY IN DIRECTORY CONTAINS *F*
A directory *I* has been found whose first entry is *F*. *Fsck* cannot resolve this problem. The file system should be mounted and the offending entry *F* moved elsewhere. The file system should then be unmounted and *fsck* should be run again.

MISSING '.' I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F*
CANNOT FIX, INSUFFICIENT SPACE TO ADD '.'
A directory *I* has been found whose first entry is not '.'. *Fsck* cannot resolve this problem as it should never happen. See a guru.

EXTRA '.' ENTRY I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F* (FIX)
A directory *I* has been found that has more than one entry for '.'.

Possible responses to the FIX prompt are:

YES remove the extra entry for '.'.

NO leave the directory unchanged.

BAD INODE NUMBER FOR '..' I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F* (FIX)
A directory *I* has been found whose inode number for '..' does does not equal the parent of *I*.

Possible responses to the FIX prompt are:

YES change the inode number for '..' to be equal to the parent of *I*.

NO leave the inode number for '..' unchanged.

MISSING '..' I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F* (FIX)
A directory *I* has been found whose second entry is unallocated.

Possible responses to the FIX prompt are:

YES make an entry for '..' with inode number equal to the parent of *I*.

NO leave the directory unchanged.

MISSING '..' I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F*
CANNOT FIX, SECOND ENTRY IN DIRECTORY CONTAINS *F*
A directory *I* has been found whose second entry is *F*. *Fsck* cannot resolve this problem. The file system should be mounted and the offending entry *F* moved elsewhere. The file system should then be unmounted and *fsck* should be run again.

MISSING '..' I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F*
CANNOT FIX, INSUFFICIENT SPACE TO ADD '..'
A directory *I* has been found whose second entry is not '..'. *Fsck* cannot resolve this problem as it should never happen. See a guru.

EXTRA '..' ENTRY I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F* (FIX)
A directory *I* has been found that has more than one entry for '..'.

Possible responses to the FIX prompt are:

YES remove the extra entry for '..'.

NO leave the directory unchanged.

4.6. Phase 3 – Check Connectivity

This phase concerns itself with the directory connectivity seen in Phase 2. This section lists error conditions resulting from unreferenced directories, and missing or full *lost+found* directories.

UNREF DIR I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* (RECONNECT)
The directory inode *I* was not connected to a directory entry when the file system was traversed. The owner *O*, mode *M*, size *S*, and modify time *T* of directory inode *I* are printed. When preen'ing, the directory is reconnected if its size is non-zero, otherwise it is cleared.

Possible responses to the RECONNECT prompt are:

YES reconnect directory inode *I* to the file system in the directory for lost files (usually *lost+found*). This may invoke the *lost+found* error condition in Phase 3 if there are problems connecting directory inode *I* to *lost+found*. This may also invoke the CONNECTED error condition in Phase 3 if the link was successful.

NO ignore this error condition. This will always invoke the UNREF error condition in Phase 4.

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 will always invoke the UNREF error condition in Phase 4. Check access modes of *lost+found*. See *fsck*(8) manual entry for further detail. This error is fatal if the file system is being preen'ed.

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 will always invoke the UNREF error condition in Phase 4. Clean out unnecessary entries in *lost+found* or make *lost+found* larger. See *fsck*(8) manual entry for further detail. This error is fatal if the file system is being preen'ed.

DIR I=*I1* CONNECTED. PARENT WAS I=*I2*
This is an advisory message indicating a directory inode *I1* was successfully connected to the *lost+found* directory. The parent inode *I2* of the directory inode *I1* is replaced by the inode number of the *lost+found* directory.

4.7. Phase 4 – Check Reference Counts

This phase concerns itself with the link count information seen in Phase 2 and Phase 3. This section lists error conditions resulting from unreferenced files, missing or full *lost+found* directory, incorrect link counts for files, directories, symbolic links, or special files, unreferenced files, symbolic links, and directories, bad and duplicate blocks in files, symbolic links, and directories, and incorrect total free-inode counts. All errors in this phase are correctable if the file system is being preen'ed except running out of space in the lost+found directory.

UNREF FILE I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* (RECONNECT)
Inode *I* was not connected to a directory entry when the file system was traversed. The owner *O*, mode *M*, size *S*, and modify time *T* of inode *I* are printed. When preen'ing the file is cleared if either its size or its link count is zero, otherwise it is reconnected.

Possible responses to the RECONNECT prompt are:

YES reconnect inode *I* to the file system in the directory for lost files (usually *lost+found*). This may invoke the *lost+found* error condition in Phase 4 if there are problems connecting inode *I* to *lost+found*.

NO ignore this error condition. This will always invoke the CLEAR error condition in Phase 4.

(CLEAR)
The inode mentioned in the immediately previous error condition can not be reconnected. This cannot occur if the file system is being preen'ed, since lack of space to reconnect files is a fatal error.

Possible responses to the CLEAR prompt are:

YES de-allocate the inode mentioned in the immediately previous error condition by zeroing its contents.

NO ignore this error condition.

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 will always invoke the CLEAR error condition in Phase 4. Check access modes of *lost+found*. This error is fatal if the file system is being preen'ed.

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 will always invoke the CLEAR error condition in Phase 4. Check size and contents of *lost+found*. This error is fatal if the file system is being preen'ed.

LINK COUNT FILE I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* COUNT=*X* SHOULD BE *Y* (ADJUST)
The link count for inode *I* which is a file, is *X* but should be *Y*. The owner *O*, mode *M*, size *S*,

and modify time *T* are printed. When preen'ing the link count is adjusted.

Possible responses to the ADJUST prompt are:

YES replace the link count of file inode *I* with *Y*.

NO ignore this error condition.

LINK COUNT DIR I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* COUNT=*X* SHOULD BE *Y* (ADJUST)

The link count for inode *I* which is a directory, is *X* but should be *Y*. The owner *O*, mode *M*, size *S*, and modify time *T* of directory inode *I* are printed. When preen'ing the link count is adjusted.

Possible responses to the ADJUST prompt are:

YES replace the link count of directory inode *I* with *Y*.

NO ignore this error condition.

LINK COUNT *F* I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* COUNT=*X* SHOULD BE *Y* (ADJUST)

The link count for *F* inode *I* is *X* but should be *Y*. The name *F*, owner *O*, mode *M*, size *S*, and modify time *T* are printed. When preen'ing the link count is adjusted.

Possible responses to the ADJUST prompt are:

YES replace the link count of inode *I* with *Y*.

NO ignore this error condition.

UNREF FILE I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* (CLEAR)

Inode *I* which is a file, was not connected to a directory entry when the file system was traversed. The owner *O*, mode *M*, size *S*, and modify time *T* of inode *I* are printed. When preen'ing, this is a file that was not connected because its size or link count was zero, hence it is cleared.

Possible responses to the CLEAR prompt are:

YES de-allocate inode *I* by zeroing its contents.

NO ignore this error condition.

UNREF DIR I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* (CLEAR)

Inode *I* which is a directory, was not connected to a directory entry when the file system was traversed. The owner *O*, mode *M*, size *S*, and modify time *T* of inode *I* are printed. When preen'ing, this is a directory that was not connected because its size or link count was zero, hence it is cleared.

Possible responses to the CLEAR prompt are:

YES de-allocate inode *I* by zeroing its contents.

NO ignore this error condition.

BAD/DUP FILE I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* (CLEAR)

Phase 1 or Phase 1b have found duplicate blocks or bad blocks associated with file inode *I*. The owner *O*, mode *M*, size *S*, and modify time *T* of inode *I* are printed. This error cannot arise when the file system is being preen'ed, as it would have caused a fatal error earlier.

Possible responses to the CLEAR prompt are:

YES de-allocate inode *I* by zeroing its contents.

NO ignore this error condition.

BAD/DUP DIR I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* (CLEAR)
Phase 1 or Phase 1b have found duplicate blocks or bad blocks associated with directory inode *I*. The owner *O*, mode *M*, size *S*, and modify time *T* of inode *I* are printed. This error cannot arise when the file system is being preen'ed, as it would have caused a fatal error earlier.

Possible responses to the CLEAR prompt are:

YES de-allocate inode *I* by zeroing its contents.

NO ignore this error condition.

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. When preen'ing, the count is fixed.

Possible responses to the FIX prompt are:

YES replace the count in the super-block by the actual count.

NO ignore this error condition.

4.8. Phase 5 - Check Cyl groups

This phase concerns itself with the free-block maps. This section lists error conditions resulting from allocated blocks in the free-block maps, free blocks missing from free-block maps, and the total free-block count incorrect.

CG *C*: BAD MAGIC NUMBER
The magic number of cylinder group *C* is wrong. This usually indicates that the cylinder group maps have been destroyed. When running manually the cylinder group is marked as needing to be reconstructed. This error is fatal if the file system is being preen'ed.

EXCESSIVE BAD BLKS IN BIT MAPS (CONTINUE)
An inode contains more than a tolerable number (usually 10) of blocks claimed by other inodes or that are out of the legal range for the file system. This error is fatal if the file system is being preen'ed.

Possible responses to the CONTINUE prompt are:

YES ignore the rest of the free-block maps and continue the execution of *fsck*.

NO terminate the program.

SUMMARY INFORMATION *T* BAD
where *T* is one or more of:
(INODE FREE)
(BLOCK OFFSETS)
(FRAG SUMMARIES)
(SUPER BLOCK SUMMARIES)
The indicated summary information was found to be incorrect. This error condition will always invoke the **BAD CYLINDER GROUPS** condition in Phase 6. When preen'ing, the summary information is recomputed.

***X* BLK(S) MISSING**
X blocks unused by the file system were not found in the free-block maps. This error condition will always invoke the **BAD CYLINDER GROUPS** condition in Phase 6. When preen'ing, the block maps are rebuilt.

BAD CYLINDER GROUPS (SALVAGE)
Phase 5 has found bad blocks in the free-block maps, duplicate blocks in the free-block maps, or

blocks missing from the file system. When preen'ing, the cylinder groups are reconstructed.

Possible responses to the SALVAGE prompt are:

YES replace the actual free-block maps with a new free-block maps.

NO ignore this error condition.

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. When preen'ing, the counts are fixed.

Possible responses to the FIX prompt are:

YES replace the count in the super-block by the actual count.

NO ignore this error condition.

4.9. Phase 6 - Salvage Cylinder Groups

This phase concerns itself with the free-block maps reconstruction. No error messages are produced.

4.10. Cleanup

Once a file system has been checked, a few cleanup functions are performed. This section lists advisory messages about the file system and modify status of the file system.

***V* files, *W* used, *X* free (*Y* frags, *Z* blocks)**

This is an advisory message indicating that the file system checked contained *V* files using *W* fragment sized blocks leaving *X* fragment sized blocks free in the file system. The numbers in parenthesis breaks the free count down into *Y* free fragments and *Z* free full sized blocks.

******* REBOOT UNIX *******

This is an advisory message indicating that the root file system has been modified by *fsck*. If UNIX is not rebooted immediately, the work done by *fsck* may be undone by the in-core copies of tables UNIX keeps. When preen'ing, *fsck* will exit with a code of 4. The auto-reboot script interprets an exit code of 4 by issuing a reboot system call.

******* FILE SYSTEM WAS MODIFIED *******

This is an advisory message indicating that the current file system was modified by *fsck*. If this file system is mounted or is the current root file system, *fsck* should be halted and UNIX rebooted. If UNIX is not rebooted immediately, the work done by *fsck* may be undone by the in-core copies of tables UNIX keeps.

C

Problem Reporting

This appendix introduces the CONVEX Technical Assistance Center (TAC) and *contact* utility. The *contact* utility is an online system for reporting problems to the TAC. To learn *contact* by using it, enter **contact** at the system prompt and then answer the questions as they appear on the screen. To find out more about using *contact*, read through this appendix. It describes prerequisites and tips for using *contact* and the step-by-step process *contact* takes you through.

C.1 Technical Assistance Center

The CONVEX Technical Assistance Center (TAC) is staffed by technical specialists who can address diverse questions and problems that arise in a supercomputing environment. If you have a hardware, software, or documentation question, contact the TAC. This group stands ready to solve such problems.

C.2 The *contact* Utility

The TAC recommends using the *contact* utility to report a hardware, software, or documentation problem. The *contact* utility is an interactive program that helps the TAC track reports and route them to the CONVEX personnel most qualified to fix a problem.

After you invoke *contact*, it prompts you for information about the problem. When you finish your report, *contact* electronically mails it to the TAC. You are notified within 48 hours that the TAC has received your report.

C.3 Prerequisites

To use *contact* requires

- a UNIX-to-UNIX Communication Protocol (UUCP) connection to the TAC
- full path name of the program or utility in question
- version number of the program or utility in question

C.3.1 UUCP Connection

Before using *contact*, check with your system administrator to be sure there is a UUCP connection to the TAC. A UUCP connection allows files to be copied from one UNIX-based system to another. The *uucp* (UNIX-to-UNIX copy) command relies on either a dial-up or hard-wired UUCP communication line.

C.3.2 Finding the Program Path Name

To determine the full path name of the program or utility in question, use the *which* command. The next screen illustrates using the *which* command to find the full path name of the loader (*ld*) utility.

```
>which ld
/bin/ld
>
```

In this example, the full path name of the loader is */bin/ld*.

For more information on the *which* command, refer to the *which*(1) man page. You can also use the *info* online information system. Enter **info which** at the system prompt.

If you use the C shell (*csh*), you can also use the *whence* command to find the program path name. The *whence* command works like *which*, but faster.

C.3.3 Finding the Program Version Number

To determine the version number of the program or utility in question, use the *vers* command. The next screen illustrates using the *vers* command to find the version number of the loader (*ld*) utility. Enter **vers**, then the path name of the program or utility.

```
>vers /bin/ld
/bin/ld: 7.0
>
```

In this example, the loader version number is 7.0.

For more information on the *vers* command, refer to the *vers*(1) man page. You can also use the *info* online information system. To do so, enter **info vers** at the system prompt.

C.4 Tips on Using the *contact* Utility

The *contact* utility is interactive and easy to use. This section lists tips to help use it efficiently. In particular, this section tells how to

- use a *.contact* file
- abort a contact session
- resubmit an aborted report
- suspend a contact session
- move from one prompt to another
- use tilde-escape sequences in the *contact* utility

C.4.1 Using a *.contact* File

When you invoke *contact*, it prompts for information regarding the problem. The first prompt is for your name, title, phone number, and company name. You can, however, create a *.contact* file to skip this first prompt. Follow these steps:

1. Create a *.contact* file in your home directory.

2. Enter your name, job title, phone number, and company name, each on a new line.

When you invoke *contact*, it automatically includes the *.contact* file as input for the first prompt and proceeds to the next prompt.

C.4.2 Aborting the Report

To abort a contact report, either press the interrupt key (usually CTRL-C) or choose the abort option when prompted by the *contact* utility. Using CTRL-C to abort does not save the contents of the report. Using the abort option saves the contents of the report in a file named *dead.report* in your home directory.

C.4.3 Submitting the *dead.report* File

After you abort a contact session, the *contact* utility saves the report in a file named *dead.report* in your home directory. Using the *contact* command with the *-r* option automatically merges the contents of the *dead.report* file into the new contact session. Enter

contact -r

and *contact* finds the *dead.report* file in your home directory and merges it into the contact report. You can then edit the report. When you end the editing session, *contact* returns to the final prompt, which asks you to review, edit, submit, or abort the report.

C.4.4 Suspending a Report

Sometimes it is necessary to stop in the middle of a contact report and return to the shell (for instance, to suspend the contact session to find the program path name or version number). To suspend the contact session, press CTRL-Z. To return to the contact session, enter **fg**. Using CTRL-Z and the *fg* (foreground) command lets you toggle back and forth between the *contact* utility and the shell. You cannot, however, use CTRL-Z and *fg* to toggle back and forth if you are using the Bourne shell (*sh*).

C.4.5 Ending a Response

The *contact* utility prompts for information pertinent to your hardware, software, or documentation question. Some prompts require one-line responses; to move to the next prompt, press RETURN. Other prompts require more than a one-line response; to move to the next prompt, press CTRL-D.

C.4.6 Tilde-Escape Sequences

The *contact* utility treats input beginning with a tilde (~) as a special sequence. The character following the tilde is considered a request for a special function. The following tilde sequences are recognized by *contact*:

~e	start the text editor (defined in the EDITOR environment variable)
~h	display a list of available tilde-escape sequences
~p	print the contact report to the terminal screen
~r *filename*	read the contents of *filename* as a response to the current prompt. Some prompts require only a one-line response. This tilde-escape sequence works only for prompts that allow more than a one-line response.
~~	insert a single tilde as the first character in the line

C.5 Using the *contact* Utility

The *contact* utility prompts for the following information:

- your name, title, phone number, and corporate name
- name and version of the product
- one-line summary of the problem
- detailed description of the problem
- priority of the problem
- instructions on how to reproduce the problem
- comments about the problem
- comments about the documentation related to the problem
- files to include in the contact report

The following is a step-by-step discussion of these prompts.

Step 1a To invoke the *contact* utility, enter **contact** at the system prompt. The system responds with a welcome message and a series of questions regarding your hardware, software or documentation question. The next screen illustrates the *contact* command and the system response.

```
>contact
Welcome to contact version 0.11 ()

Enter your name, title, phone number, and corporate name (^D to terminate)
>
```

Step 1b If there is a *.contact* file in your home directory, *contact* skips the first prompt. The next screen illustrates the *contact* command and the system response when a *.contact* file is in your home directory.

```
>contact
Welcome to contact version 0.11 ()

Enter the name of the product involved
>
```

Step 2 The *contact* utility prompts for the version number of the product. If you do not know the version number, use CTRL-Z to suspend the session. Use the *which* (or *whence* if you use *csh*) and *vers* commands to find the version number of the product. Use the *fg* command to return to the session and enter the version number in the form X.X or X.X.X.X.

Step 3 The *contact* utility prompts for a one-line summary of the problem. This summary is the subject header in any further correspondence regarding the problem. Please make this summary as descriptive as possible in one line.

Step 4 The *contact* utility prompts for a detailed description of the problem. Please make this description as complete as possible. Include source code and a stack backtrace when possible. (Refer to the *adb*(1) or *csd*(1) man page for information on obtaining a stack backtrace.) The more information you provide, the quicker the TAC can isolate and solve the problem.

Step 5 The *contact* utility prompts for the priority of the problem. The next screen illustrates this prompt and priority levels from which to choose; you must enter a priority number.

```
Enter a problem priority, based on the following:
1) Critical    - work cannot proceed until the problem is resolved.
2) Serious     - work can proceed around the problem, with difficulty.
3) Necessary   - problem has to be fixed.
4) Annoying    - problem is bothersome.
5) Enhancement - requested enhancement.
6) Informative - for informational purposes only.
>
```

Step 6 The *contact* utility prompts for an explanation of how to reproduce the problem. Please include the command syntax and options you used and anything else you did to make the program run.

Step 7 The *contact* utility prompts for any other pertinent comments. Please include all relevant information.

Step 8 The *contact* utility prompts for suggestions regarding documentation supporting the product. Indicate whether documentation could be revised to address the problem.

Step 9 The *contact* utility asks for names of files necessary to reproduce the problem. The next screen illustrates the *contact* prompt and sample user response.

```
Are there any files that should be included in this report (yes | no)?
>yes
Please enter the names of the files, one to a line (^D to terminate)
>test.f
>~/subroutines/sub.f
>
```

Note

Tilde-escape sequences are not recognized in responses to this prompt. In *contact*, a tilde in this section means your home directory. This convention is based on use of the tilde for expanding file names in *csh*.

If files specified are small text files, they are automatically included in the contact report. If the files are too large to be included in this report, *contact* gives further instructions on how to submit these files.

To specify a directory, combine directory files into a single file using the *tar* command (refer to the *tar*(1) man page for further information) or enter each file name in the directory on a single line in the contact report.

Step 10 The *contact* utility prompts you to review, edit, submit, or abort the contact report. The next screen illustrates this prompt.

```
Please select one of the following options:
1) Review the problem report.
2) Edit the problem report.
3) Submit the problem report.
4) Abort the problem report.
>
```

Choose the number of the option you want to select. These options let you do the following:

Review — review the text of the contact report. You are then prompted again to select an option.

Edit — edit the text of the contact report. If you choose to edit the report, *contact* puts you in your default text editor.

Submit — sends the report to the CONVEX TAC. You are notified within 48 hours that the TAC has received the report. This option exits the *contact* utility and returns you to the shell.

Abort — saves the text of the report in a file named *dead.report* in your home directory. This option exits the *contact* utility and returns you to the shell.

Index

B

C

E

F

T

U

ConvexOS System Manager's Guide
Document No. 710-001430-207, Tenth Edition, Revision 1

Reader's Forum

You are invited to submit your comments concerning the clarity and service of this manual. Constructive critical comments are most welcome and will help us continue in our efforts to generate quality customer documentation. (Please list the page number for questions and comments):

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

From:

Name_____ Title_____

Company_____ Date_____

Address and Phone No._____

FOR ADDITIONAL INFORMATION/DOCUMENTATION:

Location	Phone Number
Within the continental United States	1-800-952-0379
From Alaska, Hawaii, and Canada	1-214-497-4379
From all other locations	Contact nearest CONVEX office

Direct Mail Orders to:

CONVEX Computer Corporation
Customer Service
P.O. Box 833851
Richardson, Texas 75083-3851 USA

(Fold Here First)

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 1046 RICHARDSON, TEXAS

POSTAGE WILL BE PAID BY ADDRESSEE

CONVEX Computer Corporation
Customer Service
PO Box 833851
Richardson TX 75083-3851

(Fold Here Second)

(Tape or Staple)

DSW-004

CONVEX COMPUTER CORPORATION

CONVEXOS SYSTEM MANAGER'S GUIDE
710-001430-207

PRINTED IN U.S.A.