SYSTEM V/68
ADMINISTRATOR'S GUIDE

PRODUCT CODE 72901
41964-00

MOTOROLA
Computer Systems

Product Code 72912
41975G01

SYSTEM V/68 DOCUMENTATION SET

VOLUME I

SYSTEM V/68 USER'S REFERENCE MANUAL, 72905 (41968-00)

- Introduction
- Permuted Index
- Section 1 - Commands

VOLUME II

SYSTEM V/68 USER'S REFERENCE MANUAL, 72905 (41968-00)

- Section 2 - System Calls
- Section 3 - Subroutines
- Section 4 - File Formats
- Section 5 - Miscellaneous Facilities
- Section 6 - Games

VOLUME III

SYSTEM V/68 ADMINISTRATOR'S MANUAL, 72900 (41963-00)

- Introduction
- Permuted Index
- Section 1M - Commands
- Section 7 - Special Files
- Section 8 - Procedures

SYSTEM V/68 ADMINISTRATOR'S GUIDE, 72901 (41964-00)

- Introduction
- Administrative Guidelines
- Using the System
- Accounting
- File System Checking
- LP Spooling System
- System Activity Package

SYSTEM V/68 OPERATOR'S GUIDE, 72904 (41967-00)

- Chapter 1 - System Overview
- Chapter 2 - Getting Started
- Chapter 3 - Using the System
- Appendix A - System Specifications
- Appendix B - Error Messages

SYSTEM V/68 USER'S GUIDE, 72903 (41966-00)

- Introduction
- Primer
- Basics for Beginners
- Text Editors
- An Introduction to Shell
- Source Code Control System (SCCS)
- UNIX-to-UNIX CoPy: A Tutorial

VOLUME IV

SYSTEM V/68 PROGRAMMING GUIDE, 72908 (41971-00)

- Introduction
- An Introduction to Shell
- C Programming Language
- FORTRAN
- Curses and Terminfo Package
- Programming Language EFL

SYSTEM V/68 SUPPORT TOOLS GUIDE, 72909 (41972-000)

- Introduction
- Maintaining Computer Programs (MAKE)
- Augmented Version of MAKE
- The M4 Macro Processor
- The AWK Programming Language
- Desk Calculator Language (BC)
- Desk Calculator Program (DC)
- Lexical Analyzer Generator (LEX)
- Yet Another Compiler-Compiler (YACC)
- Common Object File Format

SYSTEM V/68 ASSEMBLER USER'S GUIDE, 72910 (41973-00)

- Introduction
- Warnings
- General Syntax Rules
- Segments, Location Counters, and Labels
- Types
- Expressions
- Pseudo-Operations
- Span-Dependent Optimization
- Address Mode Syntax
- Machine Instructions

SYSTEM V/68 COMMON LINK EDITOR REFERENCE MANUAL, 72911 (41974-00)

- Introduction
- Using the Link Editor
- Link Editor Command Language
- Notes and Special Procedures
- Error Messages
- Syntax Diagram for Input Directives

VOLUME V

SYSTEM V/68 DOCUMENT PROCESSING GUIDE, 72906 (41969-00)

- Introduction
- Advanced Editing
- Stream Editor
- Nroff and Troff User's Manual
- Table Formatting Program
- Mathematics Typesetting Program
- Memorandum Macros
- Viewgraphs and Slides Macros

SYSTEM V/68 ERROR MESSAGE MANUAL, 72902 (41965-00)

- Introduction
- Error Messages
- Index

SYSTEM V/68
ADMINISTRATOR'S GUIDE

Product Code 72901

Part Number 41964-00

Version 1

PREFACE

The *SYSTEM V/68 Administrator's Guide* (Part Number 41964-00, Product Code 72901) is a reference volume for those who administer SYSTEM V/68, release version FE81. This guide should be used to supplement the information contained in the *SYSTEM V/68 User's Reference Manual*, *SYSTEM V/68 Administrator's Manual*, and the *VME Series 20 Software Release Guide*.

While reasonable efforts have been made to assure the accuracy of this document, Motorola assumes no liability resulting from any omissions in this document or from the use of the information obtained therein. Motorola reserves the right to revise this document and to make changes from time to time in its content without being obligated to notify any person of such revision or changes.

CONTENTS

FIGURES

TABLES

CAUTION

Some information in this document is specific to the operation of Motorola EXORmacs, VM03, VME/10, or VMEsystem 1131 computers. If you are working with a VME Series 20 system, the *VME Series 20 Software Release Guide* will provide details specific to your system and should supersede similar information found in this document.

In addition, the *VME Series 20 Software Release Guide* describes setting system variables, generating a kernel, configuration planning, initializing disks, and reinstalling software.

1. INTRODUCTION

1.1 General

Two document types provide information about SYSTEM V/68: manuals and guides. The manuals describe commands, facilities, features, and error messages of the system. The guides provide supplemental details and instructions for system implementation, administration, and use.

The *SYSTEM V/68 Administrator's Guide* is a reference volume for those who administer SYSTEM V/68. This guide should be used to supplement the information contained in the *SYSTEM V/68 User's Reference Manual, SYSTEM V/68 Administrator's Manual*, and the *VME Series 20 Software Release Guide.*

1.2 *SYSTEM V/68 Administrator's Guide* Organization

The following paragraphs contain a brief description of each chapter of the guide.

ADMINISTRATIVE GUIDELINES — Chapter 2 contains background advice and suggestions for administrators of SYSTEM V/68.

USING VME Series 20 — Refer to the VME Series 20 Software Release Guide for details on how to set system variables, generate a kernel, and perform other operator tasks.

ACCOUNTING — Chapter 4 describes the structure, implementation, and management of the accounting system. Appendix A lists the files contained in the accounting directories.

FILE SYSTEM CHECKING — Chapter 5 describes the filesystem check program (*fsck*(1M)). *Fsck* audits the filesystem and repairs inconsistency interactively.

LP SPOOLING SYSTEM — Chapter 6 the line printing program, *LP*, and describes the role of the LP administrator in performing restricted functions and overseeing the smooth operation of *LP.*

SYSTEM ACTIVITY PACKAGE — Chapter 7 describes the design and implementation of the SYSTEM V/68 activity package. The package reports system-wide statistics.

1.3 Manual Organization

The manuals are organized as alphabetized entries within tabbed sections. The *SYSTEM V/68 User's Reference Manual* contains sections 1 through 6. The *SYSTEM V/68 Administrator's Manual* contains sections 1M, 7, and 8. Throughout the documentation, references to these manuals are given as *name* (section). For example, *chroot*(1M) is a reference to the *chroot* entry in section 1M of the *Administrator's Manual.*

1.4 Entry Format

The following conventions identify arguments, literals, and program names:

- **Boldface** strings are literals and are to be typed as they appear.
- *Italic* strings represent substitutable argument prototypes and program names.
- Square brackets [] indicate that an argument is optional.
- Ellipses (...) show that the previous argument prototype may be repeated.

In addition, SYSTEM V/68 incorporates a new convention for naming disk and tape devices. (For VME Series 20 systems, refer to the *VME Series 20 Software Release Guide* for information on new naming conventions.) In earlier releases, the standard format for naming disk devices was

/dev/dk*xy*

where *x* referred to the disk device number and *y* referred to the disk section or partition. Raw access to a disk device was indicated with an **r**, e.g., **/dev/rdk01**. Standard format for naming tape devices was

/dev/mt*x*

where *x* referred to the magnetic tape device number. Raw access to a tape device was indicated with an **r**, e.g., **/dev/rmt0**.

The new naming convention creates separate subdirectories under **/dev** for each type of disk or tape device. The new format for disk devices is:

/dev/{r} dsk/ [r] *cntrlr_* **[** *controller_number* **d]** *drive_number* **s** *section_number*

Fields in square brackets are entirely optional: they do not affect the operation of any software or hardware; they are for informational purposes only, for the convenience of administrators, operators, and users. Fields in curly brackets represent options that affect software; they must be present if that option is being selected.

r (Not Required) (The first r) indicates a raw interface to the disk. The default is normal system buffering.

dsk/ (Required) Indicates that the device is a disk.

r (Not Required) (The second r) indicates that this disk is on a remote system.

*cntrlr_* (Not Required) Indicates the appropriate disk device in systems with multiple disk drivers. In SYSTEM V/68, the controller names are present and must be used on command lines that specify a disk device. Once the system has been installed, the system administrator may elect to eliminate the disk specification on single-drive systems. The disk devices available are:

- **vm21_** Intelligent Universal Disk Controller, M68KVM21
- **vm22_** Intelligent SMD Disk Controller with Floppy Disk, M68KVM22
- **wd_** Winchester Disk Controller, M68RWIN1
- **ud_** General Universal Disk Controller
- **c_** Generic controller

*controller_number***d** (Not Required) System administrators decide whether or not to specify the controller number in the disk device name. If the controller number is specified, the **d** introduces the drive number.

drive_number (Required) The drive number. The field is free format; there is no default drive number.

s*section_number* (Required) The section number. The field is free format; there is no default section number.

As an example, the name for disk drive 0, section 0 might be **/dev/dsk/vm22_0s0**. The new format for tape device names is:

/dev/{**r**}**mt/**[**c***controller_number***d**]***drive_number***[***density***]**{**n**}

where:

r (Not Required) Indicates a raw device. The default is a blocked device.

mt/ (Required) Indicates a magnetic tape device.

c*controller_number***d** (Not Required) The **c** introduces the controller number. System administrators decide whether or not to specify the controller number in the tape device name. If the controller number is specified, the **d** introduces the device number.

device_number (Required) The drive number. The drive number is followed immediately by the density value of the tape.

density (Required) Tape density must be specified for each drive. The density is indicated with an **h**, **m**, or **l**, where:

- **h** (high) is a tape density of 6250 bpi
- **m** (medium) is a tape density of 1600 bpi
- **l** (low) is a tape density of 800 bpi

n (Not Required) Indicates no rewind on close. The default condition is to rewind.

As an example, the name for a 6250 bpi magnetic tape drive might be **/dev/mt/0h**.

To make the transition to the new names less painful, the old names can exist in SYSTEM V/68 software. However, all documentation and sample shell scripts distributed with this release use the new naming convention. System administrators are encouraged to rename existing devices and incorporate the new names into shell scripts as soon as possible.

The following table compares existing device filenames with the new filenames that will be found in the documentation.

Disk Devices		Tape Devices	
Old Disk Name	New Disk Name	Old Tape Name	New Tape Name
/dev/dk00	**/dev/dsk/***cntrlr***_0s0**	**/dev/mt01**	**/dev/mt/0l**
/dev/dk10	**/dev/dsk/***cntrlr***_1s0**	**/dev/mt5**	**/dev/mt/5mn**
/dev/rdk00	**/dev/rdsk/***cntrlr***_0s0**		

2. ADMINISTRATIVE GUIDELINES

2.1 Areas of Concern

This chapter provides system administrators with the background necessary to anticipate and avoid problem areas that can develop in the course of installing and administering SYSTEM V/68. Most of the issues addressed in this section are resolved easily with early planning and preparation. Other issues have been targeted for the administrator as areas that bear watching. In general, the discussions in this chapter provide administrators with advice and suggestions for implementing SYSTEM V/68 in the most efficient and least time-consuming manner.

The discussions in this chapter assume that the administrator possesses a basic understanding of the material contained in all sections of the *SYSTEM V/68 Administrator's Manual* and the *SYSTEM V/68 User's Reference Manual.*

Specific references are made in this guide to many of the programs described in the *SYSTEM V/68 User's Reference Manual* and the *SYSTEM V/68 Administrator's Manual.* In reviewing the material contained in the manual, special attention should be paid to: *acct*(1M), *checkall*(1M), *chmod*(1), *chown*(1), *config.68*(1M), *cpio*(1), *date*(1), *dcopy*(1M), *df*(1M), *du*(1), *ed*(1), *env*(1), *errpt*(1M), *find*(1), *fsck*(1M), *fuser*(1M), *kill*(1), *mail*(1), *mkdir*(1), *mkfs*(1M), *ncheck*(1M), *ps*(1), *rm*(1), *rmdir*(1), *shutdown*(1M), *stty*(1), *su*(1), *sync*(1M), *time*(1), *volcopy*(1M), *wall*(1M), *who*(1), and *write*(1); *acct*(4); Section 7; and *crash*(8).

The topics in this chapter are arranged in the order that an administrator will encounter them while setting up the system. Administrators should not attempt to set up SYSTEM V/68 without having read this chapter thoroughly.

2.2 Single-user and Multi-user Modes

The system as shipped moves automatically through **single-user mode** and comes up in **multi-user** mode. Note that single-user mode does **NOT** refer to a mode used to support only one user. Instead, single-user mode refers to the mode of operation used to perform functions such as installing software on the "root" device, perform filesystem checks or repairs, or do system backups; in short, to perform any function that requires one user to keep exclusive control over the system. In this sense, single-user mode might be more appropriately named "System Maintenance Mode".

2.3 Error Messages from fsck

The File System Check Program, *fsck*(1M), is an interactive filesystem check and repair program that may be invoked optionally during the system boot. *Fsck* is a multi-pass filesystem check program; each pass invokes a different phase of the *fsck* program. The error messages explained here follow the normal Phase 1 through Phase 6 checks of the program. (The "File System Checking" chapter of this guide contains detailed information concerning normal updating of the filesystem, possible causes of filesystem corruption, and recommended corrective procedures.)

Experience shows that administrators need only be concerned with a small number of the possible error messages. Generally, the *fsck* program displays each error message accompanied by a prompt for action. The prompt is usually "Fix?", "Salvage?" or

"Remove?". In nearly every case, the administrator should respond "y" meaning "yes", which gives *fsck* the go-ahead to repair the filesystem.

Occasionally, the fix requires the administrator to reboot the system immediately. When an immediate reboot is required, the following message appears:

****** BOOT UNIX (NO SYNC!) ******

This message appears when the root filesystem has been damaged and corrections have been made on the disk itself. At this point, the in-core buffer and inode cache do not agree with the disk. If the administrator enters a **sync** command, the corrupted information contained in the in-core buffer is written out to the disk, wiping out the corrections on the disk. Therefore, when this message appears, turn the RESET key switch to RESET and reboot – DO NOT perform a "sync" before resetting the machine.

Fsck makes six passes through the filesystems, Phase 1 through Phase 6. The remainder of this section describes the most commonly received error messages and the recommended responses. The error messages are arranged by Phase number and follow the normal Phase 1 through Phase 6 program checks.

2.3.1 Phase 1: Check Blocks and Sizes.

Phase 1 lists error conditions discovered while checking inode types, examining inode block numbers for bad or duplicate blocks, checking inode size and checking inode format. In general, *fsck* checks that the size given for each file agrees with the number of disk blocks reserved for the file.

In the listings that follow, the screen messages are shown in boldface type.

UNKNOWN FILE TYPE I=<inode #>

Fsck does not know file inode #I's type, e.g., directory, regular, FIFO, character special, or block special.

(NOT EMPTY)

The problem file is not empty.

CLEAR?

Fsck is asking if you want to clear the possibly damaged file. If you answer "y", the file will disappear. Typically, you should enter "y". One case where an administrator might want to type "n" is when *fsck* has provided only the inode number of a file and the filename is not known. To recover the file from backup disk (or tape), the administrator must know the filename. To learn the name of a file when the inode number is known, refer to *ncheck*(1M) or *ff*(1M).

LINK COUNT TABLE OVERFLOW

This message results from an internal *fsck* problem. Try again.

CONTINUE?

Fsck asks "Do you want to continue?" from many different places within the filesystem checking program. The implication is that there are so many problems with the filesystem that *fsck* is running out of space. A good rule of thumb is to allow *fsck* to continue three or four times before giving up. Enter "y" to continue.

Sometimes, a problem that *fsck* encountered earlier in the program can be related to a lengthy **CONTINUE?** sequence. For example, during Phase 1, *fsck* may tell you that you have damaged four files. Phase 2 may start cleaning up the files and repeatedly prompt **CONTINUE?**. Here, you know that there are only four files that have been damaged so you should continue until the four files are fixed.

PARTIALLY ALLOCATED INODE I=<inode #>

Fsck is telling you that the filesystem crashed in the process of creating a new file. Unless the partially filled file is extremely important, clear when prompted.

2.3.2 Phase 1B: Rescan For More DUPS.

DUPS refers to a "duplicate block". Each inode contains a list or pointers to lists of all the blocks claimed by the inode. *Fsck* compares each block number claimed by an inode to a list of already allocated blocks. If a block number appears in more than one inode, the block number is added to a list of duplicate blocks. *Fsck* does not find all possible duplicate blocks on its first pass. When DUPS are found on the first pass, *fsck* rescans looking for others.

2.3.3 Phase 2: Check Pathnames.

In this phase, *fsck* checks that the directory structure is consistent with the file inodes. A directory is a file formatted with records containing the inode numbers of files in the directory, along with the files' names. *Fsck* verifies that the filesystem directory is intact.

ROOT INODE UNALLOCATED. TERMINATING.

If the root directory inode is not marked as allocated, *fsck* assumes that the filesystem is nonexistent or has been so severely damaged that further analysis is useless.

ROOT INODE NOT DIRECTORY

This problem is similar to the previous one, but this one can be repaired. *Fsck* asks the administrator if it should proceed with the repair.

FIX?

Whenever damage can be fixed, *fsck* asks if the repair should be made. Unless the "fixing" will destroy extremely vital information, answer "y" (yes).

DUPS/BAD IN ROOT INODE

This problem is severe. When the root directory is damaged, it is nearly impossible to recover the filesystem intact. If *fsck* thinks it can fix it, it will ask "FIX?".

2.3.4 Phase 3: Check Connectivity.

This checking phase verifies that directories correctly point to their parent directories, and parent directories correctly point to their subdirectories.

2.3.5 Phase 4: Check Reference Counts.

This checking phase verifies that when an inode has *N* links, the inode appears in *N* places in the filesystem. The check includes directories.

UNREF

An inode is allocated but it does not appear in any directory. A "y" response will reconnect the inode to the filesystem in the **lost+found** directory. (Refer to paragraph 2.5 for additional information about the **lost+found** directory.)

FREE INODE COUNT WRONG IN SUPERBLK

The "superblock" is the filesystem description area (described in **/usr/include/sys/filsys.h** and in *fs*(4)). The superblock contains a value that should be equal to the total number of free inodes that are left to be allocated. If the number does not agree with the unallocated inodes counted during the reference check, this message appears. This condition is repairable; when prompted for action, enter "y".

2.3.6 Phase 5: Check Free List.

The filesystem structures contain a list of "free blocks" that are available for new files. This phase checks the free block list for errors.

BAD BLKS IN FREE LIST

A block number in the free list is invalid. Usually the bad number points to a block beyond the end of the filesystem. This problem can occur if the contents of a regular file are interpreted as a free list block. This condition is repairable; when prompted for action, enter "y".

DUP BLKS IN FREE LIST

A block in the free list either appears more than once in the free list or it appears in a file. In either case, the problem is repairable although some file may be damaged.

FREE BLK COUNT WRONG IN SUPERBLK, *id,devname*;

This condition is similar to the FREE INODE COUNT WRONG message above. The number of blocks counted in the free list disagrees with the value in the superblock. This condition is repairable; when prompted for action, enter "y".

BAD FREE LIST

Any free list problem generates this message.

SALVAGE?

Fsck is asking if you want to repair the free list. Users should answer "y" to this prompt nearly every time. It would be exceedingly rare that the administrator would not salvage the free list. The exception might be when repairing the free list would add blocks that had been in a damaged file back to the free list. It might be that a system administrator would prefer to first try to recover the blocks from the damaged file before letting *fsck* recover the free list.

BLK(S) MISSING

Fsck has checked the filesystem files and the free list and still cannot account for some blocks. This problem is easily remedied and should be fixed. When prompted for action, enter "y".

2.3.7 Phase 6: Salvage Free List.

During this phase, *fsck* is reconstructing the free block list.

******* BOOT UNIX (NO SYNC!) *******

This message appears when the root filesystem has been damaged and corrections have been made on the disk itself. At this point, the in-core buffer and inode cache do not agree with the disk. If the administrator enters a **sync** command, the corrupted information contained in the in-core buffer is written out to the disk, wiping out the corrections on the disk. Therefore, turn the RESET key switch to RESET and reboot – DO NOT perform a "sync" before resetting the machine.

******* FILE SYSTEM WAS MODIFIED *******

This is a notice that a repair has changed some information related to the filesystem.

EXCESSIVE BAD BLKS I=<inode #>

Fsck estimates there are too many bad blocks in the current file. Usually, it will ask whether you want to continue.

EXCESSIVE DUP BLKS I=<inode #>

Fsck estimates there are too many duplicate blocks in the current file. Usually, it will ask whether you want to continue.

DUP TABLE OVERFLOW

If this message appears, your filesystem may be hopelessly damaged. *Fsck* was unable to complete its fix. However, *fsck* may have corrected enough problems so that the fix can be completed if the filesystem check program is run a second or third time. Turn the RESET key switch to RESET and run *fsck* again.

EXCESSIVE BAD BLKS IN FREE LIST

Fsck estimates there are too many bad blocks in the free list. Usually, it will ask whether you want to continue.

EXCESSIVE DUP BLKS IN FREE LIST

Fsck estimates there are too many duplicate blocks in the free list. Usually, it will ask whether you want to continue.

POSSIBLE FILE SIZE ERROR

Experience shows this message usually occurs with FIFO files, where it is not a problem. When creating SYSTEM V/68 files, the writing program can occasionally leave "holes" in a file. When *fsck* detects this condition, it prints the "file size error" message. If the damaged file is **/etc/mnttab**, it may represent a damaged inode and possible data loss. The system administrator should notify the owners of the files so they may determine if damage has been done, and if so, recover the file from backups.

2.4 Variations in Block Sizes

SYSTEM V/68 introduced 1Kb logical blocks. For example, all system buffers and filesystem blocks are 1024 bytes each. However, remnants of 512-byte blocks can be found. Most filesystem-related commands still report in 512-byte block units. The commands *ls*(1), *du*(1), *df*(1), *labelit*(volcopy(1M)), and *grep*(1) are examples of this. The *mkfs*(1M) command expects its *blocks* argument to be expressed in 512-byte physical blocks, however it reports the number of blocks created as 1Kb logical blocks.

The root filesystem is distributed as a 1Kb block filesystem. Filesystem-related commands have been changed internally to handle both types of filesystems. While users are encouraged to convert their old filesystems to the larger size block, 512-byte block filesystems are acceptable.

2.5 Lost + Found Directory

The **lost+found** directory serves as a temporary holding location for lost files in the system. Typically, an inode will be discovered as "lost" during the filesystem check, *fsck*(1M). A lost file or directory is one that is allocated but unreferenced. If the file or directory is not empty, it is usually connected to the directory **lost+found.** Therefore, the directory **lost+found** must exist in every mountable filesystem before the *fsck* program is initiated.

In addition, the **lost+found** directory must contain empty slots into which entries can be made. This is accomplished by creating **lost+found**, creating several files in the directory, and then removing them before *fsck* is executed. An easy way to do this is to create and execute a command script file **/etc/mklost+found** that contains:

```
mkdir lost+found
cd lost+found
i=1
while [ $i  -lt 30 ]
do
        echo > $i
        i=`expr $i + 1`
done
rm *
```

Notice that in the third line, the i is assigned a value of 1, and the option in the expression **[$i -lt 30]** is a lowercase L, not the numeral 1. Include a space in the expression between the **i** and the option dash.

2.6 Passwords and New Users

The SYSTEM V/68 operating system is shipped without passwords. The system administrator is encouraged to install passwords, especially for root, as soon as possible.

The files **/etc/passwd** and **/etc/group**, as shipped, contain the minimum required set of system login IDs and group IDs, respectively. As shipped, the **/etc/passwd** file arrives "write protected". Administrators should refer to *chmod*(1) for information about changing the write permission for the file. Detailed instructions for creating and changing passwords are provided in paragraphs 2.22.6, "**/etc/password**", and 2.22.7, "**/etc/group**", of this guide. Instructions for creating dialup passwords are also included in 2.22.6.

Adding new users to the system is a three-step process. First, the system administrator makes a new entry in the password file, **/etc/passwd** (refer to *passwd*(1) and *passwd*(4)). As

part of the new entry, the administrator assigns each user a group ID, usually for reasons of security or to restrict file access. Typically, each user's group ID is based on the user's relationship to other users. (Refer to *group*(4) and paragraph 2.22.7 of this guide.)

The second step in the procedure is to create a home directory for each new user. Third, after the home directory is created, the administrator must change the ownership and the group identification of the directory from **root** to the new user. Care should be taken that the home directory has the appropriate permissions (refer to *mkdir*(1), *chown*(1), and *chgrp*(1)).

As user login IDs are added to **/etc/passwd**, the system administrator can verify additions with *pwck*(1M). Similarly, additions of user group IDs to **/etc/group** can be verified with *grpck*(1M).

After users gain access to the system, the command **passwd** enables them to enter their own unique passwords.

2.7 Setting SYSTEM V/68 Variables

The shell variable **TZ** defines the time zone. **TZ** is defined in three files: **/etc/bcheckrc**, **/etc/rc** (refer to *brc*(1M)), and **/etc/profile**. System administrators should modify these files if **TZ** needs a different definition (*ctime*(3C)). For example, for Mountain Standard or Eastern Standard time zones:

```
TZ=MST7; export TZ
TZ=EST5EDT; export TZ
```

The system environment contains the following exported variables:

- MMU
- PROCESSOR
- STACKCHECK
- SYSTEM
- DBLALIGN
- OPTIM
- STALIGN
- FPU
- FP

The object distribution file /etc/profile contains the appropriate values for each variable They are as follows:

TABLE 2-1. System Variables

VARIABLE	VALUE
MMU	M68851
PROCESSOR	M68020
STACKCHECK	OFF
SYSTEM	VME131
DBLALIGN	YES
OPTIM	BOTH
STALIGN	NO
FPU	M68881
FP	NULL

2.8 Setting Terminal Variables

Before users can use the full-screen editor *vi*(1), their terminal type must be defined to the system. The recommended approach is to define the terminal in a user's **.profile** (*profile*(4)) per some entry in **/usr/lib/terminfo** (*terminfo*(4) or *term*(4)). For example:

TERM=vme10; export TERM
or
TERM=155; export TERM

In addition, SYSTEM V/68 is distributed with each terminal's *stty*(1) erase character defined as the hash character (#) (the UNIX standard default value). To change this to the more commonly preferred backspace character, enter:

stty erase ^H

This change may be included in the file **/etc/profile** for a system-wide change. Optimally, this change should also be included in a user's **.profile**. To insert the character **^H** (CTRL H) into a file using the *vi*(1) text editor, enter insert mode and type the two-character sequence **^V^H**.

A second recommended change is to redefine the "kill" character from the "@" symbol to some other, less frequently used, character. A common choice is a ^X (CTRL X) combination.

In general, there are a number of variables a user may change to customize a terminal. For a complete explanation of the *terminfo*(4) (terminal capability data base) variables, refer to *terminfo*(4).

2.9 Job Control

Job control offers users more control over their processes and greater flexibility when working on more than one type of job. Job control can be used from a single terminal to maintain and interact with different environments, e.g. different working directories or different effective user-ids. This is managed by providing users with different invocations of the shell. The different invocations may be conceptualized as shell "layers". By using two different shell layers, a user can leave a foreground process, work on a second process on a second layer, and then return to the first process.

Job control is implemented as a user-level program, *shl*(1). The operating system terminal drivers are changed to recognize an additional control character. The *sxt*(7) driver communicates with *shl*, multiplexing output from multiple layers onto the terminal. The include files **tty.h** and **termio.h** and related commands are affected. This section describes the procedure for including job control in the operating system as an option at system configuration.

The *sxt*(7) driver supports the *shl*(1) program, which is the user interface to the job control feature. The following steps describe the installation procedures for the *sxt*(7) driver.

1. Locate the *sxt* definition in **/etc/master** (*master*(4)) and note the major number used for the software device.
2. The configuration file contains a line that provides a default value of eight *sxt* devices. This line in the configuration file includes the *sxt* driver in the operating system.
3. In **/dev**, create *sxt* devices with the following script:

```
major=(look in /etc/master for sxt)
cd /dev
mkdir sxt
chmod 755 sxt
for link in 00 01 02 03 04 #...number of devices desired
do
        for chan in 0 1 2 3 4 5 6 7
        do
                echo ${link} ${chan}
                mknod  sxt${link}${chan}  c  ${major}  ${link}${chan}
                ln sxt${link}${chan} sxt/${link}${chan}
        done
done
```

4. Verify that the *shl*(1) program has been installed in the system.

The system administrator indicates the number of *sxt* devices with an entry in **/etc/master** or in the configuration file (refer to *master*(4) and *config*(1M)). For each device, about 800 bytes of operating system kernel space are required. If too many devices are requested, the error message

sxt cannot allocate link buffers

appears on the console. If this occurs, either increase the size of the kernel or decrease the requested number of *sxt* devices.

2.10 System Kernels and make Command

The following reference table lists the system kernel names and the new kernels that result from the *make*(1) command. These kernels are needed as part of the software installation procedure described in Chapter 3. Note that the administrator should be logged in as **root** and located in directory **/usr/src/uts/m68k** before executing the *make* command.

2.11 Line Printer Configuration

Briefly, SYSTEM V/68 contains the *lp*(1) family of utilities for use in configuring the *lp*(1) system. The system administrator (root) configures the *lp*(1) system (one time per printer) from the console as follows:

```
PATH=:$PATH
cd /usr/lib
lpshut
lpadmin -pprl -mpprx -v/dev/lp0     #parallel printer
lpadmin -dprl       # only for the default printer
accept prl
enable prl
lpsched
```

where *prl* is the printer name, *pprx* is the model, and *lp0* is the device name.

The program *lpshut*(1M) (refer to *lpsched*(1M)) shuts down the current *lpsched*(1M). The first *lpadmin*(1M) command defines a printer "*prl*" (you select the name) and a model interface program ("*pprx*" for the parallel Printronix; refer to *lpadmin*(1M) for the others) connected to the device **/dev/***lp0*. The second *lpadmin*(1M) calls "*prl*" the default printer. *Accept*(1M) says "*prl*" is ready to accept spooled files. *Enable*(1M) indicates "*prl*" is ready to print. The final line restarts the scheduler.

If you have a serial printer, change the first *lpadmin*(1M) command to:

```
lpadmin -pprl -mprx -v/dev/tty400     #serial printer
```

Detailed information regarding the LP spooling system is contained in the "Lp Spooling System" chapter of this guide.

2.12 Sticky Bits

Typically, files are not stored contiguously on the disk. When the system is loading a file for execution, noncontiguous disk storage creates high overhead on the system. In SYSTEM V/68, a special mode can be assigned to a frequently used program so that an executable image is stored in temporary storage, contiguously, in the system swap space when the program is not being used. Files that are designated to be stored in the temporary swap area are assigned a "sticky mode" and the bit that controls the process is the "sticky bit". Files that are stored in the temporary swap area remain there until the system is rebooted.

The system is shipped with no sticky bits set (refer to *chmod*(1) and *chmod*(2)). The utilities selected for "sticky mode" should be utilities that are frequently used, but not used all the time. Utilities that are suggested for sticky mode include *vi*(1), *nroff*(1) if users invoke *nroff* frequently, and the C-compiler, with its associated files:

/bin/cc
/bin/as
/bin/ld
/lib/c0
/lib/c1
/lib/optim
/lib/cpp

Two methods are available to check how much space is left in the swap area. Users with "superuser" status can invoke *crash*(1M) and execute the command **map swapmap**. Other users can invoke *sysdef*(1M) which displays the device, the starting block number of the swap area, and the number of 512Kb blocks available.

2.13 UNIX-to-UNIX CoPy (uucp) Programs

Customers who plan to connect their system to the uucp network need to change the nodename present in the kernel. Before making any changes, administrators are urged to read the UNIX-to-UNIX CoPy (uucp) Tutorial in the *SYSTEM V/68 User's Guide*. The complete uucp installation procedure is contained in the tutorial.

2.14 M68KVM30 Module Installation

A example installation for the M68KVM30 Multi-Channel Communications Module is included in this section. The example is generic; administrators must furnish the kernel name appropriate to their system.

1. Login as **root**. Generate a kernel with the M68KVM30 module included:

   cd /usr/src/uts/m68k
   vi makefile

   Change the line in **makefile** for the kernel you want to generate. (For information on kernel names, refer to the "System Kernels and make Command" paragraph earlier in this chapter.) Locate the line that applies to your kernel. For this example, assume your system requires the **macs80** kernel. Locate the lines:

```
macs80:
        $(MAKE)\
                $(MACSARGS)\
                DFOTPS="-DVM21=1 -DVAM=2 -DV21KD00=CMD80        ...
```

   Change the assignments in the last line as follows:

```
                DFOPTS="-DVM21=1 -DVAM=x -DMCCM=x  -DV21DK00=CMD80  ...
```

   where *x* is equal to the number of modules in your system.

2. Generate a new kernel:

   make *kernelname*

3. The newly generated kernel is named:

   /usr/src/uts/m68k/M680*x*0/*kernelname*unix.

   where the *x* is either a **0** or **1**, depending on whether the kernel is for an EXORmacs, a VM03, or a VME/10.

Move the kernel to the root directory for testing:

mv M680*x*0i/*kernelname*unix /unix.mccm
sync
sync
sync

Turn the RESET key to RESET and reboot with the new kernel name.

4. Test the kernel by executing routine commands such as **ls -l**. If the system performs well, make the new kernel the default kernel:

   mv /unix /old.unix
   mv unix.mccm /unix
   rm /stand/unix
   ln /unix /stand/unix

5. To obtain the major device names, change directories to:

   cd /usr/src/uts/m68k/M68020/*system*cf

   where *system* refers to **MACS** for EXORmacs, **VME10** for VME/10, or **VM03** for the VM03.

   Look at the file *kernelname***conf.c** and locate the line that reads:

   /* *N* */ mccmopen, mccmclose, mccmread, mccmwrite, mccm...

   The number *N* is the major device number.

6. Add MCCM ports to the device list using the major device number obtained above:

   mknod /dev/tty*xw* c *N* 0
   mknod /dev/tty*xx* c *N* 1
   mknod /dev/tty*xy* c *N* 2
   mknod /dev/tty*xz* c *N* 3

   where *xw*, *xx*, *xy*, and *xz* are tty numbers and *N* is the major device number.

   Also make a node for the printer port:

   mknod /dev/tty*xp* c *N* 4

   where *xp* is a tty number and *N* is the major device number.

   If your system contains a second MCCM module, make a second set of nodes with the following commands:

   mknod /dev/tty*yw* c *N* 5
   mknod /dev/tty*yx* c *N* 6
   mknod /dev/tty*yy* c *N* 7
   mknod /dev/tty*yz* c *N* 8
   mknod /dev/tty*yp* c *N* 9

   where *yw*, *yx*, *yy*, *yz*, and *yp* are tty numbers, and *N* is the major device number.

7. To make the MCCM ports into login ports, append the following lines to the file **/etc/inittab**:

_xw_:2:respawn: /etc/getty tty_xw_ 9600155
_xx_:2:respawn: /etc/getty tty_xx_ 9600155
_xy_:2:respawn: /etc/getty tty_xy_ 9600155
_xz_:2:respawn: /etc/getty tty_xz_ 9600155

where *xw*, *xx*, *xy*, and *xz* are tty numbers. If there is more than one MCCM, each device must be added to **/etc/inittab**.

8. The new kernel is now the default kernel. Shut down the system (refer to *shutdown*(1M)), execute **sync** three times, and reboot.

   To login to the MCCM ports, the system must be in multi-user mode. Therefore, after booting, enter the command:

   init 2

   to enter multi-user mode.

2.15 Configuration Planning

The SYSTEM V/68 operating system supplied in the object distribution supports only the debug ports and a disk controller (disk drive 0). Each system administrator must describe the actual configuration of the system.

All operating system source code and object libraries are in **/usr/src/uts**. All configuration information is kept in the directories **/usr/src/uts/m68k/M68020/cf**. Two files must be changed to reflect system configuration: **low.s** and **conf.c**. The program *config.68*(1M) should be used to make these changes.

Config.68 requires a "system description file" (refer to *config.68*(1M)) and produces the two needed files. One file, **low.s**, provides information regarding the interface between the hardware and device handlers. The second file, **conf.c**, is a C program that defines the configuration tables for the various devices on the system.

Table 2.3 lists the values and sizes of the basic parameters for VME Series 20. For more details of syntax and structure, refer to *config.68*(1M) and the associated *master*(4).

TABLE 2-2. System Parameters

ITEM	
nswap (70Mb)	7680
buffers	256
hashbuf	64
physbuf	4
inodes	90
files	90
mounts	8
coremap	100
swapmap	75
calls	50
procs	128
texts	100
clists	150
maxproc	25

The first part of the system description file lists all of the hardware devices on the system. Next, various system information is listed. A brief explanation of this information follows:

- **root:** Specifies the device where the **root** filesystem is to be found. The device must be a block device with read/write capability because this device is mounted read/write as "/". Thus, a tape cannot be mounted as the **root** but can be mounted as some read-only filesystem. Normally, **root** is disk drive 0, section 0.
- **pipe:** Specifies where pipes are to be allocated (must be a *mounted* filesystem – the root filesystem is normally used).
- **dump:** Specifies the device to be used to dump memory after a system crash.
- **swap:** Specifies the device and blocks that are used for *swapping*. *Swplo* is the first block number used and *nswap* indicates how many blocks, starting at *swplo*, to use. Care must be taken that the swap area specified does not overlap any filesystem. For example, if section 0 is 8000 blocks long, the **root** filesystem could occupy the first 6000 blocks, and **swap** the remaining 2000 by specifying:

  root disk 0
  swap disk 0 6001 1999

- **buffers:** Specifies how many "system buffers" to allocate. Real-time response improves as more buffers are allocated. System buffers form a "data cache". Improvement in the hit rate of this cache tends to fall as the number of buffers is increased.
- **hashbuf:** Specifies how many hash buckets to allocate. These are used to search for a buffer given a device number and block number. This number must be a power of two. The default value is 64.
- **physbuf:** Specifies how many physical I/O buffer headers to allocate. One is needed for each physical read or write active. The default value is 4.
- **inodes:** Specifies how many "inode table" entries to allocate. Each entry represents a unique open inode. When the table overflows, the warning message "Inode table overflow" is printed on the console. The table size should be increased if this happens

regularly. The number of entries used depends on the number of active processes, texts, and mounts.

- **files:** Specifies how many "open-file table" entries to allocate. Each entry represents an open file. When the table overflows, the warning message "no file" is printed on the console. The table size should be increased if this happens regularly.
- **mounts:** Specifies how many "mount table" entries to allocate. Each entry represents a mounted filesystem. The **root** (/) is always the first entry. When full, the *mount*(2) system call returns the error EBUSY.
- **swapmap:** Specifies how many entries to allocate to the "list of free swap blocks". It is similar to **coremap**, except it represents free blocks in the swap area in 512-byte units.
- **calls:** Specifies how many "call-out table" entries to allocate. Each entry represents a function to be invoked at a later time by the clock handler. The time unit is 1/60 of a second. The call-out table is used by the terminal handlers to provide terminal delays and by various other I/O handlers. When the table overflows, the system crashes and prints the panic message "Timeout table overflow" on the console. This value must be greater than two.
- **procs:** Specifies how many "process table" entries to allocate. Each entry represents an active process. The scheduler is always the first entry and **init**(1M) is always the second entry. The number of entries depends on the number of terminal lines available and the number of processes spawned by each user. The average number of processes per user is in the range of 2 through 5. When full, the **fork**(2) system call returns the error EAGAIN.
- **texts:** Specifies how many "text table" entries to allocate. Each entry represents an active read-only text segment. Such programs are created by using the **–i** or **–n** option of the loader *ld*(1). When the table overflows, the warning message "out of text" is printed on the console.
- **clists:** Specifies how many "character list buffers" to allocate. The buffers are dynamically linked together to form input and output queues for the terminal lines and various other slow-speed devices. The average number of buffers needed per terminal line is in the range of 5 through 10. When full, input characters from terminals are lost and not echoed.
- **maxproc:** Specifies how many concurrent processes a non-superuser is allowed to run.
- **power:** Specifies whether to attempt restart after a power failure. A value of 0 (default) indicates no restart; a value of 1 attempts power-fail restart. On restart, device drivers are called and process 1 (i.e., *init*) is sent a hangup signal; refer to *init*(1M).
- **sema:** Specifies whether to include semaphore code. A value of 0 (default) indicates no semaphores; a value of 1 includes semaphores.
- **shmem:** Specifies whether to include shared memory code. A value of 0 (default) indicates no shared memory; a value of 1 includes shared memory.
- **mesg:** Specifies whether to include message code. A value of 0 (default) indicates no messages; a value of 1 includes messages.
- **shmmax:** Specifies the maximum size of a shared memory segment.
- **shmmin:** Specifies the minimum size of a shared memory segment.

- **shmmni:** Specifies the maximum number of shared memory segments in the system.
- **shmseg:** Specifies the maximum number of shared memory segments a user may have attached.
- **shmall:** Specifies the maximum amount of shared memory that may be allocated system wide. The default value is 512 clicks, 256Kb.
- **shmbrk:** Specifies the number of clicks between the end of the data segment, and the beginning of the first shared memory segment if the default starting address is used allowing the user to continue to use *sbrk*(2) or *brk*(2). The default value is 16 clicks, 8Kb.
- **msgmax:** Specifies the maximum message size.
- **msgmnb:** Specifies the maximum number of bytes on any one queue.
- **msgtql:** Specifies the number of system message headers, i.e., maximum number of outstanding messages.
- **msgssz:** Specifies the message segment size. Messages consist of a set of contiguous message segments large enough to fit the text. The segments are used to help eliminate fragmentation and speed message buffer allocation. A message may span several segments.
- **msgseg:** Specifies the number of message segments in the system.
- **msgmap:** Specifies the message segment map size.
- **msgmni:** Specifies the maximum number of message queues system wide. The default is 10.
- **semmap:** Specifies the number of entries in the semaphore map. The map is used by the system to allocate and free semaphore sets. This parameter should be changed to reflect changes in *semmns*.
- **semmni:** Specifies the number of semaphore identifiers, i.e., number of semaphore sets.
- **semmns:** Specifies the number of semaphores in the system.
- **semmnu:** Specifies the number of undo structures in the system.
- **semume:** Specifies the maximum number of undo entries per structure.
- **semmsl:** Specifies the maximum number of semaphores per semaphore identifier.
- **semopm:** Specifies the maximum number of semaphore operations per **semop**(2) call.

2.16 Managing Disk Space

Managing disk space is a broad term that describes several different areas that are the responsibility of the system administrator. These areas include:

- Initial system installation (Program selection)
- Filesystem partitioning
- Disk free space
- Filesystem backups
- Monitoring disk usage

- Controlling disk usage
- Filesystem reorganization
- Controlling directory size

2.16.1 Filesystem Partitioning.

Each physical pack is partitioned into eight logical sections. This partitioning is defined in the operating system by a table with eight entries. Each table entry is two words long. The first specifies how many blocks are in the section; the second specifies the starting cylinder; refer to *cmd16*(7), *cmd80*(7), *sa800fl*(7), *lark8*(7), and *lark25*(7) for default cylinder and block assignments. These values are described to the system in the header file **/usr/include/sys/io/***cntrlr***io.h.**

A filesystem starts at block 0 of a section of the disk and may be as large as the section; if it is smaller than the size of a section, the remainder of the section is unused. Note that the sections themselves may overlap physical areas of the pack, but the filesystems must never overlap.

The program *mkfs*(1M) (for 1K byte/block filesystems) or *omkfs* (for 512 byte/block filesystems) initializes a section of the disk to be a filesystem. The length of each section of the disk is specified in 512-byte blocks. When *mkfs* is used, it produces half the number of 1Kb filesystem blocks. The 1Kb blocks provide better throughput for the particular filesystem. A 512 byte/block filesystem can be made using *omkfs* in place of *mkfs*. The number of physical disk blocks (512 bytes each) used to make the entire filesystem would be the same for either command.

Next, the program *labelit* (*volcopy*(1M)) labels the filesystem with a name and the name of the pack. Finally, the filesystem may be checked for consistency with *fsck*(1M). The filesystem can be mounted with *mount*(1M). The **lost+found** directory should be created at this point. An example of the complete procedure (including the command script file for the **lost+found** directory described in paragraph 2.5) is:

mkfs /dev/dsk/*cntrlr_XsX size*
labelit /dev/rdsk/*cntrlr_XsX filesystem_name volume*
mount /dev/dsk/*cntrlr_XsX /filesystem_name*
cd *filesystem_name*
/etc/mklost+found

For information on file system partitioning for specific controllers, refer to *cmd16*(7), *cmd80*(7), *lark8*(7), lark25(7), *sa400fl22*(7), *sa400flwd*(7), *sa800fl21*(7), *sa800fl22*(7), *vm21*(7), *vm22*(7), *wd15*(7), *wd40*(7), *wd70*(7), and *wd140*(7).

2.16.2 Disk Free Space.

Maintaining enough free disk blocks and free inodes (file headers) can be a problem. If the free inode count falls below 100, the system spends most of its time rebuilding the free inode array. If a filesystem runs out of space, the system prints "no-space" messages, and does little else. The *df*(1M) program prints a summary of the free blocks and free inodes on the online filesystem. To avoid problems, the following start-of-day free counts should be maintained:

- The filesystem containing **/tmp** and **/usr/tmp** (temporary files):

- 4-user system: 1500 free kilobytes.
- 8-user system: 3000 free kilobytes.

- The filesystem containing **/usr**:

  - 3000 to 6000 free kilobytes, depending on load.

- Other user filesystems:

  - 6 to 10 percent free, depending on user habits (3000Kb minimum).

When deciding how big the filesystem should be, set aside space on each drive for a copy of **root/swap** and use the rest of the pack for a single filesystem. However, if user groups contend for disk space, it may be better to divide a pack into more than one filesystem.

2.16.3 Filesystem Backups.

The easiest and most reliable way to minimize data losses when a system crashes is to perform frequent filesystem backups. Weekly backups typically result in data that is old and out of date while daily backups can be disruptive to system operations. System administrators should evaluate their own situations to find the optimal schedule, based on the frequency of data loss, the value of the data and the amount of effort involved in a backup.

For most systems, a twice-weekly backup of filesystems is recommended. Identify one backup set as, say, the "Monday morning" set, and the other as the "Thursday morning" set. Re-use each backup set each week. The most recent disks (or tape) should be kept off premises or in a fireproof safe. Backup disks (or tapes) should be read periodically to make sure they are readable.

When the system goes down, active files can become scrambled. In addition to good backup procedures, filesystem patching expertise must be available (on-site or on-call). If the system is rebooted for general use without first checking the filesystems, problems occur. Study *checkall*(1M), *fsck*(1M), and *crash*(8) and the "File System Checking" chapter for more information.

The following backup programs are distributed with SYSTEM V/68:

- **find/cpio**: SYSTEM V/68 is distributed in *cpio* format. The **-cpio** option of the **find** command can be used for saving only those files that have changed or have been created over a defined period.
- **volcopy**: Physical filesystem copying to disk or tape. For those with a spare drive, *volcopy* to disk provides convenient file restore and quick recovery from disk disasters. Tape *volcopy* provides good long-term backup because the filesystem can be read quickly, mounted, and browsed over. Disk and tape *volcopy* are generally used together for short- and long-term backup. Note that a *volcopy* from a mounted filesystem may result in an inconsistent copy (files being written at the time can contain invalid data).

The following table summarizes attributes of these programs. The filesystem size is 65,500Kb in all cases; times are in minutes; judgments are subjective.

TABLE 2-3. Summary of Backup Programs

VARIABLE	FIND/CPIO	VOLCOPY (DISK)	VOLCOPY (TAPE)
Full dump time	40	2	15
Incremental dump time	7	-	-
Full restore time	80	2	15
Incremental restore time	10	-	-
Ease of restoring:			
one file	fair	good	fair
a directory	fair	good	good
scattered files	poor	good	good
full restore	fair	very good	good
Needs tape drive	yes	no	yes
Needs spare filesystem:			
(two CPUs can share)	-	yes	-
Maintains pack/tape labels	no	yes	-
Handles multi-reel tape	yes	-	yes
512Kb per record	1.10	88	10
Interactive			
(ties up console)	yes	yes	yes
May require separate			
I/D space	no	no*	no

* Kb per record are cut to 22 without separate I/D space.

A recommended procedure is for the system administrator to modify the **/etc/filesave** and **/etc/checklist** files to meet operational needs, and update the operator's manual accordingly.

2.16.4 Monitoring Disk Usage.

The system administrator should run *df*(1M) periodically to monitor the free space on all filesystems and check that the recommended start-of-day counts are maintained.

Some administrators may want to limit the amount of disk space that is used by individuals or groups. The *du*(1) program can be executed (after hours) regularly (e.g., daily), and the output kept in an accessible file for later comparison. In this way, users who are rapidly increasing their disk usage can be spotted. The accounting system's *acctdusg* program locates these users also (refer to *acct*(1M)) and the "Accounting" chapter of this guide).

2.16.5 Controlling Disk Usage.

The **find**(1) command can be used to locate inactive or large files. For example:

find / –mtime +90 –atime +90 –print >*filename*

records in *filename* the names of files neither written nor accessed in the last 90 days.

The administrator should balance usage between filesystems. To do this, user directories may be moved. The user's login directory name (available in the shell variable *HOME*) should be chosen to minimize pathname dependencies. User groups with more extensive filesystem structures should set up a shell variable to refer to the filesystem name (e.g., *FS*).

The *find*(1) and *cpio*(1) utilities can be used to move user directories and to manipulate the filesystem tree. For example, the following sequence moves the directory trees **user***x* and

user*y* from filesystem **filesys***1* to filesystem **filesys***2* where, presumably, more space is available):

```
cd /filesys1
find userx usery –print | cpio –pdm /filesys2
#  Make sure new copy is OK.
#  Change userx and usery login directories
#    in the /etc/passwd file.
#  Notify userx and usery via mail(1) that
#    they have been moved and that pathname
#    dependencies in their .profile and shell
#    procedures may need to be changed. Refer to the
#    discussion on $HOME above.
rm -rf /filesys1/userx /filesys1/usery
```

When moving more than one user in this way, keep users with common interests in the same filesystem (these users may have linked files). Move groups of users who may have linked files with a single **cpio** command; otherwise, linked files are unlinked and duplicated.

2.16.6 Filesystem Reorganization.

The filesystem reorganization utility is called *dcopy*(1M). On an otherwise idle system, a reorganized filesystem has almost twice the I/O throughput of a randomly organized filesystem. This applies to file copying, **finds**, **fscks**, etc. *Dcopy* can take up to 2.5 hours to initially reorganize (copy) a large filesystem. During reorganization, the system can be up, but the filesystem being copied must be unmounted.

If time allows, root reorganization once a week (requires system reboot) and user filesystem reorganization once a month improve system performance. Because filesystem reorganization helps throughput at the expense of downtime, reorganizations should be done when the terminals are asleep.

2.16.7 Controlling Directory Size.

Directories larger than 5Kb (320 entries) are inefficient because of filesystem indirection. The following identifies those directories:

```
find / –type d –size +10 –print
```

When large directories are emptied by a user, the amount of space allocated to that directory does not change. To compact a formerly large directory and make the emptied space available to another user, use the following commands:

```
mv /abc/directory   /abc/old_directory
mkdir /abc/directory
chmod 777 /abc/directory
cd /abc/old_directory
find . –print | cpio –plm ../directory
cd ..
rm –rf old_directory
```

where *abc* is a user-defined directory.

Following is a list of files and directories that grow. Most of the files and directories listed are restarted automatically by entries in **/etc/rc** at system reboot.

- Accounting files:

  /etc/wtmp--login information; grows extremely fast with terminal line difficulties; use *acctcon*(1M) to determine the offending line(s).

  /usr/adm/pacct--per process accounting records; gets big quickly; monitored automatically by *ckpacct* from *cron*(1M).

  /usr/adm/cronlog--status log of commands executed by *cron*(1M); also watch this file for error messages from the programs being executed in **/usr/lib/crontab.**

  /usr/adm/errfile--hardware error logging info; also read login **adm**'s mail periodically.

  /usr/adm/ctlog--a log of the people who use **ct**(1C) command.

  /usr/adm/sulog--a log of those who execute the superuser command.

  /usr/adm/Spacct--process accounting files left over from an accounting failure; remove these files unless the accounting files that failed are to be rerun.

- Other files:

  /usr/spool--spooling directory for line printers, **uucp**(1C), etc., and whose subdirectories should be compacted as described above.

2.17 Cron Facility

The *cron*(1M) facility is an administrative daemon process that runs jobs at scheduled times. *Cron* enables users to take advantage of the scheduling capability through the accompanying *crontab*(1) utility, and also provides for delayed execution with the *at*(1) utility.

This section describes how to establish and maintain the *cron* facility.

2.17.1 Defining Queues.

Cron can limit dynamically the number of concurrently running jobs. It can also maintain up to 26 separate queues, and control the number of jobs executed in each one. The file **/usr/lib/cron/queuedefs** is used to maintain definitions for all queues. Each line of the file follows the format:

*q.NN***j***NN***n***NN***w**

where

q is a letter from **a** through **z** indicating the job queue.

*NN***j** is the upper limit for the number of jobs that can be running at any one time for the job queue. *NN* is an integer; the default value is 100.

*NN***n** is the *nice*(1) value assigned to each command executed for the job queue. *NN* is an integer; the default value is 2.

*NN***w** is the time (in seconds) that the system waits before retrying a command that has failed (because all the criteria for running the command were not met). *NN* is an integer; the default value is 60.

Empty fields are initialized to the default values. An example **queuedefs** file is:

```
a.4j1n
b.2j2n90w
c.0n10j
n.120w4n1j
```

Changes to queue definitions take effect before the next job is executed by the *cron* daemon. If this file does not exist, the default values are used.

The ability to define queues enables administrators to restrict executions for commands. The *at*(1) utility can queue jobs in any one of 26 different queues with the *cron* daemon controlling the number of executions for each queue. Running the *at* command with **-q***c* as the first argument queues the command in queue *c*. The default queue is queue **a**. A special queue **b** is defined to be a batch queue; jobs in this queue run whenever the defined maximum level is not exceeded (as specified in the **queuedefs** file). *Cron* executions are limited by the definition of the queue *c*. Jobs in all other queues run at the time specified on the command line.

The *at*(1) utility reads commands from standard input to be executed at a later time. *At* allows a user to specify when the commands should be executed.

The prototype file provides a method to customize *at* command files by controlling the information that is written into the *at* job file. If a file named **.proto.***q* exists (where *q* indicates a queue name), the **.proto.***q* file is copied into the job file. Otherwise, the file **.proto** is used.

The following substitutions are made during creation of an *at* job file.

$m User's current file creation mask (Refer to *umask*(2))

$l User's current file size limit (Refer to *ulimit*(2))

$d Name of the current working directory

$t Time (in seconds) that the job is scheduled to execute

$< Read standard input until EOF is reached

An example prototype file is:

```
cd $d
ulimit $l
umask $m
$<
```

The **at** command exits with an error if no prototype file exists.

2.17.2 Log Information.

Cron logs all command invocations, terminations, and status information in the file **/usr/lib/cron/log**. Records that begin with the character > pertain to command invocations. Two invocation records are written for each command execution. The first displays the command that is being executed; the second contains the login name, process id, job queue, and a timestamp of when the command was invoked. Command termination records begin with the character < and are similar to the second invocation record, except that a non-zero termination status or exit status is also printed. Records that begin with an ! indicate status information.

2.17.3 Limiting Access.

System resources can be misused if *cron* and *at* are invoked irresponsibly. *Cron* provides the system administrator with the means to restrict user access. The files **/usr/lib/cron/cron.allow** and **/usr/lib/cron/at.allow** contain login names of users allowed access to the *crontab* and *at* utilities. The files **/usr/lib/cron/cron.deny** and **/usr/lib/cron/at.deny** contain login names of users denied access to the commands.

When a user submits a *crontab* file, the program checks **cron.allow** for a list of users (one name per line) permitted a *crontab* file. If **cron.allow** does not exist, **cron.deny** is checked for users specifically denied *crontab* files. If neither file exists, only root is allowed to have a *crontab* file. The same arrangement is used for determining access to the *at* utility. The null file **cron.allow** indicates no user is allowed a *crontab* file; a null file **cron.deny** indicates no user is denied a *crontab* file.

2.17.4 Setting Up Cron.

The *cron* utility uses named pipes (FIFOs) to communicate between the user-level commands and the daemon process. *Cron* is located in **/usr/src/cmd** and provides a makefile that can compile and install all components. The command

make

causes all components of the facility to be compiled. The command

make install

will make and install all components of the facility, and create the necessary directory structures. However, the **make install** command will not create the files **cron.allow** or **cron.deny**.

The default modes and owners of all *cron* files and directories are:

```
-rwx------   root   sys   /etc/cron
drwxr-xr-x   root   sys   /usr/lib/cron
-rw-r--r--   root   sys   /usr/lib/cron/at.deny
-rw-r--r--   root   sys   /usr/lib/cron/cron.deny
-rw-r--r--   root   sys   /usr/lib/cron/log
-rw-r--r--   root   sys   /usr/lib/cron/queuedefs
-rw-r--r--   root   sys   /usr/lib/cron/.proto
-rw-r--r--   root   sys   /usr/lib/cron/.proto.n
drwxr-xr-x   root   sys   /usr/spool/cron
drwxr-xr-x   root   sys   /usr/spool/cron/crontabs
-r--r--r--   root   sys   /usr/spool/cron/crontabs/bin
-r--r--r--   root   sys   /usr/spool/cron/crontabs/root
drwxr-xr-x   root   sys   /usr/spool/cron/atjobs
```

At compile time, the variable *MAXRUN* can be set to limit the maximum number of *cron* jobs executing concurrently. (The default value is 25.) The variables *ATLIMIT* and *CRONLIMIT* can be defined to limit the maximum number of *at* jobs and *cron* commands per user-id.

With the *cron* facility, each user maintains an individual *crontab* file. This requires that **/usr/lib/crontab** be split into separate files for each user. The name of the *crontab* file is used as a user-id to obtain user and group permissions. A sample entry in a *crontab* file named **sys** might look like:

```
0    19-7    *    *    *    /usr/lib/sa/sa1 > /dev/null &
```

2.17.5 Cron Errors.

The *cron* daemon attempts to report fatal errors that cause termination by printing an error message on the system console. A user of *at* or *crontab* receives a warning message if the *cron* daemon is not active.

2.18 System Accounting

SYSTEM V/68 provides methods to collect per-process resource utilization data, record connect sessions, monitor disk utilization, and charge fees to specific logins. A set of C language programs and shell procedures is provided to refine the data into summary files and reports.

Accounting programs should be run, even if there is no call for service. Accounting information can help find performance bottlenecks, unused logins, or bad phone lines.

The **who**(1) command lists the people logged in. The **ps**(1) command shows what they are doing. Unfortunately, **ps** operates from heuristics that can consistently fail to report certain processes in a busy system. Therefore, an administrator or operator must be careful about hanging up an apparently inactive line. The **acctcom**(1M) command can read the process accounting file **/usr/adm/pacct** backwards from the most recent entry. It prints entries for selected lines or login names.

Detailed information concerning files, reports and daily operation of the accounting programs is included in the "Accounting" chapter of this guide.

2.19 System Security

The current operating system is not completely secure. System administrators cannot prevent a person from "breaking into" the system, but they can usually detect if someone has done so. The following command mails (to root) a list of all "set user ID" programs owned by **root** (superuser):

find / –user root –perm –4100 –exec ls –l {} \; | mail root

Anything unusual in root 's mail should be investigated. Related advice:

- Change the superuser password regularly. Do not pick obvious passwords (choose 6- to 8-character nonsense strings that combine letter with digits or special characters).
- Dial ports that do not require dialup passwords usually cause trouble.
- The **chroot**(1M) and **su**(1) commands are inherently dangerous, as are group passwords.
- Login directories, **.profile** files, and files in **/bin**, **/usr/bin**, **/lbin**, and **/etc** are security weak spots because they are not owner-protected.
- No time-sharing system with dial ports is secure. Do not keep privileged information on this system.

2.20 Communicating with Users

The directory **/usr/news** and the **news**(1) command are provided as a way to get brief announcements to users. More pressing items (one-liners) can be entered in the **/etc/motd** (message of the day) file; *motd* and (new to the user) *news* are announced at login time.

To reach users who are already logged in, use the **wall**(1M) (write all) command. Do not use **wall** while logged-in as superuser, except in emergencies.

The **/usr/news** directory should be cleaned out once a week by removing everything older than 2 months. It has been found that on most systems a file in **/usr/news** reaches 50 percent of the users within a day and over 80 percent within a week; *motd* should be cleaned out daily.

2.21 Special Files

A special file must be made for every device on the operating system. Normally, all special files are located in the directory **/dev**. Initially, this directory contains:

console console terminal
error Refer to *err*(7)
mem, kmem, null Refer to *mem*(7)
tty Refer to *tty*(7)

These special files are of two types - block and character. This is indicated by the character *b* or *c* in the listing produced by *ls*(1) with the **–l** option.

In addition, each special file has a major device number and a minor device number. The major device number refers to the device type and serves as an index into either the *bdevsw* or *cdevsw* table in the configuration file *conf.c* The minor device number refers to a particular unit of the device type and is used only by the driver for that type. The *config.68*(M) program with the **–t** option lists major device numbers.

The program *mknod*(1M) creates special files. For example, the following would create *part* of the initially-supplied directory (on the EXORmacs):

```
cd /dev
mknod console c 0 0
mknod error c 20 0
mknod mem c 2 0; mknod kmem c 2 1; mknod null c 2 2
mknod tty c 13 0
```

After the special files have been made, their access modes should be changed to appropriate values by **chmod**(1). For example:

```
cd /dev
chmod 622 console
chmod 444 error
chmod 440 mem kmem
chmod 666 null
chmod 666 tty
```

Note that filenames have no meaning to the operating system itself; only the major and minor device numbers are important. However, many programs expect that a particular file is a certain device. Thus, by convention, special files are named as follows:

block device	**conf.c**	**/dev**
Disk Controller	disk	**dsk/***cntrlr_*

character device	**conf.c**	**/dev**
Debug ports	acia	tty*,console
M68KV7	acia	tty*
Printronix	lp	lp*
Centronics	lp	lp*
error	err	error

memory	mm	mem,kmem,null
terminal	sy	tty

For those devices with a **/dev** name ending in *, the star is replaced by a string of digits representing the minor device number. Note that for disks, an octal number scheme is maintained because each drive is logically partitioned into eight sections. Thus, **/dev/dsk/***cntrlr*_**2s4** refers to slice 4 of physical drive 2, where *cntrlr*_ refers to either a VM21, VM22, RWIN1, MVME319 or MVME320 disk controller. There is also a special file, **/dev/swap**, that is used by the program *ps*(1). This file reflects what block device is used for swapping and must be readable. For example:

```
rm /dev/swap
mknod /dev/swap b 0 0
chmod 440 /dev/swap
chown sys /dev/swap
chgrp sys /dev/swap
```

2.22 Administrative Files

2.22.1 /etc/motd.

This file contains the *message of the day*. It is printed by **/etc/profile** after every successful *login*; therefore, it should be kept short and to the point.

2.22.2 /etc/brc.

This file is executed prior to entering any of the numbered *init* states for the first time after a reboot. The file clears **/etc/mnttab**. The file **/etc/brc** is executed once per reboot and is controlled by **/etc/inittab**.

2.22.3 /etc/powerfail.

This shell script is executed according to its line in **/etc/inittab**.

2.22.4 /etc/rc.

During the transition between *init* states, **/etc/init** executes the shell script **/etc/rc** (which must have executable modes). The execution of this file is controlled by a line in **/etc/inittab**. For **/etc/rc** to properly handle the mounting and unmounting of filesystems and the opening of tty lines, etc., it may need certain information that is present in **/etc/utmp**, e.g., the new (current) state, the number of times this state has previously been entered, and the previous state. The following shell script fragment assigns this data to shell variables and checks for entering init state '2 for the first time. As an example:

```
set 'who -r'
cur_mode=$7
no_times=$8
pre_mode=$9
if [ ${cur_mode} = 2 ) -a  ${no_times} = 0 ) ]
then
# commands to be executed when entering multi-user mode
fi
```

These values are carried over from one booting sequence to the next. When the system is rebooted into a single-user initial state, these values, stored in **/etc/utmp**, reflect the state the system was in when it last went down.

The files **/etc/rc**, **/etc/brc**, **/etc/inittab**, **/etc/powerfail**, and **/etc/shutdown** must be edited to suit local conditions; refer to *brc*(1M).

2.22.5 /etc/inittab.

File **/etc/init** uses **/etc/inittab** to determine which processes to create or terminate in each *init* state. By convention, state 's' is single-user and state '2' is multi-user.

The following line indicates the default *init* state, that is, the state the system is to come up in:

```
is:s:initdefault:
```

The following lines arrange for appropriate execution of **/etc/brc**, **/etc/rc**, and **/etc/powerfail**:

```
bc::bootwait:/etc/brc 1> /dev/console 2>&1
rc::wait:/etc/rc 1> /dev/console 2>&1
pf::powerfail:/etc/powerfail 1> /dev/console 2>&1
```

For line **/dev/tty00** for use by 1200 baud asynchronous terminals, add the following:

```
00:2:respawn:/etc/getty -t60 tty00 1200
```

The arguments to **getty** are the number of seconds to allow before hanging up the line, the device name, optional speed settings that refer to an entry in **/etc/gettydefs**, optional type of terminal referenced in *getty*(1M), and optional line discipline.

To add or delete *getty-login* processes while the system is in multi-user mode, make the appropriate changes to **/etc/inittab**; then issue the command **/etc/init q**. This forces **/etc/init** to reread **/etc/inittab** without having to change **init** states.

Again, this file must be edited for local conditions; refer to *gettydefs*(4), and *inittab*(4).

2.22.6 /etc/passwd.

This file is used to describe each *user* to the system. A new line must be added for each new user. Each line has 7 fields separated by colons:

Login name:
: Normally 1 to 6 characters, first character alphabetic, the remainder alphanumeric. No uppercase characters.

Encrypted password:
: Initially null, filled by *passwd*(1). The encrypted password contains 13 bytes, although the actual password is limited to a maximum of 8 bytes. The encrypted password may be followed by a comma and up to 4 more bytes of password "age" information.

User ID:
: A number between 0 and 65,535; 0 indicates the superuser. User IDs 0 through 99 are reserved.

Group ID:
: The default is group 1 (one). Group IDs 0 through 99 are reserved.

Accounting information:
: This field is used by various accounting programs. It usually contains the user name, department number, and account number.

Login directory:
: Full pathname (should be kept reasonably short).

Program name:

If null, **/bin/sh** is invoked after a successful login. If present, the named program is invoked in place of **/bin/sh**.

For example:

dht::138:1:6824-D.Hayden Truscott(4357):/usr/dht:
hsl::244:1:6510-H.Sobel(4466):/usr/hsl:/bin/rsh

System administrators may have reason to establish dialup passwords for particular logins in their systems. To create a dialup password that affects only a specific login, it is imperative that the administrator follow the sequence of procedures as it is written in the next paragraphs.

Two files must be built. The first is **/etc/dialups**, which contains a list of the dialups against which you want to run the dialup password. The entries might be:

/dev/tty00
/dev/tty01
.
.
.
/dev/tty*xx*

The second file is **/etc/d_passwd**. To understand the structure of **/etc/d_passwd**, consider how the dialup password functions. Administrators can create different dialup passwords for different logins, or for groups of logins. This is done by running unique shells against the logins that will have different dialup passwords.

For example, consider two users, Rixi and Benito. Rixi's shell is specified as "shellA" and Benito's shell is specified as "shellB". Execute a **cd** to **/bin** and link **/bin/sh** to **/bin/shellA**. Next, link **/bin/sh** to **/bin/shellB**.

The next step is to build the **/etc/d_passwd** file. The entries might be:

/bin/sh::
/usr/lib/uucp/uucico::
/bin/shellA:*(Rixi's encrypted password)*:
/bin/shellB:*(Benito's encrypted password)*:

An empty set of colons means no password is required. A person running **/bin/sh** or **/usr/lib/uucp/uucico** in the example will not be prompted for a dialup password. (Tip: The easiest way to transfer an encrypted password into this file is to change the password and then manually copy the encryption into **/etc/d_passwd**.)

Next, **cd** to **/etc/passwd** and change Rixi's login to run the **/bin/shellA** and Benito's login to run **/bin/shellB**. For example:

rix::138:1:6824-Rixi Markoff(4457):/usr/rix:/bin/shellA:
ben::244:1:6510-Benito Garazzo(4466):/usr/ben:/bin/shellB:

Be aware that any login that will be **su**-ed to cannot have anything in the shell field in **/etc/passwd**. Therefore, the administrative logins must run the default shell. A dialup password can be created for these logins, but they must all share the same dialup password. Refer to *passwd*(4), *login*(1), and *passwd*(1).

2.22.7 /etc/group.

This file is used to describe each group to the system. The system administrator must add a new line for each new group. Each line has four fields separated by colons:

Group name:
: Normally 1 to 6 characters, with the first character alphabetic and the rest alphanumeric. No uppercase characters.

Encrypted password:
: Contains 13 bytes although the actual password is limited to a maximum of 8 bytes.

Group ID:
: A number between 0 and 65,535. Group IDs 0 through 99 are reserved.

Login names:
: List of all *login* names in the group, separated by commas; list of all *login* names that may use **newgrp**(1) to become a member of the group.

Group passwords are strongly discouraged. Refer to *group*(4).

2.22.8 /etc/profile.

When the shell is executed and is the leader of a process group, as is the case when it is invoked by **login**, the shell reads and executes the commands in **/etc/profile** before executing commands in the user's **.profile** file. The system administrator sets up a standard environment for all users (e.g., executing **umask**, and setting shell variables) and takes care of other housekeeping details (such as **news –n**). In **/etc/profile**, the shell variable $0 indicates the invocation of either normal shell (**-sh**), restricted shell (**-rsh**), or **su** command (**-su**).

2.22.9 /etc/checklist.

This file contains a list of default devices to be checked for consistency by the *fsck*(1M) program. The devices normally correspond to those mounted when the system is in multi-user mode.

The **root** device is specified as a block device while all others are specified as character devices. Character devices can be checked faster than block devices. The **root** device is specified as a block device so that the *fsck* program can detect when the **root** is being checked. Any modifications to the **root** filesystem result in an immediate reboot request, signified by the message ******BOOT UNIX (NO SYNC!) ******. Refer to paragraph 2.3.

2.22.10 /etc/shutdown.

This file contains procedures to shut down the system in preparation for file save or scheduled downtime. No procedures should appear after the transition to single-user mode.

2.22.11 /etc/filesave and /etc/tapesave.

These files contain prototypes for local file saves.

2.22.12 /usr/adm/pacct.

This file contains the process accounting information; refer to *acct*(1M).

2.22.13 /usr/lib/acct/holidays.

This file defines the local company holiday schedule and designates the start of prime and non-prime processing hours for the accounting system.

2.22.14 /etc/wtmp.

This file is the log of *login* processes.

2.23 Regenerating System Software.

System source is issued under the directory **/usr/src**. The subdirectories are named **cmd** (commands), **lib** (libraries), **uts** (the operating system), **head** (header files), and **stand** (stand-alone programs). Refer to *mk*(8) for details on how to remake system software.

2.24 Session Initialization: init, getty, login, who

In the SYSTEM V/68 environment, the initial process spawning is controlled and overseen by *init*(1M). *Init* creates a *getty* process for every active communication line. It does this by creating processes that become the *getty-login-sh* sequence. This sequence of processes allows users to *login* and sets up the initial conditions for the outgoing terminal lines so that speed and other terminal-related states are correct. *Init* and these other processes also keep an accounting file **/etc/wtmp** that is available to processes on the system. These files make it possible to determine the state of each process that *init* has spawned, and, if it is a terminal line, who the current user is. The *who*(1) program provides a means of examining the files.

This section describes the capabilities of each program used, the databases involved, and how to create and maintain these databases. In addition, debugging features of both *init* and *getty* are described.

2.24.1 init.

Four forces drive *init*: a database, *init*'s previous internal level, *init*'s current internal level, and events that cause *init* to wake up.

2.24.1.1 The Database: /etc/inittab. *Init*'s database, kept in the file **/etc/inittab**, consists of separate entries, each with the form:

id:level:type:process

id The *id* is a one- to four-letter identifier that *init* uses internally to label entries in its process table. The *id*, which should be unique, is also placed in the dynamic record file, **/etc/utmp**, and the history file, **/etc/wtmp**.

level The *level* specifies at which levels *init* should be concerned with this entry. *Level* is a string of characters consisting of [0-6a-c]. Any time that *init*'s internal level matches a level specified by *level*, this entry is active. If *init*'s internal level does not match any of the levels specified, then *init* makes certain that the process is not running. If the *level* field is empty, it is equivalent to the string "0123456".

type The *type* specifies some further condition required for or by the execution of an entry.

off The entry is not to run even if the levels match.

once The entry is to be run only if *init* is entering a level. This means if *init* has been awakened by powerfail or because a child died, the entry is not activated. This entry is activated only when a user signal requests a change of *init*'s internal state to a state that is different

from its current state, and this new state is one in which the entry should be active.

wait — *Wait* has all the characteristics of *once*, plus it requires *init* to wait until the spawned process dies before reading more entries from its database. *Wait* allows for initialization actions to be performed and completed before allowing other processes that might be affected to start running.

respawn — *Respawn* requests that this entry continue to run as long as *init* is running in a level that is in this entry's *level* field. Most processes spawned by *init* fall into this category. All *getty* processes are marked as *respawn*. Whenever *init* detects the death of a process that was marked *respawn*, it spawns a new process to take its place.

boot — *Boot* entries have the execution behavior of *once* entries. They are started only when *init* is switching to a numeric run state for the first time. Most commonly, *boot* entries have an empty *level* string, meaning that no matter which level *init* switches to the first time, the *boot* entry is run. Should there be a more specific *level* string, for example "01", then the *boot* entry would only be run if *init* switched to either the 0 or 1 run state as its first numeric level.

bootwait — *Bootwait* entries have the execution behavior of *wait* entries, and they, like *boot* entries, are only run as *init* switches to a numeric level for the first time.

power — *Power* entries act like *once* entries and are activated if *init* receives a SIGPWR signal (19) and is in a state that matches the active states for the entry.

powerwait — *Powerwait* entries act like *wait* entries and are activated if *init* receives a SIGPWR signal and is in a state that matches the active states for the entry.

initdefault — *Initdefault* is a non-standard entry in that it does not specify some process to be spawned. Instead, it only specifies the level *init* is to go to initially when it is coming up at boot time. This allows the system to be rebooted without an operator having to make entries at the system console if so desired. If there is no *initdefault* entry, then *init* asks for the initial run state. In addition to specifying the numbered states, the single-user state "s" may also be specified.

process — The *process* field is the action that *init* asks an *sh* to perform whenever the entry is activated. The string in the **process** field is given a prefix of "**exec** ", so that each entry only generates one process initially. *Init* then forks and execs

sh -c "exec*process*"

This means that the **process** string can take full advantage of all **sh** syntax. The only peculiarities arise from the string **exec** , which was prefixed to the string (because initially there is no standard input, output, or error output). The addition of **exec** to the string means that if the user wants to have a single entry generate more than one process, for example, making a list of the people on the system at the time of a powerfail and mailing it to **root** by the command "who | mail root", it would have to be put in as

pf::powerwait:sh -c "who | mail root"

to work. If it were put in simply as "**who | mail root**", it would be executed as "exec who | mail root", and only the *who* process would be created before the *sh* disappeared. The lack of standard input and output channels must be addressed by explicitly specifying them.

2.24.1.2 Levels. Eleven levels are possible: seven numeric levels, denoted 0, 1, 2, 3, 4, 5, or 6; three temporary levels, denoted a, b, or c; and the single-user level, "s". Normally, *init* runs in a numeric level. Characteristics of a particular level depend on the database and the system administrator. The temporary levels allow certain entries to be started on demand without affecting any processes that were started at a particular level. The temporary levels immediately revert to the previous numeric level, once all entries in the database have been scanned to check if they should be started at the temporary level. When an entry is started by a switch to a temporary level, it becomes independent of future level changes by *init*, unless the change is to the single-user level. The only way to kill a process that was started as a respawnable demand process, without going to the single-user level, is to modify the database. To do this, declare the entry to be *off*.

The single-user level is the one level independent of the database. For this reason, it is not a level in the normal sense. In the single-user level, *init* spawns off an *su* process on the console. The *su* is the only process that it maintains while at the single-user level. The single-user level can be entered at two different places in *init*. If it is entered at boot time, it allows the operator to look over the filesystems without having *init* attempt to do any file I/O, which might cause further problems. *Init* does not attempt to recreate **/etc/utmp** or access **/etc/wtmp** until after it has left the initial single-user level. If the single-user level is entered at any other time, *init* does the bookkeeping in the record files.

The system administrator requests *init* to change levels by running a secondary copy of *init* itself. The file **/etc/init** is linked to **/bin/telinit**. *Init* can only be run by root or a privileged group. If *init* starts running and finds that its process id is not 1, *init* assumes that it is a user initiated copy. Further, *init* sends a signal to the real *init*. The usage is:

telinit *[0123456sSqQabc]*

and the single character argument specifies the signal to be sent to *init*. If the request is to switch to the single-user level, 'S' or 's', then *init* also relinks **/dev/syscon** to the terminal originating the request so that the terminal becomes the virtual system console, insuring that future messages from *init* are directed to the terminal where the operator is located. When *init* does this relinking, it sends a message to **/dev/systty**, saying that the console is being relinked to some other terminal. This leaves a record of the fact at the physical system console.

2.24.1.3 Waking Events. Four events wake *init*: a boot, a powerfail, the death of a child process, or a user signal.

boot — *Init* operates in the boot state until it has entered a numeric state for the first time. It is not possible for *init* to reenter the boot state a second time. Commands labeled **boot** and **bootwait** are executed when changing to a numeric state for the first time, if the levels match.

powerfail — Any time power fails, the operating system sends a SIGPWR signal to all processes. *Init* executes commands with types of *power* and *powerfail*.

child death — Any time a child process of *init* dies, *init* receives a SIGCLD signal (18). The dead child process may be one of two types: a direct descendant of *init*, or a process whose own parent process died before it did. The parent of a process

automatically becomes *init*, if its real parent should die before it does. *Init* determines immediately if the defunct process was one of its own children or an orphan. If it were one of its own, it performs the necessary bookkeeping on its internal process table to note that the process died. If *init* was busy at the time it received the SIGCLD signal, it then returns to complete whatever action it was performing. If *init* was asleep, it then scans its database to determine if any other actions should be taken, such as respawning the process.

user signal — *Init* catches all signals that it is possible for a process to catch. Most signals have specific meaning to *init*, usually requesting it to change its current state in some way. There is one signal, the 'Q' signal, that is used to waken *init* and cause it to scan its database. This is often issued after a change has been made to the database, so that *init* puts the new change into effect immediately. If this were not done, the change would not become effective until *init* awoke for some other reason. The system administrator controls the internal level where *init* is running solely with signals, except during the initialization phase.

2.24.1.4 Normal Operational Behavior. *Init* scans **/etc/inittab** once or twice for each event that wakes it up. If it is in the boot or powerfail state, it scans the table once, looking for entries of these types, and then switches itself back to a normal state and scans again.

Its first action in the normal state is to scan **/etc/inittab** and remove all processes that are currently active and should not be at the current level. *Init* employs one of two methods when killing its child processes, depending on whether it is changing levels or not. If *init* is not changing levels, it forks a child process for each child that needs to be killed, and has that child process send the signals to the process targeted for removal. Killing a process involves sending it two signals. First, a SIGTERM signal (15) is sent so that it can clean up after itself and die gracefully. After waiting the amount of time defined as TWARN (the default value is 20 seconds), a SIGKILL signal (9) is sent, which guarantees that the child dies. The advantage of creating a child with the *fork*(1) system call is the main *init* process need not wait for all the processes it is killing to die before beginning the spawning of new processes. The disadvantage is that if many processes were being killed this way, there would be a very real chance of the operating system process table filling up, causing the system call to fail. This, in turn, could cause *init* to wait. For this reason, when *init* is changing levels, it assumes that it may have many processes to terminate. *Init* sends the signals itself, waits for the required 20 seconds, and sends the final termination signals before continuing. Once the old processes have been removed, *init* makes an entry in its accounting files, if it is changing levels. At this point, it either enters the single-user level or rescans its database looking for processes that need to be spawned at the current level and in the current state. In the normal state of operation, *init* is looking for entries whose types are *off*, *once*, *wait*, or *respawn*.

After the completion of the scan of the database in the normal state, *init* waits for another event. *Init* performs a *sync*(2) system call to ensure that a user who just logged off has had his or her files updated to the disk, and to insure that the bookkeeping is also updated to the disk. Then *init* pauses until it is awakened again.

If *init* finds that it is being requested to switch to the single-user level when it wakens from the pause, it saves all the *ioctl* information about the system console in the file **/etc/ioctl.syscon** before proceeding to remove its other children. Therefore, if the system is being taken down, the new *init* process knows how to set up the system console to talk to it. It is convenient not to have to change the baud rate and terminal specifications when rebooting a system remotely. Because *init* preserves the *ioctl* state of the system console

across system reboots, messages sent during reboots are legible to the operator, no matter where the system console happens to be linked.

All written messages from *init* are sent to **/dev/syscon**. In reality, *init* itself does not send the message, but forks a child to send the message. The reason is that if *init* opens a terminal line, it is assigned a controlling terminal. Since *init* has no controlling terminal, it can spawn *getty* processes which initially have no controlling terminal. When such a *getty* opens its assigned terminal, the terminal becomes the controlling terminal for it and its children. In the one instance, *init* needs input from the system administrator during the initialization phase. In this case, the child process, which is asking for the run level, opens **/dev/systty**, which is always the physical system console, before opening **/dev/syscon**, the virtual system console. This causes **/dev/systty** to be the child's controlling terminal.

2.24.1.5 Setting Tunable Variables. *Init* has several tunable timing constants that can be adjusted when it is compiled.

SLEEPTIME — *Init* awakens occasionally, even if the system is inactive, by setting an alarm timer before going to sleep. The length of time is defined by SLEEPTIME, and is initially five minutes. Since *init* does a *sync* system call each time it wakes, there is a *sync* at least once every SLEEPTIME seconds.

TWARN — TWARN is the number of seconds between the SIGTERM signal and the SIGKILL signal, when *init* is removing processes. It should be set long enough so that all processes can die gracefully on receipt of the SIGTERM signal. It is initially 20 seconds.

NPROC — This is the size of the internal process table *init* uses to keep track of its child processes. It currently defaults to 100, although it can be passed in during compilation with the **-D** option. It is best to set it to the size of the system's process table.

WARNFREQUENCY — To prevent *init* from flooding the system console with error messages when its own internal process table is full, *init* only generates an error message once each WARNFREQUENCY time that it is unable to find a slot. Proper sizing of the internal process table should prevent this condition from ever occurring.

Init cannot directly tell if there is something wrong when it tries to fork and exec a command. It assumes that there is something wrong if it has to respawn a particular entry too often. There are three related parameters controlling this feature, SPAWN_LIMIT, SPAWN_INTERVAL, and INHIBIT.

SPAWN_LIMIT — SPAWN_LIMIT is the number of times a process may respawn in a certain interval of time before further respawns are inhibited.

SPAWN_INTERVAL — SPAWN_INTERVAL is the interval of time in seconds that SPAWN_LIMIT number of respawns must occur to inhibit an entry. If an entry should respawn too often, a message is generated indicating the line in **/etc/inittab** that is at fault.

INHIBIT — INHIBIT is the number of seconds that a process that has respawned too often is inhibited.

SPAWN_LIMIT and SPAWN_INTERVAL should be set such that *init* can respawn a process fast enough to cause an inhibit, but not so low that a legal death of a process is interpreted as "inhibited". The current limits are 10 respawns in two minutes. The real

problem is that when something such as *getty* disappears, *init* becomes active, trying to respawn many processes and never gets to respawn a single process often enough to set off the alarm. The INHIBIT limit is five minutes. Once an entry is inhibited, it is possible to restart it sooner than INHIBIT seconds later by sending *init* the 'Q' signal. The normal problem is a typographic error in **/etc/inittab,** and the normal procedure is to correct the mistake and then do a "telinit Q" to cause *init* to attempt the spawning entry again.

2.24.1.6 Debugging Features. *Init* has some debugging features built in. There are three conditional debug flags that allow various kinds of debugging to be enabled.

UDEBUG This flag causes *init* to be compiled in a form that can be run as a normal user process instead of as process 1. This allows a person to use *sdb* in a normal fashion and not to disturb the rest of the system while debugging or while modifications are made and tested. There are differences in this user version of *init*. It assumes that **utmp, wtmp, inittab, ioctl.syscon,** and **debug** are all in the local directory instead of **/etc**. It also writes to **/dev/sysconx** and **/dev/systtyx**, instead of **/dev/syscon** and **/dev/systty**. It does not process all signals in the same fashion that the real *init* does. Signals SIGINT, SIGQUIT, SIGIOT, and SIGTERM, which correspond to the signals to change to levels 2, 3, 4, and "ignore", are left in their default modes, so that it is possible to terminate the user "init" from a terminal. Signals SIGUSR1 and SIGUSR2, which are normally ignored by the real *init*, are set to cause an *abort* for capturing cores of the debug *init*. The UDEBUG flag automatically sets the DEBUG flag, meaning that the first level of debug is generated by the *init* and written into the file **debug** in the current directory.

DEBUG This flag causes a version of *init* to be produced that can be run as the real *init*, but that generates diagnostic messages about process removal, level changes, and accounting, and writes them into the file **/etc/debug.**

DEBUG1 DEBUG1 causes the diagnostic output generated by DEBUG1 to be increased substantially. Specifically, it produces messages about each process being spawned from *inittab*.

2.24.2 Getty.

Getty(1M) is responsible for making appropriate setting of terminal characteristics and baud rate so that a user can communicate with SYSTEM V/68. The most important of those features is the choice of a baud rate so that input and output make sense. In *getty*, the search is controlled by an ASCII file, **/etc/gettydefs,** and changing or augmenting the search behavior only requires that the file be edited.

2.24.2.1 Usage. *Getty* is normally started from **/etc/inittab** by *init*. *Getty* takes from one to six arguments:

getty [-h] [-t time] line [speed_label] [term_type] [line _disc]

–h This switch tells *getty* that it should not drop the Data Terminal Ready (DTR) signal before resetting the line. This switch currently only works in the CB-UNIX system environment. Normally, *getty* ensures that DTR goes down so that connections to the Develcon data switch are disconnected every time. The EIA protocol requires that a dataset see DTR drop and be reasserted before answering another call. It is possible for *getty* to come back on a line before all the processes spun off by the previous user have died and closed their connections to the line. In this case, DTR would not drop if *getty* did not insure it. This switch is required for programs such as

ct, which initiate a call from the computer to a user (instead of the user calling the computer), putting a *getty* on the resulting connected line. Without the -h switch, the *getty* would immediately disconnect the user again.

-t This switch specifies that the *getty* should die after the specified number of seconds if nothing is typed. This prevents datasets from being tied up unnecessarily.

line *Line* is the name of the terminal line which *getty* is to open and set up. It is minus **/dev/** since, *getty* does a *chdir* to the **/dev** directory and expects to find it in that directory.

speed_label The *speed_label* appears to directly specify a baud rate such as "1200" or "9600", but it can be anything, since it is a label of an entry in **/etc/gettydefs** that *getty* looks for. It specifies the entry *getty* starts with, when trying to find an appropriate speed for the terminal. It defaults to "300".

term_type The *term_type* specifies which terminal discipline is to be used. If this is specified, the virtual terminal protocol becomes immediately effective on the line. Typical types might be "vt100", "hp45", or "tek". Whatever type is specified, it must be a terminal handler that has been compiled into the operating system. This argument is given for lines that are hardwired to the computer.

line_disc The *line discipline* is the last thing that can be specified. The most common is "half" or "half_duplex", when there is a half duplex terminal coming into the computer. This causes the appropriate line discipline to be associated with the line.

2.24.2.2 The Database: /etc/gettydefs. Whenever *getty* is invoked, it references its database to determine certain information about how to set up the line. Each entry in the database has a fixed format.

label# initial flags # final flags # login msg #nextlabel

Getty matches its *speed_label* argument against the *label* field. It stops searching when it finds an entry with a label that matches. The entry specifies how to setup the terminal during the initial phase, during the phase when *getty* prints out the *login msg* and reads in the user's login name, and the final phase, when *getty* exec's the *login*(1) program to continue to the *login* process. The baud rate is specified as an *ioctl* flag in both the initial and final flags' fields.

The flags themselves are strings matching the define variables found in **/usr/include/termio.h**. These flags may be partially or completely overridden if a terminal type is specified. When a terminal type is enabled, it resets various flags to suitable conditions automatically.

During the initial phase, *getty* puts the terminal into a non-echoing *raw* mode. This allows it to take each character as it comes in and to infer certain things about the terminal. For instance, if it sees uppercase alphabetic characters, but no lowercase, it assumes that the terminal is uppercase only, and sets the final configuration so that the uppercase to lowercase conversions are made.

The typical *initial flags* would include the speed, for example, B1200 CS7 PARENB HUPCL. CS7 PARENB sets the line for 7 bits, even parity characters. HUPCL sets the line to hangup on close. Typical final flags would be B1200 SANE IXANY TAB3. SANE is not a flag found in the header file, but a collection of *ioctl* flags used for normal terminal behavior. IXANY permits the use of any character to restart output. TAB3 says to expand

tabs on output.

The *login msg* field is the message that *getty* prints before waiting for the user to enter his or her login name. It may contain anything, and *getty* understands normal special character conventions, so that "\n" means <lf> as does "\012". On systems that are not using the terminal handlers and where lines are hardwired, special entries are sometimes made for different terminal types, for example:

vt100-2400# B2400 # B2400 SANE TAB3
 # 33[H 33[2JAMACCS System B \r\nLOGIN: #vt100-1200

where the *login msg* contains the special vt100 characters required to clear the screen. The entry can take more than one line. Entries are delimited by a blank line. Lines that begin with a pound sign (#) are ignored so that comments may be added to the file.

The *next_label* field tells *getty* which entry to try next if it gets an indication that the speed is wrong. In the above example, it looks for an entry with the name **vt100-1200** if this one was not at the proper speed. Normally, entries do not contain terminal specific information, and the various speed choices are linked together in a closed circle. For example, it is common to have **9600 -> 4800 -> 2400 -> 1200 -> 300 -> 9600.** No matter where the circle is entered, the speed that is correct for the terminal is eventually reached.

The system administrator can check the database for readability by *getty*, with a checking mode:

getty -c gettydefs_like_2file

When *getty* is run in this mode, it scans the entire input file specified and deciphers each entry, printing out the resulting modes that it sets. If it finds a line that it cannot read, it prints an appropriate message; the administrator can then correct the entry. By this mechanism, an administrator can avoid installing a misformatted **gettydefs** file and tying up the system.

If *getty* is unable to find **/etc/gettydefs**, *getty* has a built-in fallback entry. Should *gettydefs* disappear for some reason, a user can log in at 300 baud, since this is the default setting in the built-in entry.

2.24.2.3 Operational Behavior. *Getty* sets up a line as specified by an entry from **/etc/gettydefs** and from any additional arguments, outputs the *login msg* field, and then tries to read the user's login name from the line. During the input of the login name, *getty* checks for speed mismatches that the operating system reports as a NUL character. If such a mismatch occurs, *getty* tries the next speed specified by the current entry, and repeats the whole sequence. While reading in the login name, *getty* makes a guess whether the terminal is uppercase only. If it sees some uppercase characters, but no lowercase characters, it assumes that the terminal is uppercase only and sets the *ioctl* state of the line to translate uppercase letters to lowercase on input, and lowercase to uppercase on output.

Getty and *login* allow environmental variables to be expanded or modified when a user enters his or her login name. Refer to *login*(1) for more information about supplying arguments to *login.*

2.24.3 Login.

Login(1) writes to **/etc/utmp** and **/etc/wtmp**, and allows the user to modify the behavior of his or her *.profile* by setting environmental variables that the *.profile* script reads.

Any additional words provided in response to the basic **login:** query are placed in the environment of the *sh* executed by *login* as its last act.

The arguments may take either the form *xxx* or *xxx=yyy*. Arguments without an equal sign are placed in the environment as

L*n*=xxxx

where *n* is a number starting at 0 is incremented each time a new variable name is required. Variables containing an = are placed into the environment without modification. If they already appear in the environment, then they replace the older value. For example, **TERM=2621** would be placed in the environment unchanged, and the shell variable $TERM would be defined as "2621". There are two exceptions. The variables **PATH** and **SHELL** cannot be changed. This prevents people from logging into restricted shell environments and spawning unrestricted secondary shells.

2.24.4 Who.

Who(1) reads the history files maintained by *init*, *getty*, and *login*. The standard usage for *who* is:

who *[-uTlpdbrtas] [[am i]* or *[utmp_like_file]]*

u Returns a listing of useful information for all users. This information includes login time, activity, pid and comment from inittab file.

T Reports the writability state of the terminal for that entry.

l Reports all entries that are living *getty* processes.

p Reports all entries for living children of *init*, excluding *getty* and descendants of *getty*.

d Reports all the entries for processes that have died.

b Reports the boot time entries that *init* has made. In **/etc/utmp** there is only one such entry.

r Reports the run level entries that *init* has made. In **/etc/utmp** there is only one such entry, the current run level entry. The current state, the number of times in that state, and the previous state are also reported.

t Reports the change of date entries that have been made by the *date*(1) command when the clock was reset. These are required in the history file, **/etc/wtmp**, if accounting is to be done.

a Reports all the entries.

s Reports information for all users in short form; this is the default.

If no file is specified, then **/etc/utmp** is assumed. The **who am i** sequence returns the entry for the user typing the command. If you have **su**-ed to another user-id, the **who am i** sequence returns the login id you **su**-ed to.

There are various output formats for the different kinds of entry. In particular, entries for users and *getty* processes list the amount of time since output to the terminal occurred. This is often of interest, since it shows other users whether or not someone is working at a terminal. The comment field at the end of the entry from **/etc/inittab** is also included, which can conveniently be set up to be the location of the terminal. Dead entries report the exit status for the process that died. This can be of use because it shows whether the process terminated abnormally or not.

2.24.4.1 **Utmp Format.** The format is:

```
/*              <sys/types.h> must be included.*/

#define         UTMP_FILE               "/etc/utmp"
#define         WTMP_FILE               "/etc/wtmp"
#define         ut_name ut_user

struct utmp
 {
                char ut_user[8] ;       /* User login name */
                char ut_id[4] ;         /* /etc/lines id(usually line #) */
                char ut_line[12] ;      /* device name (console, lnxx) */
                short ut_pid ;          /* process id */
                short ut_type ;         /* type of entry */
                struct exit_status
                 {
                  short e_termination ; /* Process termination status */
                  short e_exit ;        /* Process exit status */
                 }
                ut_exit ;               /* The exit status of a process
                                         * marked as DEAD_PROCESS.
                                         */
                time_t ut_time ;        /* time entry was made */
 } ;

/*              Definitions for ut_type         */

#define         EMPTY                   0
#define         RUN_LVL                 1
#define         BOOT_TIME               2
#define         OLD_TIME                3
#define         NEW_TIME                4
#define         INIT_PROCESS            5/* Process spawned by "init" */
#define         LOGIN_PROCESS           6/* A "getty" process waiting for login */
#define         USER_PROCESS            7/* A user process */
#define         DEAD_PROCESS            8
#define         ACCOUNTING              9

#define         UTMAXTYPE               ACCOUNTING/* Largest legal value of ut_type */

/*              Special strings or formats used in the "ut_line" field when*/
/*              accounting for something other than a process.*/
/*              No string for the ut_line field can be more than 11 chars +*/
/*              a NULL in length.       */

#define         RUNLVL_MSG              "run-level %c"
#define         BOOT_MSG                "system boot"
#define         OTIME_MSG               "old time"
#define         NTIME_MSG               "new time"
```

The *ut_type* field completely identifies the type of entry, the *ut_id* field only contains the "id" as found in the "id" field of **/etc/inittab.** The *ut_exit* contains the exit status of processes that *init* has spawned and that have subsequently died.

For SYSTEM V/68, release version FE81, the information on using VME Series 20 is provided in the *VME Series 20 Software Release Guide* shipped with your software. The chapter, Using VME Series 20, will be included in this manual in a future update.

4. ACCOUNTING

SYSTEM V/68 Accounting provides methods to collect per-process resource utilization data, record connect sessions, monitor disk utilization, and charge fees to specific logins. A set of C language programs and shell procedures is provided to reduce this accounting data into summary files and reports. This section describes the structure, implementation, and management of this accounting system, as well as a discussion of the reports generated and the meaning of the columnar data.

4.1 General

The following list is a synopsis of the actions of the accounting system:

- At process termination, the system kernel writes one record per process in **/usr/adm/pacct** in the form of *acct.h*.
- The *login* and *init* programs record connect sessions by writing records into **/etc/wtmp**. Date changes, reboots, and shutdowns are also recorded in this file.
- The disk utilization programs *acctdusg* and *diskusg* break down disk usage by login.
- Fees for file restores, etc., can be charged to specific logins with the *chargefee* shell procedure.
- Each day the *runacct* shell procedure is executed via *cron* to reduce accounting data and produce summary files and reports.
- The *monacct* procedure can be executed on a monthly or fiscal period basis. It saves and restarts summary files, generates a report, and cleans up the *sum* directory.

4.2 Files and Directories

The **/usr/lib/acct** directory contains all of the C language programs and shell procedures necessary to run the accounting system. The *adm* login (currently user ID of four) is used by the accounting system and has the directory structure shown in Fig. 4.1.

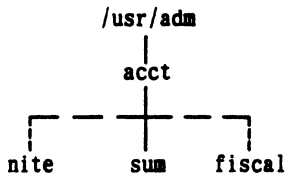

Figure 4-1. Directory Structure of the adm Login

The **/usr/adm** directory contains the active data collection files. (For a complete explanation of the files used by the accounting system, refer to Appendix A.) The **nite** directory contains files that are reused daily by the *runacct* procedure. The **sum** directory contains the cumulative summary files updated by *runacct*. The **fiscal** directory contains periodic summary files created by *monacct*.

4.3 Daily Operation

When the system is switched into multi-user mode, **/usr/lib/acct/startup** is executed, and the following occurs:

A. The *acctwtmp* program adds a "boot" record to **/etc/wtpm** This record is signified by using the system name as the login name in the *wtmp* record.

B. Process accounting is started via *turnacct*. **Turnacct on** executes the *accton* program with the argument **/usr/adm/pacct.**

C. The *remove* shell procedure is executed to clean up the saved *pacct* and *wtmp* files left in the *sum* directory by *runacct*.

The *ckpacct* procedure is run via *cron*(1M) every hour to check the size of **/usr/adm/pacct.** If the file grows past 1000 blocks (default), *turnacct switch* is executed. While *ckpacct* is not absolutely necessary, the advantage of having several smaller *pacct* files becomes apparent when trying to restart *runacct* after a failure processing these records.

The *chargefee* program can be used to bill users for file restores, etc. It adds records to **/usr/adm/fee** which are picked up and processed by the next execution of *runacct* and merged into the total accounting records.

Runacct is executed via *cron* each night. It processes the active accounting files, **/usr/adm/pacct, /etc/wtmp /usr/adm/acct/nite/disktacct,** and **/usr/adm/fee.** It produces command summaries and usage summaries by login.

When the system is shut down using *shutdown*, the *shutacct* shell procedure is executed. It writes a shutdown reason record into **/etc/wtmp** and turns process accounting off.

After the first reboot each morning, the computer operator should execute **/usr/lib/acct/prdaily** to print the previous day's accounting report.

4.4 Setting Up the Accounting System

In order to automate the operation of this accounting system, several things need to be done:

A. If not already present, add this line to the **/etc/rc** file in the state 2 section:

/bin/su — adm —c /usr/lib/acct/startup

B. If not already present, add this line to **/etc/shutdown** to turn off the accounting before the system is brought down:

/usr/lib/acct/shutacct

C. For most installations, the following three entries should be made in **/usr/spool/cron/crontab/adm** so that *cron* automatically runs the daily accounting.

```
0 4 * * 1-6 /usr/lib/acct/runacct 2> /usr/adm/acct/nite/fd2log
0 2 * * 4 /usr/lib/acct/dodisk
5 * * * * /usr/lib/acct/ckpacct
```

Note that *dodisk* is invoked with superuser privileges of *root* so that directory searching is not road blocked.

D. To facilitate monthly merging of accounting data, the following entry in **/usr/spool/cron/crontab/admf**lcrontab allows *monacct* to clean up all daily reports and daily total accounting files and deposit one monthly total report and one monthly total accounting file in the *fiscal* directory.

15 5 1 * * /usr/lib/acct/monacct

The above entry takes advantage of the default action of *monacct* that uses the current month's date as the suffix for the file names. Notice that the entry is executed at such a time as to allow *runacct* sufficient time to complete. This creates, on the first day of each month, monthly accounting files with the entire month's data.

E. The *PATH* shell variable should be set in **/usr/adm/.profile** to:

PATH=/usr/lib/acct:/bin:/usr/bin

4.5 Runacct

Runacct is the main daily accounting shell procedure. It is normally initiated via *cron* during nonprime time hours. *Runacct* processes connect, fee, disk, and process accounting files. It also prepares daily and cumulative summary files for use by *prdaily* or for billing purposes. The following files produced by *runacct* are of particular interest:

nite/lineuse	Produced by *acctcon*, which reads the *wtmp* file, and produces usage statistics for each terminal line on the system. This report is especially useful for detecting bad lines. If the ratio between the number of logoffs to logins exceeds about 3:1, there is a good possibility that the line is failing.
nite/dayacct	This file is the total accounting file for the previous day in *tacct.h* format.
sum/tacct	This file is the accumulation of each day's **nite/daytacct** which can be used for billing purposes. It is restarted each month or fiscal period by the *monacct* procedure.
sum/daycms	Produced by the *acctcms* program, it contains the daily command summary. The ASCII version of this file is **nite/daycms.**
sum/cms	The accumulation of each day's command summaries. It is restarted by the execution of *monacct*. The ASCII version is **nite/cms.**
sum/loginlog	Produced by the *lastlogin* shell procedure, it maintains a record of the last time each login was used.
sum/rprt.MMDD	Each execution of *runacct* saves a copy of the output of *prdaily*.

Runacct takes care not to damage files in the event of errors. A series of protection mechanisms are used that attempt to recognize an error, provide intelligent diagnostics, and terminate processing in such a way that *runacct* can be restarted with minimal intervention. It records its progress by writing descriptive messages into the file *active*. (Files used by *runacct* are assumed to be in the *nite* directory unless otherwise noted.) All diagnostics output during the execution of *runacct* is written into *fd2log*. To prevent multiple invocations, in the event of two *crons* or other problems, *runacct* complains if the files *lock* and *lock1* exist when invoked. The *lastdate* file contains the month and day *runacct* was last invoked and is used to prevent more than one execution per day. If *runacct* detects an error, a message is written to **/dev/console,** mail is sent to *root* and *adm*, the locks are removed, diagnostic files are saved, and execution is terminated.

In order to allow *runacct* to be restartable, processing is broken down into separate reentrant states. This is accomplished by using a *case* statement inside an endless *while* loop. Each state is one case of the *case* statement. A file is used to remember the last state completed. When each state completes, *statefile* is updated to reflect the next state. In the next loop through the *while*, *statefile* is read and the *case* falls through to the next state. When *runacct* reaches the *CLEANUP* state, it removes the locks and terminates. States are executed as follows:

SETUP	The command **turnacct switch** is executed. The process accounting files, **/usr/adm/pacct?**, are moved to **/usr/adm/Spacct?.MMDD**. The **/etc/wtmp** file is moved to **/usr/adm/acct/nite/wtmp.MMDD** with the current time added on the end.
WTMPFIX	The *wtmp* file in the **nite** directory is checked for correctness by the *wtmpfix* program. Some date changes cause *acctcon1* to fail, so *wtmpfix* attempts to adjust the time stamps in the *wtmp* file if a date change record appears.
CONNECT1	Connect session records are written to *ctmp* in the form of *ctmp.h*. The *lineuse* file is created, and the *reboots* file is created showing all of the boot records found in the *wtmp* file.
CONNECT2	*Ctmp* is converted to **ctacct.MMDD** which are connect accounting records. (Accounting records are in *tacct.h* format.)
PROCESS	The *acctprc1* and *acctprc2* programs are used to convert the process accounting files, **/usr/adm/Spacct?.MMDD**, into total accounting records in **ptacct?.MMDD**. The *Spacct* and *ptacct* files are correlated by number so that if *runacct* fails, the unnecessary reprocessing of *Spacct* files does not occur. One precaution should be noted: when restarting *runacct* in this state, remove the last *ptacct* file because it is not complete.
MERGE	Merge the process accounting records with the connect accounting records to form *daytacct*.
FEES	Merge in any ASCII *tacct* records from the file *fee* into **daytacct**.
DISK	On the day after the *dodisk* procedure runs, merge *disktacct* with *daytacct*.
MERGETACCT	Merge *daytacct* with **sum/tacct**, the cumulative total accounting file. Each day, *daytacct* is saved in **sum/tacctMMDD**, so that **sum/tacct** can be recreated in the event it becomes corrupted or lost.
CMS	Merge in today's command summary with the cumulative command summary file **sum/cms**. Produce ASCII and internal format command summary files.
USEREXIT	Any installation dependent (local) accounting programs can be included here.
CLEANUP	Clean up temporary files, run *prdaily* and save its output in **sum/rprtMMDD**, remove the locks, then exit.

4.6 Recovering from Failure

The *runacct* procedure can fail for a variety of reasons: a system crash, **/usr** running out of space, or a corrupted *wtmp* file. If the **activeMMDD** file exists, check it first for error messages. If the *active* file and lock files exist, check *fd2log* for any mysterious messages. The following are error messages produced by *runacct*, and the recommended recovery actions:

ERROR:locksfound,runaborted

The files *lock* and *lock1* were found. These files must be removed before *runacct* can restart.

ERROR: acctg already run for *date* : check /usr/adm/acct/nite/lastdate

The date in *lastdate* and today's date are the same. Remove *lastdate*.

ERROR: turnacct switch returned rc=?

Check the integrity of *turnacct* and *accton*. The *accton* program must be owned by *root* and have the *setuid* bit set.

ERROR: Spacct?.*MMDD* already exists

File setups probably already run.
Check status of files, then run setups manually.

ERROR: /usr/adm/acct/nite/wtmp.*MMDD* already exists, run setup manually

Self-explanatory.

ERROR: wtmpfix errors refer to /usr/adm/acct/nite/wtmperror

Wtmpfix detected a corrupted *wtmp* file. Use *fwtmp* to correct the corrupted file.

ERROR: connect acctg failed: check /usr/adm/acct/nite/log

The *acctcon1* program encountered a bad *wtmp* file. Use *fwtmp* to correct the bad file.

ERROR: Invalid state, check /usr/adm/acct/nite/active

The file *statefile* is probably corrupted. Check *statefile* and read *active* before restarting.

4.7 Restarting Runacct

Runacct called without arguments assumes that this is the first invocation of the day. The argument *MMDD* is necessary if *runacct* is being restarted and specifies the month and day for which *runacct* reruns the accounting. The entry point for processing is based on the contents of *statefile*. To override *statefile*, include the desired state on the command line. For example:

To start *runacct*:

nohup runacct 2> /usr/adm/acct/nite/fd2log&

To restart *runacct*:

nohup runacct 0601 2> /usr/adm/acct/nite/fd2log&

To restart *runacct* at a specific state:

nohup runacct 0601 WTMPFIX 2> /usr/adm/acct/nite/fd2log&

4.8 Fixing Corrupted Files

Occasionally, a file becomes corrupted or lost. Some of the files can simply be ignored or restored from the file save backup. However, certain files must be fixed in order to maintain the integrity of the accounting system.

4.8.1 Fixing wtmp Errors. The *wtmp* files seem to cause the most problems in the day to day operation of the accounting system. When the date is changed and the system is in multi-user mode, a set of date change records is written into **/etc/wtmp** The *wtmpfix* program is designed to adjust the time stamps in the *wtmp* records when a date change is encountered. Some combinations of date changes and reboots, however, slip through *wtmpfix* and cause *acctcon1* to fail. The following steps show how to patch up a *wtmp* file.

cd /usr/adm/acct/nite
fwtmp < wtmp.*MMDD* > xwtmp
ed xwtmp
delete corrupted records or
delete all records from beginning up to the date change
fwtmp —ic < xwtmp > wtmp.*MMDD*

If the *wtmp* file is beyond repair, create a null *wtmp* file. This prevents any charging of connect time. *Acctprc1* can not determine which login owned a particular process, but it is charged to the login that is first in the password file for that user id.

4.8.2 Fixing tacct Errors. If the installation is using the accounting system to charge users for system resources, the integrity of **sum/tacct** is quite important. Occasionally, mysterious *tacct* records appear with negative numbers, duplicate user IDs, or a user ID of 65,535. First check **sum/tacctprev** with *prtacct*. If it looks all right, the latest *sum/tacct.MMDD* should be patched up, then **sum/tacct** recreated. A simple patchup procedure would be:

cd /usr/adm/acct/sum
acctmerg —v < tacct.*MMDD* **> xtacct**
ed xtacct
remove the bad records
write duplicate uid records to another file
acctmerg —i < xtacct > tacct.*MMDD*
acctmerg tacctprev < tacct.*MMDD* > tacct

Remember that the *monacct* procedure removes all the *tacct.MMDD* files; therefore, *sum/tacct* can be recreated by merging these files together.

4.9 Updating For Holidays

The file **/usr/lib/acct/holidays** contains the prime/non-prime table for the accounting system. The table should be edited to reflect your location's holiday schedule for the year. The format is composed of three types of entries:

- Comment Lines:

  Comment lines may appear anywhere in the file as long as the first character in the line is an asterisk.

- Year Designation Line:

  This line should be the first data line (non-comment line) in the file and must appear only once. The line consists of three fields of four digits each (leading white space is ignored). For example, to specify the year as 1984, prime time at 9:00 a.m., and non-prime time at 4:30 p.m., the year designation line is:

  1982 0900 1630

  A special condition in the time field automatically converts 2400 to 0000.

- Company Holiday Lines:

  These entries follow the year designation line and have the following general format:

  day_of_year Month Day Holiday Description

  The day-of-year field is a number from 1 through 366 that indicates the day of the holiday (leading white space is ignored). The three fields that follow are commentary and are not currently used by other programs.

4.10 Reports

Runacct generates five basic reports upon each invocation. They cover the areas of connect accounting, usage by person on a daily basis, command usage reported by daily and monthly totals, and a report of the last time that users were logged in.

The following paragraphs describe the reports and the meanings of their tabulated data.

4.10.1 Daily Report. In the first part of the report, the "from/to" banner should alert the administrator to the period reported on. The times are the time the last accounting report was generated until the time the current accounting report was generated. It is followed by a log of system reboots, shutdowns, power fail recoveries, and any other record dumped into **/etc/wtmp** by the *acctwtmp* program (Refer to *acct*(1M) in the *SYSTEM V/68 Administrator's Manual*).

The second part of the report is a breakdown of line utilization. The TOTAL DURATION tells how long the system was in multi-user state (able to be accessed through the terminal lines). The columns are:

LINE	The terminal line or access port.
MINUTES	The total number of minutes that line was in use during the accounting period.
PERCENT	The total number of MINUTES the line was in use divided into the TOTAL DURATION.
# SESS	The number of times this port was accessed for a *login*(1) session.
# ON	This column does not have much meaning anymore. It used to give the number of times that the port was used to log a user on; but since *login*(1) can no longer be executed explicitly to log a new user in, this column should be identical with SESS.
# OFF	This column reflects not just the number of times a user logged off but also any interrupts that occurred on that line. Generally, interrupts occur on a port when the *getty*(1M) is first invoked when the system is brought to multi-user state. These interrupts occur at a rate of about two per event; therefore, it is not uncommon to see in excess of twice the amount of OFF than ON or SESS. This column is important when the # OFF exceeds the # ON by a large factor. It usually indicates that the multiplexer, modem or cable is going bad, or there is a bad connection somewhere. The most common cause of this is an unconnected cable dangling from the multiplexer.

During real time, **/etc/wtmp** should be monitored because this is the file that generates connect accounting. If it grows rapidly, execute *acctcon1* to see which tty line is the noisiest. If the interrupting is occurring at a furious rate, general system performance is affected.

4.10.2 Daily Usage Report. This report gives a by-user breakdown of system resource utilization. Its data consists of:

UID	User ID.
LOGIN NAME	Login name of the user. There can be more than one login name for a single user ID, this identifies which one.
CPU (MINS)	Amount of time the user's process used the central processing unit. This category is divided into PRIME and NPRIME (nonprime) utilization. The accounting system's definition of this breakdown is located in the **usr/lib/acct/holidays** file. As delivered, prime time is defined to be 0900-1700 hours.
KCORE-MINS	Represents a cummulative measure of the amount of memory a process uses while running. The amount shown reflects kilobyte segments of memory used per minute. This measurement is also divided into PRIME and NPRIME amounts.
CONNECT (MINS)	Identifies "Real Time" used. What the column identifies is the amount of time that a user was logged into the system. If this time is rather high and the later column called # OF PROCS is low, the user is what is called a "line hog". That is, a person logs in first thing in the morning and rarely uses the terminal the rest of the day. This column is also subdivided into PRIME and NPRIME utilization.
DISK BLOCKS	When the disk accounting programs have been run, their output is merged into the total accounting record (*tacct.h*) and appears in this column. This disk accounting is accomplished by the program *acctdusg*.
# OF PROCS	Reflects the number of processes that was invoked by the user. This is a good column to watch for large numbers indicating that a user may have a shell procedure that is faulty.
# OF SESS	How many times the user logged onto the system.
# DISK SAMPLES	Indicates how many times the disk accounting was run to obtain the average number of DISK BLOCKS listed earlier.
FEE	An often unused field in the total accounting record, the FEE represents the total accumulation of items charged against the user by the *chargefee* shell procedure (Refer to *acctsh*(1M)). The *chargefee* procedure is used to levy charges against a user for special services performed such as file restores, tape manipulation by operators, etc.

4.10.3 Daily Command and Monthly Total Command Summaries. These two reports are virtually the same except that the Daily Command Summary only reports on the current accounting period, while the Monthly Total Command Summary tells the story for the start of the fiscal period to the current date. In other words, the monthly report reflects the data accumulated since the last invocation of *monacct*.

The data included in these reports gives an administrator information on which commands are most heavily used, and, based on that, a hint as to what to weigh more heavily when system tuning.

These reports are sorted by TOTAL KCOREMIN which is an arbitrary yardstick, but often a good one, for calculating "drain" on a system.

COMMAND NAME	Name of the command. All shell procedures are lumped together under the name *sh* since only object modules are reported by the process accounting system. The administrator should monitor the frequency of programs called *a.out* or *core* or any other name that does not seem quite right. *Acctcom* is also a good tool to use for determining who executed a suspiciously named command and also if superuser privileges were used.
NUMBER CMDS	Total number of invocations of this particular command.
TOTAL KCOREMIN	Total cummulative measurement of the amount of kilobyte segments of memory used by a process per minute of run time.
TOTAL CPU-MIN	Total processing time this program has accumulated.
TOTAL REAL-MIN	Total real-time (wall-clock) minutes the program has accumulated. This total is the actual "waited for" time as opposed to starting a process in the background.
MEAN SIZE-K	Mean of the TOTAL KCOREMIN over the number of invocations reflected by NUMBER CMDS.
MEAN CPU-MIN	Mean derived between the NUMBER CMDS and TOTAL CPU-MIN.
HOG FACTOR	Relative measurement of the ratio of system availability to system utilization. It is computed by the formula: (total CPU time) / (elapsed time) This gives a relative measure of the total available CPU time consumed by the process during its execution.
CHARS TRNSFD	Total count of the number of characters moved around by the *read*(2) and *write*(2) system calls; it may be a negative.
BLOCKS READ	Total count of the physical block reads and writes that a process performed.

4.10.4 Last Login. Gives the date when a particular login was last used. This could be a good source for getting rid of unused logins and login directories.

4.11 Appendix A. Accounting System Files

Files in the /usr/adm directory:

diskdiag	diagnostic output during the execution of disk accounting programs
dtmp	output from the **acctdusg** program
fee	output from the **chargefee** program, ASCII tacct records
pacct	active process accounting file
pacct?	process accounting files switched via **turnacct**
Spacct?.MMDD	process accounting files for **MMDD** during execution of **runacct**

Files in the /usr/adm/acct/nite directory:

active	used by **runacct** to record progress and print warning and error messages; *active* **MMDD** same as *active* after **runacct** detects an error
cms	ASCII total command summary used by **prdaily**
ctacct.MMDD	connect accounting records in **tacct.h** format
ctmp	output of **acctconl** program, connect session records in **ctmp.h** format
daycms	ASCII daily command summary used by **prdaily**
dayacct	total accounting records for one day in **tacct.h** format
disktacct	disk accounting records in **tacct.h** format, created by **dodisk** procedure
fd2log	diagnostic output during execution of runacct (see **cron** entry)
lastdate	last day **runacct** executed in **date +%m%d** format
lock lock1	used to control serial use of **runacct**
lineuse	tty line usage report used by **prdaily**
log	diagnostic output from **acctconl**
logMMDD	same as *log* after **runacct** detects an error
reboots	contains beginning and ending dates from *wtmp*, and a listing of reboots
statefile	used to record current state during execution of **runacct**
tmpwtmp	wtmp file corrected by **wtmpfix**
wtmperror	place for **wtmpfix** error messages
wtmperrorMMDD	same as *wtmperror* after **runacct** detects an error
wtmp.MMDD	previous day's wtmp file

Files in the /usr/adm/acct/sum directory:

cms	total command summary file for current fiscal in internal summary format
cmsprev	command summary file without latest update
daycms	command summary file for yesterday in internal summary format
loginlog	created by **lastlogin**
pacct.MMDD	concatenated version of all pacct files for **MMDD**, removed after reboot by **remove** procedure
rprt.MMDD	saved output of **prdaily** program
tacct	cumulative total accounting file for current fiscal
tacctprev	same as *tacct* without latest update
tacct.MMDD	total accounting file for **MMDD**
wtmp.MMDD	saved copy of wtmp file for **MMDD**, removed after reboot by **remove** procedure

Files in the /usr/adm/acct/fiscal directory:

cms?	total command summary file for fiscal ? in internal summary format
fiscrpt?	report similar to **prdaily** for fiscal ?
tacct?	total accounting file for fiscal ?

NOTES

5. FILE SYSTEM CHECKING

The File System Check Program (*fsck*) is an interactive file system check and repair program. *Fsck* uses the redundant structural information in the 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. *Fsck* is frequently able to repair corrupted file systems using procedures based upon the order in which the system honors these file system update requests.

The purpose of this section 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.

The final segment of this section contains the *fsck* error conditions. The meanings of the various error conditions, possible responses, and related error conditions are explained.

5.1 General

When the operating system is brought up, a consistency check of the file systems should always be performed. This precautionary measure helps to ensure a reliable environment for file storage on disk. If an inconsistency is discovered, corrective action must be taken.

The section describes the file system, updating of the file system, and file system corruption. In addition, the set of heuristically sound corrective actions used by *fsck* is presented.

5.2 File System

5.2.1 Introduction. The file system features an internal block size of 1024 bytes (compared to the 512 block size in System 3); the size of the internal system buffers is also 1024 bytes. Data transfers to/from disk are, therefore, in 1024-byte operations.

5.2.2 Description. A 512-byte block file system is still supported by the operating system and file system related commands. Both file system sizes are allowed to coexist by detecting the file system type as set in the superblock. At file system mounting time, the operating system checks the magic number and type fields in the superblock. This magic number is unique in the sense that it is unlikely to be matched by an old 512-byte file system. A magic number mismatch assumes an original 512-byte block. New 1024-byte block file systems should have the special magic number set in the superblock and type field specifying a 1024-byte block. These fields are set at file system creation time (**/etc/mkfs**). Also, new file systems with 512-byte blocks may be created. These have the special magic number set and type field indicating a 512-byte block. These fields in the superblock are set at creation time (**/etc/omkfs**). *Labelit* reports the file system type.

No functional changes should be perceived by the user. File system related commands have changed internally to handle both types of file systems. These changes are transparent to the user; the user interface remains unchanged. Most commands still report in 512-byte block units.

The root file system is distributed as a 1024-byte block file system. Users are encouraged to convert their old file systems to the larger size block. However, 512-byte block file systems are still acceptable.

5.2.3 System Administrator Advice. Remember that system buffers are 1024 bytes. When configuring the operating system, take into consideration that the same number of buffers as before (i.e., System 3) uses more main memory. Weigh this against reducing the number of buffers, which reduces the cache hit ratio and degrades performance.

5.3 Update of the File System

Every time a file is modified, the operating system performs a series of file system updates. These updates yield a consistent file system. To understand what happens in the event of a permanent interruption in this sequence, it is important to understand the order in which the update requests are honored. Procedures can be developed to repair a corrupted file system when it is known which pieces of information are written to the file system first.

There are five types of file system updates. These involve the superblock, inodes, indirect blocks, data blocks (directories and files), and free-list blocks.

5.3.1 Superblock. The superblock contains information about the size of the file system, the size of the inode list, part of the free-block list, the count of free blocks, the count of free inodes, and part of the free-inode list.

The superblock of a mounted file system (the root file system is always mounted) is written to the file system whenever the file system is unmounted or a **sync** command is issued.

5.3.2 Inodes. An inode contains information about the type of inode (directory, data, or special), the number of directory entries linked to the inode, the list of blocks claimed by the inode, and the size of the inode.

An inode is written to the file system upon closure of the file associated with the inode. (All "in" core blocks are also written to the file system upon issue of a **sync** system call.)

5.3.3 Indirect Blocks. There are three types of indirect blocks: single-indirect, double-indirect, and triple-indirect. A single-indirect block contains a list of some of the block numbers claimed by an inode. Each of the 128 entries in an indirect block is a data-block number. A double-indirect block contains a list of single-indirect block numbers. A triple-indirect block contains a list of double-indirect block numbers.

Indirect blocks are written to the file system whenever they have been modified and released by the operating system. More precisely, they are queued for eventual writing. Physical I/O is deferred until the buffer is needed by the system or a **sync** command is issued.

5.3.4 Data Blocks. A data block may contain file information or directory entries. Each directory entry consists of a file name and an inode number.

Data blocks are written to the file system whenever they have been modified and released by the operating system.

5.3.5 First Free-List Block. The superblock contains the first free-list block. Free-list blocks are lists of all blocks that are not allocated to the superblock, inodes, indirect blocks, or data blocks. Each free-list block contains a count of the number of entries in that block, a pointer to the next free-list block, and a partial list of free blocks in the file system.

Free-list blocks are written to the file system whenever they have been modified and released by the operating system.

5.4 Corruption of the File System

The most common causes of file system corruption are: improper shutdown and startup procedures, and hardware failures.

5.4.1 Improper System Shutdown and Startup. File systems may become corrupted when proper shutdown procedures are not observed, e.g., forgetting to **sync** the system prior to halting the CPU, physically write-protecting a mounted file system, or taking a mounted file system off-line.

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.

5.4.2 Hardware Failure. Failures can be as subtle as a bad block on a disk pack or as blatant as a nonfunctional disk-controller.

5.5 Detection and Correction of Corruption

A quiescent file system, i.e., unmounted and and not being written on, 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. Because of the multipass nature of the *fsck* program, a quiescent state is important during a file system check.

When an inconsistency is discovered, *fsck* reports the inconsistency; this allows the operator to choose corrective action.

A discussion follows on how to discover inconsistencies (and take possible corrective actions) in the superblock, the inodes, the indirect blocks, the data blocks containing directory entries, and the free-list blocks. These corrective actions can be performed interactively by the *fsck* command under control of the operator.

5.5.1 Superblock. The superblock is prone to corruption because every change to the file system's blocks or inodes modifies the superblock.

The superblock and its associated parts are most often corrupted when the computer is halted and the last command involving output to the file system was not a **sync** command.

The superblock can be checked for inconsistencies involving file-system size, inode-list size, free-block list, free-block count, and the free-inode count.

A. File-System Size and Inode-List Size

The file-system size must be larger than the number of blocks used by the superblock and the number of blocks used by the list of inodes. The number of inodes must be less than 65,535. The file-system size and inode-list size are critical pieces of information to the *fsck* program. While there is no way to actually check these sizes, *fsck* can check for them being within reasonable bounds. All other checks of the file system depend on the correctness of these sizes.

B. Free-Block List

The free-block list starts in the superblock and continues through the free-list blocks of the file system. Each free-list block can be checked for a list count out of range, for block numbers out of range, and for blocks already allocated within the file system. A check is made to see that all the blocks in the file system are found.

The first free-block list is in the superblock. *Fsck* checks the list count for a value of less than 0 or greater than 50. It also checks each block number for a value of less than the first data block in the file system or greater than the last block in the file system. Then it compares each block number to a list of already allocated blocks. If the free-list block pointer is nonzero, the next free-list block is read in and the process is repeated.

When all the blocks have been accounted for, a check is made to see if the number of blocks used by the free-block list plus the number of blocks claimed by the inodes equals the total number of blocks in the file system.

If anything is wrong with the free-block list, then *fsck* may rebuild the list, excluding all blocks in the list of allocated blocks.

C. Free-Block Count

The superblock contains a count of the total number of free blocks within the file system. *Fsck* compares this count to the number of blocks found free within the file system. If the counts do not agree, then *fsck* may replace the count in the superblock by the actual free-block count.

D. Free-Inode Count

The superblock contains a count of the total number of free inodes within the file system. *Fsck* compares this count to the number of inodes found free within the file system. If the counts do not agree, then *fsck* may replace the count in the superblock by the actual free-inode count.

5.5.2 Inodes. An individual inode is not as likely to be corrupted as the superblock. However, because of the great number of active inodes, there is almost as likely a chance for corruption in the inode list as in the superblock.

The list of inodes is checked sequentially starting with inode 1 (there is no inode 0) and going to the last inode in the file system. Each inode can be checked for inconsistencies involving format and type, link count, duplicate blocks, bad blocks, and inode size.

A. Format and Type

Each inode contains a mode word. This mode word describes the type and state of the inode. Inodes may be one of four types:

- regular
- directory
- special block
- special character.

If an inode is not one of these types, then the inode has an illegal type. Inodes may be found in one of three states: unallocated, allocated, and neither unallocated nor allocated. This last state indicates an incorrectly formatted inode. An inode can get in this state if bad data is written into the inode list through, for example, a hardware failure. The only possible corrective action is for *fsck* to clear the inode.

B. Link Count

Contained in each inode is a count of the total number of directory entries linked to the inode. *Fsck* verifies the link count of each inode by traversing down the total directory structure, starting from the root directory, and calculating an actual link count for each inode.

If the stored link count is nonzero and the actual link count is zero, it means that no directory entry appears for the inode. If the stored and actual link counts are nonzero and unequal, a directory entry may have been added or removed without the inode being updated.

If the stored link count is nonzero and the actual link count is zero, *fsck* may link the disconnected file to the **lost+found** directory. If the stored and actual link counts are nonzero and unequal, *fsck* may replace the stored link count by the actual link count.

C. Duplicate Blocks

Contained in each inode is a list or pointers to lists (indirect blocks) of all the blocks claimed by the inode. *Fsck* compares each block number claimed by an inode to a list of already allocated blocks. If a block number is already claimed by another inode, 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* makes a partial second pass of the inode list to find the inode of the duplicated block. This is necessary because without examining the files associated with these inodes for correct content there is not enough information available to decide which inode is corrupted and should be cleared. Most times, the inode with the earliest modify time is incorrect and should be cleared. This condition can occur by using a file system with blocks claimed by both the free-block list and by other parts of the file system.

A large number of duplicate blocks in an inode may be due to an indirect block not being written to the file system. *Fsck* prompts the operator to clear both inodes.

D. Bad Blocks

Contained in each inode is a list or pointer to lists of all the blocks claimed by the inode. *Fsck* checks each block number claimed by an inode for a value lower than that of the first data block or greater than the last block in the file system. If the block number is outside this range, the block number is a bad block number.

A large number of bad blocks in an inode may be due to an indirect block not being written to the file system. *Fsck* prompts the operator to clear both inodes.

E. Size Checks

Each inode contains a 32 bit (4-byte) size field. This size indicates the number of characters in the file associated with the inode. This size can be checked for inconsistencies, e.g., directory sizes that are not a multiple of 16 characters or the number of blocks actually used not matching that indicated by the inode size.

A directory inode within the file system has the directory bit on in the inode mode word. The directory size must be a multiple of 16 because a directory entry contains 16 bytes (2 bytes for the inode number and 14 bytes for the file or directory name).

Fsck warns of such directory misalignment. This is only a warning because not enough information can be gathered to correct the misalignment.

A rough check of the consistency of the size field of an inode can be performed by computing from the size field the number of blocks that should be associated with the inode and comparing it to the actual number of blocks claimed by the inode.

Fsck calculates the number of blocks that there should be in an inode by dividing the number of characters in an inode by the number of characters per block and rounding up. *Fsck* adds one block for each indirect block associated with the inode. If the actual number of blocks does not match the computed number of blocks, *fsck* warns of a possible file-size error. This is only a warning because the system does not fill in blocks in files created in random order.

5.5.3 Indirect Blocks. Indirect blocks are owned by an inode. Therefore, inconsistencies in indirect blocks directly affect the inode that owns it.

Inconsistencies that can be checked are blocks already claimed by another inode and block numbers outside the range of the file system.

For a discussion of detection and correction of the inconsistencies associated with indirect blocks, see the parts "Duplicate Blocks" and "Bad Blocks".

5.5.4 Data Blocks. The two types of data blocks are plain data blocks and directory data blocks. Plain data blocks contain the information stored in a file. Directory data blocks contain directory entries. *Fsck* does not attempt to check the validity of the contents of a plain data block.

Each directory data block can be checked for inconsistencies involving directory inode numbers pointing to unallocated inodes, directory inode numbers greater than the number of inodes in the file system, incorrect directory inode numbers for "." and "..", and directories which are disconnected from the file system. In addition, the validity of the contents of a directory's data block is checked.

If a directory entry inode number points to an unallocated inode, then *fsck* may remove that directory entry. This condition probably occurred because the data blocks containing the directory entries were modified and written out while the inode was not yet written out.

If a directory entry inode number is pointing beyond the end of the inode list, *fsck* may remove that directory entry. This condition occurs if bad data is written into a directory data block.

The directory inode number entry for "." should be the first entry in the directory data block. Its value should be equal to the inode number for the directory data block.

The directory inode number entry for ".." should be the second entry in the directory data block. Its value should be equal to the inode number for the parent of the directory entry (or the inode number of the directory data block if the directory is the root directory).

If the directory inode numbers are incorrect, *fsck* may replace them by the correct values.

Fsck checks the general connectivity of the file system. If directories are found not to be linked into the file system, *fsck* links the directory back into the file system in the **lost+found** directory. This condition can be caused by inodes being written to the file system with the corresponding directory data blocks not being written to the file system.

5.5.5 Free-List Blocks. Free-list blocks are owned by the superblock. Therefore, inconsistencies in free-list blocks directly affect the superblock.

Inconsistencies that can be checked are a list count outside of range, block numbers outside of range, and blocks already associated with the file system.

For a discussion of detection and correction of the inconsistencies associated with free-list blocks, see the part "Free-Block List".

5.6 Appendix A. Error Conditions

5.6.1 Conventions. *Fsck* is a multipass 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 performing cleanup, checking blocks and sizes, pathnames, connectivity, reference counts, and the free-block list (possibly rebuilding it).

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. 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 is discussed under the part

"Initialization".

5.6.2 Initialization. Before a file system check can be performed, certain tables have to be set up and certain files opened. This part concerns itself with the opening of files and the initialization of tables. 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 are listed below.

- **C option?**

  C is not a legal option to *fsck*; legal options are **-y**, **-n**, **-s**, **-S**, **-t**, **-f**, **-q**, and -D. *Fsck* terminates on this error condition. See the *fsck*(1M) entry in the *SYSTEM V/68 Administrator's Manual* for further details.

- **Bad -t option**

  The **-t** option is not followed by a file name. *Fsck* terminates on this error condition. See the *fsck*(1M) entry in the *SYSTEM V/68 Administrator's Manual* for further details.

- **Invalid -s argument, defaults assumed**

  The **-s** option is not suffixed by 3, 4, or blocks-per-cylinder:blocks-to-skip. *Fsck* assumes a default value of 400 blocks-per-cylinder and 7 blocks-to-skip. See the *fsck*(1M) entry in the *SYSTEM V/68 Administrator's Manual* for further details.

- **Incompatible options: -n and -s**

  It is not possible to salvage the free-block list without modifying the file system. *Fsck* terminates on this error condition. See the *fsck*(1M) entry in the *SYSTEM V/68 Administrator's Manual* for further details.

- **Incompatible options: -n and -q**

  It is not possible to do automatic removal without modifying the file system. *Fsck* terminates on this error condition. See the *fsck*(1M) entry in the *SYSTEM V/68 Administrator's Manual* for further details.

- **Can not fstat standard input**

  Fsck's attempt to **fstat** standard input failed. *Fsck* terminates on this error condition.

- **Can not get memory**

  Fsck's request for memory for its virtual memory tables failed. *Fsck* terminates on this error condition.

- **Can not open checklist file: F**

  The default file system checklist file *F* (usually **/etc/checklist**) can not be opened for reading. *Fsck* terminates on this error condition. Check access modes of **F**.

- **Can not stat root**

  Fsck's request for statistics about the root directory "/" failed. *Fsck* terminates on this error condition.

- **Can not stat F**

  Fsck's request for statistics about the file system **F** failed. It ignores this file system and continues checking the next file system given. Check access modes of **F**.

- **FS is a mounted file system, ignored**

  This is to avoid modifying a mounted file system. It ignores this file system and continues with the next file system given.

- **F is not a block or character device**

  Fsck has been given a regular file name by mistake. It ignores this file system and continues checking the next file system given. Check file type of **F**.

- **Can not open F**

  The file system **F** can not be opened for reading. It ignores this file system and continues checking the next file system given. Check access modes of **F**.

- **Size check: fsize X isize Y**

  More blocks are used for the inode list *Y* than there are blocks in the file system **X**, or there are more than 65,535 inodes in the file system. It ignores this file system and continues checking the next file system given.

- **Can not create F**

  Fsck's request to create a scratch file *F* failed. It ignores this file system and continues checking the next file system given. Check access modes of *F*.

- **CAN NOT SEEK: BLK B (CONTINUE)**

  Fsck's request for moving to a specified block number *B* in the file system failed.

  Possible responses to CONTINUE prompt are:

YES	Attempt to continue to run file system check. Often, however, the problem persists. This error condition does not allow a complete check of the file system. A second run of *fsck* should be made to recheck this file system. If block was part of the virtual memory buffer cache, *fsck* terminates with the message "Fatal I/O error".
NO	Terminate program.

- **CAN NOT READ: BLK B (CONTINUE)**

  Fsck's request for reading a specified block number *B* in the file system failed.

  Possible responses to CONTINUE prompt are:

YES	Attempt to continue to run file system check. Often, however, the problem persists. This error condition does not allow a complete check of the file system. A second run of *fsck* should be made to recheck this file system. If block was part of the virtual memory buffer cache, *fsck* terminates with the message "Fatal I/O error".
NO	Terminate 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.

Possible responses to CONTINUE prompt are:

YES	Attempt to continue to run file system check. Often, however, the problem persists. This error condition does not allow a complete check of the file system. A second run of *fsck* should be made to recheck this file system. If block was part of the virtual memory buffer cache, *fsck* terminates with the message "Fatal I/O error".
NO	Terminate program.

5.6.3 Phase 1: Check Blocks and Sizes. This phase concerns itself with the inode list. This part 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.

- **UNKNOWN FILE TYPE I=I (CLEAR)**

  The mode word of the inode *I* indicates that the inode is not a special character inode, special character inode, regular inode, or directory inode. See the part "Format and Types" for more information.

  Possible responses to CLEAR prompt are:

YES	Deallocate inode *I* by zeroing its contents. This always invokes 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 CONTINUE prompt are:

YES	Continue with program. This error condition does not allow a complete check of the file system. A second run of *fsck* should be made to recheck this file system. If another allocated inode with a zero link count is found, this error condition is repeated.
NO	Terminate 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 always invokes the BAD/DUP error condition in Phase 2 and Phase 4. See the part "Bad Blocks" for more information.

- **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*. See the part "Bad Blocks" for more information.

  Possible responses to CONTINUE prompt are:

YES	Ignore the rest of the blocks in this inode and continue checking with next inode in the file system. This error condition does not allow a complete check of the file system. A second run of *fsck* should be made to recheck this file system.
NO	Terminate 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 always invokes Phase 1b and the BAD/DUP error condition in Phase 2 and Phase 4. See the part "Duplicate Blocks" for more information.

- **EXCESSIVE DUP BLKS I=I (CONTINUE)**

  There is more than a tolerable number (usually 10) of blocks claimed by other inodes. See the part "Duplicate Blocks" for more information.

  Possible responses to CONTINUE prompt are:

YES	Ignore the rest of the blocks in this inode and continue checking with next inode in the file system. This error condition does not allow a complete check of the file system. A second run of *fsck* should be made to recheck this file system.
NO	Terminate 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 CONTINUE prompt are:

YES	Continue with program. This error condition does not allow a complete check of the file system. A second run of *fsck* should be made to recheck this file system. If another duplicate block is found, this error condition is repeated.
NO	Terminate program.

- **POSSIBLE FILE SIZE ERROR I=I**

  The inode *I* size does not match the actual number of blocks used by the inode. This is only a warning. (See the part "Size Checks".) If the **-q** option is used, this message is not printed.

- **DIRECTORY MISALIGNED I=I**

  The size of a directory inode is not a multiple of the size of a directory entry (usually 16). This is only a warning. (See the part "Size Checks".) If the **-q** option is used, this message is not printed.

- **PARTIALLY ALLOCATED INODE I=I (CLEAR)**

  Inode *I* is neither allocated nor unallocated. See the part "Format and Types" for more information.

  Possible responses to CLEAR prompt are:

YES	Deallocate inode **I** by zeroing its contents.
NO	Ignore this error condition.

5.6.4 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 part lists the error condition when the duplicate block is found.

- **B DUP I=I**

  Inode *I* contains block number *B* which is already claimed by another inode. This error condition always invokes the BAD/DUP error condition in Phase 2. Inodes with overlapping blocks may be determined by examining this error condition and the DUP error condition in Phase 1. See the part "Duplicate Blocks" for more information.

5.6.5 Phase 2: Check Pathnames. This phase concerns itself with removing directory entries pointing to error conditioned inodes from Phase 1 and Phase 1b. This part lists error conditions resulting from root inode mode and status, directory inode pointers in range, and directory entries pointing to bad inodes.

- **ROOT INODE UNALLOCATED. TERMINATING**

  The root inode (always inode number 2) has no allocate mode bits. The program terminates. See the part "Format and Types" for more information.

- **ROOT INODE NOT DIRECTORY (FIX)**

  The root inode (usually inode number 2) is not directory inode type.

  Possible responses to 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 is produced.
NO	Terminate 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 CONTINUE prompt are:

YES	Ignore DUPS/BAD error condition in root inode and attempt to continue to run the file system check. If root inode is not correct, then this may result in a large number of other error conditions.
NO	Terminate 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. See the part "Data Blocks" for more information.

  Possible responses to 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 NAME=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. If the file system is not mounted and the **-n** option was not specified, the entry is removed automatically if the inode it points to is character size 0.

Possible responses to 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 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 REMOVE prompt are:

YES	The directory entry *F* is removed.
NO	Ignore this error condition.

- **BAD BLK B IN DIR I=I OWNER=O MODE=M SIZE=S MTIME=T**

This message only occurs when the **-q** option is used. A bad block was found in DIR inode *I*. Error conditions looked for in directory blocks are non-zero padded entries, inconsistent "." and ".." entries , and imbedded slashes in the name field. This error message indicates that the user should at a later time either remove the directory inode if the entire block looks bad or change (or remove) those directory entries that look bad.

5.6.6 Phase 3: Check Connectivity. This phase concerns itself with the directory connectivity seen in Phase 2. This part 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. *Fsck* forces the reconnection of a nonempty directory.

Possible responses to RECONNECT prompt are:

YES	Reconnect directory inode *I* to the file system in directory for lost files (usually *lost+found*). This may invoke *lost+found* error condition in Phase 3 if there are problems connecting directory inode *I* to *lost+found*. This may also invoke CONNECTED error condition in Phase 3 if link was successful.

NO — Ignore this error condition. This always invokes 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 always invokes the UNREF error condition in Phase 4. Check access modes of *lost+found*. See *fsck*(1M) in the *SYSTEM V/68 Administrator's Manual* for further details.

- **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 always invokes the UNREF error condition in Phase 4. Clean out unnecessary entries in *lost+found* or make *lost+found* larger. See *fsck*(1M) in the *SYSTEM V/68 Administrator's Manual* for further details.

- **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. See the parts "Link Count" and "Data Blocks" for more information.

5.6.7 Phase 4: Check Reference Counts. This phase concerns itself with the link count information seen in Phase 2 and Phase 3. This part lists error conditions resulting from unreferenced files, missing or full *lost+found* directory, incorrect link counts for files, directories, or special files, unreferenced files and directories, bad and duplicate blocks in files and directories, and incorrect total free-inode counts.

- **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. (See the part "Link Count".) If the **-n** option is not set and the file system is not mounted, empty files are not reconnected and are cleared automatically.

  Possible responses to RECONNECT prompt are:

  YES — Reconnect inode *I* to file system in the directory for lost files (usually *lost+found*). This may invoke *lost+found* error condition in Phase 4 if there are problems connecting inode *I* to *lost+found*.

  NO — Ignore this error condition. This always invokes CLEAR 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 file in *lost+found*. This always invokes CLEAR error condition in Phase 4. Check access modes of *lost+found*.

- **SORRY. NO SPACE IN lost+found DIRECTORY**

  There is no space to add another entry to the *lost+found* directory in the root directory of the file system; *fsck* ignores the request to link a file in *lost+found*. This always invokes the CLEAR error condition in Phase 4. Check size and contents of *lost+found*.

- **(CLEAR)**

  The inode mentioned in the immediately previous error condition can not be reconnected. See the part "Link Count" for more information.

  Possible responses to CLEAR prompt are:

YES	Deallocate inode mentioned in the immediately previous error condition by zeroing its contents.
NO	Ignore this error condition.

- **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. See the part "Link Count" for more information.

  Possible responses to ADJUST prompt are:

YES	Replace 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.

  Possible responses to ADJUST prompt are:

YES	Replace 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 file name *F*, owner *O*, mode *M*, size *S*, and modify time *T* are printed.

  Possible responses to ADJUST prompt are:

YES	Replace 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. (See the parts "Link Counts" and "Data Blocks".) If the **-n** option is not set and the file system is not mounted, empty files are cleared automatically.

  Possible responses to CLEAR prompt are:

YES	Deallocate 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.

If the **-n** option is not set and the file system is not mounted, empty directories are cleared automatically. Nonempty directories are not cleared.

Possible responses to CLEAR prompt are:

YES	Deallocate 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. See the parts "Duplicate Blocks" and "Bad Blocks" for more information.

Possible responses to CLEAR prompt are:

YES	Deallocate 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.

Possible responses to CLEAR prompt are:

YES	Deallocate 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 superblock of the file system. (See the part "Free-Inode Count".) If the **-q** option is specified, the count is fixed automatically in the superblock.

Possible responses to FIX prompt are:

YES	Replace count in superblock by actual count.
NO	Ignore this error condition.

5.6.8 Phase 5: Check Free List. This phase concerns itself with the free-block list. This part lists error conditions resulting from bad blocks in the free-block list, bad free-blocks count, duplicate blocks in the free-block list, unused blocks from the file system not in the free-block list, and the total free-block count incorrect.

- **EXCESSIVE BAD BLKS IN FREE LIST (CONTINUE)**

The free-block list contains more than a tolerable number of blocks (usually 10) with a value less than the first data block in the file system or greater than the last block in the file system. See the parts "Free-Block List" and "Bad Blocks" for more information.

Possible responses to CONTINUE prompt are:

YES	Ignore rest of the free-block list and continue execution of *fsck*. This error condition always invokes "BAD BLKS IN FREE LIST" error condition in Phase 5.
NO	Terminate program.

- **EXCESSIVE DUP BLKS IN FREE LIST (CONTINUE)**

  The free-block list contains more than a tolerable number of blocks (usually ten) claimed by inodes or earlier parts of the free-block list.

  Possible responses to CONTINUE prompt are:

YES	Ignore the rest of the free-block list and continue execution of *fsck*. This error condition always invokes "DUP BLKS IN FREE LIST" error condition in Phase 5.
NO	Terminate program.

- **BAD FREEBLK COUNT**

  The count of free blocks in a free-list block is greater than 50 or less than 0. This error condition always invokes the "BAD FREE LIST" condition in Phase 5.

- **X BAD BLKS IN FREE LIST**

  X blocks in the free-block list have a block number lower than the first data block in the file system or greater than the last block in the file system. This error condition always invokes the "BAD FREE LIST" condition in Phase 5. See the parts "Free-Block List" and "Bad Blocks" for more information.

- **X DUP BLKS IN FREE LIST**

  X blocks claimed by inodes or earlier parts of the free-list block were found in the free-block list. This error condition always invokes the "BAD FREE LIST" condition in Phase 5.

- **X BLK(S) MISSING**

  X blocks unused by the file system were not found in the free-block list. This error condition always invokes the "BAD FREE LIST" condition in Phase 5. See the part "Free-Block List" for more information.

- **FREE BLK COUNT WRONG IN SUPERBLOCK (FIX)**

  The actual count of free blocks does not match the count in the superblock of the file system. See the part "Free-Block Count" for more information.

  Possible responses to FIX prompt are:

YES	Replace count in superblock by actual count.
NO	Ignore this error condition.

- **BAD FREE LIST (SALVAGE)**

  Phase 5 has found bad blocks in the free-block list, duplicate blocks in the free-block list, or blocks missing from the file system. If the **-q** option is specified, the free-block list is salvaged automatically.

  Possible responses to SALVAGE prompt are:

YES	Replace actual free-block list with a new free-block list. The new free-block list is ordered to reduce time spent by the disk waiting for the disk to rotate into position.
NO	Ignore this error condition.

5.6.9 Phase 6: Salvage Free List. This phase concerns itself with the free-block list reconstruction. This part lists error conditions resulting from the blocks-to-skip and blocks-per-cylinder values.

- **Default free-block list spacing assumed**

  This is an advisory message indicating the blocks-to-skip is greater than the blocks-per-cylinder, the blocks-to-skip is less than one, the blocks-per-cylinder is less than one, or the blocks-per-cylinder is greater than 1000. The default values of 7 blocks-to-skip and 400 blocks-per-cylinder are used. See *fsck*(1M) in the *SYSTEM V/68 Administrator's Manual* for further details.

5.6.10 Cleanup. Once a file system has been checked, a few cleanup functions are performed. This part lists advisory messages about the file system and modify status of the file system.

X files Y blocks Z free

This is an advisory message indicating that the file system checked contained *X* files using *Y* blocks leaving *Z* blocks free in the file system.

******* BOOT (NO SYNC!) *******

This is an advisory message indicating that a mounted file system or the root file system has been modified by *fsck*. If the system is not rebooted immediately, the work done by *fsck* may be undone by the in-core copies of tables the system keeps.

******* 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 the system rebooted. If the system is not rebooted immediately, the work done by *fsck* may be undone by the in-core copies of tables.

NOTES

6. LP SPOOLING SYSTEM

6.1 General

The LP program is a system of commands which performs diverse spooling functions under the operating system. Since the primary LP application is off-line printing, this document focuses mainly on spooling to line printers. LP allows administrators to customize the system to spool to a collection of line printers of any type and to group printers into logical classes in order to maximize the throughput of the devices. Users are provided the capabilities of queuing and canceling print requests, preventing and allowing queuing to and printing on devices, starting and stopping LP from processing requests, changing configuration of printers and finding status of the LP system. This section describes the role of an LP Administrator in performing restricted functions and overseeing the smooth operation of LP.

6.2 Overview of LP Features

6.2.1 Definitions. Several terms must be defined before presenting a brief summary of LP commands. The LP was designed with the flexibility to meet the needs of users on different UNIX systems. Changes to the LP configuration are performed by the **lpadmin**(1M) command.

LP makes a distinction between printers and printing devices. A *device* is a physical peripheral device or a file and is represented by a full SYSTEM V/68 pathname. A *printer* is a logical name that represents a device. At different points in time, a printer may be associated with different devices. A *class* is a name given to an ordered list of printers. Every class must contain at least one printer. Each printer may be a member of zero or more classes. A *destination* is a printer or a class. One destination may be designated as the *system default destination.* The **lp**(1) command directs all output to this destination unless the user specifies otherwise. Output that is routed to a printer is printed only by that printer, whereas output directed to a class is printed by the first available class member.

Each invocation of **lp** creates an output request that consists of the files to be printed and options from the lp command line. An interface program which formats requests must be supplied for each printer. The LP scheduler, *lpsched*(1M), services requests for all destinations by routing requests to interface programs to do the printing on devices. An LP configuration for a system consists of devices, destinations, and interface programs.

6.2.2 Commands.

A. Commands for General Use.

The **lp**(1) command is used to request the printing of files. It creates an output request and returns a request id of the form **dest—seqno** to the user, where *seqno* is a unique sequence number across the entire LP system, and *dest* is the destination where the request was routed.

Cancel is used to cancel output requests. The user supplies request ids as returned by *lp* or printer names, in which case the currently printing requests on those printers are canceled.

Disable prevents *lpsched* from routing output requests to printers.

Enable(1) allows *lpsched* to route output requests to printers.

B. Commands for LP Administrators.

Each LP system must designate a person or persons as LP administrator to perform the restricted functions listed below. Either the superuser or any user who is logged into the system as **lp** qualifies as an LP Administrator. All LP files and commands are owned

by **lp**, except for **lpadmin** and **lpsched** which are owned by **root**. The following commands are described in more detail later in this section.

Lpadmin(1M)	Modifies LP configuration. Many features of this command cannot be used when **lpsched** is running.
Lpsched(1M)	Routes output requests to interface programs which do the printing on devices.
Lpshut	Stops *lpsched* from running. All printing activity is halted, but other LP commands may still be used.
Accept(1M)	Allows **lp** to accept output requests for destinations.
Reject	Prevents lp from accepting requests for destinations.
Lpmove	Moves output requests from one destination to another. Whole destinations may be moved at once. This command cannot be used when *lpsched* is running.

6.3 Building LP

All LP commands are built from source code that resides in the **/usr/src/cmd/lp** directory including the make file, **lp.mk**. Unless some of the definitions in **lp.mk** are changed, LP may be installed only by the superuser. Before installing a new LP system, make sure there is a login called **lp** on your system and that the spool directory, **/usr/spool/lp**, does not exist. To install LP, perform the following:

```
cd /usr/src/cmd/lp
make —f lp.mk install
```

This builds all LP commands and creates an initial LP configuration consisting of no printers, classes, or default destination. LP must be configured by an LP administrator using the **lpadmin** command in order to create a useful spooler.

In addition, add the following code to **/etc/rc**:

```
rm —f /usr/spool/lp/SCHEDLOCK
/usr/lib/lpsched
echo "LP scheduler started"
```

This starts the LP scheduler each time that the system is restarted.

Several variables in **lp.mk** may be changed before installing LP to customize the system:

Variable	Default Value	Meaning
SPOOL	**/usr/spool/lp**	spool directory
ADMIN	**lp**	logname of LP Administrator
GROUP	**bin**	group owning LP commands/data
ADMDIR	**/usr/lib**	commands of administrator
USRDIR	**/usr/bin**	user commands reside here

If an existing LP spool directory is corrupted (but not the LP programs) or if it needs to be rebuilt from scratch, make sure that *lpsched* is not running, and perform the following as superuser:

A. Make copies of any interface programs that are not standard LP software. DO NOT make these copies underneath the spool directory. The pathname for printer "p" is **/usr/spool/lp/interface/p**.

B. **rm —fr /usr/spool/lp**

C. **Make —f lp.mk new** (This recreates the bare LP configuration described above.)

6.4 Precautions

A. Some LP commands invoke other LP commands. Moving them after they are built causes some commands to fail.

B. The files under the SPOOL directory should be modified only by LP commands.

C. All LP commands require set-user-id permission. If this is removed, the commands fail.

6.5 Configuring LP — The lpadmin Command

Changes to the LP configuration should be made by using the **lpadmin** command and not by hand. **Lpadmin** does not attempt to alter the LP configuration when **lpsched** is running, except where explicitly noted below.

6.5.1 SYSTEM V/68 Configuration. SYSTEM V/68 contains two line printer facilities: the older (now obsolete) *lpr*(1)/*lpr*(1) duo and the newer *lp*(1) family of utilities. By default, it is assumed that the *lp*(1) family will be used (see **/etc/rc**). The system administrator ("root") should configure the *lp*(1) system (one time only per printer) with the following commands from the console:

```
PATH=:$PATH
cd /usr/lib
lpshut
lpadmin -ppr1 -mpprx -v/dev/lp0
lpadmin -dpr1  # only for the default printer
accept pr1
enable pr1
lpsched
```

The *lpshut*(1M) command (cf. *lpsched*(1M)) shuts down the current *lpsched*(1M). The first *lpadmin*(1M) command defines a printer "pr1" (the user specifies a name) and a model interface program "pprx" (for Parallel Printronix; see *lpadmin*(1M) for others), connected to the device **/dev/lp0**. The second *lpadmin*(1M) command calls "pr1" the default printer. The *accept*(1M) command establishes that "pr1" is ready to accept spooled files. The *enable*(1M) command indactes that "pr1" is ready to print. The final line restarts the scheduler. Descriptions of all the commands in the configuration procedure are provided in the following paragraphs.

6.5.2 Introducing New Destinations.

A. The following information must be supplied to **lpadmin** when introducing a new printer:

- The printer name. This (—**p** printer) is an arbitrary name which must conform to the following rules:

  It must be no longer than 14 characters.

  It must consist solely of alphanumeric characters and underscores.

  It must not be the name of an existing LP destination (printer or class).

- The device associated with the printer (—**v** device). This is the pathname of a hardwired printer, a login terminal, or other file that is writable by lp.

- The printer interface program. This may be specified in one of three ways:

  It may be selected from a list of model interfaces supplied with LP (**—m** model).

  It may be the same interface that an existing printer uses (**—e** printer).

  It may be a program supplied by the LP administrator (**—i** interface).

B. Information which need not always be supplied when creating a new printer includes:

- The user may specify **—h** to indicate that the device for the printer is hardwired or the device is the name of a file (this is assumed by default). If, on the other hand, the device is the pathname of a login terminal, then **—l** must be included on the command line. This indicates to *lpsched* that it must automatically disable this printer each time *lpsched* starts running. This fact is reported by *lpstat* when it indicates printer status:

  $ lpstat —pa

  printer *a* (login terminal) **disabled Oct 31 11:15 —**
  disabled by scheduler: login terminal

  This is done because device names for login terminals can be (and usually are) associated with different physical devices from day to day. If the scheduler did not take this action, somebody might log in and be surprised that LP is spooling to his/her terminal!

- The new printer may be added to an existing class or added to a new class (**—c***class*). New class names must conform to the same rules for new printer names.

C. EXAMPLES:

The following examples are referenced by further examples in later sections.

1. Create a printer called pr1 whose device is **/dev/printer** and whose interface program is the model hp interface:

   $ /usr/lib/lpadmin —ppr1 —v/dev/printer —mhp

2. Add a printer called **pr2** whose device is **/dev/tty22** and whose interface is a variation of the model prx interface. It is also a login terminal:

   $ cp /usr/spool/lp/model/prx xxx
   < edit xxx >
   $ /usr/lib/lpadmin —ppr2 —v/dev/tty22 —ixxx —l

3. Create a printer called **pr3** whose device is **/dev/tty23.** The **pr3** is added to a new class called **cl1** and uses the same interface as printer **pr2:**

   $ /usr/lib/lpadmin —ppr3 —v/dev/tty23 —epr2 —ccl1

6.5.3 Modifying Existing Destinations.

A. Modifications to existing destinations must always be made with respect to a printer name (—pprinter). The modifications may be one or more of the following:

1. The device for the printer may be changed (—vdevice). If this is the only modification, then this may be done even while *lpsched* is running. This facilitates changing devices for login terminals.

2. The printer interface program may be changed (—**m***model*, —**e***printer*, —**i***interface*).
3. The printer may be specified as hardwired (—**h**) or as a login terminal (—**l**).
4. The printer may be added to a new or existing class (—**c***class*).
5. The printer may be removed from an existing class (—**r***class*). Removing the last remaining member of a class causes the class to be deleted. No destination may be removed if it has pending requests. In that case, **lpmove** or **cancel** should be used to move or delete the pending requests.

B. EXAMPLES:

These examples are based on the LP configuration created by those in the previous section.

1. Add printer **pr2** to class **cl1**:

   $ /usr/lib/lpadmin —ppr2 —ccl1

2. Change **pr2**'s interface program to the model prx interface, change its device to **/dev/tty24**, and add it to a new class called **cl2**:

   $ /usr/lib/lpadmin —ppr2 —mprx —v/dev/tty24 —ccl2

   Note that printers **pr2** and **pr3** now use different interface programs even though **pr3** was originally created with the same interface as **pr2**. Printer **pr2** is now a member of two classes.

3. Specify printer **pr2** as a hardwired printer:

   $ /usr/lib/lpadmin —ppr2 —h

4. Add printer **pr1** to class **cl2**:

   $ /usr/lib/lpadmin —ppr1 —ccl2

   The members of class **cl2** are now **pr2** and **pr1**, in that order. Requests routed to class **cl2** are serviced by **pr2** if both **pr2** and **pr1** are ready to print; otherwise, they are printed by the one which is next ready to print.

5. Remove printers **pr2** and **pr3** from class **cl1**:

   $ /usr/lib/lpadmin —ppr2 —rcl1
   $ /usr/lib/lpadmin —ppr3 —rcl1

   Since **pr3** was the last remaining member of class **cl1**, the class is removed.

6. Add **pr3** to a new class called **cl3**.

   $ /usr/lib/lpadmin —ppr3 —ccl3

6.5.4 Specifying the System Default Destination. The system default destination may be changed even when *lpsched* is running.

EXAMPLES:

A. Establish class **cl1** as the system default destination:

   $ /usr/lib/lpadmin —dcl1

B. Establish no default destination:

$ /usr/lib/lpadmin —d

6.5.5 Removing Destinations. Classes and printers may be removed only if there are no pending requests that were routed to them. Pending requests must either be canceled using **cancel** or moved to other destinations using **lpmove** before destinations may be removed. If the removed destination is the system default destination, then the system has no default destination until the default destination is re-specified. When the last remaining member of a class is removed, then the class is also removed. The removal of a class never implies the removal of printers.

EXAMPLES:

A. Make printer **pr1** the system default destination:

$ /usr/lib/lpadmin —dpr1

Remove printer **pr1**:

$ /usr/lib/lpadmin —xpr1

Now there is no system default destination.

B. Remove printer **pr2**:

$ /usr/lib/lpadmin —xpr2

Class **cl2** is also removed, since **pr2f1 was its only member.**

C. **Remove class cl3**:

$ /usr/lib/lpadmin —xcl3

Class **cl3** is removed, but printer **pr3** remains.

6.6 Making an Output Request — The lp Command

Once LP destinations have been created, users may request output by using the **lp** command. The request id that is returned may be used to see if the request has been printed or to cancel the request.

The LP program determines the destination of a request:

- If the user specifies **id dest** on the command line, then the request is routed to **dest**
- If the environment variable **LPDEST** is set, the request is routed to the value of **LPDEST.**
- If there is a system default destination, then the request is routed there.

Otherwise, the request is rejected.

EXAMPLES:

A. There are at least four ways to print the password file on the system default destination:

```
lp /etc/passwd
lp < /etc/passwd
cat /etc/passwd | lp
lp —c /etc/passwd
```

The last three ways cause copies of the file to be printed, whereas the first way prints the file directly. Thus, if the file is modified between the time the request is made and the time it is actually printed, then the changes are reflected in the output.

B. Print two copies of file abc on printer xyz and title the output "my file":

pr abc | lp —dxyz —n2 —t my file

C. Print file xxx on a Diablo 1640 printer called **zoo** in 12-pitch and write to the user's terminal when printing has completed:

lp —dzoo —o*12* —w xxx

In this example, *12* is an option that is meaningful to the model Diablo 1640 interface program that prints output in 12-pitch mode [see *lpadmin*(1M)].

6.7 Finding LP Status — Lpstat

The **lpstat** command is used to find status information about LP requests, destinations, and the scheduler.

EXAMPLES:

A. List the status of all pending output requests made by this user:

lpstat

The status information for a request includes the request id, the logname of the user, the total number of characters to be printed, and the date and time the request was made.

B. List the status of printers **p1** and **p2**:

lpstat —pp1,p2

6.8 Canceling Requests — Cancel

The LP requests may be canceled using the **cancel** command. Two kinds of arguments may be given to the command: request ids and printer names. The requests named by the request ids are canceled and requests that are currently printing on the named printers are canceled. Both types of arguments may be intermixed.

EXAMPLE:

Cancel the request that is now printing on printer xyz:

cancel xyz

If the user who is canceling a request is not the same one who made the request, then mail is sent to the owner of the request. LP allows any user to cancel requests in order to eliminate the need for users to find LP administrators when unusual output should be purged from printers.

6.9 Allowing and Refusing Requests — Accept and Reject

When a new destination is created, *lp* rejects requests that are routed to it. When the LP administrator is sure that it is set up correctly, he or she should allow *lp* to accept requests for that destination. The **accept** command performs this function.

Sometimes it is necessary to prevent *lp* from routing requests to destinations. If printers have been removed or are waiting to be repaired or if too many requests are building for printers, then it may be desirable to cause *lp* to reject requests for those destinations. The **reject** command performs this function. After the condition that led to the rejection of requests has been remedied, the **accept** command should be used to allow requests to be taken again.

The acceptance status of destinations is reported by the **—a** option of **lpstat**.

EXAMPLES:

A. Cause *lp* to reject requests for destination xyz:

/usr/lib/reject —r"printer xyz needs repair" xyz

Any users who try to route requests to xyz encounter the following:

```
$ lp —dxyz file
lp: can not accept requests for destination xyz
-- printer xyz needs repair
```

B. Allow *lp* to accept requests routed to destination xyz:

/usr/lib/accept xyz

6.10 Allowing and Inhibiting Printing — Enable and Disable

The **enable** command allows the LP scheduler to print requests on printers. That is, the scheduler routes requests only to the interface programs of enabled printers. Note that it is possible to enable a printer but to prevent further requests from being routed to it.

The **disable** command undoes the effects of the **enable** command. It prevents the scheduler from routing requests to printers, independently of whether or not *lp* is allowing them to accept requests. Printers may be disabled for several reasons including malfunctioning hardware, paper jams, and end of day shutdowns. If a printer is busy at the time it is disabled, then the request that it was printing is reprinted in its entirety either on another printer (if the request was originally routed to a class of printers) or on the same one when the printer is re-enabled. The **—c** option causes the currently printing requests on busy printers to be canceled in addition to disabling the printers. This is useful if strange output is causing a printer to behave abnormally.

EXAMPLE:

Disable printer xyz because of a paper jam:

```
$ disable —r"paper jam" xyz
printer "xyz" now disabled
```

Find the status of printer xyz:

```
$ lpstat —pxyz
printer "xyz" disabled since Jan 5 10:15 —
paper jam
```

Now, re-enable xyz:

```
$ enable xyz
printer "xyz" now enabled
```

6.11 Moving Requests Between Destinations — Lpmove

Occasionally, it is useful for LP administrators to move output requests between destinations. For instance, when a printer is down for repairs, it may be desirable to move all of its pending requests to a working printer. This is one way to use the **lpmove** command. The other use of this command is to move specific requests to a different destination. **Lpmove** refuses to move requests while the LP scheduler is running.

EXAMPLES:

A. Move all requests for printer abc to printer xyz:

$ /usr/lib/lpmove abc xyz

All of the moved requests are renamed from abc-nnn to xyz-nnn. As a side effect, destination abc is no longer accepting further requests.

B. Move requests zoo-543 and abc-1200 to printer xyz:

$ /usr/lib/lpmove zoo-543 abc-1200 xyz

The two requests are now renamed xyz-543 and xyz-1200.

6.12 Stopping and Starting the Scheduler — Lpshut and Lpsched

Lpsched is the program that routes the output requests that were made with **lp** through the appropriate printer interface programs to be printed on line printers. Each time the scheduler routes a request to an interface program, it records an entry in the log file, **/usr/spool/lp/log**. This entry contains the logname of the user who made the request, the request id, the name of the printer that the request is being printed on, and the date and time that printing first started. In the case that a request has been restarted, more than one entry in the log file may refer to the request. The scheduler also records error messages in the log file. When *lpsched* is started, it renames **/usr/spool/lp/log** to **/usr/spool/lp/oldlog** and starts a new log file.

No printing is performed by the LP system unless *lpsched* is running. Use the command

lpstat —r

to find the status of the LP scheduler.

Lpsched is normally started by the **/etc/rc** program as described above and continues to run until the system is shut down. The scheduler operates in the **/usr/spool/lp** directory. When it starts running, it exits immediately if a file called *SCHEDLOCK* exists. Otherwise, it creates this file in order to prevent more than one scheduler from running at the same time.

Occasionally, it is necessary to shut down the scheduler in order to reconfigure LP or to rebuild the LP software. The command

/usr/lib/lpshut

causes *lpsched* to stop running and terminates all printing activity. All requests that were in the middle of printing are reprinted in their entirety when the scheduler is restarted.

To restart the LP scheduler, use the command

/usr/lib/lpsched

Shortly after this command is entered, *lpstat* should report that the scheduler is running; if not, it is possible that a previous invocation of *lpsched* exited without removing SCHEDLOCK. Try the following:

rm —f /usr/spool/lp/SCHEDLOCK
/usr/lib/lpsched

6.13 Printer Interface Programs

Every LP printer must have an interface program which does the actual printing on the device that is currently associated with the printer. Interface programs may be shell procedures, C programs, or any other executable programs. The LP model interfaces are all written as shell procedures and can be found in the **/usr/spool/lp/model** directory. At the time *lpsched* routes an output request to a printer P, the interface program for P is invoked in the directory **/usr/spool/lp** as follows:

interface/P *id user title copies options file ...*

where

id is the request id returned by *lp*

user is logname of user who made the request

title is optional title specified by the user

copies is number of copies requested by user

options is a blank-separated list of class or printer-dependent options specified by user

file is the full pathname of a file to be printed

EXAMPLES:

The following examples are requests made by user "smith" with a system default destination of printer "xyz". Each example lists an **lp** command line followed by the corresponding command line generated for printer xyz's interface program:

A. **lp /etc/passwd /etc/group**
interface/xyz xyz-52 smith "" 1 "" /etc/passwd /etc/group

B. **pr /etc/passwd | lp —t"users" —n5**
interface/xyz xyz—53 smith users 5 ""
/usr/spool/lp/request/xyz/d0—53

C. **lp /etc/passwd —oa —ob**
interface/xyz xyz—54 smith "" 1 "a b" /etc/passwd

When the interface program is invoked, its standard input comes from **/dev/null** and both the standard output and standard error output are directed to the printer's device. Devices are opened for reading as well as writing when file modes permit. If a device is a regular file, all output is appended to the end of the file.

Given the command line arguments and the output directed to a device, interface programs may format output in any way. Interface programs must ensure that the proper stty modes (terminal characteristics such as baud rate, output options, etc.) are in effect on the output device. This may be done as follows in a shell interface, only if the device is opened for reading:

stty mode ... <&1

That is, take the standard input for the stty command from the device.

When printing has completed, the interface program exits with a code indicative of the success of the print job. Exit codes are interpreted by *lpsched* as follows:

CODE	MEANING TO LPSCHED
zero	The print job has completed successfully.
1 to 127	A problem was encountered in printing this particular request (e.g., too many nonprintable characters). This problem does not affect future print jobs. **Lpsched** notifies users by mail that there was an error in

printing the request.

greater than 127 — These codes are reserved for internal use by *lpsched*. Interface programs must not exit with codes in this range.

When problems that are likely to affect future print jobs occur (e.g., a device filter program is missing), the interface programs should disable printers so that print requests are not lost. When a busy printer is disabled, the interface program is terminated with signal 15.

6.14 Setting Up Hardwired Devices and Login Terminals as LP Printers

6.14.1 Hardwired Devices. As an example of how to set up a hardwired device for use as an LP printer, using tty line 15 as printer xyz, perform the following:

A. Avoid unwanted output from non-LP processes and ensure that LP can write to the device:

```
$ chown lp /dev/tty15
$ chmod 600 /dev/tty15
```

B. Change **/etc/inittab** so that **tty15** is not a login terminal. In other words, ensure that **/etc/getty** is not trying to log users in at this terminal. Change the entries for line 15 to:

1:15:o:
2:15:o: Enter the command:

```
$ init 2
```

If there is currently an invocation of **/etc/getty** running on **tty15**, kill it. Now, and when the system is rebooted, **tty15** is initialized with default stty modes. Thus, it is up to LP interface programs to establish the proper baud rate and other stty modes for correct printing to occur.

C. Introduce printer xyz to LP using the model prx interface program:

```
$ /usr/lib/lpadmin —pxyz —v/dev/tty15 —mprx
```

D. When xyz is created, it is initially disabled and *lp* rejects requests routed to it. If it is desired, allow *lp* to accept requests for xyz:

```
/usr/lib/accept xyz
```

This allows requests to build up for xyz, and to be printed when it is enabled at a later time.

E. When printing is to occur, be sure that the printer is ready to receive output. For several printers, this means that the top of form has been adjusted and that the printer is online. Enable printing to occur on xyz:

```
enable xyz
```

When requests are routed to xyz, they begin printing.

6.14.2 Login Terminals. Login terminals may also be used as LP printers. To do this for a Diablo 1640 terminal called abc, perform the following:

A. Introduce printer abc to LP using the model 1640 interface program:

```
$ /usr/lib/lpadmin —pabc —v/dev/null —m1640 —l
```

Note that **/dev/null** is used as abc's device because the actual device is specified each time that abc is enabled. This device may be different from day to day. When abc is created, it is initially disabled, and *lp* rejects requests routed to it. **Lp** can accept requests for abc:

/usr/lib/accept abc

This allows requests to build up for abc and to be printed when it is enabled at a later time. It is not advisable to enable abc for printing, however, until the following steps have been taken.

B. Log terminal in if this has not already been done.

C. Assuming the **tty**(1) command reports that this terminal is **/dev/tty02**, associate this device with printer abc:

$ /usr/lib/lpadmin —pabc —v/dev/tty02

Note that **lpadmin** may be used only by an LPA. If others are to routinely perform this step, then an LPA may establish a program owned by **lp** or by **root** with set-user-id permission that performs this function.

D. When printing is to occur, be sure that the printer is ready to receive output. For several printers, this means that the top of form has been adjusted. Enable printing to occur on abc:

enable abc

When requests have been routed to abc, they begin printing.

E. When all printing has stopped on abc or when you want it back as a regular login terminal, you may prevent it from printing more output:

$ disable abc
printer abc now disabled

If abc is enabled when SYSTEM V/68 is rebooted or when *lpsched* is restarted, it is disabled automatically.

7. SYSTEM ACTIVITY PACKAGE

7.1 General

This section describes the design and implementation of the SYSTEM V/68 Activity Package. The operating system contains a number of counters that are incremented as various system actions occur. The system activity package reports system-wide measurements including Central Processing Unit(CPU) utilization, disk and tape Input/Output(I/O) activities, terminal device activity, buffer usage, system calls, system switching and swapping, file-access activity, queue activity, and message and semaphore activities. The package provides four commands that generate various types of reports. Procedures that automatically generate daily reports are also included. The five functions of the activity package are:

- **sar**(1): allows a user to generate system activity reports in real time and to save system activities in a file for later usage.
- **sag**(1G): displays system activity in a graphical form.
- **sadp**(1): samples disk activity once every second during a specified time interval and reports disk usage and seek distance in either tabular or histogram form.
- **timex**(1): a modified **time**(1) command that times a command and also reports concurrent system activity.
- system activity daily reports: procedures are provided for sampling and saving system activities in a data file periodically and for generating the daily report from the data file.

The system activity information reported by this package is derived from a set of system counters located in the operation system kernel. These system counters are described in the part "System Activity Counters". The part "System Activity Commands" describes the commands provided by this package. The procedure for generating daily reports is given in "Report Generation". A description for each of the files used by the system activity package can be found in Appendix A.

7.2 System Activity Counters

The operating system manages a number of counters that record various activities and provide the basis for the system activity reporting system. The data structure for most of these counters is defined in the **sysinfo** structure (see Appendix B) in **/usr/include/sys/sysinfo.h**. The system table overflow counters are kept in the **_syserr** structure. The device activity counters are extracted from the device status tables. In this version, the I/O activity of the **ud**(7) recorded.

The following paragraphs describe the system activity counters that are sampled by the system activity package.

A. Cpu time counters. There are four time counters that may be incremented at each clock interrupt 60 times per second. Exactly one of the cpu[] counters is incremented on each interrupt, according to the mode the CPU is in at the interrupt: idle, user, kernel, and wait for I/O completion.

B. Lread and lwrite. The lread and lwrite counters are used to count logical read and write requests issued by the system to block devices.

C. Bread and bwrite. The bread and bwrite counters are used to count the number of times data is transferred between the system buffers and the block devices. These actual I/Os are triggered by logical I/Os that cannot be satisfied by the current contents of the buffers. The ratio of block I/O to logical I/O is a common measure of the effectiveness of the system buffering.

D. Phread and phwrite. The phread and phwrite counters count read and write requests issued by the system to raw devices.

E. Swapin and swapout. The swapin and swapout counters are incremented for each system request initiating a transfer from or to the swap device. More than one request is usually involved in bringing a process into memory, or out, because text and data are handled separately. Frequently used programs are kept on the swap device and are swapped in rather than loaded from the file system. The swapin counter reflects these initial loading operations as well as resumptions of activity, while the swapout counter reveals the level of actual "swapping." The amount of data transferred between the swap device and memory are measured in blocks and counted by bswapin and bswapout

F. Pswitch and syscall. These counters are related to the management of multiprogramming. Syscall is incremented every time a system call is invoked. The numbers of invocations of read(2), write(2), fork(2), and exec(2) system calls are kept in counters sysread , syswrite , sysfork, and sysexec, respectively. Pswitch counts the times the switcher was invoked, which occurs when: (a) a system call resulted in a road block; (b) an interrupt occurred resulting in awakening a higher priority process; or (c) a 1-second clock interrupt occurred.

G. Iget, namei, and dirblk. These counters apply to file-access operations. Iget and namei, in particular, are the names of operating system routines. The counters record the number of times that the respective routines are called. Namei is the routine that performs file system path searches. It searches the various directory files to get the associated i-number of a file corresponding to a special path. Iget is a routine called to locate the inode entry of a file (i-number). It first searches the in-core inode table. If the inode entry is not in the table, routine iget gets the inode from the file system where the file resides and make an entry in the in-core inode table for the file. Iget returns a pointer to this entry. Namei calls iget, but other file access routines also call iget. Therefore, counter iget is always greater than counter namei.

H. dirblk. This counter records the number of directory block reads issued by the system. It is noted that the directory blocks read divided by the number of namei calls estimates the average path length of files.

I. Runque, runocc, swpque, and swpocc. These counters are used to record queue activities. They are implemented in the clock.c routine. At every 1 second interval, the clock routine examines the process table to see whether any processes are in core and in "ready" state. If so, the counter runocc is incremented and the number of such processes are added to counter runque . While examining the process table, the clock routine also checks whether any processes in the swap device are in "ready" state. The counter swpocc is incremented if the swap queue is occupied, and the number of processes in swap queue is added to counter swpque .

J. Readch and writech. The readch and writech counters record the total number of bytes (characters) transferred by the read and write system calls, respectively.

K. Monitoring terminal device activities. There are six counters monitoring terminal device activities. Rcvint, xmtint, and mdmint are counters measuring hardware interrupt occurrences for receiver, transmitter, and modem individually. Rawch, canch, and outch count number of characters in the raw queue, canonical queue, and output queue. Characters generated by devices operating in the "cooked" mode, such as terminals, are counted in both rawch and (as edited) in canch, but characters from raw devices, such as communication processors, are counted only in rawch.

L. Msg and sema counters. These counters record message sending and receiving activities and semaphore operations, respectively.

M. Monitoring I/O activities. As to the I/O activity for a disk or tape device, four counters are kept for each disk or tape drive in the device status table. Counter io_ops is incremented when an I/O operation has occurred on the device. It includes block I/O, swap I/O, and physical I/O. Io_bcnt counts the amount of data transferred between the device and memory in 512 byte units. Io_act and io_resp measure the active time and response time of a device in time ticks summed over all I/O requests that have completed for each device. The device active time includes the device seeking, rotating and data transferring times, while the response time of an I/O operation is from the time the I/O request is queued to the device to the time when the I/O completes.

N. Inodeovf, fileovf, textovf, and procovf. These counters are extracted from _syserr structure. When an overflow occurs in any of the inode, file, text and process tables, the corresponding overflow counter is incremented.

7.3 System Activity Commands

The system activity package provides three commands for generating various system activity reports and one command for profiling disk activities. These tools facilitate observation of system activity during a controlled stand-alone test of a large system, an uncontrolled run of a program to observe the operating environment, and normal production operation.

Commands **sar** and **sag** permit the user to specify a sampling interval and number of intervals for examining system activity and then to display the observed level of activity in tabular or graphical form. The **timex** command reports the amount of system activity that occurred during the precise period of execution of a timed command. The **sadp** command allows the user to establish a sampling period during which access location and seek distance on specified disks are recorded and later displayed as a tabular summary or as a histogram.

7.3.1 The sar command. The **sar** command can be used in the following ways: When the frequency arguments **t** and **n** are specified, it invokes the data collection program *sadc* to sample the system activity counters in the operating system every *t* seconds for *n* intervals and generates system activity reports in real time. Generally, it is desirable to include the option to save the sampled data in a file for later examination. In addition to the system counters, a time stamp is also included. It gives the time at which the sample was taken.

If no frequency arguments are supplied, it generates system activity reports for a specified time interval from an existing data file that was created by *sar* at an earlier time.

A convenient usage is to run *sar* as a background process, saving its samples in a temporary file but sending its standard output to **/dev/null**. Then an experiment is conducted after which the system activity is extracted from the temporary file. The *sar*(1) manual entry describes the usage and lists various types of reports. Appendix C gives the formula for deriving each reported item.

7.3.2 The sag command. **Sag** displays system activity data graphically. It relies on the data file produced by a prior run of *sar* after which any column of data or the combination of columns of data of the *sar* report can be plotted. A fairly simple, but powerful, command syntax allows the specification of cross plots or time plots. Data items are selected using the *sar* column header names. The *sar*(1G) manual entry describes its options and usage. The system activity graphical program invokes *graphics*(1G) and *tplot*(1G) commands to have the graphical output displayed on any of the terminal types supported by *tplot*.

7.3.3 The timex command. The **timex** command is an extension of the **time**(1) command. Without options, **timex** behaves exactly like **time**. In addition to giving the time information, it also prints a system activity report derived from the system counters. The manual entry *timex*(1) explains its usage. It should be emphasized that the *user* and *sys* times reported in the second and third lines are for the measured process itself, including all its children, while the remaining data (including the *cpu user %* and *cpu sys %*) are for the entire system.

While the normal use of *timex* is to measure a single command, multiple commands can also be timed: either by combining them in an executable file and timing it, or more concisely, by typing:

timex sh –*c* "*cmd1; cmd2; ... ;*"

This establishes the necessary parent-child relationships to correctly extract the user and system times consumed by *cmd1*, *cmd2*, ... (and the shell).

7.3.4 The sadp command. Sadp is a user level program that can be invoked independently by any user. It requires no storage or extra code in the operating system and allows the user to specify the disks to be monitored. The program is reawakened every second, reads system tables from **/dev/kmem**, and extracts the required information. Because of the 1 second sampling, only a small fraction of disk requests are observed; however, comparative studies have shown that the statistical determination of disk locality is adequate when sufficient samples are collected.

In the operating system, there is an *iobuf* for each disk drive. It contains two pointers which are head and tail of the I/O active queue for the device. The actual requests in the queue may be found in three buffer header pools: system buffer headers for block I/O requests, physical buffer headers for physical I/O requests, and swap buffer headers for swap I/O. Each buffer header has a forward pointer which points to the next request in the I/O active queue and a backward pointer which points to the previous request.

Sadp reads the *iobuf* of the monitored device and the three buffer header pools once every second during the monitoring period. It then traces the requests in the I/O queue, records the disk access location, and seeks distance in buckets of 8 cylinder increments. At the end of monitoring period, it prints out the sampled data. The output of *sadp* can be used to balance load among disk drives and to rearrange the layout of a particular disk pack. The usage of this command is described in manual entry **sadp**(1).

7.4 Daily Report Generation

The previous part described the commands available to users to initiate activity observations. It is probably desirable for each installation to routinely monitor and record system activity in a standard way for historical analysis. This part describes the steps that a system administrator may follow to automatically produce a standard daily report of system activity.

7.4.1 Facilities.

- **sadc** – The executable module of *sadc.c* (see Appendix A) which reads system counters from **/dev/kmem** and records them to a file. In addition to the file argument, two frequency arguments are usually specified to indicate the sampling interval and number of samples to be taken. If no frequency arguments are given, it writes a dummy record in the file to indicate a system restart.
- **sa1** – The shell procedure that invokes *sadc* to write system counters in the daily data file **/usr/adm/sadd** where **dd** represents the day of the month. It may be invoked with sampling interval and iterations as arguments.

- **sa2** -- The shell procedure that invokes the **sar** command to generate daily report **/usr/adm/sa/sardd** from the daily data file **/usr/adm/sa/sadd.** It also removes daily data files and report files after 7 days. The starting and ending times and all report options of **sar** are applicable to **sa2.**

7.4.2 Suggested Operational Setup. It is suggested that *cron*(1M) control the normal data collection and report generation operations. For example, the sample entries in **/usr/lib/crontab:**

```
0 * * * 0,6 su sys -c "/usr/lib/sa/sa1"
0 18-7 * * 1-5 su sys -c "/usr/lib/sa/sa1"
0 8-17 * * 1-5 su sys -c "/usr/lib/sa/sa1 1200 3"
```

would cause the data collection program *sadc* to be invoked every hour on the hour. Moreover, depending on the arguments presented, it writes data to the data file one to three times at every 20 minutes. Therefore, under the control of *cron*(1M), the data file is written every 20 minutes between 8:00 and 18:00 on weekdays and hourly at other times.

Note that data samples are taken more frequently during prime time on weekdays to make them available for a finer and more detailed graphical display. It is suggested that *sa1* be invoked hourly rather than invoking it once every day; this ensures that if the system crashes, data collection is resumed within an hour after the system is restarted.

Because system activity counters restart from zero when the system is restarted, a special record is written on the data file to reflect this situation. This process is accomplished by invoking **sadc** with no frequency arguments within **/etc/rc** when going to multi-user state:

```
su adm -c "/usr/lib/sa/sadc /usr/adm/sa/sa`date +%d`"
```

Cron(1M) also controls the invocation of **sar** to generate the daily report via shell procedure **sa2.** One may choose the time period the daily report is to cover and the groups of system activity to be reported. For instance, if:

```
0 20 * * 1-5 su sys -c "/usr/lib/sa/sa2 -s 8:00 -e 18:00 -i 3600 -uybd"
```

is an entry in **/usr/lib/crontab,** *cron* executes the **sar** command to generate daily reports from the daily data file at 20:00 on weekdays. The daily report furnishes information on CPU utilization, terminal device activity, buffer usage, and device activity every hour from 8:00 to 18:00.

In case of a shortage of the disk space or for any other reason, these data files and report files can be removed by the superuser.

7.5 Appendix A. Source Files

The source files and shell programs of the system activity package are in directory **/usr/src/cmd/sa.**

sa.h	The system activity header file defines the structure of data file and device information for measured devices. It is included in **sadc.c, sar.c** and **timex.c.**
sadc.c	The data collection program that accesses **/dev/kmem** to read the system activity counters and writes data either on standard output or on a binary data file. It is invoked by the **sar** command generating a real-time report. It is also invoked indirectly by entries in **/usr/lib/crontab** to collect system activity data.

sar.c	The report generation program invokes **sadc** to examine system activity data, generates reports in real time, and saves the data to a file for later usage. It may also generate system activity reports from an existing data file. It is invoked indirectly by **cron** to generate daily reports.
saghdr.h	The header file for **saga.c** and **sagb.c** which contains data structures and variables used by **saga.c** and **sagb.c.**
saga.c & sagb.c	The graph generation program that first invokes **sar** to format the data of a data file in a tabular form and then displays the **sar** data in graphical form.
sa1.sh	The shell procedure that invokes **sadc** to write data file records. It is activated by entries in **/usr/lib/crontab.**
sa2.sh	The shell procedure that invokes **sar** to generate the report. It also removes the daily data files and daily report files after a week. It is activated by an entry in **/usr/lib/crontab** on weekdays.
timex.c	The program that times a command and generates a system activity report.
sadp.c	The program that samples and reports disk activities.

7.6 Appendix B. System Information Data Structure

```
struct sysinfo {
        time_t      cpu[4];
#define   CPU_IDLE      0
#define   CPU_USER      1
#define   CPU_KERNAL    2
#define   CPU_WAIT      3
        time_t      wait[3];
#define   W_IO          0
#define   W_SWAP        1
#define   W_PIO         2
        long        bread;
        long        bwrite;
        long        lread;
        long        lwrite;
        long        phread;
        long        phwrite;
        long        swapin;
        long        swapout;
        long        bswapin;
        long        bswapout;
        long        pswitch;
        long        syscall;
        long        sysread;
        long        syswrite;
        long        sysfork;
        long        sysexec;
        long        runque;
        long        runocc;
        long        swpque;
        long        swpocc;
        long        iget;
        long        namei;
        long        dirblk;
        long        readch;
        long        writech;
        long        rcvint;
        long        xmtint;
        long        mdmint;
        long        rawch;
        long        canch;
        long        outch;
        long        msg;
        long        sema;
};
```

7.7 Appendix C. Formula for Reported Items

The derivation of the reported items is given in this attachment. Each item discussed below is the data difference sampled at two distinct times *t2* and *t1*.

- CPU Utilization

  %-of-cpu-x = cpu-x / (cpu-idle + cpu-user + cpu-kernel + cpu-wait) * 100

  where cpu-x is cpu-idle, cpu-user, cpu-kernel (cpu-sys), or cpu-wait.

- Cached Hit Ratio

  %-of-cached-I/O = (logical-I/O − block-I/O) / logical-I/O * 100

  where cached I/O is cached read or cached write.

- Disk or Tape I/O Activity

  %-of-busy = I/O-active / (t2 − t1) * 100;
  avg-queue-length = I/O-resp / I/O-active;
  avg-wait = (I/O-resp − I/O-active) / I/O-ops;
  avg-service-time = I/O-active / I/O-ops.

- Queue Activity

  avg-x-queue-length = x-queue / x-queue-occupied-time;
  %-of-x-queue-occupied-time = x-queue-occupied-time / (t2 − t1);

  where x-queue is run queue or swap queue.

- The Rest of System Activity

  avg-rate-of-x = x / (t2 − t1)

  where x is swap in/out, blks swapped in/out, terminal device activities, read/write characters, block read/write, logical read/write, process switch, system calls, read/write, fork/exec, iget, namei, directory blocks read, disk/tape I/O activities, message or semaphore activities.

USER'S COMMENTS

SYSTEM V/68 ADMINISTRATOR'S GUIDE

Product Code 72901
Part Number 41964-00

Motorola welcomes your comments and suggestions. Please use this form.

- Does this manual provide the information you need? ☐Yes ☐No
  - What is missing?

- Is the manual accurate? ☐Yes ☐No
  - What is incorrect? (Be specific.)

- Is the manual written clearly? ☐Yes ☐No
  - What is unclear? (Be specific.)

- What other comments can you make about this manual?

- What do you like about this manual?

- Was this manual difficult to obtain? ☐Yes ☐No

Please include your name and address if you would like a reply.

Name_____
Company_____
Address_____
_____

• What is your occupation?

☐ Programmer	☐ Operator	☐ Manager
☐ Systems Analyst	☐ Instructor	☐ Customer Engineer
☐ Engineer	☐ Student	☐ Other_____

• How do you use this manual?

☐ Reference Manual	☐ Introduction to the Subject
☐ In a Class	☐ Introduction to the System
☐ Self Study	☐ Other_____

fold **fold**

MOTOROLA COMPUTER SYSTEMS
3013 S. 52nd Street
Tempe, AZ 85282

Attention: Software Publications, X4

---

fold **fold**

Staple Here

MOTOROLA
Computer Systems

3013 S. 52nd St.
Tempe, AZ 85282