305-502
Issue 1

AT&T 3B2 Computer
UNIX™ System V Release 2.0

System Administration Utilities Guide

NOTICE

The information in this document is subject to change without notice. AT&T assumes no responsibility for any errors that may appear in this document.

TRADEMARKS

The following is a listing of the trademarks that are used in this manual:

- 3BNET — Trademark of AT&T
- CIPHER — Registered Trademark of Cipher Data Products, Inc.
- DOCUMENTER'S WORKBENCH — Trademark of AT&T
- UNIX — Trademark of AT&T
- WREN — Trademark of Control Data Corporation
- WRITER'S WORKBENCH — Trademark of AT&T.

ORDERING INFORMATION

Additional copies of this document can be ordered by calling

1-800-432-6600 Inside the U.S.A.

OR

1-317-352-8557 Outside the U.S.A.

OR by writing to:

AT&T Customer Information Center (CIC)
Attn: Customer Service Representative
P.O. Box 19901
Indianapolis, IN 46219

CONTENTS

Chapter 1

INTRODUCTION

PAGE

Chapter 1

INTRODUCTION

GENERAL

This guide describes the command formats (syntax) and use of the System Administration Utilities provided with the AT&T 3B2 Computer. Procedures for the administration of the AT&T 3B2 Computer are also described. The commands and procedures described in this guide are for use by a sophisticated user who needs more capabilities than those provided by Simple Administration. Simple Administration capabilities are described in the *AT&T 3B2 Computer Owner/Operator Manual.*

GUIDE ORGANIZATION

This guide is structured so you can easily find information without having to read the entire text. The remainder of this guide is organized as follows.

- Chapter 2, "ADMINISTRATIVE DIRECTORIES AND FILES," describes directories and files that are important in administering the system.

- Chapter 3, "ADMINISTRATIVE TASKS," describes system administration tasks (procedures).

- Chapter 4, "FILE SYSTEM CHECKING AND REPAIR," describes use of the **/etc/fsck** command.

- Chapter 5, "BAD BLOCK HANDLING FEATURE," describes how the life of a hard disk is extended by detecting disk access errors and mapping the bad blocks to good block addresses. The **/etc/hdeadd**, **/etc/hdefix**, and **/etc/hdelogger** commands are the basis of this feature.

- Chapter 6, "SYSTEM ADMINISTRATION COMMANDS," describes the format and use of the commands provided by the System Administration Utilities. Certain other system administration commands that are provided on the hard disk as part of the Essential Utilities are also described in this chapter.

- Appendix A, "DEVICE DEFAULT PARTITIONING," shows the hard disk, cartridge tape, and floppy disk default partitioning.

- Appendix B, "RUN LEVELS," defines the various operating states (levels) of the system. The normal operating state is run-level 2 (multi-user).

- Appendix C, "ERROR MESSAGES," provides a recommended owner/administrator action for UNIX* System, diagnostic monitor program, equipped device table completion, firmware, boot, and pump error messages. A brief description of the types of error messages is also provided in this appendix.

---

* Trademark of AT&T

MANUAL PAGE SPECIAL NOTATION

Throughout this guide, references to UNIX System V manual pages are in the form of **command***(section)*. The *section* is the section number of the UNIX System manual page in which the **command** is found. When a section number is not provided, the referenced command is in Section 1. The Section 1 manual pages for the commands described as part of the System Administration Utilities are provided in the *AT&T 3B2 Computer User Reference Manual.* The Section 1M manual pages for the commands described as part of the System Administration Utilities are provided in the *AT&T 3B2 Computer System Administration Reference Manual.* Also provided in the *AT&T 3B2 Computer System Administration Reference Manual* are Section 7 (special files) and Section 8 (system maintenance procedures) manual pages. Refer to the *AT&T 3B2 Computer Programmer Reference Manual* for the manual pages associated with other than Sections 1, 7, and 8. The following is a guide to the various section numbers used on manual pages.

Section 1 — This section describes the general use commands. General-purpose commands are identified by only the number 1. Letters are appended to the section numbers as follows:

- C — Communication-type commands
- G — Graphics-type commands
- M — Maintenance-type commands.

Section 2 — This section describes the system calls and error numbers.

Section 3 — This section describes the subroutines and libraries. Letters are appended to the section numbers as follows:

- C — C language and assembler library routines
- F — Fortran library routines
- M — Mathematical library routines
- S — Standard input/output library routines
- X — Miscellaneous routines.

Section 4 — This section outlines the formats of various system files.

Section 5 — This section describes miscellaneous facilities. Included are descriptions of character sets and macro packages.

Section 6 — This section describes games and educational programs.

Section 7 — This section describes various special files that support specific hardware devices.

Section 8 — This section describes system maintenance procedures.

WHO TO CALL FOR HELP

Throughout this guide, the term *service representative* is used to mean AT&T Service Representative or authorized dealer. When you need help to resolve operating procedures or system failures, do not hesitate to call your *service representative*.

ADMINISTRATIVE TASKS

The tasks associated with system administration are concerned with keeping the system operational and changing system configuration, as necessary. The tasks include:

- Adding, deleting, and modifying user login information
- File system space administration
- File system backup and restoral
- Application hardware installation (circuit cards and cables)
- Application software installation (utilities)
- System reconfiguration
- Network administration.

Chapter 2

ADMINISTRATIVE DIRECTORIES AND FILES

PAGE

Chapter 2

ADMINISTRATIVE DIRECTORIES AND FILES

INTRODUCTION

This chapter describes the directories and files that are of interest to a system administrator. Refer to the UNIX System V manual pages found in Section 4 of the *AT&T 3B2 Computer Programmer Reference Manual* for additional information on the formats of system files.

DIRECTORIES

The directories of the **root** file system (/) are as follows.

bck	Directory used to mount a backup file system for restoring files.
bin	Directory that contains public commands.
boot	Directory that contains configurable object files created by the **/etc/mkboot**(1M) program.
dev	Directory containing special files that define all devices on the system.
dgn	Directory that contains diagnostic programs.
etc	Directory that contains administrative programs and tables.
install	Directory used by Simple Administration to mount utilities packages for installation and removal (**/install** file system).
lib	Directory that contains public libraries.
lost+found	Directory used by **fsck**(1M) to save disconnected files.
mnt	Directory used to temporarily mount file systems during restoral of the operating system from floppy disks.
save	Directory used by Simple Administration for saving data on floppies.
tmp	Directory used for temporary files.
usr	Directory used to mount the **/usr** file system.

FILES

General

The following files/directories are important in the administration of the AT&T 3B2 Computer.

- /etc/checklist
- /etc/fstab
- /etc/gettydefs
- /etc/group
- /etc/inittab
- /etc/log/filesave.log
- /etc/master.d Directory
- /etc/motd
- /etc/passwd
- /etc/profile
- /etc/rc0, /etc/rc2, and the /etc/rc.d Directory
- /etc/save.d Directory
- /etc/shutdown (and the /etc/shutdown.d Directory)
- /etc/TIMEZONE
- /etc/utmp

- /etc/wtmp
- /usr/adm/graflog
- /usr/adm/sulog
- /usr/lib/cron Directory
- /usr/lib/help/HELPLOG
- /usr/lib/spell/spellhist
- /usr/news Directory
- /usr/options Directory
- /usr/spool/cron/crontabs.

Each of these files is briefly described in this chapter.

/etc/checklist

The **/etc/checklist** file is used to define a list of file system devices to be checked for consistency by the **fsck**(1M) default command. The character (raw) device partition for the file system should be identified. The devices listed normally correspond to those mounted when the system is in the multi-user mode (run-level 2). The **root** file system (**/dev/rdsk/c1d0s0**) SHOULD NOT be listed in this file. Except for the **root** file system, a file system must be unmounted to be checked. Therefore, the **checklist** file is convenient to use when in the single-user mode of operation with only the **root** file system mounted. As the system is delivered, the **checklist** file is empty. A typical 3B2 Computer **/etc/checklist** file is shown in Figure 2-1. See **checklist**(4) manual pages for additional information.

```
/dev/rdsk/c1d0s2
```

Figure 2-1. Typical /etc/checklist File

/etc/fstab

The **/etc/fstab** file is used as an argument to the **/etc/mountall** command. The **fstab** file specifies the file system(s) to be mounted by the **/etc/mountall** command. A typical 3B2 Computer **/etc/fstab** file is shown in Figure 2-2. The format of the file is the block device name followed by the mount point name. See **mountall**(1M) manual pages for additional information.

```
/dev/dsk/c1d1s2 /usr
/dev/dsk/c1d0s8 /usr2
/dev/dsk/c1d0s9 /usr3
```

Figure 2-2. Typical /etc/fstab File

/etc/gettydefs

The **/etc/gettydefs** file contains information used by the **getty**(1M) command to set the speed and terminal settings for a **/dev/tty** line. The **getty** command accesses the **gettydefs** with a label. The general format of the **gettydefs** file is as follows.

label# initial-flags # final-flags #login-prompt #next-label

Each line entry in the **gettydefs** file is followed by a blank line. Refer to the **gettydefs**(4) manual pages for additional information. Figure 2-3 shows a typical 3B2 Computer **/etc/gettydefs** file. Note that it is common practice to add the node (machine) name at the beginning of the login prompt fields in the **gettydefs** file.

```
19200# B19200 HUPCL # B19200 SANE IXANY TAB3 HUPCL #login: #9600

9600# B9600 HUPCL # B9600 SANE IXANY TAB3 HUPCL #login: #4800

4800# B4800 HUPCL # B4800 SANE IXANY TAB3 HUPCL #login: #2400

2400# B2400 HUPCL # B2400 SANE IXANY TAB3 HUPCL #login: #1200

1200# B1200 HUPCL # B1200 SANE IXANY TAB3 HUPCL #login: #300

300# B300 HUPCL # B300 SANE IXANY TAB3 HUPCL #login: #19200

console# B9600 HUPCL OPOST ONLCR # B9600 SANE IXANY TAB3 #Console Login: #console1

console1# B1200 HUPCL OPOST ONLCR # B1200 SANE IXANY TAB3 #Console Login: #console2

console2# B300 HUPCL OPOST ONLCR # B300 SANE IXANY TAB3 #Console Login: #console3

console3# B2400 HUPCL OPOST ONLCR # B2400 SANE IXANY TAB3 #Console Login: #console4

console4# B4800 HUPCL OPOST ONLCR # B4800 SANE IXANY TAB3 #Console Login: #console5

console5# B19200 HUPCL OPOST ONLCR # B19200 SANE IXANY TAB3 #Console Login: #console

contty# B9600 HUPCL OPOST ONLCR # B9600 SANE IXANY TAB3 #login: #contty1

contty1# B1200 HUPCL OPOST ONLCR # B1200 SANE IXANY TAB3 #login: #contty2

contty2# B300 HUPCL OPOST ONLCR # B300 SANE IXANY TAB3 #login: #contty3

contty3# B2400 HUPCL OPOST ONLCR # B2400 SANE IXANY TAB3 #login: #contty4

contty4# B4800 HUPCL OPOST ONLCR # B4800 SANE IXANY TAB3 #login: #contty5

contty5# B19200 HUPCL OPOST ONLCR # B19200 SANE IXANY TAB3 #login: #contty

pty# B9600 HUPCL OPOST ONLCR # B9600 SANE IXANY TAB3 #PC login: #pty

4800H# B4800 # B4800 SANE IXANY TAB3 HUPCL #login: #9600H

9600H# B9600 # B9600 SANE IXANY TAB3 HUPCL #login: #19200H

19200H# B19200 # B19200 SANE IXANY TAB3 HUPCL #login: #2400H

2400H# B2400 # B2400 SANE IXANY TAB3 HUPCL #login: #1200H

1200H# B1200 # B1200 SANE IXANY TAB3 HUPCL #login: #300H

300H# B300 # B300 SANE IXANY TAB3 HUPCL #login: #4800H
```

Figure 2-3. Typical gettydefs File

/etc/group

The **/etc/group** file describes each group to the system. An entry is added for each new group. Each entry in the file is one line and consists of four fields. The fields are separated by a colon (:). The format of a line is as follows.

group name:password:group id:login names

The fields are described as follows. Figure 2-4 shows a typical 3B2 Computer **/etc/group** file.

group name The first field defines the group name. The group name is from one to eight characters. The characters are alphanumeric (no uppercase characters).

password The second field contains the encrypted group password. The encrypted group password contains 13 bytes (characters). 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 aging information. The use of group passwords is discouraged.

group id The third field contains the group identification number. Specifying a number outside this range results in an unsuccessful login. A "Bad group id." message is output when attempting to log in with a group identification number larger than 59,999. The group identification field is a number between 0 and 59,999 (inclusive). Group identification numbers 0 through 99 are reserved; 0 indicates the super-user (root). Commas are not entered in this field.

login names The fourth field contains a list of all login names in the group. Names in the list are separated with commas. The names listed may use the **/etc/newgrp** command to become a member of the group.

```
root::0:root
other::1:
bin::2:root,bin,daemon
sys::3:root,bin,sys,adm
adm::4:root,adm,daemon
mail::6:root
rje::8:rje,shqer
daemon::12:root,daemon
```

Figure 2-4. Typical /etc/group File

/etc/inittab

The **/etc/inittab** file contains instructions for the **/etc/init** command. The instructions define the processes that are to be created or terminated for each initialization state. Initialization states are called run-levels or run-states. Run-levels range from 0 through 6. By convention, run-level 1 (S or s) is single-user mode; run-level 2 is multi-user mode. Appendix B, "RUN LEVELS," summarizes the various run-states and describes their uses. See **inittab**(4) manual pages for additional information. Figure 2-5 shows a typical 3B2 Computer **/etc/inittab** file. The general format of a line entry in the **/etc/inittab** file is as follows.

identification:run-state:action:process

These fields are as follows.

identification The identification field is a one- or two-character identifier for the line entry. The identifier is unique for a line.

run-state The run-state defines the run-level in which the entry is to be processed.

action The action field defines how **/etc/init** treats the process field. Refer to the **inittab**(4) manual pages for complete information.

process The process field defines the shell command that is to be executed.

```
zu::sysinit:/etc/bzapunix </dev/console >/dev/console 2>&1
fs::sysinit:/etc/bcheckrc </dev/console >/dev/console 2>&1
mt::sysinit:/etc/brc >/dev/console 2>&1
ck::sysinit:/etc/setclk </dev/console >/dev/console 2>&1
su:1:wait:/etc/shutdown -is -y -g0 </dev/console >/dev/console 2>&1
is:2:initdefault:
lt:s:once:/etc/led -o                          # turn on green LED
p1:s1234:powerfail:/etc/led -f                 # start green LED flashing
p3:s1234:powerfail:/etc/shutdown -y -p9 -i0 -g0 >/dev/console 2>&1
s2:2:wait:/etc/rc2 >/dev/console 2>&1 </dev/console
fl:056:wait:/etc/led -f >/dev/console 2>&1 </dev/console
s0:056:wait:/etc/rc0 >/dev/console 2>&1 </dev/console
of:0:wait:/etc/uadmin 2 0 >/dev/console 2>&1 </dev/console
fw:5:wait:/etc/uadmin 2 2 >/dev/console 2>&1 < /dev/console
RB:6:wait:echo Automatic Reboot >/dev/console 2>&1
rb:6:wait:/etc/uadmin 2 1 >/dev/console 2>&1 </dev/console
co:1234:respawn:/etc/getty console console
ct:2:respawn:/usr/lib/uucp/uugetty -r -t 60 contty 1200H
he:1234:respawn:sh -c 'sleep 20 ; exec /etc/hdelogger >/dev/console 2>&1'
l1:2:respawn:/etc/getty tty11 9600
12:2:respawn:/etc/getty tty12 1200
13:2:off:/etc/getty tty13 9600
14:2:respawn:/usr/lib/uucp/uugetty -r -t 60 tty14 1200H
15:2:off:/etc/getty tty15 9600
```

Figure 2-5. Typical /etc/inittab File

/etc/log/filesave.log

The **/etc/log/filesave.log** file is used by the **/etc/volcopy** command to maintain a record of file system volume copies. If you want to record this information, the **filesave.log** file must be created before you first use the **/etc/volcopy** command. The **/etc/volcopy** command will not create this log file. The command only makes entires to the file if it already exists. A typical **filesave.log** file is shown in Figure 2-6. Each record entry identifies the special device file, file system name (mount point), and the volume for both the source and destination of the volume copy.

```
/dev/rdsk/c1d0s8;usr2;2032 -> /dev/rdsk/c1d1s8;usr2; on Mon Mar 25 06:43:17 1985
```

Figure 2-6. Typical /etc/log/filesave.log File

/etc/master.d Directory

The **/etc/master.d** directory contains files that define the configuration of hardware devices, software drivers, system parameters and aliases. The files are used by **/etc/mkboot** to obtain device information for the generation of device driver and configurable module files. The **/etc/sysdef**(1M) program uses the **master.d** files to get the names of supported devices. The first step in reconfiguring the system to run with different tunable parameters is to edit the appropriate files in the **/etc/master.d** directory. Refer to the **/etc/master**(4) manual pages for additional information.

/etc/motd

The **/etc/motd** file contains the message-of-the-day. The message-of-the-day is output by instructions in the **/etc/profile** file after a successful login. The message-of-the-day should be kept short and to the point. The **/usr/news** file(s) should be used for lengthy, more explicit messages.

/etc/passwd

The **/etc/passwd** file identifies each user to the system. An entry is added for each new user. Each entry in the file is one line and consists of seven fields. The fields are separated by a colon (:). The format of a line is as follows.

login name:passwd:user:group:account:login directory:program

These fields are as follows.

login name The first field defines the login name. The login name is from one to eight characters. The characters are alphanumeric (no uppercase characters).

passwd The second field contains the encrypted login password. The encrypted login password contains 13 bytes (characters). 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 aging information.

user id The third field contains the user identification number. The user identification field is a number between 0 and 59,999 (inclusive). Specifying a number outside this range results in an unsuccessful login. A "Bad user id." message is output when attempting to log in with a user identification number larger than 59,999. User identification numbers 0 through 99 are reserved; 0 indicates the super-user (root). Commas are not entered in this field.

group id The fourth field contains the group identification number. The group identification field is a number between 0 and 59,999 (inclusive). Specifying a number outside this range results in an unsuccessful login. A "Bad group id." message is output when attempting to log in with a group identification number larger than 59,999. Group identification numbers 0 through 99 are reserved; 0 indicates the super-user (root). Commas are not entered in this field.

account The fifth field is used by accounting programs. This field typically contains the user name, department number, and the bin number.

login directory The sixth field defines the full path name of the login directory.

program The seventh field defines the program to be executed after login. If null, the shell (**/bin/sh**) is invoked.

Figure 2-7 shows a typical **/etc/passwd** file. No user logins are shown in this example. See **passwd**(4) manual pages for additional information.

```
root:zZyOd6c22FCfI:0:1:0000-Admin(0000):/:
daemon:Locked;:1:1:0000-Admin(0000):/:
bin::2:2:0000-Admin(0000):/bin:
sys:Locked;:3:3:0000-Admin(0000):/usr/src:
adm:Locked;:4:4:0000-Admin(0000):/usr/adm:
uucp:Locked;:5:5:0000-uucp(0000):/usr/lib/uucp:
nuucp:spJzqtVoUtQzk:10:10:0000-uucp(0000):/usr/spool/uucppublic:/usr/lib/uucp/uucico
rje:Locked;:18:18:0000-rje(0000):/usr/rje:
trouble:Locked;:70:1:trouble(0000):/usr/lib/trouble:
lp:Locked;:71:2:0000-lp(0000):/usr/spool/lp:
setup::0:0:general system administration:/usr/admin:/bin/rsh
powerdown::0:0:general system administration:/usr/admin:/bin/rsh
sysadm::0:0:general system administration:/usr/admin:/bin/rsh
checkfsys::0:0:check diskette file system:/usr/admin:/bin/rsh
makefsys::0:0:make diskette file system:/usr/admin:/bin/rsh
mountfsys::0:0:mount diskette file system:/usr/admin:/bin/rsh
umountfsys::0:0:unmount diskette file system:/usr/admin:/bin/rsh
```

Figure 2-7. Typical /etc/passwd File

/etc/profile

The default profile for all users is in the **/etc/profile** file. The standard (default) environment for all users is established by the instructions in the **/etc/profile** file. The system administrator can change this file to set options for the **root** login. For example, the following may be added to the **/etc/profile** file for the **root** login to cause the erase character to back up and to set the TERM variable.

```
if [ ${LOGNAME} = root ]
        then
                stty echoe
                echo "Enter TERM: \c"
                read TERM
                export TERM
fi
```

Figure 2-8 shows the 3B2 Computer default profile.

```
# The profile that all logins get before using their own .profile.

trap ""  2 3
export LOGNAME

. /etc/TIMEZONE

# Login and -su shells get /etc/profile services.
# -rsh is given its environment in its .profile.
case "$0" in
-su )
        export PATH
        stty ixon -ixany
        ;;
-sh )
        export PATH
        stty ixon -ixany

        # Allow the user to break the Message-Of-The-Day only.
        trap "trap '' 2"  2
        cat -s /etc/motd
        trap "" 2

        if mail -e
        then
                echo "you have mail"
        fi

        if [ ${LOGNAME} != root ]
        then
                news -n
        fi
        ;;
esac

umask 022
trap  2 3
```

Figure 2-8. Standard /etc/profile File

/etc/rc0

The **/etc/rc0** file contains a shell script that is executed by **/etc/shutdown** for transitions to single-user state, and by **/etc/init** on transitions to run-levels 0, 5, and 6. Files in the **/etc/shutdown.d** directory are executed by **/etc/rc0**. As the system is delivered, the **/etc/shutdown.d** directory contains one file named ANNOUNCE. The ANNOUNCE file outputs the message "System services are now being stopped." Any task that you want executed when the system is taken to run-levels 0, S, 5, or 6 may be done by adding a file to the **/etc/shutdown.d** directory. Figure 2-9 shows a typical 3B2 Computer **/etc/rc0** file.

```
# "Run Commands" for init state 0
# Leaves the system in a state where it is safe
# to turn off the power or go to firmware.

stty sane 2>/dev/null
echo 'The system is coming down.  Please wait.'
for f in /etc/shutdown.d/*
{
        if [ -f ${f} ]
        then
                /bin/sh ${f}
        fi
}
trap "" 15
kill -15 -1
sleep 10
/etc/killall  9
sleep 10
sync
/etc/umountall
stty sane 2>/dev/null
ps=`ps -fe`
psc=`echo "${ps}"  |  wc -l`
if [ ${psc} -gt 7 ]
then
        echo 'The shutdown did not complete properly.
Too many processes are still running.'
        echo "${ps}"
fi
sync;  sync
echo '\nThe system is down.'
sync
```

Figure 2-9. Typical /etc/rc0 File

/etc/rc2

The **/etc/rc2** file contains a shell script that is executed by **/etc/init** on transitions to run-level 2 (multi-user state). Executable files in the **/etc/rc.d** directory are executed by **/etc/rc2**. The **/etc/rc.d** executable files and a description of each are as follows.

.0_firstcheck Executes the first-time checks for the machine that has yet to be used as a customer machine.

.setup Explains how to set up the machine for the first time.

MOUNTFILESYS
Sets up and mounts file systems. Builds the mount table and mounts the **root** (/) and user (/usr) file systems. Makes the **/usr/tmp** directory, thus, cleaning up (deleting) any previous files in that directory.

autoconfig Makes a disk resident version of the in(hymemory **unix** operating system if self-configuration occurred during the boot sequence. The new in-memory operating system is copied to **/unix**.

cron Starts the **cron** daemon by executing **/etc/cron**.

disks Creates "**/dev**" entries for new disk(s) by executing **/etc/disks**.

sysetup Removes the **/etc/ps_data** to force the **/bin/ps** command to read the **/unix** file. Outputs the system configuration if the **/etc/prtconf** command exists. Outputs the system trademark information in the **/etc/tm** directory.

uucp When basic networking is added to the system, the **uucp** file is added to this directory. The **uucp** file cleans up (deletes) uucp lock files under the **/usr/spool/locks** directory.

ctc	This file is added when the Cartridge Tape Controller (CTC) utilities is loaded on the system. The **ctc** file is executed when the computer is brought up to "multi-user" mode. This file checks for the CTC hardware, validates the major device numbers, verifies the existence of the CTC pump firmware file, and then pumps (attempts) the CTC peripheral(s).
lp	When line printer spooling is added to the system, the **lp** file is added to this directory. The **lp** file removes the spooler lock file and starts the scheduler.
nodename	Defines the node name of the machine by executing the **uname**(1M) command. This file is created by the **sysadm setup** and **sysadm nodename** commands.

Other files may also be added to the **etc/rc.d** directory as a function of adding hardware or software to the system. Figure 2-10 shows a typical 3B2 Computer **/etc/rc2** file.

```
# "Run Commands" executed when the system is changing
#  to init state 2, traditionally called "multi-user".

. /etc/TIMEZONE

# Pickup start-up packages for mounts, daemons, services, etc.
stty sane 2>/dev/null
echo 'The system is coming up.  Please wait.'
for f in /etc/rc.d/*
{
        if [ -f ${f} ]
        then
                /bin/sh ${f}
        fi
}
echo 'The system is ready.'
```

Figure 2-10. Typical /etc/rc2 File

/etc/save.d Directory

The **/etc/save.d** directory contains files that are used by the Simple Administration commands associated with backing up data. The following files are included.

except A list of the directories and files that *should not* be copied as part of an incremental backup (**savefiles**) is maintained in this file.

timestamp/... The date and time of the last backup (volume or incremental) is maintained for each file system in the **/etc/save.d/timestamp** directory.

/etc/shutdown

The **/etc/shutdown** file contains a shell script to gracefully shut down the system in preparation for system backup or for scheduled downtime. After stopping all nonessential processes, the **shutdown** script executes files in the **/etc/shutdown.d** directory by calling **/etc/rc0** for transitions to run-level 1, s or S. For transitions to other run-levels, the **shutdown** script calls **/etc/init**. As the system is delivered, this directory contains one file named ANNOUNCE. The ANNOUNCE file outputs the message "System services are now being stopped." Figure 2-11 shows a typical 3B2 Computer **/etc/shutdown** file.

```
#   Sequence performed to change the init stat of a machine.

#   This procedure checks to see if you are permitted and allows an
#   interactive shutdown.  The actual change of state, killing of
#   processes and such are performed by the new init state, say 0,
#   and its /etc/rc0.

#   Usage:  shutdown [ -y ] [ -p<proccount> ] [ -g<grace-period> ] \
#      [ -i<init-state> ]

askconfirmation=yes
proccount=7

if [ `pwd` != / ]
then
        echo "$0:  You must be in the  /  directory to run /etc/shutdown."
        exit 1
fi

#   Check the user id.
eval `id | sed 's/[^a-z0-9=].*//'`
if [ "${uid:=0}" -ne 0 ]
then
        echo "$0:  Only root can run /etc/shutdown."
        exit 2
fi

grace=60
initstate=s
```

Figure 2-11. Typical /etc/shutdown File (Sheet 1 of 4)

```
while [ $# -gt 0 ]
do
        case $1 in
        -g[0-9]* )
                grace=`expr "$1" : '-g\([0-9]*\)'`
                ;;
        -i[0-6abcQqSs] )
                initstate=`expr "$1" : '-i\([0-6abcQqSs]*\)'`
                ;;
        -p[0-9]* )
                proccount=`expr "$1" : '-p\([0-9]*\)'`
                ;;
        -y )
                askconfirmation=
                ;;
        -* )
                echo "Illegal flag argument '$1'"
                exit 1
                ;;
        * )
                echo "Usage:  $0 [ -y ] [ -p<proccount> ] [ -g<grace> ] [ -i<initstate> ]"
                exit 1
        esac
        shift
done

if [ -n "${askconfirmation}" -a -x /etc/ckbupscd ]
        then
        #   Check to see if backups are scheduled at this time
        BUPS=`/etc/ckbupscd`
        if [ "$BUPS" != "" ]
                then
                echo "$BUPS"
                echo "Do you wish to abort this shutdown and return to
command level to do these backups? [y, n] \c"
                read YORN
                if [ "$YORN" = "y" -o "$YORN" = "Y" ]
                        then
                        exit 1
                fi
        fi
fi
```

Figure 2-11. Typical /etc/shutdown File (Sheet 2 of 4)

```
if [ -z "${TZ}"   -a  -r /etc/TIMEZONE ]
then
          /etc/TIMEZONE
fi

echo '\nShutdown started.    \c'
date
echo

sync
cd /

trap "exit 1"  1 2 15

a="`who  |  wc -l`"
if [ ${a} -gt 1  -a  ${grace} -gt 0 ]
then
        su adm -c /etc/wall<<-!
                <CTRL g>The system will be shut down in ${grace} seconds.
                Please log off now<CTRL g>.

        !
        sleep ${grace}
fi

/etc/wall <<-!
        <CTRL g>THE SYSTEM IS BEING SHUT DOWN NOW ! ! !<CTRL g>
        <CTRL g>Log off now or risk your files being damaged.<CTRL g>

!
sleep ${grace}
```

Figure 2-11. Typical /etc/shutdown File (Sheet 3 of 4)

```
if [ ${askconfirmation} ]
then
        echo "Do you want to continue? (y or n):   \c"
        read b
else
        b=y
fi
if [ "$b" != "y" ]
then
        /etc/wall <<-\!
                False Alarm:  The system will not be brought down.
        !
        echo 'Shut down aborted.'
        exit 1
fi
case "${initstate}" in
s | S )
        . /etc/rc0
esac
/etc/init ${initstate}
```

Figure 2-11. Typical /etc/shutdown File (Sheet 4 of 4)

/etc/TIMEZONE

The **/etc/TIMEZONE** file sets the time zone shell variable TZ. The TZ variable is initially established for the system via the Simple Administration **setup** function. The TZ variable in the **TIMEZONE** file is changed by the Simple Administration **timezone** command (**sysadm timezone**). The TZ variable can be redefined on a user (login) basis by setting the variable in the associated **.profile**. The **TIMEZONE** file is executed by **/etc/rc2**. Figure 2-12 shows a typical **/etc/TIMEZONE** file.

The format of the **TZ** shell variable is as follows.

TZ=*TTT#SSS*

where:

TTT is the three-character abbreviation for the local time zone.

is the number of hours that the local time zone differs from Greenwich Mean Time (GMT). This field can be entered as a positive or negative number.

SSS is the three-character abbreviation for the local daylight savings time zone. This field is entered only if daylight savings time is observed.

```
# Set timezone environment to default for the machine
TZ=EST5EDT
export TZ
```

Figure 2-12. Typical /etc/TIMEZONE File

/etc/utmp

The **/etc/utmp** file contains information on the run-state of the system. This information is accessed with a **who -a** command.

/etc/wtmp

The **/etc/wtmp** file contains a history of system logins. The owner and group of this file must be **adm**, and the access permissions must be 664 (*-rw-rw-r--*). This file is updated each time **login** is run. As the system is accessed, this file increases in size. Periodically, this file should be cleared or truncated. The command line **>/etc/wtmp** when executed by **root** creates the file with nothing in it. The following is an example command line to limit the size of the **/etc/wtmp** file to the last 100 lines in the file.

```
tail -100 /etc/wtmp > /tmp/wtmp; mv /tmp/wtmp /etc/wtmp
```

Note that **/etc/cron**, **/etc/rc0**, or **/etc/rc2** can be used to clean up the **wtmp** file. To use one of these functions to limit the size of a log file, add the appropriate command line to the **/usr/spool/cron/crontab/root**, **/etc/shutdown.d/...**, or **/etc/rc.d/...** file.

/usr/adm/graflog

If the Graphics Utilities is installed, the **/usr/adm/graflog** file will keep a history of **graphics** command usage. Each time a user executes **graphics**, an entry consisting of the login name, terminal number, and system date is made to the file. As a security measure, this file should not be readable by "others." The **/usr/adm/graflog** file should be periodically truncated to keep the size of the file within a reasonable limit. Note that **/etc/cron**, **/etc/rc0**, or **/etc/rc2** can be used to clean up the **graflog** file. To use one of these functions to limit the size of a log file, add the appropriate command line to the **/usr/spool/cron/crontab/root**, **/etc/shutdown.d/...**, or **/etc/rc.d/...** file. The following is a command line to limit the size of the log file to the last 100 lines in the file.

```
tail -100 /usr/adm/graflog > /tmp/graflog; mv /tmp/graflog /usr/adm/graflog
```

Figure 2-13 shows the contents of a typical **/usr/adm/graflog** file.

```
root    console    Mar 27 08:52
klm     tty12      Mar 27 11:47
rar     contty     Mar 28 07:15
root    console    Mar 28 09:23
rar     tty11      Mar 30 14:24
```

Figure 2-13. Typical /usr/adm/graflog File

/usr/adm/sulog

The **/usr/adm/sulog** file contains a history of switch user (**su**) command usage. As a security measure, this file should not be readable by "others." The **/usr/adm/sulog** file should be periodically truncated to keep the size of the file within a reasonable limit. Note that **/etc/cron**, **/etc/rc0**, or **/etc/rc2** can be used to clean up the **sulog** file. To use one of these functions to limit the size of a log file, add the appropriate command line to the **/usr/spool/cron/crontab/root**, **/etc/shutdown.d/...**, or **/etc/rc.d/...** file. The following is a command line to limit the size of the log file to the last 100 lines in the file.

```
tail -100 /usr/adm/sulog > /tmp/sulog; mv /tmp/sulog /usr/adm/sulog
```

Figure 2-14 shows the contents of a typical **/usr/adm/sulog** file.

```
SU 08/18 12:35 + console root-sysadm
SU 08/18 16:11 + console root-sysadm
SU 08/18 16:16 + console root-sysadm
SU 08/18 23:45 + tty?? root-uucp
SU 08/19 11:53 + console root-sysadm
SU 08/19 15:25 + console root-sysadm
SU 08/19 23:45 + tty?? root-uucp
SU 08/20 10:16 + console root-adm
SU 08/20 10:33 + tty24 rar-root
SU 08/20 10:42 + console root-sysadm
SU 08/20 10:59 + console root-root
SU 08/20 11:01 + console root-sysadm
SU 08/20 12:36 + tty11 bin-bin
SU 08/20 12:37 + tty11 tws-bin
SU 08/20 14:42 - tty24 awa-sys
SU 08/20 14:47 - tty24 awa-sys
SU 08/20 14:48 + tty24 awa-root
SU 08/20 15:44 + console root-sysadm
```

Figure 2-14. Typical /usr/adm/sulog File

/usr/lib/cron/log

A history of all actions taken by **/etc/cron** are recorded in the **/usr/lib/cron/log** file. The **/usr/lib/cron/log** file should be periodically truncated to keep the size of the file within a reasonable limit. Note that **/etc/cron**, **/etc/rc0**, or **/etc/rc2** can be used to clean up the **/usr/lib/cron/log** file. To use one of these functions to limit the size of a log file, add the appropriate command line to the **/usr/spool/cron/crontab/root**, **/etc/shutdown.d/...**, or **/etc/rc.d/...** file, as applicable. The following is a command line to limit the size of the log file to the last 100 lines in the file.

```
tail -100 /usr/lib/cron/log > /tmp/log; mv /tmp/log /usr/lib/cron/log
```

Figure 2-15 shows the typical information found in the **/usr/lib/cron/log** file.

```
! *** cron started ***   pid = 237 Sun Aug 19 14:06:45 1984
>  CMD: /usr/lib/uucp/uudemon.hour > /dev/null
>  root 251 c Sun Aug 19 14:11:00 1984
<  root 251 c Sun Aug 19 14:11:01 1984
>  CMD: /usr/lib/uucp/uudemon.poll > /dev/null
>  root 370 c Sun Aug 19 14:30:00 1984
<  root 370 c Sun Aug 19 14:30:03 1984
>  CMD: /usr/lib/uucp/uudemon.hour > /dev/null
>  root 417 c Sun Aug 19 14:41:01 1984
<  root 417 c Sun Aug 19 14:41:02 1984
>  CMD: /usr/lib/uucp/uudemon.poll > /dev/null
>  root 452 c Sun Aug 19 15:01:00 1984
<  root 452 c Sun Aug 19 15:01:04 1984
>  CMD: /usr/lib/uucp/uudemon.hour > /dev/null
>  root 460 c Sun Aug 19 15:11:00 1984
<  root 460 c Sun Aug 19 15:11:00 1984
>  CMD: /usr/lib/uucp/uudemon.poll > /dev/null
>  root 541 c Sun Aug 19 15:30:00 1984
<  root 541 c Sun Aug 19 15:30:07 1984
```

Figure 2-15. Typical /usr/lib/cron/log File

/usr/lib/help/HELPLOG

Providing that **help** monitoring has been enabled, a history of all actions taken by the **/usr/bin/help** command is kept in the **/usr/lib/help/HELPLOG** file. The **HELPLOG** file is copied to **/usr/lib/help/oHELPLOG** and a new **/usr/lib/help/HELPLOG** file is created by the **/usr/lib/help/helpclean** command. Executing the **helpclean** command twice in succession zeros out the **HELPLOG** and the **oHELPLOG** files. Note that **/etc/cron**, **/etc/rc0**, or **/etc/rc2** can be used to clean up the **HELPLOG** file. To use one of these functions to limit the size of a log file, add the appropriate command line to the **/usr/spool/cron/crontab/root**, **/etc/shutdown.d/...**, or **/etc/rc.d/...** file, as applicable. The following is a command line to limit the size of the log file to the last 100 lines in the file.

```
tail -100 /usr/lib/help/HELPLOG > /tmp/help; mv /tmp/help /usr/lib/help/HELPLOG
```

Figure 2-16 shows the typical information found in the **/usr/lib/help/HELPLOG** file.

```
login=bin       uname=wr3b2a    date=Mon Aug 20 12:51:03 EDT 1984
name=locate     response='l'    status=OK
name=locate     response='d'    status=ERROR
name=getkey     response='k'    status=OK
name=keysrch    response='list' status=OK
name=quit       response='q'    status=OK
login=bin       uname=wr3b2a    date=Mon Aug 20 12:51:41 EDT 1984
```

Figure 2-16. Typical /usr/lib/help/HELPLOG File

/usr/lib/spell/spellhist

If the Spell Utilities is installed, a history of all words that **spell**(1) fails to match is kept in the **/usr/lib/spell/spellhist** file. Periodically, this file should be reviewed for words that should be added to the dictionary. After the **spellhist** is reviewed, it can be cleared. Refer to the *AT&T 3B2 Computer Spell Utilities Guide* for information on adding words to the dictionary and cleaning up the **spellhist** file.

/usr/news

The **/usr/news** directory contains news files. The file names are descriptive of the contents of the files. The file names are analogous to head lines. When a user reads the news (**news** command), an empty file named **.news_time** is created (or written) in their login directory. The date (time) of this file is used by the **news** command to determine if a user has accessed the latest news file(s).

/usr/options Directory

The **/usr/options** directory contains files that identify the utilities that are installed on the system. Listing the contents of this directory is an easy way to determine what utilities have been installed on the system.

Figure 2-17 shows a typical **/usr/options** directory. Not all possible utilities are listed (installed). The example shown in Figure 2-17 was taken from a system with a 30-megabyte hard disk. All of the utilities identified in Figure 2-18 cannot be loaded at the same time on a 10-megabyte hard disk. The full names of the utilities are contained in the */usr/options* files. The contents of the files identified in Figure 2-17 are shown in Figure 2-18.

```
-r-xr-xr-x   1 bin      bin          22 Sep 20  1984 calc.name
-r-xr-xr-x   1 bin      bin          23 Sep 10  1984 cc.name
-r--r--r--   1 bin      bin          34 Aug 10  1984 crypt.name
-r-xr-xr-x   1 bin      bin          15 Feb  5 16:34 ctc.name
-r-xr-xr-x   1 bin      bin          40 Aug 10  1984 dfm.name
-r--r--r--   1 bin      bin          19 Aug 15  1984 dmd.name
-r--r--r--   1 bin      bin          38 Aug 16  1984 dmdapd.name
-r--r--r--   1 bin      bin          55 Aug 17  1984 dmdtxt.name
-r-xr-xr-x   1 bin      bin          30 Apr 26  1984 dwb.name
-r--r--r--   1 bin      bin          18 Aug 10  1984 ed.name
-r-xr-xr-x   1 bin      bin          39 Aug 10  1984 esg.name
-r-xr-xr-x   1 bin      bin          29 Sep  7  1984 fortran.name
-r-xr-xr-x   1 bin      bin          19 Aug 10  1984 graph.name
-r-xr-xr-x   1 bin      bin          15 Aug 15  1984 help.name
-rw-r--r--   1 root     root         20 May 24 07:38 im.name
-r-xr-xr-x   1 bin      bin          38 Sep 20  1984 ipc.name
-r--r--r--   1 bin      bin          22 Mar 13 17:09 kersrc.name
-r-xr-xr-x   1 bin      bin          32 Aug 17  1984 lp.name
-r-xr-xr-x   1 bin      bin          35 Sep 20  1984 perf.name
-r-xr-xr-x   1 bin      bin          30 Aug 17  1984 sccs.name
-r-xr-xr-x   1 bin      bin          30 Aug 10  1984 sgs.name
-r-xr-xr-x   1 bin      bin          28 Sep 20  1984 shell.name
-r-xr-xr-x   1 bin      bin          16 Aug 10  1984 spell.name
-r-xr-xr-x   1 bin      bin          32 Sep 20  1984 sysadm.name
-r-xr-xr-x   1 bin      bin          27 Aug 10  1984 term.name
-r--r--r--   1 bin      bin          31 Aug 17  1984 terminf.name
-r-xr-xr-x   1 bin      bin          27 Sep 20  1984 usrenv.name
-rwxr-xr-x   1 root     sys          27 Oct 24 11:23 uucp.name
-rw-r--r--   1 bin      bin          42 Apr  4  1984 wwb.name
```

Figure 2-17. Typical /usr/options Directory

```
Calculation Utilities
C Programming Language
Security Administration Utilities
Tape Utilities
Directory and File Management Utilities
DMD Core Utilities
DMD Application Development Utilities
DMD Text Processing and Graphics Application Utilities
DOCUMENTER'S WORKBENCH System
Editing Utilities
Extended Software Generation Utilities
FORTRAN Programming Language
Graphics Utilities
HELP Utilities
AT&T Integral Modem
Inter-Process Communication Utilities
Kernel Source Package
Line Printer Spooling Utilities
Performance Measurements Utilities
Source Code Control Utilities
Software Generation Utilities
Shell Programming Utilities
SPELL Utilities
System Administration Utilities
Terminal Filters Utilities
Terminal Information Utilities
User Environment Utilities
Basic Networking Utilities
Writer's Workbench Software Release 2.0.2
```

Figure 2-18. Typical /usr/options File Contents

/usr/spool/cron/crontabs

The **/usr/spool/cron/crontabs** directory contains crontab files for **adm**, **root**, and **sys** logins. Providing that the user's logname is in the **/usr/lib/cron/cron.allow** file, users can establish their own crontabs file using the **crontab** command. If the **cron.allow** file does not exist, the **/usr/lib/cron/cron.deny** file is checked to determine if the user is denied the use of the **crontab** command.

As **root**, you can either use the **crontab**(1) command or edit the appropriate file under **/usr/spool/cron/crontabs** to establish the appropriate entries. Refer to the **crontab**(1) command manual page for additional information. The **/etc/cron** program checks the **/usr/spool/cron/crontabs** files for changes during process initialization and when a file changes. The line entry format of a **/usr/spool/cron/crontabs/***logname* file is as follows.

minute hour day month day-of-week command

The various fields of a **crontabs/***logname* line entry are as follows.

minute	The minutes field is a one- or two-digit number in the range of 0 to 59, inclusive.
hour	The hour field is a one- or two-digit number in the range of 0 to 24, inclusive.
day	The day field is the numerical day of the month in the range of 1 to 31, inclusive.
month	The month field is the numerical month of the year in the range of 1 through 12, inclusive.
day-of-week	The day-of-week field is the numerical day of the week where Sunday is 0, Monday is 1, . . . and Saturday is 6.
command	The command field is the program or command that is executed at the time specified by the first five fields.

The following syntax applies to the first five fields.

- Two numbers separated by a minus indicate an inclusive range of numbers between the two specified numbers.
- A list of numbers separated by commas specifies all of the numbers listed.
- An asterisk specifies all legal values.

In the command field (sixth field), a percent sign (%) is translated to a new-line character. Only the first line of a command field (character string up to the percent sign) is executed by the shell. Any other lines are made available to the command as standard input.

Figure 2-19 shows a typical **/usr/spool/cron/crontabs/***logname* file. The data shown is the **root** file. The file entries support the functions of the **calendar** reminder service and basic networking. The major point to remember is that you can use the **cron** function to decrease the number of data terminal driven system administration tasks. Any task that needs to be done repeatedly as a function of time is a candidate for inclusion in your crontab file.

```
0 1 * * * /usr/bin/calendar -
41,11 * * * * /usr/lib/uucp/uudemon.hour > /dev/null
45 23 * * * ulimit 5000; /bin/su uucp -c "/usr/lib/uucp/uudemon.cleanup" > /dev/null 2>&1
1,30 * * * * /usr/lib/uucp/uudemon.poll > /dev/null
```

Figure 2-19. Typical /usr/spool/cron/crontabs/root File

Chapter 3

ADMINISTRATIVE TASKS

PAGE

Chapter 3

ADMINISTRATIVE TASKS

GENERAL

Overview

This chapter describes some of the tasks (responsibilities) of an AT&T 3B2 Computer system administrator. Additional tasks such as adding users, removing users, etc. are supported by Simple Administration (**sysadm**) commands. Refer to the *AT&T 3B2 Computer Owner/Operator Manual* for a complete description of the Simple Administration commands. The tasks described in this chapter include:

- Maintaining System Log
- Formatting Floppy Disks
- Duplicating Floppy Disks
- Verifying Floppy Disk Usability

- Creating and Identifying File Systems on Floppy Disk
- Formatting Cartridge Tapes
- Creating and Identifying File Systems on Cartridge Tape
- Populating a Lost+Found Directory
- Running Diagnostics
- Setting Time-of-Day/Date
- Rebuilding File System Free List
- File System Reorganization
- Monitoring Disk Space
- Copying/Moving Directories
- Installing and Removing Utilities
- Determining System Status After Trouble
- File System Checking and Repair
- Reloading UNIX Operating System
- File System Backup
- File System Restoral from Backup
- Allocating System Resources
- System Reconfiguration
- Establishing/Changing the System and Node Names

- Security Administration
- Forgotten **root** Password Recovery
- Forgotten FIRMWARE Password Recovery

- Firmware Procedures
- Making a Bootable Device
- Assignment of Default Boot Program and Device.

Most of these tasks are done on an as required basis. Certain of these tasks should be done on a routine (scheduled) basis. Use the chapter table of contents and the Index to quickly find a particular task description.

Conventions

In the sample task descriptions, command inputs and 3B2 Computer responses are shown as follows.

This style of type is used to show system generated responses displayed on your screen.

This style of bold type is used to show inputs entered from your keyboard that are displayed on your screen.

These bracket symbols, < > identify inputs from the keyboard that are not displayed on your screen, such as: <CR> carriage return, <CTRL d> control-d, <ESC g> escape-g, passwords, and tabs.

This style of italic type is used for notes that provide you with additional information.

This chapter contains sample displays that will help you understand described procedures. The sample displays in this chapter and the displays on your terminal screen may differ slightly due to improvements in the product after this document was finalized. Therefore, use the displays in this document as samples of the type of data available. However, the data displayed on your terminal screen accurately reflects the software on your computer.

Administrative Etiquette

Many administrative tasks require the system to be shut down to a run level other than multi-user (run-level 2). This means that the conventional users cannot access the system. When the machine is taken out of the multi-user mode, the users logged(hyon the machine at the time are logged-off. These types of tasks should, therefore, be done to the extent possible on a noninterference basis with the user population. Sometimes situations arise that require the system to be taken down with little or no notice provided to the users. Try to provide the user community as much notice as possible about events affecting the use of the machine. When the system must be taken out-of-service, also tell the users when to expect the system to be available. Use the news (**/etc/news/***headline*) and the message-of-the-day (**/etc/motd**) to keep users informed about changes in hardware, software, policies, and procedures.

At your discretion, the following items should be done as prerequisites for most tasks described in this chapter.

a. When possible, schedule service-affecting tasks to be done during periods of low system use. For scheduled actions, use the message-of-the-day (**/etc/motd**) to inform users of future actions.

b. Check who is logged-in before taking any actions that would affect a logged-in user. The **/etc/whodo** and **/bin/who** commands can be used to see who is on the system.

c. If the system is in use, provide the users advanced warning about changes in system states or pending maintenance actions. For immediate actions, use the **/etc/wall** command to send a broadcast message announcing that the system will be taken down at a given time. Give the users a reasonable amount of time to terminate their activities and log off before taking the system down.

CONSOLE TERMINAL CONFIGURATION

In general, all system administration functions are done at the console terminal. The baud rate of the console terminal should be set at 9600. If the console port is operated at another data rate, you may lose communication with the system when you shut down to the firmware mode (run-level 5). If this happens, set your input/output terminal speed option to 9600 and hit RETURN. Note that the baud rate of the CONSOLE and contty ports can be changed as a function of the optional firmware DEbug MONitor (DEMON).

It is recommended that a printer be part of the console equipment configuration. When doing system administration tasks, the printer should be enabled. This provides a record of exactly what was done and how the system responded. It is especially advantageous to have the printer enabled when running diagnostics. The use of a printer can also simplify the task of maintaining a system log.

MAINTAINING A SYSTEM LOG

Maintaining a system log book may not be necessary in all system installations. However, a system log is a valuable tool when trouble shooting transient problems or when trying to establish system operating characteristics over a period of time. For example, maintenance records are very important when trying to determine the system operating cost. Printouts can be easily added to the log if it is maintained in a ring-binder. Keeping track of equipment and system configuration changes is another aspect of maintaining a system log. In a multi-user environment it is strongly recommended that a complete set of records be maintained.

The format of the system log and the types of items noted in the log should follow a logical structure. The log may be thought of as a diary that is maintained on a periodic basis. To a large measure, the nature of YOUR use of the system will dictate the form and importance of maintaining a system log.

FORMATTING FLOPPY DISKS

Caution: Formatting a floppy disk in one drive and using it in a different drive may be a problem when one or both drives are out of alignment. If you experience this problem, call your service representative for help.

General

Floppy disks are formatted by the UNIX System **fmtflop**(1M) command.

Before a new floppy disk can be used for the storage of information, it must be formatted. Formatting defines the tracks (cylinders) on a new floppy disk. Formatting also erases any data that may exist on a used floppy disk in addition to redefining the tracks. It is suggested that you format an entire box of floppy disks at a time. By formatting floppy disks on a box basis, the problem of keeping track of which floppies are or are not formatted is avoided. If the box is opened, all floppies are formatted.

Before you insert a floppy disk into the floppy disk drive, check that the media is free to move within the floppy disk jacket. Floppy disks that bind or do not move freely within the floppy disk jacket can be fixed by holding the floppy disk perpendicular to the edge of a table and sliding the edges of the floppy disk jacket across the edge of a table. This action increases the clearance between the media and the jacket by making the edges of the jacket more rounded. Be careful not to touch the media.

fmtflop Formatting Procedure

The time required to format a box of ten floppy disks using a single floppy disk drive is about 35 minutes (about 3.5 minutes per floppy). The time required to format a box of ten floppy disks using two floppy disk drives is about 18 minutes.

The major steps in formatting floppy disks are as follows.

1. Insert the floppy disk into the floppy disk drive.

2. Execute the **fmtflop** command, specifying the verify (-v) option. The raw (character) device partition for the entire floppy disk must be specified. For the integral floppy disk drive, use **/dev/rSA/diskette1**. (The partition name **/dev/rSA/diskette1** is linked to **/dev/rdsk/c0d0s6**.) For the second floppy disk drive, use **/dev/rSA/diskette2**. (The partition name **/dev/rSA/diskette2** is linked to **/dev/rdsk/c?d1s6**. The question mark (**?**) represents the controller number for the second floppy disk drive.) To simultaneously use two drives, execute the commands in the background; redirect the standard and error output as desired.

3. Remove the floppy disk from the floppy disk drive when complete (floppy disk indicator is off). Check the error output for successful completion. No messages are output for a successful completion.

The following shows the typical command line entries and system responses for formatting and verifying a floppy disk using the first (integral) floppy disk drive. As shown, the **fmtflop** command is silent when the floppy disk successfully formats and verifies. The floppy disk to be formatted must be inserted into the integral floppy disk drive BEFORE executing **fmtflop**.

Note: Insert the floppy disk into the drive before executing the command.

```
# fmtflop -v /dev/rSA/diskette1<CR>
#
```

Note: The command has no output when successful

The following shows the typical command line entries and system responses for simultaneously formatting and verifying two floppy disks. The intent of this example is to show one of many possible approaches to simultaneously using two floppy disk drives. As shown, the **fmtflop** command is silent when the floppy disk successfully formats and verifies. The floppy disks to be formatted are inserted into the floppy disk drives BEFORE executing **fmtflop** commands. The commands are executed in the background. The standard and error outputs are redirected to temporary files by the **2>&1** string. The results of the processes are checked by examining the temporary files AFTER the completion of the last **fmtflop** command. In this example, a **wait** command is used to delay the execution of the **cat** command until the last **fmtflop** command (process 5626) completes execution. A **wait** command with no arguments waits for the completion of all background processes belonging to the envoking user identification.

Note: Insert the floppy disks into the drives before executing the commands.

```
# fmtflop -v /dev/rSA/diskette1 > /tmp/floppy1 2>&1&<CR>
5625
# fmtflop -v /dev/rSA/diskette2 > /tmp/floppy2 2>&1&<CR>
5626
# wait 5626;cat /tmp/floppy?<CR>
#
```

Note: The **fmtflop** *command has no output when successful.*

DUPLICATING FLOPPY DISKS

General

The contents of an existing floppy disk are copied to another formatted floppy disk by using the **dd** command. A block size of 4608 bytes is specified when using the **dd** command to duplicate floppy disks. This block size matches the 79 tracks per side of the floppy disk and results in the most efficient data transfer to and from a floppy disk.

When using one floppy disk drive, the contents of the floppy disk to be duplicated are first copied to a temporary file space. Either the character (**/dev/rSA/diskette1**) or the block (**/dev/SA/diskette1**) device can be specified in this procedure. The same device (character or raw) must be specified for the entire procedure. The source floppy disk is then replaced with a formatted floppy disk and the temporary file space copied to the destination floppy. A minimum of 1430 blocks must be available (free) in the **root** file system to use **/tmp** as the temporary file space. This free space requirement includes 1422 blocks for the contents of the floppy disk plus free space to create a temporary file. Any file system can be used for the temporary file space. A **tmp** directory is normally used since the files are automatically deleted during the transition to the multi-user mode (run-level 2). This is also done for the **/usr/tmp** directory. Thus, use the **/usr/tmp** directory as the temporary file space if enough free space is not available in the **root** file system.

When using two floppy disk drives, a disk-to-disk copy method is used to duplicate the contents of one floppy disk to another floppy disk. Temporary file space is not required for a disk-to-disk copy.

Single Floppy Disk Drive Copy Procedure

The time required to duplicate a floppy disk is about 5 minutes (2.5 minutes for each transfer) when using a block size of 4608 bytes (9b). This a minimum time that increases with system load.

The steps necessary to duplicate (copy) a floppy disk are as follows.

1. At the console terminal, log in as **root**. (If "others" have read/write permission to **/dev/rSA/diskette1**, any login can be used to do the copy. As the system is delivered, only **root** has read/write permission.)

2. Write protect the source floppy disk.

3. Insert the source floppy disk into the floppy disk drive.

4. For the integral floppy disk drive, enter the following command:

   ```
   dd bs=4608 if=/dev/rSA/diskette1 of=/tmp/tmpfile
   ```

   The indicator on the floppy disk drive will turn-on in response to this command.

5. When the indicator on the floppy drive is off, remove the source floppy disk. Insert a formatted destination floppy disk into the floppy disk drive. Be sure to properly label the duplicate floppy disk. At this point, the contents of the source floppy is in **/tmp/tmpfile**.

6. Enter the following command:

   ```
   dd bs=4608 if=/tmp/tmpfile of=/dev/rSA/diskette1
   ```

7. When the indicator on the floppy disk drive is off, remove the floppy disk. Store the floppy disks (original and copy) in a safe place.

The following shows the command line entries and system responses associated with duplicating (copying) a floppy disk using a single drive. The source floppy disk is write protected.

```
# dd bs=4608 if=/dev/rSA/diskette1 of=/tmp/tmpfile<CR>
158+0 blocks in
158+0 blocks out

Note: Wait for prompt, then:
      (1) Remove original (source) floppy disk.
      (2) Insert formatted (destination) floppy disk.

# dd bs=4608 if=/tmp/tmpfile of=/dev/rSA/diskette1<CR>
158+0 blocks in
158+0 blocks out
#
```

Dual Floppy Disk Drive Copy Procedure

The time required to duplicate a floppy disk is about 4 minutes when using a block size of 4608 (9b). This a minimum time that increases with system load.

The steps necessary to duplicate (copy) a floppy disk are as follows.

1. At the console terminal, log in as **root**. (If "others" have read/write permission to **/dev/rSA/diskette1** and **/dev/rSA/diskette2**, any login can be used to do the copy. As the system is delivered, only **root** has read/write permission.)

2. Label the destination floppy disk as appropriate. Write protect the source floppy disk.

3. Insert the source floppy disk into the integral floppy disk drive (diskette1).

4. Insert the destination floppy disk into the second floppy disk drive (diskette2).

5. Enter the following command:

   ```
   dd bs=4608 if=/dev/rSA/diskette1 of=/dev/rSA/diskette2
   ```

   The indicators on the floppy disk drives turn-on in response to this command.

6. When the indicators on the floppy drives turn off, remove the source and destination floppy disks. Store the floppy disks (original and copy) in a safe place.

The following shows the command line entries and system responses associated with duplicating (copying) a floppy disk using two drives.

```
# dd bs=4608 if=/dev/rSA/diskette1 of=/dev/rSA/diskette2<CR>
158+0 blocks in
158+0 blocks out

Note: Wait for prompt and indicators to turn off, then
      remove original (source) and duplicate (destination) floppy disks.

#
```

VERIFYING FLOPPY DISK USABILITY

General

The integrity of the storage medium of a formatted floppy disk can be verified without changing the data on the disk by using the **dd** command to copy the data stored on the disk to **/dev/null**. Either the character (**/dev/rSA/diskette1**) or the block (**/dev/SA/diskette1**) device can be specified as the source. The **dd** command reports the number of whole and partial data blocks that are processed (input and output). A "good" floppy disk provides 158 blocks (4608-byte blocks). The block size of 4608 bytes is the number of bytes per track on one side of a floppy disk (9 tracks times 512 bytes). Specifying a block size matching the number of bytes per side is the most efficient way to do the verify procedure using the **dd** command. Since no destination file is created by directing the output to **/dev/null**, this procedure is independent of the amount of free disk space and requires no file cleanup at the end of the copy. You can direct the output to a temporary file as described for duplicating a floppy disk. Essentially, the verify procedure is the first half of a single floppy disk drive copy procedure.

Verifying Formatted Floppy Disk Procedure

The time required to verify a floppy disk is less than three minutes. The steps in verifying a formatted floppy disk are as follows.

1. At the console terminal, log in as **root**. (If "others" have read/write permission to **/dev/rSA/diskette1**, any login will work. As the system is delivered, only **root** has read/write permission.)

2. Insert the floppy disk to be verified into the floppy disk drive.

3. Enter the following command:

   ```
   dd bs=4608 conv=noerror if=/dev/rSA/diskette1 of=/dev/null
   ```

   The indicator on the floppy disk drive will turn-on in response to this command.

4. When the indicator on the floppy is off, remove the floppy disk. The number of blocks copied in and out should be 158+0 (158 whole blocks). If the number of blocks copied in/out equals 158+0 (158 whole blocks), the storage medium is "good." If the number of blocks is not equal to 158+0, the disk medium is suspect.

The following shows the command line entries and system responses associated with verifying a formatted floppy disk. The floppy disk used in this example passed the verification.

```
# dd bs=4608 conv=noerror if=/dev/rdiskette of=/dev/null<CR>
158+0 blocks in
158+0 blocks out
#
```

The following shows the command line entries and system responses associated with verifying a formatted floppy disk having a problem.

```
# dd bs=4608 conv=noerror if=/dev/rdiskette of=/dev/null<CR>
NOTICE:
Floppy Access Error: Consult the Error Message Section
of the System Administration Utilities Guide


NOTICE:
Floppy Access Error: Consult the Error Message Section
of the System Administration Utilities Guide
dd read error: No such device or address
4+0 blocks in
4+0 blocks out
157+1 blocks in
157+1 blocks out
#
```

CREATING AND IDENTIFYING FILE SYSTEMS ON FLOPPY DISK

General

A file system is created and identified by the **mkfs**(1M) and **labelit**(1M) commands. The maximum size of a file system that can be created on a floppy disk is 1422 blocks (512-byte blocks). Normally, the entire floppy disk (partition 6) is specified when making a file system. When making a bootable floppy disk, part of the storage is used for the boot partition (partition 7). Therefore, bootable floppy disks use partition 5 for the file system. As shown in Appendix A, "DEVICE DEFAULT PARTITIONING," partition 5 of a floppy disk is 1404 blocks.

When making a file system, all specified blocks are not available for files depending on the number of information nodes (i-nodes) allocated. For example, specifying 1422 blocks and 192 i-nodes results in 1392 available blocks; 400 i-nodes results in 1366 available blocks. The number of i-nodes needed is a function of the number of directories and files to be created on the floppy disk. When using a floppy disk file system to store many small files, it is possible to use up the available i-nodes before running out of free space. If the number of i-nodes is not specified, the number of i-nodes defaults to one-fourth the number of logical (1024-byte) blocks, rounded down to a modulo 16 value. I-nodes are allocated in modulo 16 groups. When either a calculated or specified i-node number is not a multiple of 16, the number is rounded down to the next modulo 16 value. Therefore, the default i-node value for 1422 blocks is 176 (711 divided by 4 equals 177.78).

File System Creation Procedure

The steps in making and identifying a file system on a floppy disk are as follows.

1. At the console terminal, log in as **root**. (If "others" have read/write permission to **/dev/diskette**, any login will work. As the system is delivered, only **root** has read/write permission.)

2. Insert a formatted floppy disk into the integral floppy disk drive.

3. Make a file system of 1422 blocks and 192 information nodes using the following command. The rotational gap is 1 and the blocks-per-cylinder is 18. For optimum performance, always use a rotational gap of 1 and 18 blocks-per-cylinder for floppy disk file systems.

```
mkfs /dev/SA/diskette1 1422:192 1 18
```

4. Label the floppy disk file system using the **labelit** command. Use the same identification information for the stick-on label that you specify for the **labelit** command.

5. File systems are mounted at **root** (/) as directories. Make a directory at **root** that is the name of the file system. A file system is mounted using the **mount**(1M) command. Mounting a file system at a mount point that does not match the file system label produces an output message defining what has been mounted.

6. Mount the floppy disk file system and make a **lost+found** director. Refer to the "POPULATING A LOST+FOUND DIRECTORY" procedure described in this chapter. The file system creation procedure is complete after populating the **lost+found** directory. Unmount the file system and remove the floppy disk from the drive if no further use is to be made of the newly created file system. (Remember that a file system must be idle to be unmounted. To unmount a file system no processes can be executing from the file system and no one's present working directory can be within the file system.)

The following shows the command line entries and system responses associated with making and identifying a file system on a floppy disk. The **mkfs** command outputs the message **(DEL if wrong)** and then waits 10 seconds before executing. If you realize that the command is not what you wanted to do, typing a **DEL** at this time terminates the command before execution. The **labelit** command provides the same capability. In this example, the file system (rar) is mounted as **/rar** and then mounted as **/install** to show the applicable output messages.

```
# mkfs /dev/SA/diskettel 1422:192 1 18<CR>
Mkfs: /dev/SA/diskettel?
(DEL if wrong)
bytes per logical block = 1024
total logical blocks = 711
total inodes = 192
gap (physical blocks) = 1
cylinder size (physical blocks) = 18
mkfs: Available blocks = 696
# labelit /dev/rSA/diskettel rar 2032<CR>
Current fsname: , Current volname: , Blocks: 1422, Inodes: 192
FS Units: 1Kb, Date last mounted: Mon Apr  1 12:25:18 1985
NEW fsname = rar, NEW volname = 2032 -- DEL if wrong!!
# mkdir /rar<CR>
# mount /dev/diskette /rar<CR>
# mount<CR>
/ on /dev/dsk/cld0s0 read/write on Sat Mar 30 07:55:43 1985
/usr on /dev/dsk/cld0s2 read/write on Sat Mar 30 07:57:40 1985
/usr2 on /dev/dsk/cld1s8 read/write on Sat Mar 30 09:25:51 1985
/usr3 on /dev/dsk/cld1s9 read/write on Sat Mar 30 09:26:00 1985
/rar on /dev/SA/diskettel read/write on Mon Apr  1 12:32:11 1985
# mkdir /rar/lost+found<CR>
```

Populate the lost+found directory
by creating and deleting an appropriate number of files.
Refer to the POPULATING A LOST+FOUND DIRECTORY
procedure described in this chapter.

```
# umount /dev/SA/diskettel<CR>
# mount /dev/dsk/diskettel /install<CR>
mount: warning! <rar> mounted as </insta>
# mount<CR>
/ on /dev/dsk/cld0s0 read/write on Sat Mar 30 07:55:43 1985
/usr on /dev/dsk/cld0s2 read/write on Sat Mar 30 07:57:40 1985
/usr2 on /dev/dsk/cld1s8 read/write on Sat Mar 30 09:25:51 1985
/usr3 on /dev/dsk/cld1s9 read/write on Sat Mar 30 09:26:00 1985
/install on /dev/SA/diskettel read/write on Mon Apr  1 12:35:14 1985
# umount /dev/diskette<CR>
#
```

FORMATTING CARTRIDGE TAPES

Caution: Formatting a cartridge tape in one drive and using it in a different drive may be a problem when one or both drives are out of alignment. If you experience this problem, call your service representative for help.

General

Cartridge tapes are formatted by the UNIX System **ctcfmt**(1M) command.

Before a new cartridge tape can be used for the storage of information, it must be formatted. Formatting defines the tracks and pass count threshold on a cartridge tape. The recommended pass count threshold is 4,000 when the cartridge tape is used as a streaming device. When used as a file system, the recommended pass count is 1,000. The default pass count threshold of the **ctcfmt** command is 4,000. A different pass count threshold can be specified by using the **-p** *number* option. Formatting erases any data that may exist on a used cartridge tape. Cartridge tapes wear-out with use. Therefore, a pass count is maintained on a cartridge tape. When the pass count equals the pass count threshold, a warning message is output. **The pass count is not erased by reformatting a cartridge tape.** Formatting also establishes a Volume Table Of Contents (VTOC) for the tape. The cartridge tape VTOC is fixed within the **ctcfmt** process. The VTOC defines the number and size of partitions. Partitioning is used to divide the storage space of a physical device into a number of more manageable pieces. The VTOC for a cartridge tape is displayed by the **ctcinfo -v** command.

ctcfmt Formatting Procedure

The time required to format (and verify) a cartridge tape is about 20 minutes. This time estimate is a minimum time that does not include tape re-tensioning (about 3 minutes). The time required to format a cartridge tape increases with increased system load. The steps in formatting a cartridge tape are as follows.

1. Insert the cartridge tape into the cartridge tape drive and wait for the re-tensioning pass to complete.

   Caution: Once started, the** ctcfmt **command cannot be interrupted; the program ignores break, delete, etc. To keep the CONSOLE available for input during the cartridge tape formatting process, execute the command in the background.

2. Execute the **ctcfmt** command in the background, specifying the verify (-v) option. The raw (character) device partition for the entire cartridge tape must be specified. For the first cartridge tape drive, specify **/dev/rSA/ctape1** as the raw device partition. The following command line executes the **ctcfmt** command in the background with the standard and error outputs going to the terminal.

   ```
   ctcfmt -vt /dev/rSA/ctape1&
   ```

   If desired, the command can be executed in the background with the standard output and error output redirected to a file. The following command line executes the **ctcfmt** command in the background with the standard and error outputs going to a file named **/tmp/ctc**. The format process is done when the cartridge tape drive indicator turns off. The **/tmp/ctc** file is then checked to determine the completion status of the process.

   ```
   ctcfmt -vt /dev/rSA/ctape1 > /tmp/ctc 2>&1&
   ```

3. The formatting procedure is complete when a "successfully formatted" message is output/displayed. If no other procedures are to be run using the newly formatted cartridge tape, remove the cartridge tape from the drive. If an error message is displayed, refer to the "Recognizing a Bad Cartridge Tape" procedure.

The following shows the typical command line entry and system responses for formatting and verifying a cartridge tape. The command is executed in the background to keep the console terminal available for input during the formatting process. The number (2723 in the sample output) displayed after the command input line is the process number.

```
    Insert the cartridge tape into the drive and wait for
    re-tensioning to complete before executing the command.

# ctcfmt -vt /dev/rSA/ctape1&<CR>
2723
#
Insert tape, wait for re-tension pass to complete,
and press the <RETURN> key when ready [q]: Format completed successfully.
```

Recognizing a Bad Cartridge Tape

When an attempt to format a cartridge tape results in error messages that include an error number 5, do the following, as required, to determine if a defective cartridge tape is the cause of the error.

1. Check that the cartridge tape is NOT write protected (SAFE). If it is SAFE, rotate the protection tab to the non-protected position and retry the process.

2. Try to format a different cartridge tape. If the same errors occur, suspect a hardware problem with either the CTC interface board or the tape drive. Clean the tape drive and then try to format a cartridge tape.

3. If another cartridge tape formats successfully, try to format the previous cartridge tape that failed. If the same errors occur, the cartridge tape is defective.

The following shows the command line entry and system responses for an attempt to format a write-protected (SAFE) cartridge tape. A defective cartridge tape may also include an error message from the cartridge tape controller driver. Typically, this additional error message is error number 215 (tape read or write error).

Note: Insert the cartridge tape into the drive and wait for re-tensioning to complete before executing the command.

```
# ctcfmt -vt /dev/rSA/ctape1&<CR>
2804
#
Insert tape, wait for re-tension pass to complete,
ctcfmt: format failed for /dev/rSA/ctape1, errno=5
and press the <RETURN> key when ready [q]: Format completed successfully.
```

CREATING AND IDENTIFYING FILE SYSTEMS ON CARTRIDGE TAPE

General

Warning: Cartridge tape file systems should be re-tensioned every four hours. Re-tensioning is done by unmounting the file system and operating the tape drive latch. After re-tensioning, the tape file system can be remounted. Failure to re-tension can result in damage to the cartridge tape and/or loss of data.

Caution: The primary function of cartridge tape is as a streaming device for doing file system backups. As a file system, cartridge tape should only be used to contain seldomly accessed data. Tape file systems are very slow compared to floppy disk or hard disk file systems. A mounted tape file system degrades the overall system performance because of the time required to write the system buffers to all mounted devices.

Caution: When cartridge tape is used as a file system, format the cartridge tape with a pass count threshold of 1,000 or less to compensate for the increased wear factor.

A file system is created and identified by the **mkfs**(1M) and **labelit**(1M) commands. The maximum size of a file system that can be created on a 23-megabyte cartridge tape is 45539 blocks (512-byte blocks) on partition 3. Normally, the partition 3 (/dev/SA/ctape1) is specified when making a tape file system. Tape partition 3 CANNOT be used when making a bootable tape. When making a bootable cartridge tape, part of the storage is used for the boot partition (partition 7). Therefore, bootable cartridge tapes use partition 0 for the file system. As shown in Appendix A, "DEVICE DEFAULT PARTITIONING," partition 0 of a cartridge tape is 8928 blocks. The tape partition values are fixed by the tape formatting process [**/etc/ctcfmt**(1M) command].

When making a file system, all specified blocks are not available for files depending on the number of information nodes (i-nodes) allocated. For example, specifying 1422 blocks and 192 i-nodes results in 1392 available blocks; 400 i-nodes results in 1366 available blocks. The number of i-nodes needed is a function of the number of directories and files to be created on the cartridge tape. When using a cartridge tape file system to store many small files, it is possible to use up the available i-nodes before running out of free space. If the number of i-nodes is not specified, the number of i-nodes defaults to one-fourth the number of logical (1024-byte) blocks, rounded down to a modulo 16 value. I-nodes are allocated in modulo 16 groups. When either a calculated or specified i-node number is not a multiple of 16, the number is rounded down to the next modulo 16 value. Therefore, the default i-node value for 1422 blocks is 176 (711 divided by 4 equals 177.78).

While it is important to create a file system large enough to support the intended application, it is equally important to avoid creating a significantly over-sized tape file system. The size of a tape file system should be limited to what is appropriate for the use of the file system. The time required to create a tape file system and use the resulting file system increases with the size of the file system.

File System Creation Procedure

The steps in making and identifying a file system on a cartridge tape are as follows.

1. At the console terminal, log in as **root**. (If "others" have read/write permission to the tape device partitions, any login will work. As the system is delivered, only **root** has read/write permission.)

2. Insert a formatted cartridge tape into the tape drive.

3. Make a file system specifying the appropriate partition, number of blocks, and i-nodes using the following command format. The rotational gap is 1 and the blocks-per-cylinder is 31. For optimum performance, always use a rotational gap of 1 and 31 blocks-per-cylinder for tape file systems.

   mkfs *partition blocks:i-nodes* **1 31**

4. Label the cartridge tape file system using the **labelit** command. Use the same identification information for the stick-on label that you specify for the **labelit** command.

5. File systems are normally mounted at **root** (/) as directories. Make a directory at **root** that is the name of the file system. A file system is mounted using the **mount**(1M) command. Mounting a file system at a mount point that does not match the file system label produces an output message defining what has been mounted.

6. Mount the cartridge tape file system and make a **lost+found** director. Refer to the "POPULATING A LOST+FOUND DIRECTORY" procedure described in this chapter. The file system creation procedure is complete after populating the **lost+found** directory. Unmount the file system and remove the cartridge tape from the drive if no further use is to be made of the newly created file system. (Remember that a file system must be idle to be unmounted. To unmount a file system no processes can be executing from the file system and no one's present working directory can be with in the file system.)

The following shows the command line entries and system responses associated with making and identifying a file system on a cartridge tape. The **mkfs** command outputs the message **(DEL if wrong)** and then waits 10 seconds before executing. If you realize that the command is not what you wanted to do, typing a **DEL** at this time terminates the command before execution. The **labelit** command provides the same capability. In this example, a small file system (2000 blocks) is created with a default number of i-nodes (240). The file system (labeled rar) is mounted as **/rar** and then mounted as **/install** to show the applicable output messages.

```
# mkfs /dev/SA/ctape1 2000 1 31<CR>
Mkfs: /dev/SA/ctape1?
(DEL if wrong)
bytes per logical block = 1024
total logical blocks = 1000
total inodes = 240
gap (physical blocks) = 1
cylinder size (physical blocks) = 31
mkfs: Available blocks = 982
# labelit /dev/rSA/ctape1 rar 2032<CR>
Current fsname: , Current volname: , Blocks: 1422, Inodes: 192
FS Units: 1Kb, Date last mounted: Mon Apr  1 12:25:18 1985
NEW fsname = rar, NEW volname = 2032 -- DEL if wrong!!
# mkdir /rar<CR>
# mount /dev/SA/ctape1 /rar<CR>
# mount<CR>
/ on /dev/dsk/c1d0s0 read/write on Sat Mar 30 07:55:43 1985
/usr on /dev/dsk/c1d1s2 read/write on Sat Mar 30 07:57:40 1985
/usr2 on /dev/dsk/c1d0s8 read/write on Sat Mar 30 09:25:51 1985
/usr3 on /dev/dsk/c1d0s9 read/write on Sat Mar 30 09:26:00 1985
/rar on /dev/SA/ctape1 read/write on Mon Apr  1 12:32:11 1985
# mkdir /rar/lost+found<CR>
```

Populate the lost+found directory
by creating and deleting an appropriate number of files.
Refer to the POPULATING A LOST+FOUND DIRECTORY
procedure described in this chapter.

```
# umount /dev/SA/ctape1<CR>
# mount /dev/SA/ctape1 /install<CR>
mount: warning! <rar> mounted as </insta>
# mount<CR>
/ on /dev/dsk/c1d0s0 read/write on Sat Mar 30 07:55:43 1985
/usr on /dev/dsk/c1d1s2 read/write on Sat Mar 30 07:57:40 1985
/usr2 on /dev/dsk/c1d0s8 read/write on Sat Mar 30 09:25:51 1985
/usr3 on /dev/dsk/c1d0s9 read/write on Sat Mar 30 09:26:00 1985
/install on /dev/SA/ctape1 read/write on Mon Apr  1 12:35:14 1985
# umount /dev/SA/ctape1<CR>
#
```

POPULATING A LOST+FOUND DIRECTORY

General

Each file system should contain a **lost+found** directory in the root directory of the file system. The **lost+found** directory is used by **/etc/fsck**(1M) to reconnect unreferenced directories and files. Unreferenced information nodes (i-nodes) are relinked to available i-nodes in the **lost+found** directory. Therefore, each **lost+found** directory must be populated with enough i-nodes to support the size of the file system in the event of file system corruption.

The **/etc/mkdir**(1M) command is used to created the **lost+found** directory. Populating the this directory to reserve i-node slots is best done via a program (shell script). I-node slots are reserved in a **lost+found** directory by creating the appropriate number of files in the directory and then removing the files. As a general guide, the number of slots to reserve is about ten percent of the number of i-nodes allocated for the file system.

Sample Shell Procedure

The following is a sample shell script for populating a **lost+found** directory. To use this program first change directory to the **lost+found** directory to be populated. Then execute the program with an argument specifying the number of i-nodes allocated for the file system. This is a prototype shell script. It is intended to provide a starting point for the development of your own procedure.

```
# cat popl+f<CR>
# Populate lost+found directory PROTOTYPE.
# Creates files in modulo 50 file increments
# for ten percent of the supplied number

if [ "${#}" -gt "1"   ] ! [ "${#}" = "0" ]
then
echo "Usage: pop n0
echo "        where n = number of i-nodes"
exit
fi

n=`expr \( \( $1 / 10 \) + 1 \) / 50`
if [ "$n" = "0" ]
then
n=1
fi
echo "Creating $n times 50 files (`expr $n  50`)."

while [ "${n}"  -gt  "0" ]
do
>a${n}; >b${n}; >c${n}; >d${n}; >e${n}
>f${n}; >g${n}; >h${n}; >i${n}; >j${n}
>k${n}; >l${n}; >m${n}; >n${n}; >o${n}
>p${n}; >q${n}; >r${n}; >s${n}; >t${n}
>u${n}; >v${n}; >w${n}; >x${n}; >y${n}
>A${n}; >B${n}; >C${n}; >D${n}; >E${n}
>F${n}; >G${n}; >H${n}; >I${n}; >J${n}
>K${n}; >L${n}; >M${n}; >N${n}; >O${n}
>P${n}; >Q${n}; >R${n}; >S${n}; >T${n}
>U${n}; >V${n}; >W${n}; >X${n}; >Y${n}

n=`expr ${n} - 1`
done
# Remove (delete) files
echo "Removing files"
rm [A-z][0-9]*
exit
#
```

RUNNING DIAGNOSTICS

Caution: Diagnostic phases should not be run by a casual user. If you have a system failure and you are not sure you can run the diagnostic phases properly, contact your service representative for help.

General

Diagnostics are intended to be used as tools for locating hardware problems in the 3B2 Computer. By running the diagnostic phases, you should be able to isolate the source of the problem to a specific area of hardware or possibly to a specific board. This will help the service representative determine the nature of the problem and what needs to be done to solve the problem.

After powering up the system, the system administrator should periodically run all diagnostics. Since only the NORMAL diagnostics are run during a powerup sequence, diagnostics should be periodically run to execute the DEMAND and INTERACTIVE phases. In certain applications, the system may be left powered-up and only powered down for maintenance. In this situation it is very important that a schedule be established to periodically run diagnostics.

Types of Diagnostics

There are three types of diagnostic phases: normal, demand, and interactive. They are defined as follows.

NORMAL Normal diagnostics are automatically run each time the system is powered up. The normal diagnostics are run manually via the diagnostic monitor (dgmon).

DEMAND Demand diagnostics are diagnostics that run only on a manual request basis via the diagnostic monitor. These diagnostic phases DO NOT run automatically as part of a powerup sequence.

INTERACTIVE Interactive diagnostics are diagnostics that are manually run via the diagnostic monitor and require operator intervention. The operator intervention usually consists of inserting a floppy disk into the floppy disk drive and/or entering data via the keyboard.

Diagnostic Monitor Execution

Diagnostics are run from the firmware mode via the diagnostic monitor program (dgmon). To get to the firmware mode from the multi-user mode, you must be logged in at the console as **root**. The steps necessary to run diagnostics are as follows.

1. At the console terminal, take the system to the firmware mode (run-level 5).

   ```
   shutdown -y -i5
   ```

   If you are the only one logged in, you can use an express shutdown (-g0) where a grace period of zero seconds is used.

2. Enter the firmware password. (**mcp** is the default firmware password.) What you type is not echoed to the terminal. The system will respond with an "Enter" message.

3. Execute the **dgmon** program from the hard disk (option **1**).

4. Run diagnostics as required. Refer to the "Diagnostic Command Examples" description for command format.

5. When you are finished running diagnostics, **boot** the UNIX Operating System (**unix**) from the hard disk (option **1**).

The following shows the typical command line entries and system responses associated with accessing the diagnostic monitor.

```
# shutdown -y -i5<CR>

                        series of messages are displayed
                        ending with the following

INIT: New Run level: 5

The system is down.


SELF-CHECK

FIRMWARE MODE
<mcp><CR>

Enter name of program to execute [ ]: dgmon<CR>
            Possible load devices are:

Option Number   Slot     Name
---------------------------------------
       0          0     FD5
       1          0     HD30
       2          0     HD30
       3          3     CTC

Enter Load Device Option Number [1 (HD30)]: <CR>


        3B2 DIAGNOSTIC MONITOR
DGMON > h<CR>
        3B2
     DIAGNOSTIC
      COMMANDS   OPTIONS                                    DESCRIPTION
      ==========  =======                                   ===========
         DGN     [DEVICE [DEVICE # ¦ REP=? ¦ PH=?-? ¦    DIAGNOSE DEVICES(S)
                                     UCL ¦ SOAK ]]

        H(ELP)   (NONE)                                     PRINT HELP MENU

        L(IST)   DEVICE                                     LIST DEVICE PHASE TABLE

        Q(UIT)   (NONE)                                     EXIT DGMON

        S(HOW)   (NONE)                                     SHOW EDT
```

Continued

Continued from previous screen

```
DGMON > s<CR>

Current System Configuration

System Board memory size: 2 megabyte(s)

00 - device name = SBD       ,occurrence = 0, slot = 00, ID code = 0x01
     boot device = y, board width = double, word width = 2 byte(s),
     req Q size = 0x00, comp Q size = 0x00, console ability = y, pump file = n
     subdevices(s)
     #00 = FD5       , ID code = 0x00, #01 = HD30      , ID code = 0x03
     #02 = HD30      , ID code = 0x03

Press any key to continue<CR>


01 - device name = PORTS      ,occurrence = 0, slot = 01, ID code = 0x03
     boot device = n, board width = single, word width = 2 byte(s),
     req Q size = 0x03, comp Q size = 0x23, console ability = y, pump file = y
     subdevices(s)

Press any key to continue<CR>


02 - device name = PORTS      ,occurrence = 1, slot = 02, ID code = 0x03
     boot device = n, board width = single, word width = 2 byte(s),
     req Q size = 0x03, comp Q size = 0x23, console ability = y, pump file = y
     subdevices(s)

Press any key to continue<CR>


03 - device name = CTC        ,occurrence = 0, slot = 03, ID code = 0x05
     boot device = y, board width = single, word width = 2 byte(s),
     req Q size = 0x10, comp Q size = 0x20, console ability = n, pump file = y
     subdevices(s)
     #00 = FT25      , ID code = 0x04, #01 = FD5       , ID code = 0x01

DONE

DGMON > q<CR>
```

Continued

Continued from previous screen

```
Enter name of program to execute [ ]: unix<CR>
            Possible load devices are:

Option Number    Slot    Name
----------------------------------------
       0          0     FD5
       1          0     HD30
       2          0     HD30
       3          3     CTC

Enter Load Device Option Number [1 (HD30)]: <CR>

UNIX System V Release 2.0.3 3B2 Version 2
unix
Copyright (c) 1984 AT&T.
All Rights Reserved

        fsstat: root file system needs checking
        The root file system (/dev/dsk/c1d0s0) is being checked automatically.

  /dev/dsk/c1d0s0
  File System: root Volume: 2032

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  717 files 12376 blocks 1796 free
```

Note: File systems listed in **/etc/fstab** *may also be checked.*

```
The system is coming up.  Please wait
AT&T 3B2 SYSTEM CONFIGURATION:

Memory size: 2 Megabytes
System Peripherals:

         Device Name         Subdevices

         SBD
                          Floppy Disk
                          30 Megabyte Disk
                          30 Megabyte Disk
         PORTS
         PORTS
         CTC
The system is ready.

Console Login:
```

Execution Options

The devices that can be checked by running the diagnostic phases are listed in the Equipped Device Table (EDT). The **s** command is used to show the EDT. To see the list of devices, enter: **s**<CR>.

The diagnostic phases for a given device are listed in the Diagnostic Phase Tables. The **l** command (lowercase letter "l") is used to display these tables. To display the diagnostic phases for a specific device, enter: **l** *device_type*<CR>.

The **dgn** command is used to run the diagnostic phases. The optional arguments of the **dgn** command are as follows.

DEVICE (type) Represents the option of running diagnostic phases on a certain type of device. For example, the command **dgn sbd** runs all the NORMAL diagnostic phases on the system board.

DEVICE # Represents the option of running diagnostic phases on a certain device. For example, the command **dgn ports 0** runs all the NORMAL diagnostic phases on the first ports board (the first occurrence equals 0).

rep=? Represents the number of times you want the phases to run. Valid numbers are in the range of 1 through 65536.

ph=?[-?] Represents the option to run a specific phase or string of phases. When running specific phases, be sure you know which phase you want or you could cause some problems. INTERACTIVE phases are run if they are included in a string of phases. When possible, run INTERACTIVE phases separately.

ucl Represents the option to run the phases in the unconditional mode. In this mode testing continues when a phase fails. The results of each phase are displayed as it is completed. This mode cannot be used with the **soak** option.

soak Represents the option of running the phases continuously and storing all the results until testing is completed. This allows you to check for intermittent problems by comparing the number of failures against the number of times the phase ran. For each specified device, soak runs all NORMAL and DEMAND phases in sequence within the requested range of phases. Soak is stopped by either entering a character at the console, or using the **rep** option. The **soak** option cannot be used with the INTERACTIVE phases.

Examples of Diagnostic Commands

The following are some examples of valid **dgn** commands using the various options. All the following commands should be followed by a carriage return.

dgn Runs all NORMAL phases once on the devices in the Equipped Device Table. The results of each phase are displayed as it completes. If any of the phases fail, testing stops, and a failure message is displayed.

dgn sbd 0
Runs all NORMAL phases once on the system board. The results of each phase are displayed as it completes. If any of the phases fail, testing stops and a failure message is displayed.

dgn ports
Runs all NORMAL phases once on all the ports boards. If any of the phases fail, testing stops and a failure message is displayed.

dgn ports 0
Runs all NORMAL phases once on the first ports board (the first occurrence equals 0). The results of each phase are displayed as it completes. If any of the phases fail, testing stops and a failure message is displayed.

dgn ports 1 ucl
Runs all NORMAL phases once on the second ports board (the second occurrence equals 1). The results of each phase are displayed as it completes. Testing continues if a phase fails.

dgn sbd ph=3
Runs phase 3 (CPU #4 Normal DGN) once. The results of the phase are displayed as it completes.

dgn sbd ph=1-3

Runs phases 1 (CPU #2 Normal DGN), 2 (CPU #3 Normal DGN), and 3 (CPU #4 Normal DGN) once or until a failure occurs. The results of each phase are displayed as it completes.

dgn sbd rep=3 ph=3

Runs phase 3 (CPU #4 Normal DGN) three times. Testing stops if any part of the phase fails, and a failure message is displayed.

dgn ucl

Runs all NORMAL phases once on every device in the Equipped Device Table. Results of each phase are displayed as it runs. Testing continues if a phase fails.

dgn ucl rep=3

Runs all NORMAL phases 3 times. Testing continues if a phase fails. The results of each phase are displayed as it runs.

dgn soak

Runs all NORMAL and DEMAND phases on all boards until a character is entered on the console terminal. Testing stops when a character is entered, and the results are displayed.

dgn ports 1 soak

Runs all NORMAL and DEMAND phases on the second ports board until a character is entered on the console. Testing stops when a character is entered, and the results are displayed.

dgn sbd soak ph=1-3

Runs phases 1 (CPU #2 Demand DGN), 2 (CPU #3 Demand DGN), and 3 (CPU #4 Demand DGN), until a character is entered on the console terminal. Testing results are displayed when testing stops.

dgn sbd soak rep=10 ph=11

Runs phase 11 (Control Status Register Normal DGN) 10 times and then displays a summary of the results. Testing continues when a phase fails.

dgn soak rep=25

Runs all NORMAL and DEMAND phases on all boards 25 times and displays the results when testing is completed. Testing continues when a phase fails.

Note: When specific phases are requested, the device(s) to be tested must be designated.

Suggested Sequence for Running Phases

The Diagnostic MONitor (DGMON) program is used to run diagnostic phases. Decide what phases you want to run before starting. The following sequence is a guideline to follow when running diagnostic phases. Random running of phases is not suggested.

> ***Note:*** Be sure to record the results of all phases so the service representative will have an idea of the trouble before making the service call.

1. Use the **s** command to identify what devices there are in the Equipped Device Table.

2. Use the **dgn** command to locate which device is causing the trouble.

3. If one of the devices returns a **DIAGNOSTICS FAILED** message, run all the NORMAL phases on that device.

4. If any of the NORMAL phases failed, you may want to repeat those phases using either the **soak** option or the **ucl** with a number of repetitions (**rep**=*?*).

   > ***Note:*** The phase number for the individual diagnostic phases is found by using the **l** command (lowercase letter "l").

5. If you are testing because of a system failure or intermittent trouble, but all the NORMAL phases passed, the next logical step is to run some of the DEMAND phases.

6. If any of the DEMAND phases failed, you may want to repeat those phases using either the **soak** option or the **ucl** with a number of repetitions (**rep**=*?*).

7. After running any phases that failed with the **soak** or **ucl** option, omit those phases and continue running the other phases.

8. When a phase fails, the phases that follow may not be executed. If any of the devices in the Equipped Device Table were not tested because of a premature test termination, those devices should be tested by using the specific device number.

9. After running all the NORMAL and all the DEMAND phases, you may want to run the INTERACTIVE phases.

10. Once you have completed running diagnostic phases, check to make sure you have a complete record of the results. If the service representative cannot understand your results, the diagnostics must be run again.

How to Leave the Diagnostic Monitor

There are two procedures available for leaving the Diagnostic Monitor.

- If you are in the Diagnostic Monitor because of a system failure message or an intermittent problem, or if any of the phases failed during testing, go to the "Procedure for Shutdown When a Problem is Present."
- If you are in the Diagnostic Monitor to do non-trouble testing and all the phases passed, go to the "Procedure for Rebooting the UNIX System."

When you receive the Diagnostic Monitor prompt (DGMON >), you can quit the Diagnostic Monitor. Proceed to the appropriate procedure for leaving the DGMON program.

Procedure for Shutdown When a Problem is Present

If you entered the Diagnostic Monitor because of the system failure message or if any of the diagnostic phases failed do the following:

1. Depress the power switch to STANDBY. This causes the computer to execute a "soft" shutdown.
2. Contact your service representative.

Procedure for Rebooting the UNIX System

The following procedure should only be used if all the diagnostic phases passed and the system failure message was not the reason for entering the Diagnostic Monitor. Simply quit the Diagnostic Monitor and execute **unix** from the hard disk. The following command line entries and system responses show how to quit the Diagnostic Monitor and return to the operating system.

```
DGMON > q<CR>

Enter name of program to execute [ ]: unix<CR>
             Possible load devices are:

Option Number    Slot    Name
-----------------------------------------
       0          0     FD5
       1          0     HD30
       2          0     HD30
       3          3     CTC

Enter Load Device Option Number [1 (HD30)]: <CR>

UNIX System V Release 2.0.3 3B2 Version 2
unix
Copyright (c) 1984 AT&T.
All Rights Reserved

       fsstat: root file system needs checking
       The root file system (dev/dsk/c1d0s0) is being checked automatically.

  /dev/dsk/c1d0s0
  File System: root Volume: 2032

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List (Ignored)
  ** Phase 6 - Salvage Free List
  507 files 10158 blocks 2154 free
The system is coming up.  Please wait.
AT&T 3B2 SYSTEM CONFIGURATION:

Memory size: 2 Megabytes
System Peripherals:

         Device Name        Subdevices

         SBD
                          Floppy Disk
                          30 Megabyte Disk
                          30 Megabyte Disk
         PORTS
         PORTS
         CTC
The system is ready.

Console Login:
```

Sample Diagnostic Execution

The following examples are provided to show you what to expect when running diagnostics. The system responses will vary according to the command you execute.

NORMAL Diagnostic Phase

Phase 1 (CPU #2) is a NORMAL type phase run on the system board and takes about 1 second to execute. The following command line entry and system responses show the successful execution of phase 1 on the system board.

```
DGMON > dgn sbd ph=1<CR>

     <<<  DIAGNOSTIC MODE  >>>

Test: CPU_2 NORMAL
Time Taken = ~1 second


1 2 3 4 5 6 7
CPU_2 Diagnostic Completed ATP

     SDB 0 (IN SLOT 0) DIAGNOSTICS PASSED
```

DEMAND Diagnostic Phase

Phase 16, Permanent Interrupt diagnostic, is a DEMAND type phase run on the system board. This phase takes about 1 second to execute. The following command line entry and system responses show the successful execution of the Permanent Interrupt demand diagnostic phase.

Caution: A failure of this test may affect the other tests that assume the system is clear of interrupts.

```
DGMON > dgn sbd ph=16<CR>

    <<<  DIAGNOSTIC MODE  >>>

Test: Permanent Interrupt DEMAND
Time Taken = ~1 second

Permanent Interrupt Diagnostic Completed ATP

    SDB 0 (IN SLOT 0) DIAGNOSTICS PASSED

DGMON >
```

INTERACTIVE Diagnostic Phase

Phase 12, Non-Volatile Memory diagnostic, is an INTERACTIVE type phase run on the system board. This phase takes about 1 second to execute. The following command line entry and system responses show the successful execution of this diagnostic phase.

```
DGMON > dgn sbd ph=12<CR>

    <<<  DIAGNOSTIC MODE  >>>

Test: Non-Volatile Static RAM INTERACTIVE
Time Taken = ~1 second

WARNING: This test can destroy NVRAM contents!
Are you certain you wish to execute this diagnostic [y/n]: y
Non-Volatile RAM Diagnostic Completed ATP

     SDB 0 (IN SLOT 0) DIAGNOSTICS PASSED

DGMON >
```

SETTING TIME-OF-DAY/DATE CLOCK

Caution: Setting the date ahead by one or more days should be done in the single-user mode. Setting the date ahead while in the multi-user mode with *cron* ***running should be avoided. The*** *cron* ***program will try to "catch-up" for the time interval involved. All the processes that were scheduled to run in the time interval are started by*** *cron.*

When the UNIX Operating System is booted, you may be prompted to set the time-of-day clock. This is normally required when you first get the system, when you have reset Nonvolatile Random Access Memory (NVRAM) with the floppy key, or when you have run the time-of-day clock interactive diagnostic test (system board diagnostic phase 22). The clock is also set using the **date** command when you are in the operating system. You must be logged-in as **root** to set the clock using the **date** command. The Time Zone (TZ variable) can be changed using the **timezone** Simple Administration command.

The following command line entries and system responses show the setting of the time-of-day-clock using Simple Administration.

```
# sysadm datetime(CR>

Running Subcommand 'datetime' from menu 'sysetup',
SYSTEM SETUP


Current time and time zone is: 04:59 EDT
Change the time zone? [y, n, q, ?] y<CR>
Current date and time:  Tue. 08/28/84 05:00
Change the date and time: [y, n, q, ?] y<CR>
Month   default 08      (1-12): <CR>
Day     default 28      (1-31): <CR>
Year    default 84      (70-99): <CR>
Hour    default 05      (0-23): <CR>
Minute  default 00      (0-59): 04<CR>
Date and time will be set to:  08/28/84 05:04. OK? [y, n, q] y<CR>
The date and time are now changed.
# date<CR>
Tue Aug 28 05:04:16 EDT 1984
#
```

The following shows the messages associated with setting the time-of-day clock using the **date** command. You must be logged in as **root** to set the clock. When setting the date substantially ahead of the current value, first take the system to the single-user mode. The arguments to the **date** command are in the sequence of month, day, hour, minute, and year.

```
# date 0216131684<CR>
Thu Feb 16 13:16:00 EST 1984
#
```

REBUILDING FILE SYSTEM FREE LIST

General

File system reorganization is necessary to maintain an efficient file system. As files and directories are created and removed, the file system becomes randomly organized. As files are created, the directory files increase in character count when a new information node (i-node) slot is created for a file name. When a file is removed, the slot in the i-node table is marked as free (zero). When a file is created, the operating system searches for a free slot in which to put the file name. If a free slot is found, the file name is placed in it. If a free slot is not found, a new slot is created and the directory increases in size. Reorganization of a directory involves removing all free slots, thereby decreasing the size of the directory file. Another consideration in maintaining an efficient file system is the condition of the free list. Rebuilding the free list improves the accessibility of the data. The free list is rebuilt using the **fsck -s -b** command. The free list is also rebuilt using the **dcopy**(1M) command. When the time and system resources required to use the **dcopy** command to reorganize a file system and rebuild the free list are not available, use the **fsck** command to rebuild the free list. Refer to the "FILE SYSTEM REORGANIZATION" task description for an example of a complete file system reorganization.

Sample Free List Rebuild

The following shows the typical command line entries and system responses associated with checking the **/usr** and **root** (/) file systems and rebuilding the free lists. The system is taken to the single-user mode (run-level S) at the start of the example. In this mode, the **/usr** file system is unmounted.

```
# shutdown -y -il<CR>

Shutdown started.

     Series of messages
     ending with the following.

INIT: SINGLE USER MODE
# fsck -s -b /dev/dsk/c1d0s2 /dev/dsk/c1d0s0<CR>

  /dev/dsk/c1d0s2
  File System: usr Volume: 2032

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List (Ignored)
  ** Phase 6 - Salvage Free List
  1833 files 22400 blocks 20742 free
  ***** FILE SYSTEM WAS MODIFIED *****


  /dev/dsk/c1d0s0
  File System: root Volume: 2032

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List (Ignored)
  ** Phase 6 - Salvage Free List
  507 files 10158 blocks 2154 free

  *** ROOT FILE SYSTEM WAS MODIFIED ***
  *** SYSTEM WILL REBOOT AUTOMATICALLY ***


SELF-CHECK

DIAGNOSTICS  PASSED
```

Continued

Continued from the previous screen

```
UNIX System V Release 2.0.3 3B2 Version 2
unix
Copyright (c) 1984 AT&T.
All Rights Reserved

        fsstat: root file system needs checking
        The root file system (dev/dsk/cld0s0) is being checked automatically.

  /dev/dsk/cld0s0
  File System: root Volume: 2032

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List (Ignored)
  ** Phase 6 - Salvage Free List
  507 files 10158 blocks 2154 free
The system is coming up.  Please wait.
AT&T 3B2 SYSTEM CONFIGURATION:

Memory size: 2 Megabytes
System Peripherals:

         Device Name         Subdevices

         SBD
                         Floppy Disk
                         30 Megabyte Disk
                         30 Megabyte Disk
         PORTS
         PORTS
         CTC
The system is ready.

Console Login:
```

FILE SYSTEM REORGANIZATION

General

File system data blocks become randomly organized as files and directories are created and removed. File system reorganization is necessary to maintain an efficient file system. The optimum disk access performance is achieved when the data blocks of a file are sequential. File system reorganization involves compressing the directories by removing empty entries and spacing consecutive blocks of each file by the optimal rotational gap (9). The **/etc/dcopy**(1M) command is used to copy and reorganize an unmounted file system. The **dcopy** command also provides options to control the placement of files and subdirectories within the file system. The **dcopy** command produces the best results when the source file system is copied from the raw device partition to a block device of another device partition. The destination partition must be large enough to accommodate the file system being reorganized. File system reorganization using the **dcopy** command is only feasible when another hard disk partition is used as the intermediate storage device. Therefore, file system reorganization must be considered when partitioning the hard disk(s). When another hard disk partition cannot be made available, the Simple Administration file system compression approach may be used to improve the disk access performance of a user-type file system (See Note.).

> ***Note:*** The Simple Administration compress function (**sysadm compress**) calls the **/etc/cmpress** shell script. The **/etc/cmpress** shell script only removes empty entries and DOES NOT reorganize the placement of directories and files in the file system. File system compression using the **/etc/cmpress** shell script uses cartridge tape as the intermediate storage device. The compress function is used only on user-type file systems; the root and **/usr** file systems cannot be compressed.

A copy of a file system made using the **dcopy** command must be checked using the **/etc/fsck**(1M) command before the file system can be mounted as a read/write file system. Executing **fsck** sets the status of the file system to OK thus enabling the file system to be mounted as a read/write file system. The source file system (partition) should be checked before executing the **dcopy** command. The reorganized file system is copied back to the original device partition using the **/etc/volcopy**(1M) command and again checked.

Recommended File System Reorganization Strategy

The following paragraphs describe the recommended approach to file system reorganization. As with the recommended file system backup strategy, the user-type file systems are the major concern.

Root File System

The root (/) file system does not require reorganization under normal circumstances. The only time the root file system may require reorganization is before the installation of software utilities. Reorganization before software installation guarantees sequential allocation of blocks for each file added to the file system. The best file system performance is achieved when the data blocks for a file are sequential blocks in the file system. The recommended method of reorganizing the root file system is to do a full restore of the operating system. Reloading the software utilities and the user-type file systems from backups at the end of the full restore procedure results in the reorganization of all file systems.

Usr File System

As described in "ALLOCATING SYSTEM RESOURCES" task, it is recommended that the **/usr** file system contain only software utilities. As previously described, reorganization of the root file system also reorganizes the rest of the file systems. User login directories (user-type data) should be in another file system or systems, separate from the root and **/usr** file systems.

User-type File Systems

The file systems used for login directories are the prime candidates for frequent reorganization. How often reorganization needs to be done for the user-type file systems is application dependent. As part of the strategy for allocating system resources, several small disk partitions should be allocated to facilitate file system reorganization.

General File System Reorganization Procedure

The major steps necessary to reorganize a file system are as follows.

1. At the console, take the system to the single-user mode.
2. Unmount all file systems (except for root) using the **/etc/umountall**(1M) command.
3. Check the file system to be reorganized using the **/etc/fsck**(1M) command.
4. Reorganize the target (source) file system using the **/etc/dcopy**(1M) command. The source disk partition is "dcopied" to a destination disk partition of appropriate size to hold the file system.
5. Check the reorganized file system on the destination disk partition using the **/etc/fsck**(1M) command.
6. Volume copy the reorganized file system back to the original disk partition using the **/etc/volcopy**(1M) command.
7. Check the file system on the original disk partition using the **/etc/fsck**(1M) command.
8. Return the system to multi-user mode by using the **/etc/init**(1M) command.

Sample File System Reorganization

Using Hard Disk Intermediate Storage

The following command line entries and system responses show the use of the **dcopy** command to reorganize a file system maintained on hard disk. In this example the intermediate hard disk partition (partition c1d1sa) could also be used as a backup to the file system on partition c1d1s8. The file system on partition c1d1s8 of the second hard disk is copied to partition c1d1sa on the same hard disk using the **dcopy** command. The default options of the **dcopy** command are used; only the verbose (**-v**) option is specified in the example. Disk partitions c1d1s8 and c1d1sa are equally sized partitions. After copying partition c1d1s8 to partition c1d1sa, **fsck** is run on partition c1d1sa. Partition c1d1sa is then copied back to partition c1d1s8 using the **volcopy** command.

At the start of this example, the system is in run-level S (single user) with partitions c1d1s8 and c1d1sa of the second hard disk unmounted.

```
# fsck -D /dev/rdsk/c1d1s8<CR>

  /dev/rdsk/c1d1s8
  File System: usr2 Volume: 2032

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  111 files 874 blocks 14480 free
# dcopy -v /dev/rdsk/c1d1s8 /dev/dsk/c1d1sa<CR>
From: /dev/rdsk/c1d1s8, to: /dev/dsk/c1d1sa? (DEL if wrong)
old filesize = 7800, old inode size = 123
old stepsize = 9, old cylinder size = 90
new filesize = 7800, new inode size = 123
new stepsize = 9, new cylinder size = 90
Available mem 466944, got 63488 for inodes (that's 992 inodes)
Pass 1: Reorganizing file system
Pass 2: Fixing inums in directories
Pass 3: Remake freelist
Files:          111
Free blocks in:         7240
Free blocks out:        7240
Complete
# fsck -D /dev/rdsk/c1d1sa<CR>

  /dev/rdsk/c1d1sa
  File System: usr2 Volume: 2032

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  FILE SYSTEM STATE SET TO OKAY
  111 files 874 blocks 14480 free
  *** FILE SYSTEM WAS MODIFIED ***
```

Continued

Continued from previous screen

```
# volcopy usr2 /dev/rdsk/c1d1sa  - /dev/rdsk/c1d1s8 -<CR>
/dev/rdsk/c1d1s8 less than 48 hours older than /dev/rdsk/c1d1sa
To filesystem dated:  Sat Feb  9 05:57:17 1985
Type 'y' to override:      y<CR>
From: /dev/rdsk/c1d1sa, to: /dev/rdsk/c1d1s8? (DEL if wrong)
  END: 15600 blocks.
volcopy: cannot access /etc/log/filesave.log
#
# fsck -D /dev/rdsk/c1d1s8<CR>

  /dev/rdsk/c1d1s8
  File System: usr2 Volume: 2032

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  111 files 874 blocks 14480 free
#
```

Using Cartridge Tape Intermediate Storage

An example of *dcopying* to a cartridge tape is not provided. The use of cartridge tape as an intermediate storage device for the reorganization of a file system requires that the cartridge tape be used as a block device. Because of the time required to **dcopy** to a cartridge tape and then copy the information back to the hard disk, it is more efficient to use a hard disk partition as the intermediate storage device. This is true even if you must first make a hard disk partition available for use by copying the existing data to tape using **/etc/ctccpio**. For these reasons, it is recommended that cartridge tape NOT BE used as an intermediate storage device for file system reorganization using the **dcopy** command.

MONITORING DISK SPACE

The amount of free disk space should be checked on a regular basis. How often you check the disk storage availability depends on the rate at which the free disk storage space is consumed. A heavily used system must be checked more frequently than a system with only a few users. The habits of the user population also affect how often the free space should be checked. As free space decreases, the need for close monitoring increases.

As new files are created and existing files grow, the number of available data blocks and information nodes (i-nodes) decrease. Administratively, both the number of free disk blocks and the number of free i-nodes can be a problem. When the number of free blocks in a file system is less than 1430 blocks, the entire contents of a floppy disk cannot be transferred to that file system. This free space requirement includes 1422 blocks for the contents of the floppy disk plus free space to create a temporary file. When the free i-node count falls below 100, the operating system spends most of its time rebuilding the free i-node array. When a file system runs out of space, the operating system prints "no-space" messages and does little else. One of the more important system administration tasks is to monitor the system free disk space. At the start of each day, the free disk space for the file systems that are normally mounted (root and usr) should be examined. The **/usr** file system is where most expansion takes place.

The following shows a 30-megabyte disk system with several optional utilities installed. As indicated by the large amount of **/usr** free space, the system is relatively new, with little user activity.

```
# df -t / /usr<CR>
/usr       (/dev/dsk/c1d0s2):    20736 blocks    3637 i-nodes
                   total:    43830 blocks    5472 i-nodes
/          (/dev/dsk/c1d0s0):     1660 blocks    1041 i-nodes
                   total:    12510 blocks    1552 i-nodes
#
```

A comparison between the number of free blocks and free i-nodes versus the number of allocated blocks and i-nodes shows that no free space problem exists on this system. For the **/usr** file system, the number of free blocks and i-nodes are well within the total number of blocks and i-nodes that have been allocated to the file system.

When the space used for a file system approaches 90 percent of the allocated space, action must be taken to clean up the file system. Users should be informed of diminishing system resources and requested to clean up their directories. If after such a cleanup, sufficient space is still a problem, utilities and data files that have not been accessed (used) for some time should be removed from the system. A **find** such as follows can be used to identify files that have not been accessed for some time interval. In this example, **/usr** files with a group identification of **other** that have not been accessed for 30 days are output.

```
# find / -atime +30 -group 'other' -print<CR>
/usr/adm/sulog
/usr/hrp/.profile
/usr/hrp/305-330/ctlpgm.c
/usr/hrp/305-330/msgget
/usr/hrp/305-330/msgctl
/usr/hrp/305-330/getpgm.c
/usr/hrp/305-330/clean
/usr/hrp/305-330/set.c
/usr/hrp/305-330/getpgm
/usr/hrp/305-330/ctlpgm
/usr/eb/.profile
/usr/eb/berens/3b2plan
/usr/eb/berens/tools
/usr/eb/berens/jeb
/usr/eb/berens/rfpex
#
```

Based on the output of the **find** command, mail should be sent to users **hrp** and **eb** to clean up their files. Given that the users do not accept the responsibility of cleaning up their own files, the system administrator may need to copy certain data to floppy disk and remove the files from the hard disk. See "CREATING AND IDENTIFYING FILE SYSTEMS ON FLOPPY DISK" and "COPYING/MOVING DIRECTORIES" in this chapter.

When the amount of free disk space drops to about ten percent of the total number of blocks allocated to the file system, action must be taken to "clean up" the file system. If no action is taken in these circumstances, the file system will run out of space. The following are the traditional corrective actions.

1. Post a message-of-the-day or news telling the user population to clean up their file space or else ... run out of space.

2. Clean up system log files that increase in size as the system operates. Refer to Chapter 2, "ADMINISTRATIVE DIRECTORIES AND FILES," for information on these files that increase in size as the system is used. Note that **/etc/cron**, **/etc/rc0**, or **/etc/rc2** can be used to clean up log files on a periodic basis. To use one of these functions to limit the size of a log file, add the appropriate command line to the **/usr/spool/cron/crontab/root**, **/etc/shutdown.d/...**, or **/etc/rc.d/...** file.

3. Copy data files that have not been accessed for some time to a backup media (such as floppy disks) and remove the files. Notify affected users of the action taken.

4. Determine if any optional utilities should be removed to provide additional free space. Notify the user community of the removal of any utilities.

COPYING/MOVING DIRECTORIES

General

Copying entire directory structures is done using the **find** and **cpio** commands to copy (pass) information from one structure to another structure. The destination directory must already exist. When the selected information is copied to the new structure, the old structure (source directory) is deleted if the task is to move the data. The **rm –rf** *directory* silently removes an entire directory structure. Also see the **/etc/mvdir**(1M) command description in Chapter 6, "SYSTEM ADMINISTRATION COMMANDS."

Copy/Move Directories Procedure

The following are the major steps necessary to copy/move a directory structure.

1. If the destination directory DOES NOT already exist, make the destination directory using the **mkdir** command.

2. Change directory [**cd**(1) command] to the directory structure to be copied. Enter the following command line to duplicate all files and directories of the current directory in the destination directory. You must be **root** to use the **m** option of the **cpio**(1) command.

   find . -print | cpio -pdmuv *destination*

3. If the task is to move the structure, delete the current directory information. The **rm -rf** *source* where *source* is the path name of the directory that was copied, removes the directory.

4. Check that the owner and group names are correct for destination directory and the information copied. Also check that the access permissions are correct. Use the **chown**, **chgrp**, and **chmod** commands to establish the desired names and access permissions.

The following shows the command line entries and system responses associated with copying a directory structure **/usr/rar/305-323** to an existing directory **/rar**. The **/rar** is a file system on the floppy disk (**/dev/SA/diskette1**). The directory structures are then listed using the **ls -l** command to compare the contents of the directories. Note that other options can be used with the **cpio -p** command.

```
# cd /usr/rar/305-323<CR>
# find . -print | cpio -pdmuv /rar<CR>
/rar/ch6.package
/rar/disk.stats
/rar/trademarks
/rar/ch1.general
/rar/ch2.install
/rar/ch3.files
/rar/ch4.functions
/rar/ch5.fsck
/rar/cmds
/rar/appendix.d
/rar/chx.advice
/rar/toc
/rar/runstates
/rar/appendix.a
/rar/appendix.b
/rar/appendix.c
449 blocks
#
```

Continued

Continued from previous screen

```
# ls -l /usr/rar/305-323 /rar<CR>

/rar:
total 468
-rw-rw-rw-   1 rar      rar         1241 Mar 21 06:17 appendix.a
-rw-rw-rw-   1 rar      rar         1890 Mar 23 10:30 appendix.b
-rw-rw-rw-   1 rar      rar         1089 Mar 23 13:53 appendix.c
-rw-rw-rw-   1 rar      rar          503 Mar 21 06:17 appendix.d
-rw-rw-rw-   1 rar      rar         3127 Mar 22 05:51 ch1.general
-rw-rw-rw-   1 rar      rar         8140 Mar 22 06:00 ch2.install
-rw-rw-rw-   1 rar      rar        24670 Mar 23 13:52 ch3.files
-rw-rw-rw-   1 rar      rar        72215 Mar 24 09:03 ch4.functions
-rw-rw-rw-   1 rar      rar        45723 Mar 21 06:17 ch5.fsck
-rw-rw-rw-   1 rar      rar        47594 Mar 24 09:15 ch6.package
-rw-rw-rw-   1 rar      rar        15436 Mar 21 06:17 chx.advice
-rw-r--r--   1 rar      rar          267 Mar 21 06:17 cmds
-rw-r--r--   1 rar      rar         1489 Mar 21 10:00 disk.stats
-rw-rw-rw-   1 rar      rar          997 Mar 21 06:17 ff
-rw-rw-rw-   1 rar      rar         2658 Mar 21 06:16 fuser
-rw-rw-rw-   1 rar      rar         1128 Mar 21 06:17 mkfs
-rw-r--r--   1 rar      rar          770 Mar 22 06:24 runstates
-rw-rw-rw-   1 rar      rar          718 Mar 22 05:53 toc
-rw-rw-rw-   1 rar      rar          720 Mar 21 06:16 trademarks
```

Continued

Continued from previous screen

```
/usr/rar/305-323:
total 457
-rw-rw-rw-   1 rar      rar          1241 Mar 21 06:17 appendix.a
-rw-rw-rw-   1 rar      rar          1890 Mar 23 10:30 appendix.b
-rw-rw-rw-   1 rar      rar          1089 Mar 23 13:53 appendix.c
-rw-rw-rw-   1 rar      rar           503 Mar 21 06:17 appendix.d
-rw-rw-rw-   1 rar      rar          3127 Mar 22 05:51 ch1.general
-rw-rw-rw-   1 rar      rar          8140 Mar 22 06:00 ch2.install
-rw-rw-rw-   1 rar      rar         24670 Mar 23 13:52 ch3.files
-rw-rw-rw-   1 rar      rar         72215 Mar 24 09:03 ch4.functions
-rw-rw-rw-   1 rar      rar         45723 Mar 21 06:17 ch5.fsck
-rw-rw-rw-   1 rar      rar         47594 Mar 24 09:15 ch6.package
-rw-rw-rw-   1 rar      rar         15436 Mar 21 06:17 chx.advice
-rw-r--r--   1 rar      rar           267 Mar 21 06:17 cmds
-rw-r--r--   1 rar      rar          1489 Mar 21 10:00 disk.stats
-rw-r--r--   1 rar      rar           770 Mar 22 06:24 runstates
-rw-rw-rw-   1 rar      rar           718 Mar 22 05:53 toc
-rw-rw-rw-   1 rar      rar           720 Mar 21 06:16 trademarks
#
```

INSTALLING AND REMOVING UTILITIES

The general procedures and special instructions for installing and removing the various software utilities are documented in the *AT&T 3B2 Computer Release Notes*.

The installation and removal of software and hardware are system administration tasks. The removal of one or more utilities is one of the actions that can be taken to free-up some disk space. Determining which utilities to remove requires a survey of the functions needed by the user population. This should only be done after the user population has cleaned up their file space and you have cleaned up the system files. Refer to Chapter 2, "ADMINISTRATIVE DIRECTORIES AND FILES," for information on some of the system files that increase in size as the system operates.

DETERMINING SYSTEM STATUS AFTER TROUBLE

General

Following panic error messages (See Appendix C "ERROR MESSAGES."), the current state of the system should be recorded. Several tools are available for recording and analyzing system status. The tools used depend on the operating state of the system following the trouble.

The **/etc/errdump** command dumps (displays) an error history file. You must be running the UNIX Operating System to use this command. If the system cannot boot the operating system, the following tools are used to determine the status of the system.

- Diagnostics (diagnostic monitor program [**dgmon**])
- System dump (**sysdump**).

The root file system (/) must be accessible (undamaged) to run diagnostics from the hard disk. How to run diagnostics is covered elsewhere in this chapter. The **sysdump** command executes from firmware level and depends on the integrity of random access memory. The **sysdump** program is loaded from the operating system. A system dump is analyzed using **/etc/crash**. The floppy disks generated by **sysdump** on one system can be analyzed on another system, providing that the **/unix** is copied to the other machine. When you reboot the system, be sure to boot (execute) **/unix**. If you boot (execute) **/etc/system**, the resulting **/unix** that is generated may not match the previous version. The **/etc/ldsysdump** command is used to combine several system dump floppy disks into one file on hard disk. The following paragraphs address the **errdump** and **sysdump** commands. Refer to the *AT&T 3B2 Computer Crash Analysis Guide* for more information.

System Dump (sysdump)

The **sysdump** program is executed from the firmware level. The program writes the system image to one or more floppy disks. Three formatted floppy disks are needed to dump the contents of a 2-megabyte memory. Six formatted floppy disks are needed to dump the contents of a 4-megabyte memory. Existing floppy disks may be used; however, the contents of the floppy disks are overwritten. The time required to dump the system image is about 5 to 6 minutes per floppy disk.

If a system dumb is to be executed following a crash, the firmware mode must be entered and the **sysdump** program executed as the first action taken. Rebooting the system or executing another program will overwrite the system image, making subsequent crash analysis a futile effort.

The following command line entries and system responses show the execution of the **sysdump** on a system with 2 megabytes of memory.

```
FIRMWARE MODE
<mcp><CR>

Enter name of program to execute [ ]: sysdump<CR>

Do you want to dump the system image to the floppy diskette?
Enter 'c' to continue, 'q' to quit: c<CR>

Insert first sysdump floppy.
Enter 'c' to continue, 'q' to quit: c<CR>

Dumping mainstore
................................................................................
.............................................................
If you wish to dump more of mainstore,
insert new floppy.

Enter 'c' to continue, 'q' to quit: c<CR>

Dumping more main store
................................................................................
.............................................................
If you wish to dump more of mainstore,
insert new floppy.

Enter 'c' to continue, 'q' to quit: c<CR>

Dumping more main store
................................................................................
.........................................
Dump completed.
three floppies written

Returning to firmware

SELF-CHECK

FIRMWARE MODE
```

Error Dump (errdump)

The following command line entry and system responses show a typical error dump. Note that the last five panic error messages are displayed at the end of the error dump output.

```
# errdump<CR>
nvram status:   sane

csr:    0x0258  (unassigned) (clock) (pir9) (uart)

psw:    rsvd CSH_F_D QIE CSH_D OE NZVC TE IPL CM PM R I ISC TM FT
(hex)     0       0   0     0  0    0  0   f  0  0 1 0   5  0  3

r3:     0xffffffff
r4:     0x400d554c
r5:     0x866f6300
r6:     0xc002001e
r7:     0x0021413f
r8:     0x866f62cc
oap:    0x400806e0
opc:    0x40010d3f
osp:    0x40080708
ofp:    0x40080708
isp:    0x40080004
pcbp:   0x4004la9c

fltar:  0x866f62cc
fltcr:  reqacc  xlevel  ftype
        0xb     0x0     0x3

        srama           sramb
[0]     0x02083000      0x0000011f
[1]     0x02083900      0x00000030
[2]     0x0209b860      0x00000074
[3]     0x0209bc00      0x00000015

        Panic log

[0]     Thu Aug 23 07:37:11 1984
        KERNEL MMU FAULT (F_SDTLEN)


[1]     Sun Aug 19 15:43:59 1984
        SYSTEM BUS TIME OUT INTERRUPT

[2]     Wed Aug  8 15:47:50 1984
        KERNEL BUS TIMEOUT

[3]     Wed Dec 31 18:59:59 1969
         D,Lx TV

[4]     (0xffffffff,0xffffffff,0xffffffff,0xffffffff)

#
```

FILE SYSTEM CHECKING AND REPAIR

The ability to check a file system and repair any damage is described in Chapter 5, "FILE SYSTEM CHECKING AND REPAIR." The importance of file system repair is inversely proportional to how often file system backups are done. The greater the time interval between file system backups, the larger is the amount of data that can be lost. When the system crashes and file damage occurs, the only way to recover the damaged data that has not been backed-up is to attempt to repair the file system.

Two commands are provided for the checking and repairing of a file system: **fsck** and **fsdb**. The **fsck** program provides the ability to check a file system and do a limited amount of repair. The **fsdb** program is a file system debugger that provides a more in-depth means of analyzing and patching a damaged file system. A high level of system expertise is necessary to recover a damaged file system using **fsdb**.

RELOADING UNIX OPERATING SYSTEM

General

The **UNIX** Operating System is reloaded from the five **3B2 Computer Core System** floppy disks. Two levels of restoral are provided: the full and partial restore.

The full restore erases all data on the target hard disk(s) and redefines the hard disk partitions. Either your own partition values or default partition values may be specified as part of the full restore procedure. If the previous user files and software utilities are to be part of the system, these files must be either restored from file system backups or rebuilt using your own procedure. Refer to the *AT&T 3B2 Computer Release Notes* for information on the installation of the various utilities.

The partial restore overwrites only the core system files. User files and previously loaded software utilities are not affected. For a partial restore, only the input/output terminal configuration, system tunable parameters, and the password file must be reconstructed. The files associated with the previous system configuration are saved in the **/usr/old** directory by the partial restore process.

Before you begin reloading the UNIX Operating System, make sure that you know what you need to do. The major items needed to reload the operating system are:

- The five UNIX System V Core floppy disks
- The Maintenance Control Program (MCP) password
- Any backup data that has to be reloaded.

Read through the procedure for reloading the operating system before you do the procedure. If you understand all steps, reload the operating system. If you do not understand the reload procedure, contact your

service representative for help. If you are in doubt as to which procedure to do, start with the partial restore procedure. If the partial restore cannot be completed or completes but does not solve the problem, do the full restore.

Partial Restore Procedure

Caution 1: ***The first*** *3B2 Computer Core System* ***floppy disk must be writable. If write protected, remove the write protect tab before using the floppy disk. The restore process will fail if the first floppy disk is write protected.***

Caution 2: ***The partial restore procedure saves the previous system configuration in the*** */usr/old* ***directory. The files in this directory are used to make the job of restoring the system configuration easier. Care must be used when restoring the previous configuration from the*** */usr/old* ***information if the reason for doing the partial restore was bad data in one of these system files. Hardware and software drivers may have to be reloaded (reinstalled). Examine the*** */boot* ***directory (if valid) for a list of drivers.***

The "user" files are not affected by a partial restoral. Certain system files are overwritten. For example, the terminal configuration (**/etc/inittab**) and password (**/etc/passwd**) files must be rebuilt to support your previous system configuration. The **adm**, **root**, and **sys** crontab files are overwritten. The **at.allow** and **cron.allow** files are overwritten. A partial restoral replaces (overwrites) the core system files on the hard disk with those originally distributed. These files include the Essential Utilities. Certain system files are saved in the **/usr/old** directory by the partial restore process. The files in the **/usr/old** directory are used to make the job of restoring the system configuration easier. Care must be used when restoring the previous configuration from the **/usr/old** information if the reason for doing the partial restore was bad data in one of these system files.

The following steps are for a PARTIAL restoral of the system.

1. If the operating system is running, at the console terminal log in as root and take the system to the firmware mode (run-level 5). If you cannot log in, depress RESET to get to the firmware mode.

2. Insert the first **3B2 Computer Core System** floppy disk into the integral floppy disk drive. See Caution 1.

3. Execute (boot) the operating system (**unix**) from the integral floppy disk (**0**).

4. Select the Partial Restore option (option number 2).

5. Follow the displayed instructions to remove and insert the **3B2 Computer Core System** floppy disks. When the last **3B2 Computer Core System** floppy disk has been loaded, the operating system restarts from the hard disk.

   Note: Since the new operating system is using a default equipped device table, certain optional hardware device equipage is not recognized by the new operating system. This results in "unknown device" error messages when the operating system restarts from the hard disk.

6. When the system is ready, rebuild the various system files from either backup or from the **/usr/old** directory. Install utilities as required to restore drivers. Refer to the *AT&T 3B2 Computer Release Notes* for information on the installation of the various utilities. Use caution when restoring system files from the **/usr/old** directory. See Caution 2.

7. Refer to the "SYSTEM RECONFIGURATION" task description in this chapter for information on changing tunable system parameters.

The following command line entries and system responses show a typical partial restore procedure. In this example, system is shut down to the firmware mode (run-level 5) and a partial restore started. At the end of this sample procedure, the system must be set up and the applicable system files restored.

```
# shutdown -y -g0 -i5<CR>

Shutdown started.    Wed Apr  3 07:41:36 EST 1985

Broadcast message from root (console) Wed Apr  3 07:42:16 EST 1985
THE SYSTEM IS BEING SHUT DONW NOW ! ! !
Log off now or risk your files being damaged.


#
INIT: New run level: 5
The system is coming down.  Please wait.
System services are now being stopped.

The system is down.


SELF-CHECK

FIRMWARE MODE
<mcp><CR>
```

Continued

Continued from the previous screen

Insert the first 3B2 Computer Core System floppy disk into the integral floppy disk drive.

```
Enter name of program to execute [ ]: unix<CR>
     Possible load devices are:

Option Number   Slot     Name
-------------------------------------
       0          0     FD5
       1          0     HD30
       2          0     HD30
       3          3     CTC

Enter Load Device Option Number [1 (HD30)]: 0<CR>


UNIX System V Release 2.0.3 3B2 Version 2
unix
Copyright (c) 1984 AT&T.
All Rights Reserved

3B2 Version 2 Installation

        1  Full Restore
        2  Partial Restore
        3  Dual-Disk Upgrade
        4  Release Upgrade

When responding to a question, you may use the "backspace" key
to erase the last character  typed or the "@" key to erase the
entire line. Enter "help" for additional information.

Selection? [ 1 2 3 4 quit help ] 2<CR>

-- Partial Restore --

The "Partial Restore" replaces the core files with 3B2 Version 2 files on
the integral hard disk(s).  Other files are not affected.  This will UNDO
your terminal and login configuration.

Continue? [ y n help ] y<CR>
```

Continued

Continued from the previous screen

```
Checking the hard disk filesystems.

  /dev/rdsk/c1d0s0
  File System: root Volume: 2032

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  604 files 11040 blocks 3132 free

  /dev/rdsk/c1d1s2
  File System: usr Volume: 2032

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  1984 files 24762 blocks 16608 free

  /dev/rdsk/c1d0s8
  File System: usr Volume: 2032

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  600 files 10696 blocks 9766 free

  /dev/rdsk/c1d0s9
  File System: usr3 Volume: 2032

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  427 files 10322 blocks 10140 free

Saving system configuration files in /usr/old.

Installing the initial core system files. This should take
no more than ten minutes.
1246 blocks
1 blocks
```

Continued

Continued from the previous screen

```
The system is restarting itself from the hard disk. This should
take no more than five minutes. The installation procedure
will then continue automatically.
```

The following message is caused by the use of a default equipped device table that is installed by the partial restore process. The unknown device in this example is the Cartridge Tape Controller card.

```
SELF-CHECK
UNKNOWN ID CODE 0x5 FOR DEVICE IN SLOT 3
EQUIPPED DEVICE TABLE COMPLETION WILL CONTINUE.
CHECK EDT.


UNIX System V Release 2.0.3 3B2 Version 2
unix
Copyright (c) 1984 AT&T.
All Rights Reserved

Checking the hard disk file systems.

  /dev/dsk/c1d0s0
  File System: root Volume: 2032

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  656 files 11044 blocks 3128 free

  /dev/dsk/c1d1s2
  File System: usr Volume: 2032

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  2021 files 24932 blocks 16438 free

  /dev/dsk/c1d0s8
  File System: usr2 Volume: 2032

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  600 files 10696 blocks 9766 free
```

Continued

```
Continued from the previous screen
  /dev/dsk/c1d0s9
  File System: usr3 Volume: 2032

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  427 files 10322 blocks 10140 free


Please insert the 3B2 Core System floppy number 2.

Type "go" when ready [go quit help ] go<CR>

Installing additional core system files. This should take
no more than ten minutes.
1233 blocks

Please insert the 3B2 Core System floppy number 3.

Type "go" when ready [ go quit help ] go<CR>

Installing additional core system files. This should take
no more than ten minutes.
1281 blocks

Please insert the 3B2 Core System floppy number 4.

Type "go" when ready [ go quit help ] go<CR>

Installing additional core system files. This should take
no more than ten minutes.
1078 blocks

Please insert the 3B2 Core System floppy number 5.

Type "go" when ready [ go quit help ] go<CR>

Installing additional core system files. This should take
no more than ten minutes.
1146 blocks

You may now remove the last 3B2 Core System floppy.
83 blocks

See /usr/old/README regarding changes to system configuration files.

Installation is now complete. The system is restarting itself from
the hard disk. It will be ready to use when you receive the "Console
Login" prompt. This should take no more than five minutes.
```

Continued

Continued from the previous screen

The following message is caused by the use of a default equipped device table that is installed by the partial restore process. The unknown device in this example is the Cartridge Tape Controller card.

```
SELF-CHECK
UNKNOWN ID CODE 0x5 FOR DEVICE IN SLOT 3
EQUIPPED DEVICE TABLE COMPLETION WILL CONTINUE.
CHECK EDT.


DIAGNOSTICS  PASSED

Driver not found for *VOID* device (board code 3)

UNIX System V Release 2.0.3 3B2 Version 2
unix
Copyright (c) 1984 AT&T.
All Rights Reserved
The system is coming up.  Please wait.
Generating a new /unix
Error 6: Can't open /dev/rdsk/c4d0s6
Error 6: Can't open /dev/rdsk/c4d0s6
Error 6: Can't open /dev/rdsk/c4d0s6
Error 6: Can't open /dev/rdsk/c4d0s6
Error 6: Can't open /dev/rdsk/c4d0s6
ct: CTC firmware pump failed on 5 successive attempts
ct: CTC initialization fialed on CTC Major 3
error: cannot open /dev/prf
cannot open /dev/prf
AT&T 3B2 SYSTEM CONFIGURATION:

Memory size: 2 Megabytes
System Peripherals:

        Device Name         Subdevices

        SBD
                        Floppy Disk
                        30 Megabyte Disk
                        30 Megabyte Disk
        PORTS
        PORTS
        *VOID*
The system is ready.

Console Login:
```

Full Restore Procedure

> ***Caution:*** ***The first*** *3B2 Computer Core System* ***floppy disk must be writable. If write protected, remove the write protect tab before using the floppy disk. The restore process will fail if the first floppy disk is write protected.***

The following steps are for a FULL restoral of the system. A full restoral erases everything on the integral hard disk and then loads the core system files. These files include the Essential Utilities. Refer to the *AT&T 3B2 Computer Release Notes* for information on the installation of the various utilities.

1. If the operating system is running, at the console terminal log in as root and take the system to the firmware mode (run-level 5). If you cannot log in, depress RESET to get to the firmware mode.

2. Insert the first **3B2 Computer Core System** floppy disk into the integral floppy disk drive. See Caution.

3. Boot the operating system (**unix**) from the integral floppy disk (**0**).

4. Select the Full Restore (option number 1).

5. If you are defining your own disk partition values, refer to the "ALLOCATING SYSTEM RESOURCES" task description for information on disk partitioning.

6. Follow the displayed instructions to remove and insert the **3B2 Computer Core System** floppy disks. When the last **3B2 Computer Core System** floppy disk has been loaded, the system will restart from the hard disk.

7. When the system is ready, either follow the displayed instructions to do the Simple Administration setup procedure, or rebuild the various system files from backup using your own procedure. Install utilities as necessary. Refer to the *AT&T 3B2 Computer Release Notes* for information on the installation of the various utilities.

8. Refer to the "SYSTEM RECONFIGURATION" task description for information on changing tunable system parameters.

The following command line entries and system responses show a typical full restore procedure for a system equipped with two 30-megabyte hard disks. Default partitioning is used in the example by answering "yes" to the "use default partitioning" question. The system is in the firmware mode at the start of the example. At the end of this sample procedure, the system must be set up to support a particular application (utilities must be loaded and logins defined).

```
Enter name of program to execute [ ]: unix<CR>
     Possible load devices are:

Option Number   Slot     Name
---------------------------------------
       0          0     FD5
       1          0     HD30
       2          0     HD30
       3          3     CTC

Enter Load Device Option Number [1 (HD30)]: 0<CR>


UNIX System V Release 2.0.3 3B2 Version 2
unix
Copyright (c) 1984 AT&T.
All Rights Reserved


3B2 Version 2 Installation

        1  Full Restore
        2  Partial Restore
        3  Dual-Disk Upgrade
        4  Release Upgrade

When responding to a question, you may use the "backspace" key
to erase the last character  typed or the "@" key to erase the
entire line. Enter "help" for additional information.

Selection? [ 1 2 3 4 quit help ] 1<CR>

-- Full Restore --

The "Full Restore" will destroy EVERYTHING on both hard disks and install a
3B2 Version 2 core UNIX system.

Continue? [ y n help ] y<CR>
```

Continued

Continued from the previous screen

```
Use the default hard disk partitioning? [y n help ] y<CR>

Setting up the initial hard disk with default partition sizes; this should
take no more than twenty-five minutes.

Installing the initial core system files. This should take no more than
seven minutes.
1246 blocks
1 blocks


The system is restarting itself from the hard disk. This should
take no more than five minutes. The installation procedure will
then continue automatically.
```

The following message is caused by the use of the default equipped device table that is installed by the full restore process. The unknown device in this example is the Cartridge Tape Controller card.

```
SELF-CHECK
UNKNOWN ID CODE 0x5 FOR DEVICE IN SLOT 3
EQUIPPED DEVICE TABLE COMPLETION WILL CONTINUE.
CHECK EDT.
```

Continued

Continued from the previous screen

```
UNIX System V Release 2.0.3 3B2 Version 2
unix
Copyright (c) 1984 AT&T.
All Rights Reserved

Checking the hard disk file systems.

  /dev/dsk/c1d0s0
  File System: root Volume: 2032

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  201 files 1376 blocks 10936 free

  /dev/dsk/c1d0s8
  File System: usr2 Volume: 2032

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  3 files 4 blocks 43226 free

  /dev/dsk/c1d1s2
  File System: usr Volume: 2032

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  3 files 6 blocks 61476 free
```

Continued

Continued from the previous screen

```
Please insert the 3B2 Core System floppy number 2.

Type "go" when ready [go quit help ] go<CR>

Installing additional core system files. This should take no more than
ten minutes.
1233 blocks

Please insert the 3B2 Core System floppy number 3.

Type "go" when ready [ go quit help ] go<CR>

Installing additional core system files. This should take no more than
ten minutes.
1281 blocks

Please insert the 3B2 Core System floppy number 4.

Type "go" when ready [ go quit help ] go<CR>

Installing additional core system files. This should take no more than
ten minutes.
1078 blocks

Please insert the 3B2 Core System floppy number 5.

Type "go" when ready [ go quit help ] go<CR>

Installing additional core system files. This should take no more than
ten minutes.
1146 blocks

You may now remove the last 3B2 Core System floppy.
83 blocks

Installation is now complete. The system is restarting itself from
the hard disk. It will be ready to use when you receive the "Console
Login" prompt. This should take no more than five minutes.


SELF-CHECK
UNKNOWN ID CODE 0x5 FOR DEVICE IN SLOT 3
EQUIPPED DEVICE TABLE COMPLETION WILL CONTINUE.
CHECK EDT.

DIAGNOSTICS  PASSED

Driver not found for *VOID* device (board code 3)
```

Continued

Continued from the previous screen

```
UNIX System V Release 2.0.3 3B2 Version 2
unix
Copyright (c) 1984 AT&T.
All Rights Reserved
The system is coming up.  Please wait.

This machine has not been used as a customer machine yet.  The messages that
follow are from checking the built-in file systems for damage that might have
occurred during shipment.  As long as you do not see either of the messages
                                    BOOT UNIX
or                          FILE SYSTEM WAS MODIFIED
all is well.  If either message does come out, call your service representative.
However, the machine is still usable unless I tell you otherwise.
I will now start checking file systems.

  /dev/dsk/c1d0s8
  File System: usr2 Volume: 2032

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  3 files 4 blocks 43226 free

  /dev/dsk/c1d1s2
  File System: usr Volume: 2032

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  214 files 1552 blocks 59930 free

Generating a new /unix

        Welcome!
This machine has to be set up by you.  When you see the "login" message type
                                    setup
followed by the RETURN key.  This will start a procedure that leads you through
those things that should be done the "first time" the machine is used.

AT&T 3B2 SYSTEM CONFIGURATION:

Memory size: 2 Megabytes
System Peripherals:

        Device Name         Subdevices

        SBD
                            Floppy Disk
                            30 Megabyte Disk
                            30 Megabyte Disk
        PORTS
        PORTS
        *VOID*
The system is ready.

Console Login:
```

FILE SYSTEM BACKUP

General

The importance of establishing and following a file system backup plan is too often not appreciated until data is lost and cannot be recovered. Backing up a file system takes time. Trying to recover lost or damaged data from paper records and best-guess-work takes even more time. The purpose of an effective system backup plan lies in the ability to easily recover lost or damaged data.

The backup procedures described in this guide use the **find** command and variations of cpio-type commands. These commands are used to do all types of backups. For multiple disk systems, the **volcopy** command is used to make a literal copy (complete backup) of a file system to an equal or larger sized partition.

Another method of backing up the information stored on the 3B2 Computer is to copy file systems to another computer system over a high-speed data link. The link between the machines must be a high-speed data link so that data transfers are done in a timely fashion. The other machine should be a larger system with mass storage capability. The AT&T 3B Local Area Network (AT&T 3BNET*) provides the speed necessary to efficiently transfer large amounts of data between systems.

* Trademark of AT&T

There are also Simple Administration procedures provided under the File Management menu for doing "complete" file system backups and "incremental" file system backups. Simple Administration also provides a procedure for restoring backup data to the hard disk.

The backup plan that you use can include any or all of these methods. The important consideration is that you evaluate the need for system backup and form a backup plan.

Planning and Scheduling Backups

There is no recommended plan for doing backups. However, there is a default backup schedule delivered in the Essential Utilities that may be satisfactory for your use. Each application and system configuration has different aspects that must be considered when developing a backup strategy. Personal preference also plays a part in establishing backup procedures. The important issue is that you form a backup plan. The backup plan should be reevaluated as the use and configuration of the machine changes.

There are many things to consider when formulating a backup schedule. Some of the main considerations are:

- The method to be used to restore (recover) lost or damaged data.
- The importance of the stored data.
- The downtime of the machine needed to do backup.
- The use of the system. That is, how the 3B2 Computer is being used to support a particular application.
- The size of the file system to be backed up/restored.
- The type(s) of backup to do.
- The device(s) used to store the backup data. The storage device could be floppy disk(s), hard disk(s), and/or cartridge tape(s).

Restoral/Recovery

One important consideration in formulating a backup plan is the method of recovery for lost or damaged data. The recovery process should be as quick and easy as possible. The speed of recovery depends on many things. The device used to store the data is very important. For example, an individual file can be restored from a floppy disk faster than a file can be restored from a cartridge tape. However, if there needs to be a restoral of a complete file system, a cartridge tape would be faster than inserting and removing several floppy disks.

The ease of recovery depends on the organization of the data within the file system and the size of the file system. It is better if all files and file systems are structured logically and efficiently. For instance, if all file systems are configured smaller than the block capacity of a cartridge tape, it will be easier to backup/restore the file systems. A file system structure consisting of several small disk partitions facilitates file system reorganization as opposed to the same data being allocated to one large file system (partition).

Organization also refers to how the backup media is physically stored. Rambling through a drawer of floppy disks for the one disk labeled "john's files" would be ridiculous. Each time a backup of a file or file system is done, the storage medium (cartridge tape or floppy disk) should be stored properly and labeled with the following information:

1. The type of backup done (complete, incremental, selective).
2. The complete name of the file, directory, or file system.
3. The date and time.
4. The order or sequence number for the storage medium (for example: 1 of 5, 2 of 5). If it takes more than one medium, you will need to know the sequence of the media to do a restoral.
5. Some instructional information for someone else who may have to restore the data from the storage medium.

The following is an example of what a storage medium label could look like. (This particular example is for a cartridge tape.)

```
Complete Backup of all files in: /usr
Fri. 02/22/85    5:38:04 pm
PART 1 of 2
TO LOAD:  ctccpio -idumvT /dev/rSA/ctape1
```

This information should be used to organize the storage media into a library of stored data. The library should be purged of old backups. Obsolete data should be removed and the media reused instead of building a historical library of numerous floppy disks and cartridge tapes. Also, you do not want to create work for yourself by having to look through all the saved data to find the one file that was lost or damaged.

Importance of Data

Very important or critical data should be saved more often than data of average importance. For something very critical, the data should be saved every time there is an update. This assures that the data is safe if a system crash occurs before the planned backup can be done. For example, if your location has frequent power interruptions, backups should be done more often than in most other environments.

Downtime of System

The policy set forth in this guide is for the 3B2 Computer to be in the single-user mode (run-level S or 1) when either a backup or complete restore is being done. Therefore, the amount of backup downtime (time that the machine is unavailable to users) should be considered. For this reason, you may want to schedule backups for "nonworking" hours.

When a file recovery from backup will be needed is not predictable. Therefore, recovery downtime is another reason you should structure and organize the backups to match configuration of the system. If you know where the stored file is, it will be easier and faster to replace (recover). Also, if there are just a few files, you can restore them in multi-user mode, notify the user that needs them, and no other users are disturbed.

System Use

The way the 3B2 Computer is being used influences the planning of backup schedules. Assuming that the **root** and **/usr** file systems are used only for the operating system and utilities, these file systems need to be backed up only when a change in utilities has occurred. The user-type (**/usr?**) file systems need to be backed up more often because of the activity in the system. Any multi-user system needs to be backed up more often than a single-user system. An increase of users automatically increases the chances of files being updated, damaged, or lost.

Another example would be a data base which is accessed by many customers. This data base must be kept as current as possible so customers will not be conducting business using data from last week. If there is a system failure, you would want the restored data to match the latest accessed data as accurately as possible. This data base application might require a hard disk backup plan.

File Size

The size of the data to be saved is an important consideration when formulating a backup schedule. You would probably want to store large file systems on a cartridge tape so you do not have to load several floppy disks just to restore one file system. A complete backup of all the file systems could be done on one to three cartridge tapes as opposed to 30 or more floppy disks.

On the other hand, storing small files frequently could be done rapidly on floppy disks. Perhaps you have a small file system which contains critical data. This data could be backed up to floppy disks at the end of each day.

Type of Backup

A complete backup is the first backup that should be done on a file system. Once a complete backup is done on a file system, a series of incremental or selective backups can be done. The decision of when to stop doing incremental backups and do a new complete backup is YOUR decision. A selective backup has no restrictions; it can be done at any time on any file/directory. Some of the considerations are:

- How much storage space do you want to dedicate to incremental backups? For example, if floppy disks are used for incremental backups each day and complete backups once each month, over 40 floppy disks could be required for backing up the **/usr** file system. Incremental backups to cartridge tapes would still require one per day, thus the number of media would be about the same. Also, the blocks of data that had changed in one day would probably be insignificant when compared to capacity of a cartridge tape. Therefore, cartridge tapes are NOT recommended for incremental backups.

- How often do you want to take the time to do a complete backup? A complete backup requires less storage media than a complete backup and several incremental backups. A complete backup may be more efficient during recovery since the last complete backup and ALL incremental backup data must be read to completely recover a file system from backups. Therefore, complete backups may be more efficient than incremental backups for file systems that frequently change a large percentage of data.

  However, restoring an entire file system may affect many users' files, causing each user to recreate all file changes since the last complete backup. An incremental backup may affect only a small number of users.

Storage Device

The device chosen to store the backup data falls in the personal preference category. The storage device can be floppy disk(s), hard disk(s), and/or cartridge tape(s). Each storage device has advantages and disadvantages. Some of the advantages and disadvantages are as follows:

- The hard disk is the fastest means of backing up or restoring data. Duplicate partitions on separate hard disks provide the best hard disk backup, as opposed to partitions on the same disk. In either case, the backup partition is normally left unmounted. However, any space used for storing data cannot be used for any other purpose. The space requirements of the system and the stored data should be considered.

- It takes about 20 minutes to format and verify a cartridge tape for use; it takes about 4.5 minutes to format and verify a floppy disk. The time required to fill a cartridge tape or floppy disk with data depends on the structure of the data.

- The time required to recover data from cartridge tapes or floppy disks also depends on the structure of the data. Generally, it takes 6 minutes to restore data from a full floppy disk and about 20 minutes to restore data from a full cartridge tape.

- The cartridge tape can hold more data than a floppy disk (45539 blocks per cartridge tape as opposed to 1422 blocks per floppy disk). Therefore, cartridge tapes are NOT recommended for incremental backups since the amount of data that has been modified is probably small in comparison to the capacity of a cartridge tape.

- Also consider the lower cost of floppy disks versus the higher cost of a cartridge tape.

A general comparison of the two removable storage devices, including physically inserting and removing the storage media, is as follows:

10 floppy disks = 60 min. = 14200 blocks of data
1 cartridge tape = 20 min. = 45539 blocks of data

After combining all respective considerations into one overall picture, YOU must review the trade-offs and form a backup plan that is best for YOU. Each person has their own preferences and each system has a different set of requirements. The only common factor is that a backup plan is essential for all systems.

Complete Backup

General

A complete backup copies directories and files of a specified mounted file system to the desired storage medium. A complete backup can be done at any time and must be done once before an incremental backup can be done on the file system.

> ***Note:*** The **root** (/) file system is backed up using a selective backup procedure. Refer to the "Selective Backup" description in this chapter.

Complete Backup Procedure To Removable Media

You should read the entire procedure before starting any backup. The general sequence for doing a backup is as follows:

1. At the console terminal, log in as **root**.
2. Determine the number of storage media required to back up each file system.
3. If enough formatted storage media are not available, obtain and format the required number of storage media.
4. Take the system to the single-user mode (run-level 1).
5. Run a check on the file system(s) to be backed up.
6. Mount the file systems. A file system must be mounted before it can be backed up.

> ***Note:*** The **/usr** file system contains the **find** command and, therefore, must be mounted to do the backup procedure.

7. Execute the backup command.

   Note: Backups of numerous small files (10 blocks or less) require a significantly longer amount of time.

8. If not previously established, create a directory in the **/etc/save.d** directory containing a file named the same as each file system just backed up. This will serve as a reference time for later incremental backups. If the files are present, **touch** the files associated with the backed up file systems.

9. Unmount the file system(s) and return the system to the normal operating configuration.

   Note: The **root** file system (/) should be the only file system mounted before returning the system to the normal operating configuration (multi-user mode).

10. Store the backup media in a safe place.

Doing a backup should be relatively easy if you follow these operations carefully. Remember to read through the entire procedure before actually starting any backup.

Estimating Number of Storage Media Required for Backup

Before you do any backups, you should know how many blocks of data you are going to save. The **du**(1M) command is used to display the amount of blocks used by the specified file, directory, or file system.

> ***Caution: Always use the -s option with the* du *command to include block counts for linked files. Otherwise, the links to files are not counted.***

This is important when using floppy disks and if more than one cartridge tape is required.

It is important to have enough formatted storage media on hand to contain the directories and files being backed up before the backup process begins. The backup process can NOT be interrupted without having to start from the beginning. As a rule-of-thumb, the number of floppy disks needed for a complete backup is equal to the number of data blocks used in the file system divided by 1400, rounded UP to a whole number and incremented by one.

For example, a file system that occupies 13294 blocks requires 11 floppy disks for a complete backup. This is shown in the following formula.

$$Number\ of\ Disks = \frac{blocks\ used}{1400} + 1$$

$$Number\ of\ Disks = \frac{13294}{1400} = 10 + 1 = 11$$

A rule-of-thumb for the number of cartridge tapes needed for a complete backup is the number of blocks used in the file system divided by 45539, rounded to a whole number. For example, a file system that occupies 72348 blocks requires two (2) cartridge tapes for a complete backup. This is shown in the following equation.

$$Number\ of\ Tapes = \frac{blocks\ used}{45539}$$

$$Number\ of\ Tapes = \frac{72348}{45539} = 2$$

If the number of blocks is close to a multiple of 45539, have an extra formatted cartridge tape to be safe.

> ***Note:*** It is recommended that you NOT back up the entire system onto one cartridge tape. Restorals will be easier if there is one file system per cartridge tape.

The number of blocks needed to back up data to a hard disk is equal to the number of blocks being stored. The disk partition used to hold the backup must be large enough to hold the file system being copied (backed up).

Obtaining and Formatting Removable Media

If enough formatted storage media are not available, you must either get and format enough storage media, or alter your backup plans to exclude those files for which there is not enough storage space. How to format cartridge tapes and floppy disks is described in this chapter.

Single-User Mode

The system should be in single-user mode when either a backup or complete restore is being done. This will assure that no one is accessing some of the files that you are trying to save or restore. Single-user mode is accomplished through the **/etc/shutdown**(1M) command described in detail in Chapter 6, "SYSTEM ADMINISTRATION COMMANDS."

Check File Systems

The file systems to be saved should be checked before they are backed up. The **/etc/fsck**(1M) command (described in Chapter 4, "FILE SYSTEM CHECKING AND REPAIR") is used to check and repair inconsistencies in file systems. With the exception of **root**, a file system must be unmounted before it can be checked.

Mount File Systems

A file system has to be mounted before it can be backed up. The command to mount all normally available file systems is:

```
mountall /etc/fstab
```

Note: The **/etc/fstab** file is described in Chapter 2, "ADMINISTRATIVE FILES AND DIRECTORIES." Device partitions and their mount points are identified in the **/etc/fstab** file for automatic mounting in the multi-user mode.

Executing Backup

The next step is the execution of the backup command. The cpio-type command to transfer data to cartridge tapes is **/etc/ctccpio**. The command used for a complete backup to cartridge tape is of the form:

```
find <file system> -print | ctccpio -ovKT /dev/rSA/ctape1
```

The cpio-type command to transfer data to floppy disks is "**/usr/lbin/ncpio**." You must specify **/usr/lbin/ncpio** or place **/usr/lbin** in your **.profile** search path (PATH). The commands presented in this guide are depicted as though **/usr/lbin** is in your search path. The command used for a complete backup to floppy disks is of the form:

```
find <file system> -print | ncpio -ocv > /dev/rSA/diskette?
```

Note: At your discretion, you can verify that the medium is readable by using one of the following commands:

ctccpio -it /dev/rSA/ctape1 for cartridge tape

or

ncpio -it < /dev/rSA/diskette? for floppy disk

This command will display a table of contents (list) of the files on the medium.

Label each storage medium as instructed. You may add other information to the labels at your discretion.

Create Reference Directory

Create a directory under **/etc/save.d** containing files named for each file system that was backed up. For example, if the **/usr** and **/usr2** file systems were backed up, the new directory would contain two-files: **usr** and **usr2**. This new directory should be named something logical (like **lastbackup**) for future reference. These files will be used as a reference point for incremental backups.

Unmount File Systems

Unmount the **/usr** file system (and any other mounted file systems other than **root**) and return the system to the normal operating configuration (multi-user mode). This two step operation is accomplished with the following commands:

```
umountall
init 2
```

Store the Backup Media

The backup operation is complete. You should store the backup media in a safe place. Any backup media containing obsolete data can be erased and used for other operations.

Sample File System Complete Backup

The following shows the typical command line entries and system responses for doing a complete backup of the **/usr** file system to cartridge tape. The system is in the single-user mode (run-level 1) with the **/usr** file system mounted at the start of the sample. The **/usr** file system was checked (**fsck**) before it was mounted.

```
   Insert the cartridge tape into the ctape1 drive.

# find /usr -print | ctccpio -ovKT /dev/rSA/ctape1<CR>
/usr/bin/cut
/usr/bin/egrep
/usr/bin/checkfsys
     |
     |
  List of all files in the /usr file
  system as they are copied.
     |
     |
/usr/old/bin/ed
/usr/old/bin/red
/usr/old/README

Verify pass begins.
Verify pass completed.


43234 blocks
#
```

The following shows the typical command line entries and system responses for doing a complete backup of the **/usr** file system to floppy disks. The system is in the single-user mode (run-level 1) with the **/usr** file system mounted at the start of the sample. The **/usr** file system was checked (**fsck**) before it was mounted.

Insert the floppy disk into the floppy disk drive.

```
# find /usr -print | ncpio -ocv > /dev/rSA/diskette?<CR>
/usr/bin/cut
/usr/bin/egrep
/usr/bin/checkfsys
      |
      |
  List of files in the /usr file
  system as they are copied.
      |
      |
/usr/bin/cal
/usr/bin/calendar
Reached end of medium on output.
If you want to go on, type device/file name when ready
```

Insert another floppy disk into the floppy disk drive.

```
/dev/rSA/diskette?<CR>
/usr/bin/crontab
/usr/bin/csplit
      |
      |
  List of files in the /usr file
  system as they are copied.
      |
      |
/usr/old/bin/ed
/usr/old/bin/red
/usr/old/README
43234 blocks
#
```

Complete Backup Procedure To Hard Disk

The procedures for backing up to hard disk and restoring from that backup are identical. The general sequence for making copies of hard disk partitions is as follows:

1. At the console terminal, log in as **root**.
2. Take the system to the single-user mode (run-level 1).
3. Make sure the destination area is as large as the original file system.
4. Run a check on the file system(s) to be backed up.
5. Execute the **volcopy** command.
6. Run a check on the new file system copy (**fsck**).
7. Return the system to the normal operating configuration (multi-user mode).

Single-User Mode

The system should be in single-user mode when a **volcopy** is being done. This will assure that no one is accessing the file system that you are trying to save or restore. Single-user mode is accomplished through the **/etc/shutdown**(1M) command described in detail in Chapter 6, "SYSTEM ADMINISTRATION COMMANDS."

Check File Systems

The file systems to be saved should be checked before they are copied. The **/etc/fsck**(1M) command (described in Chapter 4, "FILE SYSTEM CHECKING AND REPAIR") is used to check and repair inconsistencies in file systems. With the exception of **root**, a file system must be unmounted before it can be checked.

Executing Backup

The next step is the execution of the **volcopy** (backup) command. The command used for a complete backup to hard disk is of the form:

```
volcopy <fsname> <srcdevice> <srcvolname> <destdevice> <destvolname>
```

Refer to the **/etc/volcopy**(1M) command description contained in Chapter 6 of this guide for more information.

Check Copy Of File Systems

The file systems that were copied should be checked for reliability as soon as they have been made. The "**/etc/fsck**(1M)" command (described in Chapter 4, "FILE SYSTEM CHECKING AND REPAIR") is used to check and repair inconsistencies in file systems. With the exception of **root**, a file system must be unmounted before it can be checked.

Return To Normal Operation

Returning the system to the multi-user mode is accomplished using the following command:

```
init 2
```

Sample Complete Backup To Hard Disk

The following shows the typical command line entries and system responses for doing a file system copy from hard disk to hard disk and checking the new file system copy. The system is in the single-user mode and the **/usr2** file system has been checked at the start of this sample.

```
# volcopy usr2 /dev/rdsk/c1d0s8 usr2 /dev/rdsk/c1d1sa usr2<CR>

   Informational messages will be displayed.
   Refer to the "volcopy" description in
   Chapter 6 for information on these messages.

From: /dev/rdsk/c1d0s8, to: /dev/rdsk/c1d1sa? (DEL if wrong)

   You have 10 seconds to interrupt
   the command process if desired.
   If not, the process continues.

  END: 20790 blocks.
# fsck -D /dev/rdsk/c1d1sa<CR>

  /dev/rdsk/c1d1sa
  File System: usr2 Volname: usr2

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  139 files 768 blocks 19694 free
#
```

Incremental Backup

General

An incremental backup copies files that have changed since the last backup was done. The incremental file system backup is fast in comparison to the time required to do a complete backup.

> ***Note:*** The **root** (/) file system can NOT be backed up using a complete backup. The **root** file system can only be backed up using a selective backup procedure described in this chapter. These procedures should not be confused with the procedures provided by Simple Administration.

A complete backup is the first backup that is done on a file system. Once a complete backup is done on a file system, a series of incremental backups can be done. The decision of when to stop doing incremental backups and do a new complete backup is the focal point of YOUR system backup plan. However, if the incremental backup takes longer than 6 minutes, then a complete backup is probably required. Some of the considerations are as follows.

- How many floppy disks do you want to dedicate to incremental backups? For example, if incremental backups are done for each day and complete backups are done once each month, over 40 floppy disks could be required for backing-up the **/usr** file system.

  Note: Cartridge tapes are NOT recommended for incremental backups.

- How often do you want to take the time to do a complete backup? Fewer floppy disks are required to maintain a file system backup library the more often a complete file system backup is done. The trade-off is system down-time and operator time versus the cost of floppy disks. In evaluating the time factor, also consider the amount of time that would be necessary to recover (restore) a file system from backup floppy disks. Note that each floppy disk of all incremental backups must be read to completely recover a file system from backups. Therefore, a complete backup may be more efficient than an incremental backup for file systems that frequently change a large percentage of the data.

- The backup plan is then based on risk versus time. Remember that the most frequent cause of damage to a file system is the result of user actions (not system failures). The more users that are on the system, the more updates to data and possible loss of data. The accidental removal of files is part of the risk factor that must be considered in establishing a backup plan.

System Incremental Backup Procedure

The following steps are necessary to do an incremental backup of a file system.

1. At the console terminal, log in as **root**.
2. Take the system to the single-user mode (run-level S or 1).
3. Run **fsck** on the file system(s) to be backed up.
4. Mount the file systems using the following command.

```
mountall /etc/fstab
```

   Note: Note that the **usr** file system contains the **find** command and, therefore, must be mounted to do an incremental backup.

5. Execute the backup operation with the "-newer" option to the **find** command. This option will compare the files listed in the file system to the reference file created at the time of the last backup, and keep only those files which have been updated. Label the floppy disk as desired.
6. Unmount the file system(s) and return the system to the normal operating configuration using the following commands.

```
umountall
init 2
```

7. At your discretion, verify the backup floppy disks. See the "VERIFYING FLOPPY DISK USABILITY" procedure in this chapter. Store the backup floppy disks in a safe place.

Sample File System Incremental Backup

The following shows the typical command line entries and system responses for doing an incremental backup of the **/usr** file system. There is a file named **usr** in an **/etc/save.d/lastbackup** directory that was created when the last backup of the **/usr** file system was done. The system is in the single-user mode (run-level 1) with the **/usr** file system mounted at the start of the sample. The **/usr** file system was checked (**fsck**) before it was mounted.

```
# find /usr -newer /etc/save.d/lastbackup/usr -print | ncpio -ocv > /dev/rSA/diskette?<CR>
/usr/rar/booklist
/usr/rar/functions
/usr/klm/glossary
96 blocks
#
```

Selective Backup

General

Specific directories and files can be quickly saved by using the **find** command and a variation of the cpio-type command. This type of storing data is sometimes referred to as "selective backups." Before you copy the selected directory(ies) or file(s), check the number of blocks involved. The amount of data copied to a single floppy disk cannot exceed 1422 blocks. The amount of data copied to a cartridge tape cannot exceed 45539 blocks. This method of backup is used to save the **root** directory.

Selective Backup Procedure

The general sequence for copying selected information to a storage medium is as follows:

1. At the console terminal, log in as **root**. (If "others" have read/write permission to **/dev/rSA/diskette?**, any login will work. As the system is delivered, only **root** has write permission.)

2. Change directory (**cd**) to the desired directory.

   If you are backing up the **root** file system (/), then you would also do the following two steps.

   a. Execute an **ls** command and redirect the output into a file (**ls > list**).

   b. Edit the file and remove the lines naming the **/usr?** file systems and any other directories/files that you **DO NOT** want to back up.

Caution: Always use the* -s *option of the* du *command to include block counts for linked files. Otherwise, the links to files are not counted.

3. Check that the number of data blocks involved (**du -s** command or **df -t** command if backing up **root**) is less than the number of data blocks on the formatted storage medium. If the **du -s** command is used on more than one directory/file, you will have to add the block sizes together for the required number of blocks. The **df -t** command reports the number of free and allocated blocks for file systems. You must calculate the number of blocks actually used.

4. Label the formatted medium to indicate the directory or file. Insert the formatted medium into the appropriate drive.

5. Copy the selected data to the medium using the proper **find** and cpio-type command.

6. Verify that you can read the data by using the cpio-type command with a **t** option. This will display a table of contents of the specified data without creating any files.

Sample Selective Backup For root

The following shows the command lines and system responses associated with backing up the **root** file system. First a list is formed from the **ls** command and edited to name only those files/directories which should be backed up. For this sample, the /**list** file is as follows.

```
# cat list<CR>
bck
bin
boot
dev
dgmon
dgn
etc
filledt
install
lib
mnt
save
unix
#
```

Next, a **df** is executed to see how many blocks of data are contained in the **root** file system. Finally, the backup command line is entered and the **root** file system is copied to a cartridge tape. At the start of this sample, the current directory is **root** (/) and the **list** file is as desired.

```
# df -t /<CR>
/          (/dev/dsk/c1d0s0):     1796 blocks     1075 i-nodes
                     total:     14400 blocks     1792 i-nodes
# find 'cat list' -print | ctccpio -ovKT /dev/rSA/ctape1<CR>
bck
bin
bin/cat
bin/cpio
bin/echo
    |
    |
List of files as they are being
copied to the cartridge tape.
    |
    |
lib/comp
lib/cpp
mnt
save
unix

Verify pass begins.
Verify pass completed.


1796 blocks
#
```

At your discretion, you can verify that the tape is readable by using the following command:

```
ctccpio -it /dev/rSA/ctape1
```

Sample Selective Backup For A Directory

The following shows the command line entries and system responses associated with copying a selected directory of information to a floppy disk. Refer to the **find** command and the cpio-type manual pages for an explanation of the various options. This sample also shows the command line to verify the readability of the data.

```
# cd /usr/rar/305-323<CR>
# du -s<CR>
258     .
# find . -print | ncpio -ocv > /dev/rSA/diskette?<CR>
ch6.package
chx.functions
trademarks
ch1.general
ch2.install
ch3.files
ch4.advice
ch5.fsck
248 blocks
# ncpio -it < /dev/rSA/diskette?<CR>
.
ch6.package
chx.functions
trademarks
ch1.general
ch2.install
ch3.files
ch4.advice
ch5.fsck
248 blocks
#
```

FILE SYSTEM RESTORAL FROM BACKUP

General

The restoral of directories and files from system backups can be done on the basis of a single file, a directory structure, or all files of a system backup. The procedure for loading the contents of a hard disk backup is the same as described in the "Sample Complete Backup To Hard Disk" procedure of this chapter.

When copying (writing) data to a mounted file system using a cpio-type command, a "Text file busy" error message is output for each write attempt to an executing program. This message means that the open program file cannot be unlinked and the backup copy cannot be written to the file system. When restoring the root file system it is normal to get this message for the **/etc/init** file, the **/bin/sh** file, and the open **/etc/***cpio-type* file, regardless of the system run-level (S or 2). A "Partial Restore" of the operating system using the **3B2 Computer Core System** floppy disks is done to restore these types of root files. Additional files, as applicable, are reported if the system is in run-level 2 during a restore process.

File System Restore Procedure

The following steps are necessary to restore a portion or all of a file system from a removable media backup.

1. At the console, log in as **root**.

2. Take the system to the single-user mode (run-level S or 1).

3. Mount the normal operating file systems using the following command.

   ```
   mountall /etc/fstab
   ```

   If the file system to be restored is NOT mounted by the **mountall** command, it must be mounted before it can be restored. See the **/etc/mount**(1M) command described in Chapter 6 of this guide.

4. Execute the appropriate cpio-type command specifying the **u** (unconditional) option. This option will unconditionally overwrite the current files with the stored files.

When restoring from a complete or incremental backup, all removable media of the series must be loaded. Even if you intend to restore only a single file, all removable media of a backup must be loaded in sequence. In all cases, the first removable medium of a series must be the first medium read by the restore process.

5. Unmount the file systems and return the system to the normal operating configuration using the following commands.

```
umountall
init 2
```

6. Store the backup media in a safe place.

Sample File System Restoral

The following shows the typical command line entries and system responses for restoring from a complete backup of the **/usr2** file system. At the start of this sample, the system is in the single-user mode (run-level 1), the current directory is **root** (/), and the **/usr2** file system is mounted.

```
   Insert the cartridge tape into the cartridge tape drive.

# ctccpio -idumvT /dev/rSA/ctape1
/usr2/mrk/.profile
/usr2/mrk/updates
/usr2/mrk/305-327/chapter1
/usr2/mrk/305-327/chapter2
     |
     |
  List of all files in the /usr file
  system as they are copied.
     |
     |
/usr2/dth/.profile
/usr2/dth/docplans
/usr2/dth/index

5398 blocks
#
```

Sample Restoral From Incremental Backup

The following shows the typical command line entries and system responses for restoring a portion of the **/usr** file system. At the start of this sample, the system is in the single-user mode (run-level 1), the current directory is **root** (/) and the **/usr** file system is mounted. The restoral is from an incremental backup (single floppy disk).

Insert the floppy disk into the floppy disk drive.

```
# ncpio -idumcv < /dev/rSA/diskette?
/usr/klm/glossary
/usr/rar/booklist
/usr/rar/functions
96 blocks
#
```

Sample Restoral From Selective Backup

The following shows the typical command line entries and system responses for restoring the data of a selective backup to the hard disk. The command used in this example copies the information from the integral floppy disk (**/dev/rSA/diskette1**) to the current directory of the integral hard disk.

In this example, the current directory is the **/tmp** directory. The data is from a selective backup of the **/usr/rar/305-323** directory. The contents of the **/tmp** and **/usr/rar/305-323** directories are listed to compare the source data with the copied data. Refer to the **cpio** manual pages for a complete explanation of the various options.

```
# cd /tmp<CR>
# cpio -icdumv < /dev/rSA/diskette1<CR>

ch6.package
chx.functions
trademarks
ch1.general
ch2.install
ch3.files
ch4.advice
ch5.fsck
248 blocks
# ls -l /tmp /usr/rar/305-323<CR>

/tmp:
total 256
-rw-r--r--   1 rar      other       2783 Feb 11 07:53 ch1.general
-rw-r--r--   1 rar      other       7819 Feb 11 14:14 ch2.install
-rw-r--r--   1 rar      other      18340 Feb 12 13:00 ch3.files
-rw-r--r--   1 rar      other      17627 Feb 14 08:14 ch4.advice
-rw-r--r--   1 rar      other      45725 Feb 12 10:24 ch5.fsck
-rw-r--r--   1 rar      other      19294 Feb 12 10:43 ch6.package
-rw-r--r--   1 rar      other      13465 Feb 17 07:53 chx.functions
-rw-r--r--   1 rar      other        751 Feb 13 07:06 trademarks
#

/usr/rar/305-323:
total 256
-rw-r--r--   1 rar      other       2783 Feb 11 07:53 ch1.general
-rw-r--r--   1 rar      other       7819 Feb 11 14:14 ch2.install
-rw-r--r--   1 rar      other      18340 Feb 12 13:00 ch3.files
-rw-r--r--   1 rar      other      17627 Feb 14 08:14 ch4.advice
-rw-r--r--   1 rar      other      45725 Feb 12 10:24 ch5.fsck
-rw-r--r--   1 rar      other      19294 Feb 12 10:43 ch6.package
-rw-r--r--   1 rar      other      13465 Feb 17 07:53 chx.functions
-rw-r--r--   1 rar      other        751 Feb 13 07:06 trademarks
#
```

Sample File Restoral

The following shows the typical command line entries and system responses for restoring a single file from a cartridge tape containing a complete backup of the **/usr2** file system. This process works on a file basis and will not restore directories without specifying to restore all files in that directory. For example, **/usr2/klm/*** will restore all files under the **/usr2/klm** directory.

At the start of this example, the **/usr2** file system is mounted and the cartridge tape containing the backup of the **/usr2** file system is in the cartridge tape drive. The file to be restored is **/usr2/klm/allchaps** consisting of 76 blocks. Notice that the block count given at the end of the process is the total block count of stored data.

```
# ctccpio -idumvT /dev/rSA/ctape1 /usr2/klm/allchaps<CR>
/usr2/klm/allchaps

16460 blocks
#
```

ALLOCATING SYSTEM RESOURCES

Caution: Enough storage space must be allocated for the various file systems to support the operation of the system application. If you are not experienced in allocating system resources, it is recommended that the system be operated with a default configuration for a time interval sufficient to determine the partitioning requirements of the particular application.

General

The purpose of system resource allocation is to define a hard disk system configuration matched to the software application and user community needs. The process of defining a hard disk system configuration is called disk partitioning. Part of this process involves the structuring of the user community into groups and the assignment of user groups to file systems. Structural decisions made here carry over into and become part of other system operational considerations (for example, backup strategy). The proper user community and hard disk system structure for the support of a particular application establishes the foundation for reasonable system performance and system operation. After the disk memory usage is defined via partitioning, the operating system tunable parameters can be adjusted to maximize the system performance. Refer to the "SYSTEM RECONFIGURATION" task description in this chapter for information on optimizing system performance. Incorrect disk partitioning usually results in running out of disk space on a particular partition ("NOTICE: no space on . . . " messages). Incorrect disk partitioning can also cause degraded system performance in the form of increased disk access time.

The major system resource to be allocated is hard disk memory. The allocation of hard disk memory defines how the available hard disk system memory is used to support a particular application. A default configuration is established by the Full Restore option of the 3B2 Computer Core System floppy disks. The default configuration is matched to the number and size of equipped hard disks. For a single hard disk system, the default configuration assigns the root (/) and usr (**/usr**) file systems to disk

partitions 0 and 2, respectively. For a dual hard disk system, the default configuration assigns the root (/) and usr2 (**/usr2**) file systems to partitions 0 and 8 on the first disk, and the usr (**/usr**) file system to partition 2 on the second disk. The default number of blocks and information nodes (i-nodes) allocated for each of the various configurations of hard disk memory are shown in Appendix B, "DEFAULT DEVICE PARTITIONING." The various disk partitioning considerations are generally better understood after the system has been operating for some time. Calculating the partition values requires a previous knowledge of the system application and operation. If you are not experienced in allocating system resources, it is recommended that the system be operated with a default configuration for a time interval sufficient to determine the partitioning requirements of the particular application.

On systems equipped with a single hard disk, the allocation of hard disk memory space is concerned with:

- Providing enough free space in the root (/) and user (**/usr**) file systems to support the particular application. Certain utilities (software features) require operational space in addition to the space required to install the feature.

- Separating the user community into a usr2, usr3, ... file systems ONLY IF enough free disk space exists to support such a configuration. There are system operational advantages in having login directories in file systems separate from the root (/) and **usr** file systems. When user login directories and utilities are in the same file system, the file system free space is available to both functions. Certain functions use large amounts of disk space for a short time. This is referred to as temporary space. The **/usr/tmp** directory provides temporary space. The **/usr/spool** directory also requires a certain amount of free space which is dynamically used. Providing a separate **/usr2** file system for the user-type data (logins) requires sufficient free space for the **/usr** and **/usr2** file systems. Sufficient free space must remain for the **/usr** file system to support spooling and other similar functions.

On systems equipped with multiple hard disks, the allocation of hard disk memory space is more complex than on single disk systems. One added consideration is how to best make use of the dual port disk access available with two hard disks. For example, certain performance gains can be realized by having the root (/) and user (**/usr**) file systems on separate disk drives. However, the particular application may require that the second disk be used for usr2, usr3, ... file systems. The capability to easily repartition the second disk without destroying the operating system and utilities is done by using the second disk only for user-type data (logins). Also, using the second disk only for user-type file systems is a good starting approach because of the ability to easily change the user community partitioning. Refer to the **/etc/fmthard**(1M) command description in Chapter 6, "SYSTEM ADMINISTRATION COMMANDS," for an example of second hard disk repartitioning.

Therefore, to intelligently allocate system resources a complete profile of the application being supported must be known. Certain of the miscellaneous partitioning considerations are further described in the following paragraphs.

Miscellaneous Considerations

Many factors must be considered to correctly allocate system resources. In general, some of these factors are also factors in selecting the proper computer system and the features supported on the system. Some of the considerations are:

- The number and size of hard disk drives.
- The number of users to be supported.
- What users should be members of the same group.
- The amount of storage space needed per user.

- The amount of storage space needed for the root (/) and user (**/usr**) file systems to support a particular software application. This includes the storage space needed for the various utilities which are to be installed. This information can be found in the *AT&T 3B2 Computer Release Notes* or can be empirically determined for the various utilities.

- The amount of free space needed for the root (/) and usr (**/usr**) file systems to support a given application.

- The method (strategy) for doing file system backups. For example, as dictated by the application, duplicate hard disk partitions can be used to maintain an active (mounted) and inactive (unmounted backup) of a file system.

- The amount of flexibility built into a given configuration is a function of the number and size of the disk partitions. When experimenting with disk partitioning for the first time, consider allocating several small partitions of free space. The use of small disk partitions for the purpose of "tacking on" additional disk space where necessary is a quick-fix-alternative to having to repartition the entire system. Also, as previously mentioned, consider using the second disk only for user-type file systems to be able to repartition the second disk without having to repartitioning the entire system. Making use of these suggestions can significantly reduce the time required to empirically determine the proper partitioning arrangement for your application.

Some aspects of these and other considerations are further described in the following paragraphs.

Disk System Structure

The assignment of a file system to a physical device (normally a hard disk partition) requires that the dimensions and use of each file system be known. The first consideration is the number and size of the hard disk drives. All other partitioning considerations ultimately depend on the disk system structure. For a system equipped with two hard disk drives, the best system performance is typically achieved by having **root**, **swap**, and **usr2** partitions on the first hard disk (disk 1); the **/usr** partition is the second hard disk (disk 2). For a system equipped with a single hard disk drive, dual port access is not a consideration.

Assignment of Users to File Systems

Before assigning users to a file system, the user community should be organized into groups based on functions or projects. System security is another important consideration in the assignment of users to groups. At one extreme, each user can be assigned to his/her own group. If all users do the same thing or are assigned to the same project, group divisions can still be done to keep file system size within limits derived from your backup/restore strategy. Users with common interests and needs should be placed in the same group. Remember that file linking is only done within a file system.

After assigning users to groups and groups to file systems, estimate the total number of blocks (512-byte blocks) needed for each user. The estimate includes free space. The function(s) being done by one group could require more disk space and impose a more frequent backup requirement than the functions being done by another group.

Group identifications are assigned by editing the **/etc/group** file and adding the applicable group definitions. The **/etc/passwd** must also be modified to change the group identification number for the applicable logins. The format of the **/etc/group** and **/etc/passwd** files are described in Chapter 2, "ADMINISTRATIVE DIRECTORIES AND FILES."

File System Backup Strategy

Several partitioning considerations are imposed by the strategy used to maintain file system backups. The backup strategy is primarily concerned with the user-type file systems. These are the file systems containing the login directories. Remember that a user-type file system requires frequent backup as opposed to a file system containing the operating system and utilities. Important considerations are as follows.

a. User-type file systems offer a simpler backup strategy than file systems containing a mix of utilities and login directories.

b. If the backup strategy includes the use of duplicate hard disk partitions, space must be defined on one or more hard disk drives as applicable. The best hard disk backup strategy is to define duplicate partitions (file systems) on separate hard disk drives, as necessary, to contain the user-type file systems. The duplicate partitions can be on the same disk drive; however, successful access to both partitions then depends on the same hard disk drive. The potential amount of free disk space is reduced by the size of the duplicate partition when hard disk space is used as backup for another disk partition.

c. Unless the system application dictates otherwise, file systems should be less than the capacity of a cartridge tape (45539 blocks). The tape backup can support multiple cartridge tapes; however, large file systems are difficult to administer. In general, it is better to have a hard disk partitioned into several small partitions as opposed to one large partition. Several smaller partitions facilitate the reorganization of file systems. Also consider the amount of time to backup/restore a small disk partition as compared with the time required for a large partition (file system). This consideration parallels the assignment of users to groups, and the assignment of groups to files systems.

Hard Disk Partitioning

General

Device partitions should fall on cylinder boundaries to obtain the best possible file system performance. For the root device, the boot and swap partitions are special in this regard. The number of blocks assigned to the boot and swap partitions are collectively chosen to cause the subsequent partition values to fall on cylinder boundaries. Partition size for file systems is therefore even divisible by the number of blocks per cylinder for the device (modulo 90 for a 30-megabyte disk). This approach eliminates wasted space that would result from the strict assignment of partition values based on a modulo cylinder size for each partition of the root device. The otherwise wasted space in the boot block is used in the swap area without degrading the system performance. Refer to Appendix B, "DEVICE DEFAULT PARTITIONING," for blocks per cylinder information for the various hard disks.

The first two hard disks are partitioned using the Full Restore option of the 3B2 Computer Core System floppy disks. The second and subsequent hard disks (other than root disks) can also be partitioned using the **/etc/fmthard** command. Refer to the **/etc/fmthard**(1M) command description in Chapter 6, "SYSTEM ADMINISTRATION COMMANDS," for a sample command use.

Root Disk Partitioning

The root disk is partitioned using the Full Restore option provided by the 3B2 Computer Core System floppy disks. The Full Restore option only supports the use of the first hard disk as the root device. When opting to define your own block sizes, remember that the number of blocks allocated are in 512-byte blocks. The number of information nodes (i-nodes) allocated for a partition is fixed in the full restore script to be one-fourth of the number of 512-byte blocks (one-eighth of the 1024-byte logical blocks). This fixed value is the result of the default i-node argument of the **/etc/mkfs** command. The **/etc/mkfs** command allocates i-nodes is groups of 16 (modulo 16). If a number is not a multilple of 16, the next lower multiple of 16 is allocated. Refer to Appendix B, "DEVICE DEFAULT PARTITIONING," for a summary of the partition use, starting sector, size, and information node default values.

Other Than Root Disk Partitioning

The second or other than root disk can be partitioned at the same time the root disk is partitioned during the full restore process. The second or other than root disk can also be partitioned using the **/etc/fmthard**(1M) command. Only the knowledgeable user should attempt to use the **/etc/fmthard** command.

Sample System Resource Allocation

System Description

The following sample allocation of system resources is based on a model system having the following characteristics.

a. The system is equipped with two 30-megabyte hard disks and 2 megabytes of random access memory.

b. The primary use of the system is for text processing. The primary editor used in this application is **/usr/bin/vi**. This editor uses **/tmp** file space for buffer memory. The application requires sufficient free space in the root (/) file system to support editing and text formatting. The initial target is for about 4000 blocks of free space in the root file system for this application.

c. The user community consists of eight users. Two projects are being supported, with four users assigned to each project. Based on previous experience in the projects being supported, it is estimated that each user needs 5000 blocks. Therefore, two 20000 block file systems are initially assumed to be required.

d. Basic networking is used to transfer data to and from a larger machine. Line printer spooling is used for draft copies. Enough free space must exist in the **/usr** file system to support the basic networking and spooling functions.

e. The utilities required for this application are estimated to require about 10000 blocks in the root file system and 27000 blocks in the usr file system.

f. The combined utilities and free space estimate for the root file system is about 14000 blocks. The combined utilities and free space estimate for the **/usr** file system is about 42500 blocks.

Based on the analysis of the application, the following partition values are derived. The file system block sizes have been adjusted from the initial estimates to fall on partition boundaries (modulo 90 blocks for 30-megabyte disks). The first disk values for the boot and swap partitions are collectively assigned to cause the subsequent file system partitions to fall on partition boundaries.

PARTITION	USE	SIZE
c1d0s0	root	14400
c1d0s1	swap	6020
c1d0s6	entire disk	62550
c1d0s7	boot	100
c1d0s8	usr2	20970
c1d0s9	usr3	21060
c1d1s2	usr	42480
c1d1s6	entire disk	62550
c1d1s7	boot	90
c1d1s8	free	19980

Installing Partition Values Using Full Restore

The following command line entries and system responses show the installation of the previously defined partition values using the Full Restore option of the **3B2 Core System** floppy disks. This example places the root (/) and /**usr** file systems on separate disk drives. At the start of the example, the system is in the firmware mode. At the end of the example, the system is ready to be loaded from either backups or individual utilities can be installed or any other system setup procedure can be executed. If backups are on cartridge tape, the Cartridge Tape Utilities must be installed first.

```
Enter name of program to execute [ ]: unix<CR>
     Possible load devices are:

Option Number   Slot     Name
---------------------------------------
       0          0     FD5
       1          0     HD30
       2          0     HD30
       3          3     CTC

Enter Load Device Option Number [1 (HD30)]: 0<CR>


UNIX System V Release 2.0.3 3B2 Version 2
unix
Copyright (c) 1984 AT&T.
All Rights Reserved


3B2 Version 2 Installation

        1  Full Restore
        2  Partial Restore
        3  Dual-Disk Upgrade
        4  Release Upgrade

When responding to a question, you may use the "backspace" key
to erase the last character  typed or the "@" key to erase the
entire line. Enter "help" for additional information.

Selection? [ 1 2 3 4 quit help ] 1<CR>


-- Full Restore --

The "Full Restore" will destroy EVERYTHING on both hard disks and install a
3B2 Version 2 core UNIX system.

Continue? [ y n help ] y<CR>

Use the default hard disk partitioning? [y n quit help ] n<CR>

How many blocks for the "swap" partition?
[ (3500 - 53522) quit again help ] (default 6020) <CR>

How many blocks for the "root" partition?
[ (8928 - 56430) quit again help ] (default 12510) 14400<CR>
```

Continued

Continued from the previous screen

```
There are 42030 blocks remaining of disk 1 and 62460 blocks on
disk 2. The "usr" partition requires at least 9360 blocks; by
default, 62460 blocks would have been allocated. The system will
typically perform better with "usr" on disk 2.

Which disk should hold the "usr"
partition? [1 2 quit again help ] (default 2) <CR>

How many blocks for the "usr" partition?
[ (9360 - 62460) quit again help ] (default 62460) 42480<CR>

There are 42030 blocks remaining on disk 1.

How many blocks for the disk 1 partition 8?
[ (0 - 42030) again quit help ] (default 42030) 20970<CR>

Upon what directory should the file system within disk 1 partition 8
be mounted? [ (pathname) again quit help ] (default /usr2) <CR>

How many blocks for the disk 1 partition 9?
[ (0 - 21060) again quit help ] (default 21060) <CR>

Upon what directory should the file system within disk 1 partition 9
be mounted? [ (pathname) again quit help ] (default /usr3) <CR>

There are 19980 blocks remaining on disk 2.

How many blocks for the disk 2 partition 8?
[ (0 - 19980) again quit help ] (default 19980) <CR>

Upon what directory should the file system within disk 2 partition 8
be mounted? [ (pathname) again quit help ] (default /usr4) <CR>

Disk 1 partitioning:
*   Partition  Tag  Flags  First Sector  Sector Count   Mount Directory
        0       2    00        6120         14400
        1       3    01         100          6020
        6       0    01           0         62550
        7       0    01           0           100
        8       0    00       20520         20970       /usr2
        9       0    00       41490         21060       /usr3

Disk 2 partitioning:
*   Partition  Tag  Flags  First Sector  Sector Count   Mount Directory
        2       0    00          90         42480       /usr
        6       0    01           0         62550
        7       0    01           0            90
        8       0    00       42570         20790       /usr4

This completes the interactive partitioning of your disks. Enter
"go" if you are ready to proceed. Enter "again" to specify
different partitioning for both disks.

Type "go" to proceed, "again" to start over [ go again quit ] go<CR>

Installing the initial core system files. This should take
no more than ten minutes.
1246 blocks
1 blocks
```

Continued

Continued from the previous screen

```
The system is restarting itself from the hard disk. This should
take no more than five minutes. The installation procedure will
then continue automatically.


SELF-CHECK
UNKNOWN ID CODE 0x5 FOR DEVICE IN SLOT 3
EQUIPPED DEVICE TABLE COMPLETION WILL CONTINUE.
CHECK EDT.


UNIX System V Release 2.0.3 3B2 Version 2
unix
Copyright (c) 1984 AT&T.
All Rights Reserved

Checking the hard disk file systems.

  /dev/dsk/c1d0s0
  File System: root Volume: 2032

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  202 files 1378 blocks 12794 free

  /dev/dsk/c1d1s2
  File System: usr Volume: 2032

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  3 files 4 blocks 41810 free

  /dev/dsk/c1d0s8
  File System: usr2 Volume: 2032

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  3 files 4 blocks 20636 free

  /dev/dsk/c1d0s9
  File System: usr2 Volume: 2032

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  3 files 4 blocks 20724 free
```

Continued

Continued from the previous screen

```
  /dev/dsk/c1d1s8
  File System: usr4 Volume: 2032

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  3 files 4 blocks 19660 free

Please insert the 3B2 Core System floppy number 2.

Type "go" when ready [go quit help ] go<CR>

Installing additional core system files. This should take no more than
ten minutes.
1233 blocks

Please insert the 3B2 Core System floppy number 3.

Type "go" when ready [ go quit help ] go<CR>

Installing additional core system files. This should take no more than
ten minutes.
1281 blocks

Please insert the 3B2 Core System floppy number 4.

Type "go" when ready [ go quit help ] go<CR>

Installing additional core system files. This should take no more than
ten minutes.
1078 blocks

Please insert the 3B2 Core System floppy number 5.

Type "go" when ready [ go quit help ] go<CR>

Installing additional core system files. This should take no more than
ten minutes.
1146 blocks

You may now remove the last 3B2 Core System floppy.
83 blocks

Installation is now complete. The system is restarting itself from
the hard disk. It will be ready to use when you receive the "Console
Login" prompt. This should take no more than five minutes.


SELF-CHECK
UNKNOWN ID CODE 0x5 FOR DEVICE IN SLOT 3
EQUIPPED DEVICE TABLE COMPLETION WILL CONTINUE.
CHECK EDT.

DIAGNOSTICS  PASSED

Driver not found for "VOID" device (board code 3)
```

Continued

Continued from the previous screen

```
UNIX System V Release 2.0.3 3B2 Version 2
unix
Copyright (c) 1984 AT&T.
All Rights Reserved
The system is coming up.  Please wait.

This machine has not been used as a customer machine yet.  The messages that
follow are from checking the built-in file systems for damage that might have
occurred during shipment.  As long as you do not see either of the messages
                                    BOOT UNIX
or                          FILE SYSTEM WAS MODIFIED
all is well.  If either message does come out, call your service representative.
However, the machine is still usable unless I tell you otherwise.
I will now start checking file systems.


  /dev/dsk/c1d1s2
  File System: usr Volume: 2032

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  214 files 1552 blocks 40262 free

  /dev/dsk/c1d0s8
  File System: usr2 Volume: 2032

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  3 files 4 blocks 20636 free

  /dev/dsk/c1d0s9
  File System: usr3 Volume: 2032

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  3 files 4 blocks 20724 free

  /dev/dsk/c1d1s8
  File System: usr4 Volume: 2032

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  3 files 4 blocks 19660 free
```

Continued

Continued from the previous screen

```
Generating a new /unix

        Welcome!
This machine has to be set up by you.  When you see the "login" message type
                                    setup
followed by the RETURN key.  This will start a procedure that leads you through
those things that should be done the "first time" the machine is used.

AT&T 3B2 SYSTEM CONFIGURATION:

Memory size: 2 Megabytes
System Peripherals:

        Device Name         Subdevices

        SBD
                        Floppy Disk
                        30 Megabyte Disk
                        30 Megabyte Disk
        PORTS
        PORTS
        "VOID"
The system is ready.

Console Login:
```

Another solution for the same model system is to use the second hard disk only for user-type file systems. The following partition values support this configuration. The procedure to install this configuration is the same as the previous example, except for assigning **/usr** to the first hard disk.

PARTITION	USE	SIZE
c1d0s0	root	14400
c1d0s1	swap	6020
c1d0s2	user	42030
c1d0s6	entire disk	62550
c1d0s7	boot	100
c1d1s6	entire disk	62550
c1d1s7	boot	90
c1d1s8	usr2	20790
c1d1s9	usr3	20790
c1d1sa	usr4	20880

SYSTEM RECONFIGURATION

Caution 1: When reconfiguring the operating system DO NOT arbitrarily change the node name (NODE) of the 3B2 Computer. Once basic networking has been established, a change in node name must be coordinated with all interfacing systems. If not properly coordinated, a calling system will fail sequence checks because the returned name does not match the name stored in */usr/lib/uucp/Systems.*

Caution 2: System parameters are used to alter the size of various tables and control structures according to expected system load. System performance may be enhanced or degraded by changing tunable parameters. It is recommended that you copy the bootable */unix* ***to*** */oldunix* ***so that you will have a bootable operating system if you create an unbootable*** */unix.*

Caution 3: Do not change system and hardware device variables. The fine-tuning of a system is concerned only with the tunable parameters.

General

System reconfiguration consists of changing system tunable parameters in relation to the available Random Access Memory (RAM), sometimes called core memory or mainstore. Tunable parameters allow for the fine-tuning of the operating system. Sometimes before fine-tuning is effective, partitioning of the hard disk(s) is required. Hard disk partitioning is a means of allocating blocks of disk storage to accommodate the operating system, software utilities, and the user population. Refer to the "ALLOCATING SYSTEM RESOURCES" task description in this chapter for information about hard disk partitioning.

The system configuration as delivered is based on a compromise configuration that is satisfactory for most applications. System performance can be improved for a specific hardware configuration and system application via system reconfiguration. The goal in reconfiguring the system is to achieve optimum performance for a specific application. What is optimum for one application is generally unsatisfactory for a different application. Optimum machine performance is achieved when a given set of functions that make up a particular application execute in the shortest possible time interval. Unless you are a UNIX System guru and know internal activities of the machine, you will spend a lot of time experimenting to find the optimum configuration for your application.

System reconfiguration variables are in files of the **/etc/master.d** directory. These variables are for hardware devices, system devices, and tunable parameters. In general, the fine-tuning of a system is concerned only with the tunable parameters.

Tunable Parameters

Caution: System performance may be enhanced or degraded by changing tunable parameters. Raising or lowering a parameter too far can cause strange results. It is recommended that you copy the bootable */unix* ***to*** */oldunix* ***so that you will have a bootable operating system if you create an unbootable*** */unix.*

Tunable system parameters are used to alter the size of various tables and control structures according to expected system load. There are no maximum values for the parameters described here unless stated in the definition. Caution should be used when changing these variables since certain changes directly affect system performance. System performance can be increased to a maximum efficiency when the operating system is in tune with the available hardware, and the kernel parameters are matched with the software application. You may have to experiment with different parameter values before you find one that best matches your application. The stimulus for changing the value of a parameter is either poor system performance or repeated error messages that point to a particular resource deficiency.

Figure 3-1 shows the default and recommended tunable parameter values for 3B2/300 Computer systems equipped with different ranges of Random Access Memory (RAM).

Figure 3-2 shows the default and recommended tunable parameter values for 3B2/400 Computer systems equipped with different ranges of Random Access Memory (RAM).

These models are based on systems supporting from two to six users. The parameters shown are defined in the following four files:

- **/etc/master.d/kernel**
- **/etc/master.d/msg**
- **/etc/master.d/sem**
- **/etc/master.d/shm**.

The parameters defined in the **/etc/master.d/kernel** file are delivered with the Core Utilities. The message (MSG...), semaphore (SEM...), and shared memory (SHM...) parameters are added with the Inter-Process Communication Utilities.

The value of certain parameters are calculated by entries in the **/etc/master.d** files. A calculated parameter value is replaced by adding the parameter name and value to the appropriate **/etc/master.d** file. Setting the value of an otherwise calculated parameter replaces the calculated value with the set value each time a new kernel (**/unix**) is generated. When calculated parameter values are overridden, a subsequent change in hardware configuration must be accompanied by a change in the overridden parameter values to optimize the new configuration performance. Other parameter values are set equal to specific values.

The default value and the size in bytes for each entry are also shown in the figures. A dash (—) is used in the size information column to indicate parameters that only set flags in the kernel. Parameters that set flags do not affect the size of the kernel when their values are changed, only the values of the specific flags are changed. The size of the kernel is found by executing the command **size /unix**.

PARAMETER	EQUIPPED RAM			DEFAULT VALUE	SIZE PER ENTRY IN BYTES
	0.5 MEGABYTE	1 MEGABYTE	2 MEGABYTES		
NBUF	60	200	400	CALCULATED	1076
NCALL	30	30	30	30	16
NINODE	60	100	160	100	92
NFILE	60	100	160	100	12
NMOUNT	5	5	5	25	20
NPROC	30	40	60	60	72
NTEXT	28	38	58	58	16
NCLIST	60	80	120	80	72
MAXUP	20	25	25	25	—
SMAPSIZ	60	80	100	80	8
CMAPSIZ	60	80	100	80	8
NHBUF	32	128	256	128	12
NPBUF	4	4	4	8	52
NAUTOUP	10	10	10	10	—
MAXMEM	64	256	256	256	—
REL	2.0.3	2.0.3	2.0.3	2.0.3	—
NODE	unix	unix	unix	unix	—
SYS	unix	unix	unix	unix	—
VER	2	2	2	2	—
MSGMAP	100	100	100	100	8
MSGMAX	8192	8192	8192	8192	—
MSGMNB	16384	16384	16384	16384	—
MSGMNI	10	10	10	10	48
MSGSSZ	8	8	8	8	1024
MSGTQL	40	40	40	40	12
MSGSEG	1024	1024	1024	1024	8
SEMMAP	10	10	10	10	8
SEMMNI	10	10	10	10	32
SEMMNS	60	60	60	60	8
SEMMNU	30	30	30	30	8X(SEMUME+2)
SEMMSL	25	25	25	25	—
SEMOPM	10	10	10	10	8
SEMUME	10	10	10	10	240
SEMVMX	32767	32767	32767	326767	—
SEMAEM	16384	16384	16384	16384	—
SHMMAX	8192	8192	8192	8192	—
SHMMIN	1	1	1	1	—
SHMMNI	8	8	8	8	56
SHMSEG	4	4	4	4	—
SHMALL	32	32	32	32	—
FLCKREC	100	100	100	100	28
FLCKRIL	20	20	20	20	20

Figure 3-1. Parameter Values, 3B2/300 Computer

PARAMETER	EQUIPPED RAM				DEFAULT VALUE	SIZE PER ENTRY IN BYTES
	1 MEGABYTE	2 MEGABYTES	3 MEGABYTES	4 MEGABYTES		
NBUF	200	400	500	600	CALCULATED	1076
NCALL	30	30	30	30	30	16
NINODE	100	160	220	280	100	92
NFILE	100	160	220	280	100	12
NMOUNT	5	5	10	10	25	20
NPROC	40	60	80	100	60	72
NTEXT	38	58	78	98	58	16
NCLIST	80	120	160	200	80	72
MAXUP	25	25	30	30	25	—
SMAPSIZ	80	100	120	140	80	8
CMAPSIZ	80	100	120	140	80	8
NHBUF	128	256	256	256	128	12
NPBUF	4	4	8	8	8	52
NAUTOUP	10	10	15	15	10	—
MAXMEM	256	256	512	1024	256	—
REL	2.0.3	2.0.3	2.0.3	2.0.3	2.0.3	—
NODE	unix	unix	unix	unix	unix	—
SYS	unix	unix	unix	unix	unix	—
VER	2	2	2	2	2	—
MSGMAP	100	100	100	100	100	8
MSGMAX	8192	8192	8192	8192	8192	—
MSGMNB	16384	16384	16384	16384	16384	—
MSGMNI	10	10	10	10	10	48
MSGSSZ	8	8	8	8	8	1024
MSGTQL	40	40	40	40	40	12
MSGSEG	1024	1024	1024	1024	1024	8
SEMMAP	10	10	10	10	10	8
SEMMNI	10	10	10	10	10	32
SEMMNS	60	60	60	60	60	8
SEMMNU	30	30	30	30	30	8X(SEMUME+2)
SEMMSL	25	25	25	25	25	—
SEMOPM	10	10	10	10	10	8
SEMUME	10	10	10	10	10	240
SEMVMX	32767	32767	32767	326767	326767	—
SEMAEM	16384	16384	16384	16384	16384	—
SHMMAX	8192	8192	8192	8192	8192	—
SHMMIN	1	1	1	1	1	—
SHMMNI	8	8	8	8	8	56
SHMSEG	4	4	4	4	4	—
SHMALL	32	32	32	32	32	—
FLCKREC	100	100	100	100	100	28
FLCKRIL	20	20	20	20	20	20

Figure 3-2. Parameter Values, 3B2/400 Computer

Kernel Parameters

The following parameters are defined in the **/etc/master.d/kernel** file. The order in which they are described follows the order in which they are defined in the output of the **/etc/sysdef** command.

NBUF

Specifies the number of system buffers to allocate. Real-time response improves as more buffers are allocated. The UNIX System buffers form a data cache in RAM. The data cache is a memory array containing disk file information. Improvement in the hit rate of this cache increases with the number of buffers. Cache hits reduce the number of disk accesses.

If available memory is more than what is required for the application, the number of buffers may be increased. NBUF should normally be limited to 600 through 800 buffers. However, if NBUF is increased beyond 800, then the tunable parameter (NAUTOUP) which specifies the interval for buffer writes to disk must be increased to take advantage of the extra buffers. Otherwise, the system will spend most of its time writing buffers to disk. Consequently, the cache hit ratio may also be affected. Each buffer contains 1076 bytes. Hash buffers (NHBUF) should be increased along with system buffers (NBUF) for optimal performance. The value for NBUF is calculated automatically by the system, unless specifically set to a value by an entry in the **/etc/master.d/kernel** file. The calculated value for NBUF is 400 for a 2-megabyte system and never exceeds this value. For 3-megabyte or 4-megabyte systems, the system administrator is expected to increase the NBUF and NHBUF parameters according to the values provided in Figure 3-2.

NCALL Specifies the number of call-out table entries to allocate. Each entry represents a function to be invoked at a later time by the clock handler. This value must be greater than 2 and is normally in the range of 10 to 70. The default value is 30. Each entry contains 16 bytes. This parameter only controls the associated resource and has minimal affect on performance.

Only the kernel is allowed to use the call-out table. User programs are not allowed to use the clock handler. Software drivers may use call entries to check hardware device status. For example, the ports card driver uses one entry to check and see if the peripheral ports card has input information. When the call-out table overflows, the system crashes and outputs the following message on the system console.

```
PANIC: Timeout table overflow
```

NINODE Specifies the number of information-node table entries to allocate. Each table entry represents an in-core i-node that is an active file. For example, an active file might be a current directory, an open file, or a mount point. The file control structure is modified when changing this variable. The number of entries used depends on the number of opened files. The number of entries entries must exceed the maximum number of processes allowed (NPROC). Each NINODE entry contains 92 bytes. If processes manipulate fewer than 10 files on the average, the values specified for NINODE and NFILE can be halved. The value for NINODE pertains directly to the NFILE value. (NINODE is equal to or greater than NFILE). NINODE is a resource parameter and has little to contribute to system performance. When the i-node table overflows, the following warning message is output on the system console.

```
WARNING: inode table overflow
```

NFILE Specifies the number of open file table entries to allocate. Each entry represents an open file. Like NINODE, NFILE must exceed NPROC. Each entry contains 12 bytes. The NFILE entry relates directly to the NINODE entry. A value of NFILE larger than NINODE is of no value. The NFILE control structure operates in the same manner as the NINODE structure. The NFILE parameter is primarily a resource control. When the file table overflows, the following warning message is output on the system console.

```
NOTICE: File table overflow
```

As a reminder, this parameter does not affect the number of open files per process. The operating system allows 20 open files per process, the **/bin/sh** uses 3 processes: standard input, standard output, and standard error (0, 1, and 2 are normally reserved for stdin, stdout, and stderr, respectively).

NMOUNT Specifies the number of mount table entries to allocate. The number of table entries determines the number of file systems that can be mounted simultaneously. Each entry represents a mounted file system. Each entry has the size of 20 bytes. The root (/) file system is always the first entry. When full, the **mount**(2) system call returns the error EBUSY. The default value for NMOUNT is 25. The maximum value for NMOUNT is 100. Since the mount table is searched linearly, this value should be as low as possible.

NPROC Specifies the number of process table entries to allocate. Each table entry represents an active process. The swapper is always the first entry and **/etc/init** is always the second entry. The number of entries depends on the number of terminal lines available and the number of processes spawned by each user. The average number of processes per user is in the range of 2 to 5 (also see MAXUP, default value 25). When full, the **fork**(2) system call returns the error EAGAIN. The NPROC entry is only a resource control. However, system performance will decrease as more processes are allowed in the system. Each entry contains 72 bytes.

NTEXT Specifies the number of text table entries to allocate. Each entry represents an active read-only shared text segment. Every executing program (process) has a text table entry. Although, multiple copies of sharable text share a single entry. The value of NTEXT should never be set larger than NPROC minus 2 If NTEXT is larger than NPROC minus 2, the additional text table entires are never used. The special kernel processes named **swapper** and **init** *do not use NTEXT entries; hence the reason for NPROC minus 2. The NTEXT parameter is only for resource control. Each entry contains 16 bytes. When the table overflows, the following warning message is output on the system console.*

```
WARNING: out of text
```

NCLIST Specifies the number of character list buffers to allocate. Each buffer contains up to 64 bytes. The buffers are dynamically linked to form input and output queues for the terminal lines and other slow speed devices. The average number of buffers needed per terminal is in the range of 5 to 10. Each entry contains 72 bytes. When full, input and output characters dealing with terminals are lost, although echoing continues. If terminals attached to the system are run at high baud rates (9600 or 19200), NCLIST can be lowered to 50 percent of the recommended values.

MAXUP Specifies the number of concurrent processes a nonsuper-user is allowed to run. The entry is normally in the range of 15 to 25. This value should not exceed the value of NPROC (NPROC should be at least 10% more than MAXUP). This value is per user identification number, not per terminal. For example, if 12 people are logged in with the same user identification, the default limit would be reached very quickly. This parameter has almost no effect on individual process performance; however, it may allow one user to hog the machine if it is set too high.

SMAPSIZ Specifies the number of entries to allocate for the control list (swap space management table) used to manage free system swap space. The number of entries used depends on the number of processes active (NPROC), their sizes, and the amount of memory and swap space available. The SMAPSIZ value should be slightly larger than the NPROC value. Each entry contains 8 bytes. The following two error messages relate directly to this parameter.

```
NOTICE: swap space running out: needed # blocks

WARNING: out of swap space: needed # blocks
```

The first message indicates that insufficient space was found on the swap device when attempting to swap out a given process or swap out a copy of a shared text program. The number of blocks requested is given by #. After the first message is output on the console, an attempt is made to clean up the swap area. If this action is unsuccessful, the second message is output. The system may hang, crash, or it may recover and resume normal operations if swap space becomes available. If the system hangs, or crashes, reboot the system. This error is usually caused by an excessive user load on the system. If this problem persists, change the disk partitioning to increase the size of the swap area on the disk.

CMAPSIZ Specifies the number of entries to allocate to the core map (RAM) memory table. This table is used to manage available main memory space on the system. The number of entries depends on the number of processes active (NPROC), their sizes, and the amount of system (RAM) memory. Each entry contains 8 bytes. When the list overflows due to excessive fragmentation, the system outputs the following message on the console.

```
WARNING: mfree map overflow #. Lost # items at #
```

This message may apply to several different structures. The map overflow # is the hexadecimal address of the map structure that is too small. The command **nm -x|unix|grep #** is used to get the name of the map structure. If the name is CMAPSIZ, then increase the value of CMAPSIZ.

NHBUF Specifies the number of "hash buckets" to allocate. These are used to search for a buffer given a device number and block number rather than a linear search through the entire list of buffers. **This value must be a power of 2.** Each entry contains 12 bytes. As a general rule, choose NHBUF and NBUF such that the ratio NBUF/NHBUF is equal to the average number of disk read/write operations. If this information is not available, set NHBUF to a power of 2 that is within the limits of the following equation.

$$\frac{NBUF}{2} \leq NHBUF \leq NBUF\ 2$$

Note that a large value of NHBUF is preferred because a small value may not be able to utilize all of the available buffers (NBUF).

NPBUF Specifies the number of physical input/output buffers to allocate. One input/output buffer is needed for each physical read or write activity. The value is normally in the range of 4 to 8. Each entry contains 52 bytes. This value can be increased as needed if more than 4 to 8 physical read/write operations are expected to be active concurrently. The default value is 8.

NAUTOUP The NAUTOUP entry specifies the time interval in seconds for automatic file system updates. The system buffers are written to the hard disk at the interval specified by the NAUTOUP parameter. The default value is 10. Specifying a smaller limit decreases system performance and increases system reliability by writing the buffers to disk more frequently. Specifying a larger limit increases system performance at the expense of reliability.

MAXMEM Specifies the maximum amount of memory that a user can have per process. The size is in terms of the number of pages (one page equals 2048 bytes). A value larger than the size of physical memory is adjusted down to the maximum allowed value.

REL Specifies the UNIX System release. This value is 2.0.3.

NODE Specifies the node name of the system. The default node name is **unix**. Refer to the "ESTABLISHING/CHANGING THE SYSTEM AND NODE NAMES" task description for additional information.

SYS Specifies the system name. The default system name is **unix**. Refer to the "ESTABLISHING/CHANGING THE SYSTEM AND NODE NAMES" task description for additional information.

VER Specifies the hardware version. The default version is 2.

FLCKREC Specifies the number of records that can be locked by the system. The default value is 100. Each entry contains 28 bytes.

FLCKRIL Specifies the number of file headers configured for file locking. Each entry contains 20 bytes. The default value is 20.

Message Parameters

The following tunable parameters are associated with the message type of inter-process communication. These parameters are defined in the **/etc/master.d/msg** file. The order in which they are described follows the order in which they are defined in the output of the **/etc/sysdef** command.

MSGMAP Specifies the size of the control map used to manage message segments. Default value is 100. Each entry contains 8 bytes.

MSGMAX Specifies the maximum size of a message. The default value is 8192. The maximum size is 128 kilobytes.

MSGMNB Specifies the maximum length of a message queue. The default value is 16384.

MSGMNI Specifies the maximum number of message queues system-wide (id structure). The default value is 10.

MSGSSZ Specifies the size, in bytes, of a message segment. Messages consist of a contiguous set of message segments large enough to fit the text. The default value is 8. The value of MSGSSZ times the value of MSGSEG must be less than or equal to 131,072 bytes (128 kilobytes).

MSGTQL Specifies the number of message headers in the system and, thus, the number of outstanding messages. The default value is 200. Each entry contains 12 bytes.

MSGSEG Specifies the number of message segments in the system. The default value is 1024. The value of MSGSSZ times the value of MSGSEG must be less than or equal to 131,072 bytes (128 kilobytes).

Semaphore Parameters

The following tunable parameters are associated with the semaphore type of inter-process communication. These parameters are defined in the **/etc/master.d/sem** file. The order in which they are described follows the order in which they are defined in the output of the **/etc/sysdef** command.

SEMMAP Specifies the size of the control map used to manage semaphore sets. The default value is 10. Each entry contains 8 bytes.

SEMMNI Specifies the number of semaphore identifiers in the kernel. This is the number of unique semaphore sets that can be active at any given time. The default value is 10. Each entry contains 32 bytes.

SEMMNS Specifies the number of semaphores in the system. The default value is 60. Each entry contains 8 bytes.

SEMMNU Specifies the number of undo structures in the system. The default value is 30. The size is equal to 8 x (SEMUME + 2) bytes.

SEMMSL Specifies the maximum number of semaphores per semaphore identifier. The default value is 25.

SEMOPM Specifies the maximum number of semaphore operations that can be executed per **semop**(2) system call. The default value is 10. Each entry contains 8 bytes.

SEMUME Specifies the maximum number of undo entries per undo structure. The default value is 10. Each entry contains 240 bytes.

SEMVMX Specifies the maximum value a semaphore can have. The default value is 32767. The default value is the maximum value for this parameter.

SEMAEM Specifies the adjustment on exit for maximum value, alias **semadj**. This value is used when a semaphore value becomes greater than or equal to the absolute value of **semop**(2), unless the program has set its own value. The default value is 16384. The default value is the maximum value for this parameter.

Shared Memory Parameters

The following tunable parameters are associated with the shared memory type of inter-process communication. These parameters are defined in the **/etc/master.d/shm** file. The order in which they are described follows the order in which they are defined in the output of the **/etc/sysdef** command.

SHMMAX Specifies the maximum shared memory segment size. The default value is 8192.

SHMMIN Specifies the minimum shared memory segment size. The default value is 1.

SHMMNI Specifies the number of shared memory identifiers. The default value is 8. Each entry contains 56 bytes.

SHMSEG Specifies the number of attached shared memory segments per process. The default value is 4. The maximum value is 15.

SHMALL Specifies the maximum number of in-use shared memory text segments. The default value is 32.

How to Reconfigure the System

Caution 1: Always copy the existing bootable */unix* ***to*** */oldunix* ***so you will have a bootable operating system if you create an unbootable*** */unix*. ***If you create an unbootable*** */unix* ***and do not have a bootable version of the operating system on the hard disk, you will have to do a partial restore using the*** *3B2 Computer Core System* ***floppy disks.***

Caution 2: Execute a separate *mkboot* ***command for each driver or module. DO NOT execute*** *mkboot* ***with multiple modules (drivers) specified. The internal buffers used by the command are not cleared between modules when multiple modules are specified. Executing a separate*** *mkboot* ***command for each driver produces repeatable results for the purpose of binary comparisons.***

Normally, the stimulus for reconfiguring the system is either the addition of random access memory or UNIX System messages indicating insufficient table entries of some category.

The major steps in reconfiguring the operating system are as follows.

1. At the console terminal, log in as **root**.
2. Copy the existing **/unix** to **oldunix**. See Caution 1.
3. Change the present working directory to **/etc/master.d**.
4. Edit the applicable files in the **/etc/master.d** directory to modify (increasing or decreasing the value of) the tunable parameters.
5. Change the present working directory to **/boot**.

6. Execute the **/etc/mkboot** command to create a bootable object file for each of the files modified in **/etc/master.d**. The **-k** option must be used to create a bootable KERNEL object file. See Caution 2.

7. Take the system to the firmware mode and boot the **/etc/system** file.

> ***Note:*** An alternate boot procedure is to **touch** the **/etc/system** file and then execute **shutdown -i6**. This assumes that the time-of-day clock is sane. If the clock is insane, **touch 0101000070 /unix** file and then execute **shutdown -i6**. (January 1, 1970 is the beginning of time in the UNIX Operating System.) The major point is that the **/etc/system** file must have a more recent date (time) than the **/unix** file for the reboot process to generate a new operating system.

Sample System Reconfiguration

This sample system reconfiguration assumes that additional memory has been added to the 3B2 Computer. Self-configuration automatically sets the value of NBUF to the optimum value for the size of RAM. The new size of RAM is now 2 megabytes. Because of the additional memory, the following parameters are being increased from default values.

NBUF changed from 100 to 400.

> ***Note:*** The NBUF parameter value is calculated by the system reconfiguration program based on the amount of equipped RAM. NBUF is calculated to be 60, 200, and 400 for 0.5-, 1-, and 2-megabyte RAM configurations. NBUF will never exceed 400 even if 3-megabyte or 4-megabyte RAM configurations are used and must be changed manually if required.

NINODE changed from 100 to 160

NFILE changed from 100 to 160

SMAPSIZ changed from 80 to 100

CMAPSIZ changed from 80 to 100

NHBUF changed from 128 to 256

All of the parameters being modified are in the **/etc/master.d/kernel** file. The following command line entries and system responses show how to reconfigure the operating system to support these new parameters. The editing of the **/etc/master.d/kernel** file is not shown. The example begins with copying the **/unix** to **/oldunix**.

```
# cp /unix /oldunix<CR>
# cd /etc/master.d<CR>
# ed kernel<CR>

Note: Editing of kernel is not shown.

q<CR>
# cd /boot<CR>

Note: If parameters in other /etc/master.d files are changed,
execute mkboot on the uppercase name for each changed file.
Only the KERNEL file requires the -k option.

# mkboot -k KERNEL<CR>
# cd<CR>
# shutdown -y -i5<CR>

     series of messages are displayed
     ending with the following

INIT: New Run level: 5

The system is coming down.  Please wait.
System services are now being stopped.

The system is down.


SELF-CHECK

FIRMWARE MODE
<mcp><CR>
```

Continued

Continued from the previous screen

```
Enter name of program to execute [ ]: /etc/system<CR>
             Possible load devices are:

Option Number     Slot     Name
---------------------------------------
       0           0     FD5
       1           0     HD30
       2           0     HD30
       3           3     CTC

Enter Load Device Option Number [1 (HD30)]: <CR>

CONFIGURATION SUMMARY
=====================
     ----driver---- #devices major
     XT                  1     49
     PRF                 1     29
     SXT                 1     48
     CTC                 1      3
     TTY                 1     20
     PORTS               2      1,   2
     MEM                 1     19
     IUART               1      0
     IDISK               1     17
     HDELOG              1     16

     ----module---
     SHM
     SEM
     MSG
     IPC
     MAU
```

Continued

Continued from the previous screen

```
LOAD MAP
========
     --------section--------   --------address--------
     UNIX(boott)               p 2004000
     UNIX(bootd)               p 2004a6c
     UNIX(.gate)               p 2005800   v       0
     UNIX(.text)               p 2006000   v 40000000
     UNIX(.data)               p 202b000   v 40040000
     UNIX(.bss)                p 2030800   v 40080000
```

All of load map not shown.

```
     HDELOG(.bss)              p 2086d9c   v 400d659c
     END


UNIX System V Release 2.0.3 3B2 Version 2
unix
Copyright (c) 1984 AT&T.
All Rights Reserved
The system is coming up.  Please wait.
Generating a new /unix
AT&T 3B2 SYSTEM CONFIGURATION:

Memory size: 2 Megabytes
System Peripherals:

         Device Name         Subdevices

         SBD
                         Floppy Disk
                         30 Megabyte Disk
                         30 Megabyte Disk
         PORTS
         PORTS
         CTC
The system is ready.

Console Login:
```

Unbootable Operating System Recovery

If you create a **/unix** that is unbootable or operates so poorly that recovery while operating in that version is impossible, do the following.

1. Take the system to firmware by the best means possible. If you can use **/etc/shutdown**, do so; otherwise, use RESET.

2. Enter the firmware password and boot the **oldunix** that you should have made before reconfiguring the system. If you did not make this copy, reload the operating system using the **3B2 Computer Core System** floppy disks (select the partial restore). See the "RELOADING **UNIX** OPERATING SYSTEM" procedure in this chapter.

Supplementary Stimulus for Reconfiguration

If the optional 3B2 Computer Performance Measurements Utilities are available, they can be used to develop reports which will help in fine-tuning the system. Refer to the *AT&T 3B2 Computer Performance Measurements Utilities Guide* for complete information.

ESTABLISHING/CHANGING THE SYSTEM AND NODE NAMES

Caution: When basic networking is installed, changes in the system node name must be coordinated with all other computer systems that originate networking calls to your system. Incoming network calls will fail sequence checks at the originating system when the received node name and expected node name are different.

General

The system and node names are normally set to the same name when basic networking is installed on a system. Once basic networking is established, other computer systems use the node name as part of the communication protocol. The calling system compares the received node name with the name used in its **/usr/lib/uucp/Systems** file. Arbitrarily changing the node name of your system will cause sequence check failures for systems attempting to originate a transfer to your system. The "wrong system" status message in basic networking indicates that the received node name does not match the name in the **Systems** file for the system called.

The system and node names of the 3B2 Computer can be set by any one of the following means.

- Using the FLOPPY KEY to reset Nonvolatile Random Access Memory (NVRAM)
- Using the **uname** command
- Reconfiguring the operating system and defining the SYS and NODE names
- Using the **sysadm nodename** command.

When Simple Administration is used to establish the node name of a machine, a **nodename** file is created in the **/etc/rc.d** directory. The **/etc/rc.d/nodename** file contains a **uname -S** command. The **/etc/rc.d** files are executed by **/etc/rc2** on transitions to run-level 2. Figure 3-3 shows the default values used for NVRAM and system reconfiguration. The format used in Figure 3-3 is that of the fields displayed in the output of the **uname -a** command.

NAME SOURCE	SYSTEM NAME	NODE NAME	RELEASE	VERSION	HARDWARE NAME
uname -S ABcd5678	ABcd5678	ABcd5678	2.0.3	2	3B2
sysadm nodename	ABcd5678	ABcd5678	2.0.3	2	3B2
Floppy Key	unix	unix	2.0.3	2	3B2
Reconfiguration	unix	unix	2.0.3	2	3B2

Figure 3-3. System/Node Name Examples

"uname" Command

The **uname** command is used to output information about the UNIX System and to change the system and node names. This command changes both the system and node names to the name argument that you specify. The name argument must be eight or fewer characters. The following shows how to display/change the system and nodes names.

```
# uname -a<CR>
unix unix 2.0.3 2 3B2
# uname -S abcdefghijk<CR>
uname: name must be <= 8 letters
# uname -S ABcd5678<CR>
# uname -a <CR>
ABcd5678 ABcd5678 2.0.3 2 3B2
#
```

Figure 3-3 defines the output of the **uname -a** command for the values used in this example.

Floppy Key

When the system is powered up with the FLOPPY KEY, the contents of the Nonvolatile Random Access Memory (NVRAM) is reset. The NVRAM is also reset with the FLOPPY KEY when the system enters or leaves the firmware mode. Thus, a reboot with the FLOPPY KEY floppy disk in the integral floppy disk drive also resets NVRAM.

Executing the **uname -a** command after resetting NVRAM shows the new values. The NVRAM default values for the system and node names are as follows.

```
# uname -a<CR>
unix unix 2.0.3 2 3B2
#
```

Figure 3-3 defines the output of the **uname -a** command for the NVRAM values. Thus, using the FLOPPY KEY to reset the firmware password requires that the node name be reset using the **uname -S** command if basic networking is installed. If a **/etc/rc.d/nodename** file exists, the node name of the machine is automatically reset to the argument of the **uname -S** command when the machine is brought to run-level 2.

System Reconfiguration

The system name (SYS) and node name (NODE) are specified in the **/etc/master.d/kernel** file along with the release (REL), version (VER), and hardware machine (MACH) names. System reconfiguration default values in combination with NVRAM default values are used when making a new operating system if values are NOT specified in the kernel source **name.c** file. Figure 3-3 defines the output of the **uname -a** command for the system reconfiguration values. If a **/etc/rc.d/nodename** file exists, the node name of the machine is automatically reset to the argument of the **uname -S** command when the machine is brought to run-level 2.

Simple Administration nodename

The following command line entries and system responses show the setting of the node name using the **sysadm nodename** command. The node name is then output using the **uname** command. The contents of the **/etc/rc.d/nodename** file is the result of the execution of **sysadm nodename**.

```
# sysadm nodename<CR>

Running subcommand 'nodename' from menu 'syssetup',
SYSTEM SETUP

This machine is currently called "unix".
What name do you want to give it? [q] ABcd5678<CR>
# uname -a<CR>
ABcd5678 ABcd5678 2.0.3 2 3B2
# cat /etc/rc.d/nodename<CR>
#         Node name changed 09/01/84 14:55:49.
uname -S ABcd5678
#
```

SECURITY ADMINISTRATION

General

The security of the system is ultimately the responsibility of all who have access to the system. No system is totally secure. The system is not tamperproof. Some of the items to consider are as follows.

- Especially with a small computer, physical access to the machine must be considered. Anyone with physical access to the machine can literally walk off with it.
- Set the access permissions to directories and files to allow only the necessary permissions for owner, group, and others.
- All logins should have passwords. Change passwords regularly. Do not pick obvious passwords. Six-to-eight character nonsense strings using letters and numbers are recommended over standard names. Logins that are not needed should be either removed or blocked.
- Dial-up ports that do not have passwords usually cause trouble.
- Any system with dial-up ports is not really secure. Top secret information should not be kept on a system with dial-up ports.
- The **su** command is inherently dangerous to security, since knowledge of another login/password is required by the user. The more people who know a given login and password, the less secure access is to the system. Therefore, a log is kept on the use of the command. Check the **/usr/adm/sulog** to monitor use of the **su** command.
- Login directories, **.profile** files, and files in **/bin**, **/usr/bin**, **/lbin**, and **/etc** that are writable by others are security give-aways.

- Encrypt sensitive data files. The **crypt** command along with the encryption capabilities of the editors (**ed** and **vi**) provide protection for sensitive information. This capability is provided by the Security Administration Utilities that is available only within the United States.

- Log off the system if you must be away from the data terminal. Do not leave a logged-in terminal unattended, especially if you are logged in as **root**.

Password Aging

General

The password aging mechanism forces users to change their password on a periodic basis. Provisions are made to prevent a user from changing a new password before a specified time interval. Password aging is selectively applied to logins by editing the **/etc/passwd** file. Realistically, password aging forces a user to adopt at least two passwords for a login. If you require more access control than what is provided by password aging, you can change **/etc/profile** to require a second access code as part of the login process.

The password aging information is appended to the encrypted password field in the **/etc/password** file. The password aging information consists of a comma and up to four bytes (characters) that specify:

- The duration of the password,
- The time interval before the existing password can be changed by the user,
- The week (counted from the beginning of 1970) when the password was last changed. You do not enter this information. The system automatically adds these characters to the password aging information.

All times are specified in weeks (0 through 63) by a 64 character alphabet. Figure 3-4 shows the relationship between the numerical values and character codes.

CHARACTER	NUMBER OF WEEKS
. (period)	0 (zero)
/ (slash)	1
0 through 9	2 through 11
A through Z	12 through 37
a through z	38 through 63

Figure 3-4. Password Aging Character Codes

In establishing the password aging information, the following syntax applies where **M** is the duration of the valid password and **m** is the minimum number of weeks before a change can be made to a new password.

- The password aging information in the form of **,Mm** is appended to the 13 bytes of the encrypted password in the **/etc/passwd** file.
- If **M** and **m** are equal to zero, the user is forced to change the password at the next log in. No further password aging is then applied to that login.
- If **m** is greater than **M**, only **root** is able to change the password for that login.

Sample /etc/passwd Entries

The following shows the password aging information required to establish a new password every 2 weeks (0) and to deny changing the new password for 1 week (/). In this example **M** (0) is greater than **m** (/). This information (**,0/**) is appended to the encrypted password field by editing the **/etc/passwd** file. The first line shows a passworded login entry in the **/etc/passwd** file. The second line shows the addition of the password aging information to the **jqu** login. After the login entry is changed, the user (jqu) is required to change the password at the next log in and every 2 weeks thereafter. After the first log in following the change, the third line shows the addition of the "last change information" to the password field. The added password aging information is shown in bold print.

```
jqu:RTKESmMOE2m.E:100:1:J. Q. Username:/usr/jqu:
jqu:RTKESmMOE2m.E,0/:100:1:J. Q. Username:/usr/jqu:
jqu:.K.d779.mfkNo,0/W9:100:1:J. Q. Username:/usr/jqu:
```

The following shows the password aging information required to establish a new password when the user logs in and then disappears (**M** and **m** equal zero weeks). This information (,,..) is appended to the encrypted password field by editing the **/etc/passwd** file. The first line shows a passworded login entry in the **/etc/passwd** file. The second line shows the addition of the password aging information to the **jqu** login. After the login entry is changed, the user (jqu) is required to change the password at the next log in. After the new password is supplied, the password aging information disappears as shown in the third line. Note that the encrypted password information has changed in the third line.

```
jqu:RTKESmMOE2m.E:100:1:J. Q. Username:/usr/jqu:
jqu:RTKESmMOE2m.E,..:100:1:J. Q. Username:/usr/jqu:
jqu:EFDNLqsFUj.fs:100:1:J. Q. Username:/usr/jqu:
```

The following shows the password aging information required to establish a password for a login such that only **root** can change the password. In this example, **M** (.) is less than **m** (/). This information (,./) is appended to the encrypted password field by editing the **/etc/passwd** file. The first line shows a passworded login entry in the **/etc/passwd** file. The second line shows the addition of the password aging information to the **jqu** login. Only **root** can change the password for the **jqu** login. If the user tries to change the password, a permission denied message is displayed.

```
jqu:RTKESmMOE2m.E:100:1:J. Q. Username:/usr/jqu:
jqu:RTKESmMOE2m.E,./:100:1:J. Q. Username:/usr/jqu:
```

Set-UID and Set-GID

The set-user identification (set-UID) and set-group identification (set-GID) must be used very carefully if any security is to be maintained. Certain programs are conditioned to execute as **root**. A writable set-UID file can have another program copied onto it. For example, if the switch user (**su**) command has the write access permission allowed for others, anyone can copy the shell onto it and get a password-free version of **su**. The following paragraphs provide a few examples of command lines that can be used to identify the files with a set-UID.

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

```
# find / -user root -perm -4100 -exec ls -l {} \; | mail root<CR>
you have mail
# mail<CR>
From root Mon Aug 27 07:20 EDT 1984
-r-sr-xr-x   1 root     bin        38836 Aug 10 16:16 /usr/bin/at
-r-sr-xr-x   1 root     bin        19812 Aug 10 16:16 /usr/bin/crontab
-r-sr-xr-x   1 root     bin        27748 Aug 10 16:16 /usr/bin/shl
---s--x--x   1 root     sys        46040 Aug 10 15:18 /usr/bin/ct
-r-sr-xr-x   1 root     sys        12092 Aug 11 01:29 /usr/lib/mv_dir
-r-sr-sr-x   1 root     bin        33208 Aug 10 15:55 /usr/lib/lpadmin
-r-sr-sr-x   1 root     bin        38696 Aug 10 15:55 /usr/lib/lpsched
---s--x---   1 root     rar        45376 Aug 18 15:11 /usr/rar/bin/sh
-r-sr-xr-x   1 root     sys        11416 Aug 11 01:26 /bin/mkdir
-r-sr-xr-x   1 root     sys        11804 Aug 11 01:26 /bin/rmdir
-r-sr-xr-x   1 root     bin        12524 Aug 11 01:27 /bin/df
-rwsr-xr-x   1 root     sys        21780 Aug 11 01:27 /bin/newgrp
-r-sr-sr-x   1 root     sys        23000 Aug 11 01:27 /bin/passwd
-r-sr-xr-x   1 root     sys        23824 Aug 11 01:27 /bin/su

? d<CR>
#
```

The following command line reports all files with a set-UID for a file system. The **ncheck** command, by itself, can be used on a mounted or unmounted file system. The normal output of the **ncheck -s** command includes special files. The **grep** command is used to remove the device files from the output. The filtering done in this example to remove the device files is applicable only for the root file system (**/dev/dsk/c1d0s0**). The output of the modified **ncheck** is used as an argument to the **ls** command. The use of the **ls** command is possible only if the file system is mounted.

```
# ls -l `ncheck -s /dev/dsk/c1d0s0 | cut -f2 | grep -v dev`<CR>
-r-sr-xr-x   1 root     bin         12524 Aug 11 01:27 /bin/df
-rwxr-sr-x   1 root     sys         32272 Aug 10 15:53 /bin/ipcs
-r-xr-sr-x   2 bin      mail        32852 Aug 11 01:28 /bin/mail
-r-sr-xr-x   1 root     sys         11416 Aug 11 01:26 /bin/mkdir
-rwsr-xr-x   1 root     sys         21780 Aug 11 01:27 /bin/newgrp
-r-sr-sr-x   1 root     sys         23000 Aug 11 01:27 /bin/passwd
-r-xr-sr-x   1 bin      sys         27964 Aug 11 01:28 /bin/ps
-r-xr-sr-x   2 bin      mail        32852 Aug 11 01:28 /bin/rmail
-r-sr-xr-x   1 root     sys         11804 Aug 11 01:26 /bin/rmdir
-r-sr-xr-x   1 root     sys         23824 Aug 11 01:27 /bin/su
-r-xr-sr-x   1 bin      sys         21212 Aug 10 16:08 /etc/whodo
#
```

The following command line entry shows the use of the **ncheck** command to examine the user file system (**/dev/dsk/c1d0s2**) for files with a set-UID. In this example, the complete path names for the files start with **/usr**. The **/usr** is not part of the **ncheck** output. In this example, the **/usr/rar/bin/sh** should be investigated.

```
# ncheck -s /dev/dsk/c1d0s2 | cut -f2<CR>
/dev/dsk/c1d0s2:
/bin/at
/bin/crontab
/bin/shl
/bin/sadp
/bin/timex
/bin/cancel
/bin/disable
/bin/enable
/bin/lp
/bin/lpstat
/bin/ct
/bin/cu
/bin/uucp
/bin/uuname
/bin/uustat
/bin/uux
/lib/mv_dir
/lib/expreserve
/lib/exrecover
/lib/accept
/lib/lpadmin
/lib/lpmove
/lib/lpsched
/lib/lpshut
/lib/reject
/lib/mailx/rmmail
/lib/sa/sadc
/lib/uucp/uucico
/lib/uucp/uusched
/lib/uucp/uuxqt
/rar/bin/sh
#
```

Blocking Unused Logins

Logins that are not used or needed should be either removed from the **/etc/passwd** file or the ability to use the logins should be blocked (disabled). A login is blocked by editing the **/etc/passwd** file and changing the encrypted password field to contain one or more characters that are not used by the encryption process. One way to do this is to use the expression **Locked;**. The text serves to remind you that the login is blocked. The semicolon (;) is an unused encryption character. A space is another character that is not used by the encryption process. Therefore, the expression **not valid** could also be used to block the use of a login. The following line entry from the **/etc/passwd** file shows the "blocked" **bin** login.

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

FORGOTTEN ROOT PASSWORD RECOVERY

General

The execution of certain commands and the performance of certain system tasks require the use of the **root** login. Not being able to access the system as **root** prevents the performance of certain important system tasks. Some of these tasks include:

- Execution of the **/etc/shutdown** command
- File system checking and repair
- Installation and removal of utilities
- Proper execution of file system backups and restorals.

Root Password Recovery Procedure

The following steps are necessary to recover the ability to log in as **root** based on the assumptions that you are not able to log in and restore the **/etc/passwd** file from another login.

1. Depress the power switch to STANDBY. Wait for the power-down sequence to complete.

2. Power up the system. When the word DIAGNOSTICS is output, depress the RESET switch. This action takes the system to the firmware mode.

3. Enter the firmware password (**mcp** is the default password).

4. Reload the UNIX Operating System from floppy disks (**3B2 Computer Core System** floppy disks 1 through 5), selecting the partial restore option. Refer to the "RELOADING **UNIX** OPERATING SYSTEM" procedure described in this chapter.

5. Log in as **root** and restore the root (/) file system from **/usr/old** directory, as required. Refer to the "RELOADING **UNIX** OPERATING SYSTEM" procedure described in this chapter.

6. Execute **passwd** to change the root password.

7. Store the **3B2 Computer Core System** floppy disks in a safe place.

FORGOTTEN FIRMWARE PASSWORD RECOVERY

General

The **FLOPPY KEY** floppy disk is read by the machine on transition to and from the firmware mode. The following procedure can be done by anyone with access to the machine and the **FLOPPY KEY** floppy disk. Knowledge of any logins or passwords is not required.

If you have forgotten your firmware password, get your **FLOPPY KEY** (the special floppy disk made during initial turn-on of the machine) and do the following:

1. Turn off the 3B2 Computer by depressing the power switch to STANDBY.

2. After the power has been removed, insert the **FLOPPY KEY** into the integral floppy disk drive.

3. Depress the power switch to the ON position.

4. The FLOPPY KEY can be removed from the drive after the FW WARNING message is output. Store the FLOPPY KEY in a safe place.

The following messages are output when the system is powered up using the **FLOPPY KEY**.

```
SELF-CHECK

FW WARNING:  NVRAM DEFAULT VALUES ASSUMED


DIAGNOSTICS  PASSED


UNIX System V Release 2.0.3 3B2 Version 2
unix
Copyright (c) 1984 AT&T.
All Rights Reserved

Time of Day Clock needs Restoring:
Change Using "sysadm datetime" utility
The system is coming up.  Please wait.
Generating a new /unix
AT&T 3B2 SYSTEM CONFIGURATION:

Memory size: 2 Megabytes
System Peripherals:

         Device Name         Subdevices

         SBD
                         Floppy Disk
                         30 Megabyte Disk
                         30 Megabyte Disk
         PORTS
         PORTS
         CTC
The system is ready.

Console Login:
```

Note 1: When NVRAM is cleared, the firmware password is reset to its default value (**mcp**), and if the bits-per-second rate was previously changed, it is reset to 9600. The time-of-day clock must be restored. Refer to the "SETTING TIME-OF-DAY/DATE CLOCK" procedure in this chapter.

Note 2: A feature of the 3B2 Computer is a battery powered calendar that preserves the correct time and date even when the power is turned off. If anything does happen to the battery, or if NVRAM is cleared, the time and date will be lost and the ''Time of Day Clock needs Restoring'' message will be displayed during powerup.

FIRMWARE PROCEDURES

General

Firmware procedures include the use of the firmware-resident programs and the use of the bootable programs provided as part of the Essential Utilities. The firmware-level programs include the following firmware programs (commands):

edt	Display equipped device table
newkey	Make floppy key
passwd	Change firmware password
sysdump	Dump system image to floppy disks
version	Display firmware version.

The bootable programs are **dgmon**, **filledt**, and **unix**. All of these programs are accessible in response to the prompt "*Enter name of program to execute ....*" firmware prompt. When the "*Enter path name:*" prompt is displayed, only bootable programs on the defined boot device are accessible. When in the firmware mode, the erase character is the backspace and the kill character is the at sign (@).

Display Firmware Program Menu

The firmware program menu is displayed by entering a question mark (?) in response to the "*Enter name of program to execute ...*" message displayed when the firmware mode is initiated. The following command line entries and system responses show how to display the firmware program menu. At the start of the example, the system has been taken to the firmware mode (run-level 5). As a security measure, echo is turned off by programs requesting a password entry. Echo is enabled after the password is entered. Therefore, the characters entered for the firmware password are not displayed. The default firmware password is **mcp**.

```
FIRMWARE MODE

<password><CR>

Enter name of program to execute [    ]: ?<CR>

Enter an executable or system file, a directory name,
or one of the possible firmware program names:

edt     newkey     passwd     sysdump     version     q(uit)


Enter name of program to execute [    ]:
```

Display Bootable Device Programs

By definition, bootable programs are installed at root on the boot device. There the root directory (/) is normally the directory of interest. The contents of the root directory for a bootable device are displayed by entering a null (carriage return, only) in response to the "*Enter name of program to execute ...*" message displayed when the firmware mode is initiated. The bootable device is then selected. The following command line entries and system responses show how to display the contents of a bootable device. At the start of the example, the system has been taken to the firmware mode (run-level 5). Directories are identified by a trailing slash (/) in contents listing. At the end of the example, the UNIX Operating System is booted for the selected device (the first hard disk). The bootable programs provided as part of the Essential Utilities are **dgmon**, **filledt**, and **unix**. Refer to the "RUNNING DIAGNOSTICS" task description for an example of booting the **dgmon** program.

```
FIRMWARE MODE

<password><CR>

Enter name of program to execute [    ]: <CR>
     Possible load devices are:

Option Number   Slot     Name
-------------------------------------
       0         0      FD5
       1         0      HD30
       2         0      HD30
       3         0      CTC

Enter Load Device Option Number [1 (HD30)]: <CR>
 :No such file or directory

Enter path name: <CR>
Files in / are:
bck/            bin/         boot/           dev/
dgmon           dgn/         etc/            filledt
install/        lib          lost+found      mnt/
save/           tmp/         unix            usr/
usr2/           usr3/        usr4/

Enter path name: unix

UNIX System V Release 2.0.3 3B2 Version 2
unix
Copyright (c) 1984 AT&T.
All Rights Reserved

        fsstat: root file system needs checking
        The root file system (/dev/dsk/c1d0s0) is being checked automatically.

                The remaining Output messages vary among systems.
                The system is ready when the following messages
                are displayed.

The system is ready.

Console Login:
```

Display Equipped Device Table

The current Equipped Device Table (EDT) is displayed using the firmware **edt**(8) program. The following command line entries and system responses show how to display the equipped device table. The system is in the firmware mode (run-level 5) at the start of the example. A password is required to enter the firmware mode. The default firmware password is **mcp**.

```
FIRMWARE MODE

<password><CR>

Enter name of program to execute [    ]: edt<CR>


Current System Configuration

System Board memory size: 4 megabyte(s)

00 - device name = SBD       , occurrence =  0, slot = 00, ID code = 0x01
     boot device = y, board width = double, word width = 2 byte(s),
     req Q size = 0x00, comp Q size = 0x00, console ability = y, pump file = n
     subdevice(s)
     #00 = FD5       , ID code = 0x01, #01 = HD30      , ID code = 0x03
     #02 = HD30      , ID code = 0x03

Press any key to continue<CR>


01 - device name = PORTS     , occurrence =  0, slot = 01, ID code = 0x03
     boot device = n, board width = single, word width = 2 byte(s),
     req Q size = 0x03, comp Q size = 0x23, console ability = n, pump file = y

Press any key to continue<CR>


02 - device name = PORTS     , occurrence =  1, slot = 02, ID code = 0x03
     boot device = n, board width = single, word width = 2 byte(s),
     req Q size = 0x03, comp Q size = 0x23, console ability = n, pump file = y

Press any key to continue<CR>


03 - device name = CTC       , occurrence =  0, slot = 03, ID code = 0x05
     boot device = y, board width = single, word width = 2 byte(s),
     req Q size = 0x10, comp Q size = 0x20, console ability = n
     subdevice(s)
     #00 = FT25      , ID code = 0x04, #01 = FD5       , ID code = 0x01
DONE


Enter name of program to execute [    ]:
```

Make a Floppy Key

A floppy key is a floppy disk used to back up the certain system parameters stored in Nonvolatile Random Access Memory (NVRAM). These parameters include the system name, node name, speed of the console terminal, and the firmware password. A floppy key is used to reset these parameters to known values. For example, the floppy key is used to recover from a forgotten firmware password. The **newkey**(8) program is used to make a floppy key. The following command line entries and system responses show how to make a floppy key. The default firmware password is **mcp**.

```
FIRMWARE MODE

<password><CR>

Enter name of program to execute [    ]: newkey<CR>

Creating a floppy key to enable clearing of saved NVRAM information

Insert a formatted floppy, then type 'go' (q to quit): go<CR>

Creation of floppy key complete

Enter name of program to execute [    ]:
```

Change Firmware Password

The **passwd**(8) program is used to change the firmware password. The existing firmware password must be known to change the password. The following command line entries and system responses show how to change the firmware password. As a security measure, echo is turned off by programs requesting a password entry. Echo is enabled after password entry. Therefore, the characters entered for the firmware password are not displayed. The new password must be entered twice (confirmation).

```
FIRMWARE MODE

<password><CR>

Enter name of program to execute [    ]: passwd<CR>

enter old passwd: <old password><CR>
enter new passwd: <new password><CR>
confirmation: <new password><CR>


Enter name of program to execute [    ]:
```

Dump System Image to Floppy Disk(s)

The **sysdump**(8) program is used to write the system image to floppy disk(s). Three formatted floppy disks are needed to dump the contents of a 2-megabyte memory. Six formatted floppy disks are needed to dump the contents of a 4-megabyte memory. Existing floppy disks can be used; however, the contents of the floppy disks are overwritten. The time required to dump the system image is about 5 to 6 minutes per floppy disk.

If a system dump is to be executed following a crash, the firmware mode must be entered and the **sysdump** program executed as the first action taken. Rebooting the system or executing another program will overwrite the system image, making subsequent crash analysis a futile effort. Refer to the *AT&T 3B2 Computer Crash Analysis Guide* for more information.

The following command line entries and system responses show the execution of the **sysdump** on a system with 2 megabytes of memory. The default password is **mcp**.

```
FIRMWARE MODE

<password><CR>

Enter name of program to execute [ ]: sysdump<CR>

Do you want to dump the system image to the floppy diskette?
Enter 'c' to continue, 'q' to quit: c<CR>

Insert first sysdump floppy.
Enter 'c' to continue, 'q' to quit: c<CR>

Dumping mainstore
................................................................................
............................................................
If you wish to dump more of mainstore,
insert new floppy.

Enter 'c' to continue, 'q' to quit: c<CR>

Dumping more main store
................................................................................
............................................................
If you wish to dump more of mainstore,
insert new floppy.

Enter 'c' to continue, 'q' to quit: c<CR>

Dumping more main store
................................................................................
.....................................
Dump completed.
three floppies written

Returning to firmware

SELF-CHECK

FIRMWARE MODE
```

Display Firmware Version

The **version**(8) program is used to display the firmware version. The following command line entries and system responses show how to display the firmware version.

```
FIRMWARE MODE

<password><CR>

Enter name of program to execute [ ]: version<CR>

Created: 07/24/84
Issue: 0b0b0b0b
Release: 1.1
Load: 1S4.3
Serial Number: 0b300016


Enter name of program to execute [ ]:
```

Fill Equipped Device Table (Boot filledt)

The bootable **filledt**(8) program is executed to build the equipped device table from data in the **/etc/edt_data** file. The following command line entries and system responses show how to manually build the equipped device table. Refer to Appendix C, "ERROR MESSAGES," for descriptions of the FILLEDT error messages.

```
FIRMWARE MODE

<password><CR>

Enter name of program to execute [ ]: filledt<CR>
             Possible load devices are:

Option Number    Slot      Name
---------------------------------------
       0          0       FD5
       1          0       HD30
       2          0       HD30
       3          3       CTC

Enter Load Device Option Number [1 (HD30)]:

BEGIN FILLING EDT

EDT SUCCESSFULLY COMPLETED

CONSOLE VALUES:
slot = 0, port = 0
cflags = 0x4bd

Enter name of program to execute [ ]:
```

Boot the Operating System

The UNIX Operating System can be booted in either of two ways: **unix** or **/etc/system**. Booting **unix** loads and runs the **/unix** file on the boot device. Booting **/etc/system** causes a new operating system core image to be generated. The new system configuration and load map are displayed when **/etc/system** is booted. Auto-configuration (**/etc/rc.d/autoconfig**) copies the operating system core image to the **/unix** file on the boot device. Refer to the "SYSTEM RECONFIGURATION" task description for an example of booting **/etc/system**.

MAKING A BOOTABLE DEVICE

General

The devices that can be used as a boot device are the integral floppy disk (diskette1), the integral hard disks (hard disks 1 and 2), and the cartridge tape drive (ctape1). The second floppy disk (diskette2) in a 3B2 Expansion Module CANNOT be used as a boot device even though it appears in the subdevice option list.

A device is made bootable by copying the applicable boot programs to the boot partition (7) of the device. The **/etc/newboot** command is used to write these programs to the boot partition of the device. Except for a floppy disk, the boot partition must be at least 100 blocks (512-byte blocks). The floppy disk boot partition must be at least 50 blocks.

The major steps in making a device bootable are as follows.

1. Format/partition the media to include a boot partition if not already properly formatted/partitioned.
2. Copy the applicable boot programs to the boot partition using the **/etc/newboot** command.
3. If not already created, make a file system on the applicable device partition.
4. Label the file system.
5. Mount the applicable partition and copy the program(s) to be booted to the file system. Also copy any supporting files required by the programs.
6. Unmount the file system when you are finished copying programs and data.

7. If the bootable device is a removable media (floppy disk or cartridge tape), clearly label the floppy disk or cartridge tape.

Making a Bootable Floppy Disk

The important points to remember in making a bootable floppy disk are as follows.

- The boot partition for the integral floppy disk is c0s0s7.
- The file system partition is c0d0s5 and has a maximum of 1404 blocks.
- The **newboot** command must specify **/lib/olboot** and **/lib/mboot**.

The following shows how to make a bootable floppy disk for running diagnostics. The floppy disk is first formatted, if not already formatted, using the **/etc/fmtflop** command. The boot programs are copied to boot partition (/dev/rdsk/c0d0s7) using the **/etc/newboot** command. After making a file system on partition 5 (/dev/dsk/c0d0s5), the partition is labeled to identify the floppy disk. Partition 5 (c0d0s5) is then mounted as **/mnt** and the applicable programs are copied to the /mnt file system. The floppy disk file system (c0d0s5) is then unmounted using the **/etc/umount** command. The system is then shut down to the firmware mode and the diagnostic monitor (**dgmon**) is booted from the floppy disk.

```
# fmtflop -v /dev/rdsk/c0s0s6<CR>
# newboot /lib/olboot /lib/mboot /dev/rdsk/c0d0s7<CR>
newboot: confirm request to write boot programs to /dev/rdsk/c0d0s7: y<CR>
# mkfs /dev/dsk/c0d0s5 1404 1 18<CR>
Mkfs: /dev/dsk/c0d0s5?
(DEL if wrong)
bytes per logical block = 1024
total logical blocks = 702
total inodes = 160
gap (physical blocks) = 1
cylinder size (physical blocks) = 18
mkfs: Available blocks = 689
# labelit /dev/rdsk/c0d0s5 dgn 2032<CR>
Current fsname: , Current volname: , Blocks: 1404, Inodes: 160
FS Units: 1Kb, Date last mounted: Thu Mar 28 14:19:08 1985
NEW fsname = dgn, NEW volname = 2032 -- DEL if wrong!!
# mount /dev/dsk/c0d0s5 /mnt<CR>
mount: warning! <dgn> mounted as </mnt>
# find /dgmon /dgn /filledt -print | cpio -pdumv /mnt<CR>
/mnt/dgmon
/mnt/dgn/edt_data
/mnt/dgn/SBD
/mnt/dgn/X.SBD
/mnt/dgn/PORTS
/mnt/dgn/X.PORTS
/mnt/dgn/CTC
/mnt/dgn/X.CTC
/mnt/filledt
442 blocks
# umount /dev/dsk/c0d0s5<CR>
```

Continued

Continued from the previous screen

```
# shutdown -y -g0 -i5<CR>

                    series of messages are displayed
                    ending with the following

FIRMWARE MODE
<mcp><CR>

Enter name of program to execute [   ]: dgmon<CR>
             Possible load devices are:

Option Number   Slot     Name
---------------------------------------
       0          0      FD5
       1          0      HD30
       2          0      HD30
       3          3      CTC

Enter Load Device Option Number [1 (HD30)]: 0<CR>


        3B2 DIAGNOSTIC MONITOR
DGMON >
```

Making a Bootable Cartridge Tape

The important points to remember in making a bootable cartridge tape are as follows.

- The boot partition is c?d0s7. The question mark (?) represents the controller number for the Cartridge Tape Controller card.
- The **newboot** command must specify **/lib/lboot** and **/lib/mboot**.
- The file system partition is c?d0s0 and has a maximum of 8928 blocks. The question mark (?) represents the controller number for the Cartridge Tape Controller card. Make the file system no larger than necessary to accommodate the application.
- A bootable cartridge tape is configured as a file system. As such in this application, the cartridge tape must be re-tensioned every four hours. This requires that the system be returned to the firmware mode and the cartridge tape drive latch operated.

The following shows how to make a bootable cartridge tape for running diagnostics. The cartridge tape is first formatted, if not already formatted, using the **/etc/ctcfmt** command. The boot programs are copied to boot partition (/dev/rdsk/c4d0s7) using the **/etc/newboot** command. After making a file system on the root partition (/dev/dsk/c4d0s0), the partition is labeled to identify the cartridge tape. Partition 0 (/dev/dsk/c4d0s0) is then mounted as **/mnt** and the applicable programs are copied to the **/mnt** file system. The cartridge tape file system is then unmounted using the **/etc/umount** command. The system is shut down to the firmware mode and the diagnostic monitor (**dgmon**) is booted from the cartridge tape.

```
# ctcfmt -vt /dev/rSA/ctape1&<CR>
2723
#
Insert tape, wait for re-tension pass to complete,
and press the <RETURN> key when ready [q]: Format completed successfully.

# newboot /lib/lboot /lib/mboot /dev/rdsk/c4d0s7<CR>
newboot: confirm request to write boot programs to /dev/rdsk/c4d0s7: y<CR>
# mkfs /dev/dsk/c4d0s0 2000 1 31<CR>
Mkfs: /dev/dsk/c4d0s0?
(DEL if wrong)
bytes per logical block = 1024
total logical blocks = 1000
total inodes = 240
gap (physical blocks) = 1
cylinder size (physical blocks) = 31
mkfs: Available blocks = 982
# labelit /dev/rdsk/c4d0s0 dgn 2032<CR>
Current fsname: , Current volname: , Blocks: 2000, Inodes: 240
FS Units: 1Kb, Date last mounted: Thu Mar 29 09:12:51 1985
NEW fsname = dgn, NEW volname = 2032 -- DEL if wrong!!
# mount /dev/dsk/c4d0s0 /mnt<CR>
# find /dgmon /dgn /filledt -print | cpio -pdumv /mnt<CR>
/mnt/dgmon
/mnt/dgn/edt_data
/mnt/dgn/SBD
/mnt/dgn/X.SBD
/mnt/dgn/PORTS
/mnt/dgn/X.PORTS
/mnt/dgn/CTC
/mnt/dgn/X.CTC
/mnt/filledt
442 blocks
# umount /dev/dsk/c4d0s0<CR>
```

Continued

```
Continued from the previous screen

# shutdown -y -g0 -i5<CR>

                    series of messages are displayed
                    ending with the following

FIRMWARE MODE
<mcp><CR>

Enter name of program to execute [   ]: dgmon<CR>
             Possible load devices are:

Option Number   Slot     Name
---------------------------------------
       0          0     FD5
       1          0     HD30
       2          0     HD30
       3          3     CTC

Enter Load Device Option Number [1 (HD30)]: 3<CR>
Possible subdevices are:

Option Number   Subdevice     Name
---------------------------------------
       0          0           FT25
       1          0           FD5

Enter Load Device Option Number [0(FT25)]: <CR>


        3B2 DIAGNOSTIC MONITOR
DGMON >
```

Making the Second Hard Disk Bootable

The important points to remember in making the second hard disk bootable are as follows.

- The boot partition is c1d1s7 and must be 100 blocks, minimum.
- The **newboot** command must specify **/lib/lboot** and **/lib/mboot**.
- The swap partition is c1d1s1 and is sized to make the subsequent partitions fall on partition boundaries.
- The file system partition is c1d1s0 and has a block size of whatever you assigned when you partitioned the disk. File system partitions should begin on partition boundaries. The **/etc/prtvtoc** command is used to display the current partition values.
- Copy any existing file systems to floppy disks or cartridge tape backup before repartitioning an existing second hard disk.
- If configuring the second hard disk to boot the UNIX Operating System, remember to edit the **/etc/fstab** file on the second disk to define the appropriate file system mount information.
- The first time the UNIX Operating System is booted from another device, **/etc/system** must be booted.
- The first time the UNIX Operating System is booted from another device, the swap partition and the swap device are a mismatch. An error message is output when the operating system is booted for the first time from the new device. The swap device is changed automatically.
- The UNIX System Core Full Restore process supports only the configuration of root and swap being on the first hard disk drive.

The following example shows how to make the second hard disk bootable for running the UNIX Operating System. The second hard disk is first partitioned using the **/etc/fmthard** command. In this example, the default volume table of contents for the first hard disk is used for the partition values by using the output of the **/etc/prtvtoc** command to create a VTOC file for the use with **/etc/fmthard** command. The boot programs are copied to boot partition (/dev/rdsk/c1d1s7) using the **/etc/newboot** command. The root (/) file system (c1d0s0) is copied to partition 0 (c1d1s0) using the **/etc/volcopy** command. The root file system on the second disk is then checked using **/etc/fsck**. The system is then shut down to the firmware mode and **/etc/system** is booted from the second disk.

```
# prtvtoc  /dev/rdsk/c1d0s0 | tee /etc/vtoc/newvtoc<CR>
         0      2    00        6120          14400
         1      3    01         100           6020
         2      4    00       20520          42030       /usr
         6      0    01           0          62550
         7      0    01           0            100
# fmthard -m -s /etc/vtoc/newvtoc -n root -v /dev/rdsk/c1d1s0<CR>
fmthard:  New volume table of contents now in place.
+(fmthard) mkfs /dev/rdsk/c1d1s0 14400 9 90
Mkfs: /dev/rdsk/c1d1s0?
(DEL if wrong)
bytes per logical block = 1024
total logical blocks = 7200
total inodes = 1792
gap (physical blocks) = 9
cylinder size (physical blocks) = 90
mkfs: Available blocks = 7085
+(fmthard) mkfs /dev/rdsk/c1d1s2 42030 9 90
Mkfs: /dev/rdsk/c1d1s2?
(DEL if wrong)
bytes per logical block = 1024
total logical blocks = 21015
total inodes = 5248
gap (physical blocks) = 9
cylinder size (physical blocks) = 90
mkfs: Available blocks = 20684
# prtvtoc -s /dev/rdsk/c1d1s0<CR>
*   Partition  Tag  Flags  First Sector  Sector Count   Mount Directory
         0      2    00        6120          14400
         1      3    01         100           6020
         2      4    00       20520          42030
         6      0    01           0          62550
         7      0    01           0            100
# newboot /lib/lboot /lib/mboot /dev/rdsk/c1d1s7<CR>
newboot: confirm request to write boot programs to /dev/rdsk/c1d1s7: y<CR>
# volcopy root /dev/rdsk/c1d0s0 - /dev/rdsk/c1d1s0 -<CR>
/dev/rdsk/c1d1s0 less than 48 hours older than /dev/rdsk/c1d0s0
To filesystem dated:  Sat Mar 30 06:22:44 1985
Type 'y' to override:     y<CR>
warning! from fs(root) differs from to fs()
Type 'y' to override:     y<CR>
From: /dev/rdsk/c1d0s0, to: /dev/rdsk/c1d1s0? (DEL if wrong)
  END: 14400 blocks.
volcopy: cannot access /etc/log/filesave.log
```

Continued

Continued from the previous screen

```
# fsck -D /dev/rdsk/c1d1s0<CR>

  /dev/rdsk/c1d1s0
  File System: root Volume:

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  FILE SYSTEM STATE SET TO OKAY
  614 files 11096 blocks 3076 free
  *** FILE SYSTEM WAS MODIFIED ***
# shutdown -y -g0 -i5<CR>
```

series of messages are displayed
ending with the following

```
FIRMWARE MODE
<mcp><CR>

Enter name of program to execute [   ]: /etc/system<CR>
             Possible load devices are:

Option Number   Slot     Name
----------------------------------------
       0          0      FD5
       1          0      HD30
       2          0      HD30
       3          3      CTC
```

Continued

Continued from the previous screen

```
Enter Load Device Option Number [1 (HD30)]: 2<CR>

CONFIGURATION SUMMARY
=====================
```

series of messages are displayed
ending with the following

```
UNIX System V Release 2.0.3 3B2 Version 2
unix
Copyright (c) 1984 AT&T.
All Rights Reserved

NOTICE: /dev/swap doesn't match swapdev; changing it on fs

The system is coming up.  Please wait.
```

series of messages are displayed
ending with the following

```
Console Login:
```

ASSIGNMENT OF DEFAULT BOOT PROGRAM AND DEVICE

Caution: Before changing the default boot device from option 1 (the first hard disk) to another device option, make the other device bootable. This helps to avoid the possibility of auto-booting from a device that is not properly configured.

General

The assignment of a default program name and device to be used when the system is manually booted are assigned using the **/etc/fltboot** command. The term "boot" is used to mean the loading and executing a program from the firmware mode. When the system is delivered, the boot program name is null and the boot device is the first hard disk (option 1). When the boot program name is null, the auto boot program name defined in Nonvolatile Random Access Memory (NVRAM) is passed to the boot process. This name is **/unix**. Only **/unix** or **/etc/system** can be used as auto boot programs. Other program names must be manually booted. Booting **/unix** or **/etc/system** from a particular device can be done either manually or automatically. Any equipped and properly configured hard disk (disk 1 or disk 2), floppy disk (diskette1), or cartridge tape drive (ctape1) can be used as a boot device. If the **/unix** file on a given device was not generated after booting the device, the operating system parameters will not be properly set and the operating system will not run. Therefore, when booting the UNIX Operating System from a device for the first time, or when in doubt, boot **/etc/system**. Booting **/etc/system** remakes the operating system core image. Auto-configuration (**/etc/rc.d/autoconfig**) then copies the core image to the **/unix** file on the boot device. Refer to the "MAKING A BOOTABLE DEVICE" task description for information on conditioning a particular device to be used as a boot device.

Floating Boot Assignment Procedure

The **/etc/fltboot** command is used to output and/or change the current default boot program name and device. The command can be executed in any run level in which the UNIX Operating System is running (run-levels 1, 2, 3, 4, s, or S). Note that the program name can be changed to a null by entering a space for the program name.

The following shows the use of the **/etc/fltboot** command to display the current values.

```
# fltboot<CR>

Enter name of default program for manual load [    ]: <CR>
NULL response detected, current value will be retained
To clear value,enter space before return
        Possible load devices are:

Option Number    Slot     Name
----------------------------------------
       0          0          FD5
       1          0          HD30
       2          0          HD30
       3          3          CTC
enter number corresponding to autoload device desired [ 1 ]:<CR>
NULL response detected, current value will be maintained

 MANUAL LOAD DEFAULT PROGRAM:

 AUTO LOAD DEVICE:

 LOAD PARAMETER UPDATE COMPLETE
#
```

The following shows the use of the **/etc/fltboot** command to change the current values. The default program name is changed to **/unix**; the default device option is changed to a 2 (the second hard disk).

```
# fltboot<CR>

Enter name of default program for manual load [    ]:/unix<CR>
        Possible load devices are:

Option Number     Slot      Name
----------------------------------------
       0           0          FD5
       1           0          HD30
       2           0          HD30
       3           3          CTC
enter number corresponding to autoload device desired [ 1 ]:2<CR>

 MANUAL LOAD DEFAULT PROGRAM:/unix

 AUTO LOAD DEVICE: HD30

 LOAD PARAMETER UPDATE COMPLETE
#
```

The following shows the use of the **/etc/fltboot** command to change the current values. The default program name is changed to a null; the default device option is changed to a 1 (the first hard disk).

```
# fltboot<CR>

Enter name of default program for manual load [ /unix ]:<space><CR>
        Possible load devices are:

Option Number     Slot      Name
----------------------------------------
       0           0          FD5
       1           0          HD30
       2           0          HD30
       3           3          CTC
enter number corresponding to autoload device desired [ 2 ]:1<CR>

 MANUAL LOAD DEFAULT PROGRAM:

 AUTO LOAD DEVICE: HD30

 LOAD PARAMETER UPDATE COMPLETE
#
```

Chapter 4

FILE SYSTEM CHECKING AND REPAIR

PAGE

Chapter 4

FILE SYSTEM CHECKING AND REPAIR

GENERAL

When the UNIX Operating System is brought up, a consistency check of the file systems should always be done. The status of the file systems is automatically checked when the AT&T 3B2 Computer is powered up. If file system status checks indicate the need to check a file system, the **fsck** program is automatically executed. The root file system is automatically checked via the **fsck** call in the **/etc/bcheckrc** shell script. The **fsck** options used for checking the root file system are the **y**, **D**, and **b** options. The other file systems identified in the **/etc/fstab** file are automatically checked via an **fsck** call in the **/etc/mountall** shell script. The file systems listed in the **/etc/fstab** file are automatically checked using **fsck** options **D** and **y**. This precautionary measure during system initialization helps to ensure a reliable environment for file storage on disk. If an inconsistency is found, corrective action is automatically taken as the result of the **y** (yes) **fsck** option.

The file system check (**fsck**) program is an interactive file system check and repair program. The **fsck** program uses the redundant structural information in the UNIX System file system to do several consistency checks. If an inconsistency is detected, a message describing the

inconsistency is output. You may elect to fix or ignore each inconsistency. These inconsistencies result from the permanent interruption of the file system updates, which are done every time a file is modified. The **fsck** program is frequently able to repair corrupted file systems using procedures based on the order in which the UNIX System honors these file system update requests.

The following paragraphs describe the normal updating of the file system, the possible causes of file system corruption, and the corrective actions that can be taken by **fsck**. The meanings of the various error conditions, possible responses, and related error conditions are explained.

FILE SYSTEM UPDATE

Every time a file is modified, the UNIX Operating System does a series of file system updates. These updates, when written on disk, produce a consistent file system. To understand what happens when there is an interruption in this sequence, it is important to understand the order in which the update requests are processed. Procedures can be developed to repair a corrupted file system based on the type of information written to the file system.

There are five types of file system updates. These involve the super-block, information nodes (i-nodes), indirect blocks, data blocks (directories and files), and free-list blocks.

Super-Block

The super-block contains information about the size of the file system, the size of the i-node list, part of the free-block list, the count of free blocks, the count of free i-nodes, and part of the free i-node list.

The super-block of a mounted file system (the root file system is always mounted) is written to the disk when the file system is unmounted or when a **sync** command is executed.

Information Nodes (i-nodes)

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

An i-node is written to the file system when the file associated with the i-node is closed. All in-core blocks are also written to the file system when a **sync** command is executed.

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 i-node. 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 when they have been modified and released by the operating system. More precisely, they are queued for eventual writing. Physical input/output is deferred until the buffer is needed by the UNIX System or until a **sync** command is executed.

Data Blocks

A data block can either contain file information or directory entries. Each directory entry consists of a file name and an i-node number.

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

First Free-List Block

The super-block contains the first free-list block. The free-list blocks are a list of all blocks that are not allocated to the super-block, i-nodes, indirect blocks, or data blocks. Each free-list block contains a count of the number of entries in this free-list 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 when they have been modified and released by the operating system.

FILE SYSTEM CORRUPTION

A file system can become corrupted in a variety of ways. Improper shutdown procedures and hardware failures are the most common causes of file system corruption.

Improper System Shutdown and Startup

File systems may become corrupted when proper shutdown procedures are not observed. For example, forgetting to **sync** the system before stopping the Central Processing Unit (CPU), physically write-protecting a mounted file system, or taking a mounted file system off-line can corrupt the file system.

File systems can also become further corrupted by allowing a corrupted file system to be used (and, thus, to be modified further).

Hardware Failure

Any piece of hardware can fail at any time. Failures can be as subtle as a bad block on a disk platter or as major as a nonfunctional disk controller.

DETECTION AND CORRECTION OF CORRUPTION

An unmounted file system that is not being written on can be checked for structural integrity by doing consistency checks on the redundant file system data. The redundant data is either read from the file system or computed from other known values. A quiescent state is important during the checking of a file system because of the multipass nature of the **fsck** program.

When an inconsistency is found, **fsck** reports the inconsistency for the user to chose a corrective action.

The following paragraphs describe file system inconsistencies and the possible corrective actions that can be taken for the super-block, the i-nodes, the indirect blocks, the data blocks containing directory entries, and the free-list blocks. These corrective actions can be done interactively by the **fsck** command under control of the user.

Super-Block

The super-block is prone to corruption because every change to the file system blocks or i-nodes modifies the super-block. The super-block and its associated parts are most often corrupted when the CPU is halted and the last command involving output to the file system was not a **sync** command.

The super-block can be checked for inconsistencies involving file system size, i-node list size, free-block list, free-block count, and the free i-node count.

File System Size and I-Node List Size

The file system size must be larger than the number of blocks used by the super-block and the number of blocks used by the list of i-nodes. The number of i-nodes must be less than 65,488. The file system size and i-node list size are critical pieces of information to the **fsck** program. While there is no way to 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.

Free-Block List

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

The first free-block list is in the super-block. The **fsck** program 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. Each block number is compared to a list of already allocated blocks. If the free-list block pointer is not zero, 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 i-nodes equals the total number of blocks in the file system.

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

Free-Block Count

The super-block contains a count of the total number of free blocks within the file system. The **fsck** program compares this count to the number of blocks it found free within the file system. If the counts do not agree, then **fsck** can replace the count in the super-block with the calculated free-block count.

Free I-Node Count

The super-block contains a count of the total number of free i-nodes within the file system. The **fsck** program compares this count to the number of i-nodes it found free within the file system. If the counts do not agree, then **fsck** can replace the count in the super-block with the calculated free i-node count.

I-Nodes

An individual i-node is not as likely to be corrupted as the super-block. However, because of the great number of active i-nodes, there is almost as likely a chance for corruption in the i-node list as in the super-block.

The list of i-nodes is checked sequentially starting with i-node 1 (there is no i-node 0) and goes to the last i-node in the file system. Each i-node is checked for inconsistencies involving format and type, link count, duplicate blocks, bad blocks, and i-node size.

Format and Type

Each i-node contains a mode word. This mode word describes the type and state of the i-node. I-nodes may be one of four types:

- Regular
- Directory
- Special block
- Special character.

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

Link Count

Each i-node contains a count of the total number of directory entries linked to the i-node. The **fsck** program verifies the link count of each i-node by examining the total directory structure, starting from the root directory, and calculating the link count for each i-node.

If the stored link count is not zero and the calculated link count is zero, it means that no directory entry appears for the i-node. If the stored and calculated link counts are not zero and unequal, a directory entry may have been added or removed without the i-node being updated.

If the stored link count is not zero and the calculated link count is zero, **fsck** can, under user control, link the disconnected file to the **lost+found** directory. If the stored and calculated link counts are not zero and unequal, **fsck** can replace the stored link count by the calculated link count.

Duplicate Blocks

Each i-node contains a list or pointers to lists (indirect blocks) of all the blocks claimed by the i-node. The **fsck** program compares each block number claimed by an i-node to a list of already allocated blocks. If a block number is already claimed by another i-node, 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 i-node list to find the i-node of the duplicated block. This is necessary because without examining the files associated with these i-nodes for correct content there is not enough information available to determine which i-node is corrupted and should be cleared. Most of the time, the i-node with the earliest modify time is incorrect and should be cleared. This condition may 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 i-node may be caused by an indirect block not being written to the file system. The **fsck** program prompts the user to clear both i-nodes.

Bad Blocks

Each i-node contains a list or pointer to lists of all the blocks claimed by the i-node. The **fsck** program checks each block number claimed by an i-node 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.

If there is a large number of bad blocks in an i-node, this may be caused by an indirect block not being written to the file system. The **fsck** program prompts the user to clear the i-node.

Size Checks

Each i-node contains a 32-bit (4-byte) size field. This size indicates the number of characters in the file associated with the i-node. This size can be checked for inconsistencies such as directory sizes that are not a multiple of 16 characters and a mismatch between the number of blocks versus the number stored for the i-node size.

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

The **fsck** program 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 i-node can be done by computing the number of blocks that should be associated with the i-node and comparing it to the stored number of blocks claimed by the i-node.

The **fsck** program calculates the number of blocks that there should be in an i-node by dividing the number of characters in an i-node by the number of characters per block and rounding up. One block is added for each indirect block associated with the i-node. If the stored 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 UNIX System does not fill in blocks in files created in random order.

Indirect Blocks

Indirect blocks are owned by an i-node. Therefore, inconsistencies in indirect blocks directly affect the i-node that owns it.

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

For a description of detection and correction of the inconsistencies associated with indirect blocks, see the descriptions on "Duplicate Blocks" and "Bad Blocks" in this chapter.

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. The **fsck** program 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 i-node numbers pointing to unallocated i-nodes, directory i-node numbers greater than the number of i-nodes in the file system, incorrect directory i-node numbers for "." and ".." directories, and directories disconnected from the file system. In addition, the validity of the contents of a directory data block is checked.

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

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

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

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

If the directory i-node numbers for '.' and ".."'are incorrect, **fsck** can replace them with the correct values.

The **fsck** program 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 i-nodes being written to the file system with the corresponding directory data blocks not being written to the file system.

Free-List Blocks

Free-list blocks are owned by the super-block. Therefore, inconsistencies in free-list blocks directly affect the super-block.

Inconsistencies that can be checked are an out-of-range list count, out-of-range block number, and blocks already associated with the file system (duplicate blocks).

For an explanation of detection and correction of the inconsistencies associated with free-list blocks, see the "Free-Block List" description.

FSCK ERROR CONDITIONS

General

The **fsck** program is a multipass file system check program. Each pass invokes a different phase of the **fsck** program. After the initial setup, **fsck** runs successive phases on each file system doing 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 user. If a response is required, **fsck** prints a prompt message and waits for a response. The following paragraphs explain 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 are described under "Initialization."

Initialization

Before a file system check can be done, certain tables have to be set up and certain files opened. This section describes 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. The **fsck** program terminates on initialization errors.

Legal Options

Legal **fsck** options are **–b**, **–y**, **–n**, **–s**, **–S**, **–t**, **–r**, **–q**, and **–D**. Refer to the **fsck**(1M) manual pages for more information.

Bad –t option

The **–t** option is not followed by a file name. The **fsck** program terminates on this error condition. Refer to the **fsck**(1M) manual pages for more information.

Invalid –s argument, defaults assumed

The **–s** option is not suffixed by 3, 4, or blocks-per-cylinder:blocks-to-skip. The **fsck** program assumes a default value of 400 blocks-per-cylinder and 9 blocks-to-skip. Refer to the **fsck**(1M) manual pages for more information.

Incompatible options: –n and –s

It is not possible to salvage the free-block list without modifying the file system. The **fsck** terminates on this error condition. Refer to the **fsck**(1M) manual pages for more information.

Can not fstat standard input

The attempt to **fstat** standard input failed. The occurrence of this error condition indicates a serious problem which may require additional help. The **fsck** program terminates on this error condition.

Can not get memory

The request for memory for virtual memory tables failed. The occurrence of this error condition indicates a serious problem which may require additional help. The **fsck** program terminates on this error condition.

Can not open checkall file: F

The default file system **checkall** file *F* (usually **/etc/checkall**) cannot be opened for reading. The **fsck** program terminates on this error condition. Check access modes of *F*.

Can not stat root

The request for statistics about the root directory "/" failed. The occurrence of this error condition indicates a serious problem which may require additional help. The **fsck** program terminates on this error condition.

Can not stat F

The request for statistics about the file system *F* failed. The **fsck** program ignores this file system and continues checking the next file system given. Check access modes of *F*.

F is not a block or character device

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

Can not open F

The file system *F* cannot be opened for reading. The **fsck** program ignores this file system and continues checking the next file system given. Check the access modes of *F*.

Size check: fsize X isize Y

More blocks are used for the i-node list *Y* than there are blocks in the file system *X*, or there are more than 65,488 i-nodes in the file system. The **fsck** program ignores this file system and continues checking the next file system given.

Can not create F

The request to create a scratch file *F* failed. The **fsck** program ignores this file system and continues checking the next file system given. Check the access modes of *F*.

CAN NOT SEEK: BLK B (CONTINUE)

The request for moving to a specified block number *B* in the file system failed. The occurrence of this error condition indicates a serious problem which may require additional help.

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 the block was part of the virtual memory buffer cache, **fsck** terminates with the message "Fatal I/O error".
NO	Program terminated.

CAN NOT READ: BLK B (CONTINUE)

The request for reading a specified block number *B* in the file system failed. The occurrence of this error condition indicates a serious problem which may require additional help.

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	Program terminated.

CAN NOT WRITE: BLK B (CONTINUE)

The 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	Program terminated.

Phase 1: Check Blocks and Sizes

This phase concerns itself with the i-node list. This part lists error conditions resulting from checking i-node types, setting up the zero-link-count table, examining i-node block numbers for bad or duplicate blocks, checking i-node size, and checking i-node format.

UNKNOWN FILE TYPE I=I (CLEAR)

The mode word of the i-node *I* indicates that the i-node is not a special character i-node, regular i-node, or directory i-node.

Possible responses to CLEAR prompt are:

YES	Deallocate i-node *I* by zeroing its contents. This invokes the UNALLOCATED error condition in Phase 2 for each directory entry pointing to this i-node.
NO	Ignore this error condition.

LINK COUNT TABLE OVERFLOW (CONTINUE)

An internal table for **fsck** containing allocated i-nodes with a link count of zero has no more room.

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 i-node with a zero link count is found, this error condition is repeated.
NO	Program terminated.

B BAD I=I

I-node *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 i-node *I* has too many block numbers outside the file system range. This error condition invokes the BAD/DUP error condition in Phase 2 and Phase 4.

EXCESSIVE BAD 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 the last block in the file system associated with i-node *I*.

Possible responses to CONTINUE prompt are:

YES	Ignore the rest of the blocks in this i-node and continue checking with next i-node 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	Program terminated.

B DUP I=I

I-node *I* contains block number *B* which is already claimed by another i-node. This error condition may invoke the EXCESSIVE DUP BLKS error condition in Phase 1 if i-node *I* has too many block numbers claimed by other i-nodes. This error condition invokes Phase 1B and the BAD/DUP error condition in Phase 2 and Phase 4.

EXCESSIVE DUP BLKS I=I (CONTINUE)

There is more than a tolerable number (usually 10) of blocks claimed by other i-nodes.

Possible responses to CONTINUE prompt are:

YES	Ignore the rest of the blocks in this i-node and continue checking with next i-node 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	Program terminated.

DUP TABLE OVERFLOW (CONTINUE)

An internal table in **fsck** containing duplicate block numbers has no more room.

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 repeats.
NO	Program terminated.

POSSIBLE FILE SIZE ERROR I=I

The i-node *I* size does not match the calculated number of blocks used by the i-node. This is only a warning. If the **–q** option is used, this message is not printed.

DIRECTORY MISALIGNED I=I

The size of a directory i-node is not a multiple of the size of a directory entry (usually 16). This is only a warning. If the **–q** option is used, this message is not printed.

PARTIALLY ALLOCATED INODE I=I (CLEAR)

I-node *I* is neither allocated nor unallocated.

Possible responses to CLEAR prompt are:

YES	Deallocate i-node I by zeroing its contents.
NO	Ignore this error condition.

Phase 1B: Rescan for More DUPS

When a duplicate block is found in the file system, the file system is rescanned to find the i-node which previously claimed that block. This part lists the error condition when the duplicate block is found.

B DUP I=I

I-node *I* contains block number *B* which is already claimed by another i-node. This error condition invokes the BAD/DUP error condition in Phase 2. I-nodes with overlapping blocks may be determined by examining this error condition and the DUP error condition in Phase 1.

Phase 2: Check Path Names

This phase concerns itself with removing directory entries pointing to error conditioned i-nodes from Phase 1 and Phase 1B. This part lists error conditions resulting from root i-node mode and status, directory i-node pointers in range, and directory entries pointing to bad i-nodes.

ROOT INODE UNALLOCATED. TERMINATING

The root i-node (always i-node number 2) has no allocate mode bits. The occurrence of this error condition indicates a serious problem which may require additional help. The program stops.

ROOT INODE NOT DIRECTORY (FIX)

The root i-node (usually i-node number 2) is not directory i-node type.

Possible responses to FIX prompt are:

YES	Replace the root i-node type to be a directory. If the root i-node data blocks are not directory blocks, a *very* large number of error conditions are produced.
NO	Program terminated.

DUPS/BAD IN ROOT INODE (CONTINUE)

Phase 1 or Phase 1B has found duplicate blocks or bad blocks in the root i-node (usually i-node number 2) for the file system.

Possible responses to CONTINUE prompt are:

YES	Ignore DUPS/BAD error condition in root i-node and attempt to continue to run the file system check. If root i-node is not correct, then this may result in a large number of other error conditions.
NO	Program terminated.

I OUT OF RANGE I=I NAME=F (REMOVE)

A directory entry *F* has an i-node number *I* which is greater than the end of the i-node list.

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 i-node *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 i-node 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 has found duplicate blocks or bad blocks associated with directory entry *F*, directory i-node *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 has found duplicate blocks or bad blocks associated with directory entry *F*, i-node *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 i-node *I*. Error conditions looked for in directory blocks are nonzero padded entries, inconsistent "." and ".." entries, and embedded slashes in the name field. This error message indicates that the user should at a later time either remove the directory i-node if the entire block looks bad or change (or remove) those directory entries that look bad.

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 i-node *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 i-node *I* are printed. The **fsck** program forces the reconnection of a nonempty directory.

Possible responses to RECONNECT prompt are:

YES	Reconnect directory i-node *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 i-node *I* to **lost+found**. This may also invoke CONNECTED error condition in Phase 3 if link was successful.
NO	Ignore this error condition. This 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 invokes the UNREF error condition in Phase 4. Check access modes of **lost+found**. Refer to the **fsck**(1M) manual pages for more information.

SORRY. NO SPACE IN lost+found DIRECTORY

There is no space to add another entry to the **lost+found** directory in the root directory of the file system; **fsck** ignores the request to link a directory in **lost+found**. This invokes the UNREF error condition in Phase 4. Clean out unnecessary entries in **lost+found** or make **lost+found** larger. Refer to the **fsck**(1M) manual pages for more information.

DIR I=I1 CONNECTED. PARENT WAS I=I2

This is an advisory message indicating a directory i-node *I1* was successfully connected to the **lost+found** directory. The parent i-node *I2* of the directory i-node *I1* is replaced by the i-node number of the **lost+found** directory.

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-i-node counts.

UNREF FILE I=I OWNER=O MODE=M SIZE=S MTIME=T (RECONNECT)

I-node *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 i-node *I* are printed. If the **–n** option is omitted and the file system is not mounted, empty files are cleared automatically. Nonempty directories are not cleared.

Possible responses to RECONNECT prompt are:

YES	Reconnect i-node *I* to file system in the directory for lost files (usually **lost+found**). This can cause a **lost+found** error condition in Phase 4 if there are problems connecting i-node *I* to **lost+found**.
NO	Ignore this error condition. This invokes a 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 invokes the 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 invokes the CLEAR error condition in Phase 4. Check size and contents of **lost+found**.

(CLEAR)

The i-node mentioned in the immediately previous error condition cannot be reconnected.

Possible responses to CLEAR prompt are:

YES	Deallocate i-node 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 i-node *I*, which is a file, is *X* but should be *Y*. The owner *O*, mode *M*, size *S*, and modify time *T* are printed.

Possible responses to ADJUST prompt are:

YES	Replace link count of file i-node *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 i-node *I*, which is a directory, is *X* but should be *Y*. The owner *O*, mode *M*, size *S*, and modify time *T* of directory i-node *I* are printed.

Possible responses to ADJUST prompt are:

YES	Replace link count of directory i-node *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* i-node *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 i-node *I* with *Y*.
NO	Ignore this error condition.

UNREF FILE I=I OWNER=O MODE=M SIZE=S MTIME=T (CLEAR)

I-node *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 i-node *I* are printed. If the **–n** option is omitted and the file system is not mounted, empty files are cleared automatically. Nonempty directories are not cleared.

Possible responses to CLEAR prompt are:

YES	Deallocate i-node *I* by zeroing its contents.
NO	Ignore this error condition.

UNREF DIR I=I OWNER=O MODE=M SIZE=S MTIME=T (CLEAR)

I-node *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 i-node *I* are printed. If the **–n** option is omitted 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 i-node *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 has found duplicate blocks or bad blocks associated with file i-node *I*. The owner *O*, mode *M*, size *S*, and modify time *T* of i-node *I* are printed.

Possible responses to CLEAR prompt are:

YES	Deallocate i-node *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 has found duplicate blocks or bad blocks associated with directory i-node *I*. The owner *O*, mode *M*, size *S*, and modify time *T* of i-node *I* are printed.

Possible responses to CLEAR prompt are:

YES	Deallocate i-node *I* by zeroing its contents.
NO	Ignore this error condition.

FREE INODE COUNT WRONG IN SUPERBLK (FIX)

The calculated free i-nodes count does not match the count in the super-block of the file system. If the **–q** option is specified, the count will be fixed automatically in the super-block.

Possible responses to FIX prompt are:

YES	Replace count in super-block by calculated count.
NO	Ignore this error condition.

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-block 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 (usually 10) of blocks with a value less than the first data block in the file system or greater than the last block in the file system.

Possible responses to CONTINUE prompt are:

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

EXCESSIVE DUP BLKS IN FREE LIST (CONTINUE)

The free-block list contains more than a tolerable number (usually 10) of blocks claimed by i-nodes 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 will always invoke "DUP BLKS IN FREE LIST" error condition in Phase 5.
NO	Program terminated.

BAD FREEBLK COUNT

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

X DUP BLKS IN FREE LIST

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

FREE BLK COUNT WRONG IN SUPERBLOCK (FIX)

The calculated free block count does not match the count in the super-block of the file system.

Possible responses to FIX prompt are:

YES	Replace count in super-block by calculated 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 will be salvaged automatically.

Possible responses to SALVAGE prompt are:

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

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 1, the blocks-per-cylinder is less than 1, or the blocks-per-cylinder is greater than 500. The default values of 9 blocks-to-skip and 400 blocks-per-cylinder are used. Refer to the **fsck**(1M) manual pages for more information.

Cleanup

Once a file system has been checked, a few cleanup functions are done. 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 UNIX (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 UNIX System is not rebooted immediately without **sync**, the work done by **fsck** may be undone by the in-core copies of tables the UNIX System keeps.

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

This is an advisory message indicating that the current file system was modified by **fsck**.

Chapter 5

BAD BLOCK HANDLING FEATURE

PAGE

Chapter 5

BAD BLOCK HANDLING FEATURE

OVERVIEW

The AT&T 3B2 Computer has a software feature called bad block handling. The purpose of this feature is to extend the useful life of the integral hard disk. The useful disk life is extended by providing mechanisms for:

- Detecting and remembering blocks that are no longer usable
- Reminding you that you need to "fix" some remembered bad blocks
- Restoring the usability of the disk in spite of the bad blocks that exist.

New bad blocks seldom occur, particularly with the sealed environment of the integral disk, as long as you take reasonable precautions against movement or vibration of the computer while the disk is still spinning. But when a new bad block occurs, the data stored in the bad block is lost and the disk may be unusable in its current state.

The bad block handling feature addresses the problem of restoring the usability of the disk. However, you must address the data loss yourself using the appropriate backup/restore procedures. Backup procedures are also needed to protect against operational errors and other types of hardware failures. These other problems, particularly operational errors, are the dominant cause of data loss on a system. System backups provide protection against operational errors as well as protection against lost data caused by new bad blocks.

What Is a Bad Block?

A bad block is a disk block that cannot reliably store data. This is found only when an attempt is made to read the data and the read fails.

A read can fail even though the block is not a bad block. If a write fails, this generally means that there is a problem with the format of the disk or there is a more basic failure in the disk or disk controller hardware. A read could also fail for these reasons. To fix these problems, you must reformat the disk or get the hardware repaired. One reason for the strong recommendation that you contact your service representative is to help you determine which type of failure has occurred.

Although all failures are reported through the bad block handling mechanisms, the mechanism is designed around the properties of bad blocks. (For example, mapping the new bad blocks to substitute blocks to make the disk usable.) It is beyond the scope of bad block handling to distinguish genuine bad blocks from format problems or from more basic failures in the disk or disk controller hardware. If several distinct failures occur at about the same time, the other failures are the more likely cause.

What Makes a Block Unreliable?

A disk is an analog media used to store digital data. The analog phenomena used in the disk involve magnetic properties of the film coating on its platters. The data is recorded with a high bit density to get millions of bits in a small space. Therefore, any small scale variations in the magnetic properties of the recording media become important. These

inevitable variations mean that any given portion of the media prefers to represent some bit patterns and dislikes representing other bit patterns. Normally, these preferences are insignificant compared to the signal level thresholds. When variations in magnetic properties stop being insignificant, the disk has a bad block. However, if the data pattern matches these preferences in the bits where the preferences have become significant, the block will still appear to be good.

How Can You Fix Bad Blocks?

A small portion of the disk media is set aside from the normally accessible portion of the disk. Normal UNIX System commands and system calls cannot access this area of the disk. This reserved portion of the disk media contains a description of the properties of the disk and other media-specific data.

The mechanism for preserving the apparent accessibility of most disk blocks is to use surrogate image blocks to contain the data for bad blocks. The media-specific data portion of the disk includes a set of blocks called the surrogate image region. The media-specific data also includes a mapping table that maps bad blocks to surrogate image blocks. The disk driver software in the operating system translates disk accesses such that the data is read/written from/to the surrogate image disk address. This disk address translation is transparent to the calling software.

The disk comes with the few manufacturing defects already mapped, but there is plenty of room left over for new defects. This feature provides special software for remembering bad blocks that have been found and for mapping the remembered bad blocks. If a surrogate block becomes bad, this feature remaps the original bad block to a new surrogate block. A list of disk-manufacturer identified defects is provided on a label on the hard disk.

A Few Blocks Cannot Be Mapped

A few special blocks cannot be mapped; however, they are all in the media-specific data portion of the disk. In particular, the disk block containing the physical description of the disk and the disk block(s) containing the mapping table cannot be mapped. All other blocks, including the block used to remember reported bad blocks, can be mapped.

Detecting Bad Blocks

Bad blocks are detected when input/output disk operations fail for several successive attempts. This means that the data being input or output is lost, but the system can restore use of the disk by mapping bad blocks to surrogate blocks which are readable.

Often Asked Questions

Why doesn't the system try to discover that a given block is bad while the system still has the data in memory? Besides the undesirable increase in system size and complexity, severe performance degradation would result. Also, a block can become a bad block after the copy in memory no longer exists.

Why doesn't the system periodically test the disk for bad blocks? Reading blocks with their current contents may not show a bad block to be bad. A thorough bit pattern test would take so long that you would never run it, even assuming a thorough test could be devised using ordinary write/read operations. The disk manufacturer already has tested the disk using extensive bit pattern tests and special hardware. All manufacturing defects have been dealt with already.

Why are disks with manufacturing defects used? Allowing the disks to contain a modest number of manufacturing defects greatly increases the yield, thereby, considerably reducing the cost. Many systems, including this one, take advantage of this cost reduction to provide a more powerful system at lower cost.

HOW BAD BLOCK HANDLING WORKS

The bad block handling feature provides the mechanisms to detect, remember (where feasible), and add new bad blocks to the existing map. To the extent that is feasible, the mechanisms are automated. This automation allows the bad block handling feature to be used with minimal knowledge for most cases. But some special cases do exist and even the automated cases have special properties at some stages of the processing.

Detecting, Reporting, and Logging New Bad Blocks

The automated mechanisms are part of the normal UNIX System execution environment. The first phase of automated mechanisms involves detecting new bad blocks and remembering them for later mechanisms.

Ways of Referring to Disk Blocks

An understanding of the various ways of referring to disk blocks is necessary to comprehend the following examples. Inherent in the design of the system are several ways of referring to disk blocks. In each, a block number is an integer counter. A physical block number is the integer counter form of the way the sectors are numbered on the media. (For example, physical block 3 is sector 3 of head 0 of cylinder 0.) A logical block number counts in sectors starting with 0 at the beginning of the partitioned portion of the disk. A partition block number counts in sectors starting with 0 at the beginning of the partition. A file system block number counts in file system blocks starting with 0 at the beginning of its partition.

An Example of Detecting, Reporting, and Logging a Bad Block

Suppose you have a 3B2 Computer with a 30-megabyte integral disk. It has a few manufacturing defects and physical disk block 3 is a surrogate block for physical block X in the **/usr** file system.

Block X is in a text file that you created. Block 3 has become a bad block sometime after the last time you looked at that text file. Now, you try to read that text file. When you get to the point of trying to read block X, the

integral disk driver sees that block X is mapped to block 3 and attempts to read block 3. But block 3 is bad and cannot be read. When the integral disk driver determines that block 3 is unreadable, the following messages are output to the system console.

```
WARNING: unreadable CRC hard disk error: maj/min = 17/0

        block # = 3


WARNING:
hard disk: cannot access sector 3, head 0, cylinder 0, on drive 0

Disk Error Daemon: successfully logged error for block 3 on disk maj=17 min=0
```

Your attempt to read the text file fails. Then the system administrator notices the message on the console and runs shutdown to go to single-user mode (run-level S). While shutdown is running, the following message is output to the system console.

```
Disk Error Daemon: Disk maj=17 min=0: 1 errors logged
```

The following paragraphs describe what was going on behind the scenes.

Disk Identification

The mechanisms used for bad block handling are designed to be general purpose mechanisms. These mechanisms support both single-disk and multi-disk models of the AT&T 3B2 Computer. To handle all possible configurations, disks are identified by using their external major/minor device number as their name. Messages printed out by bad block handling include this name. The utilities of this feature can be given these names as arguments when more specialized operations must be used. To handle all possible configurations, the hard disk is identified by using its external major/minor device number as its name. Messages printed out by bad block handling include this name. The utilities of this feature can be given this name as an argument when more specialized operations must be used.

To find out what the current set of these disk names is for bad block handling, use the **/etc/hdeadd -e** command. The following command line entry and system responses show the use of this command.

```
# hdeadd -e<CR>
The 1 equipped disks for bad block handling are:
 MAJOR MINOR
   17     0
#
```

Detecting New Bad Blocks

The disk driver software detects the new bad blocks for the disk in question. The disk driver determines that a block is not accessible by attempting several accesses. The disk driver also repositions the disk read/write heads between some of the retries to be sure that the problem is not a head positioning error.

Reporting and Logging New Bad Blocks

When a block is determined to be unaccessible, the disk driver tells the bad block logging mechanism about the bad block. The logging mechanism reports the problem on the system console. This report for the previous example is as follows.

```
WARNING: unreadable CRC hard disk error: maj/min = 17/0

         block # = 3
```

The data in this message is in the correct form for use with the specialized options of the bad block handling commands.

After reporting the error to the logging mechanism, the disk driver also reports the error in a different, device-dependent form, as follows.

```
WARNING:
hard disk: cannot access sector 3, head 0, cylinder 0, on drive 0
```

This form is useful for recognizing when a track has a problem, since the head and cylinder numbers would be the same.

The logging mechanism then attempts to remember (write) the error in the disk error log. The disk error log is in the media-specific portion of the disk. If the error gets successfully logged, a message similar to the following is output to the system console.

```
Disk Error Daemon: successfully logged error for block 3 on disk maj=17 min=0
```

Normally, the sequence of events previously described is what happens when the automated mechanisms can handle the identification and reporting and logging of new bad blocks. Most of the cases of new bad blocks are handled automatically. The unusual cases are described in the following paragraphs.

The logging mechanism is implemented using a special driver (hdelog-Hard Disk Error Log driver) in the operating system and a disk error daemon process (**hdelogger**) run by the **/etc/init** process. The special driver provides the special disk access needed by this feature, as well as providing the mechanisms for reporting and queuing up reports until they get logged. This special driver can queue up as many as 18 reports that are waiting to be logged. Since a track on a 10-megabyte or 30-megabyte disk has 18 sectors, an entire bad track is handled by 18 reports. The disk error daemon gets reports from this queue and attempts to add each report to the disk error log.

The disk error daemon also has another reporting role. When the system changes its run level (for example, when you turn on the 3B2 Computer, shut it off, or shut down to the single-user mode), the daemon checks the error log. If the daemon finds outstanding bad block reports in a log, it outputs a message on the system console. In the previous example, that message was as follows.

```
Disk Error Daemon: Disk maj=17 min=0: 1 errors logged
```

The normal run-level for the system is the multi-user mode (run-level 2). The **hdelogger** daemon is running in run-level 2. The daemon also runs in run-levels 3 and 4. Run-levels 5 and 6 are used for returning to firmware and for rebooting the operating system, respectively. Run-level 1, s, or S is the single-user mode. The bad block handling daemon is not running in run-levels 1, 5, or S. Run-level 6 is a transient state.

Unusual Cases and How to Handle Them

Errors in Single-User Mode

If errors happen while the system is in the single-user mode, the errors stay queued until the system is returned to one of the run-levels where the bad block handling daemon is running. However, if you shut your system off or reboot it without going to one of the other run-levels, any error reports in the queue are lost. When errors occur while in the single-user mode, only the messages from the logging mechanism and disk driver are output to the system console.

If you get errors while in the single-user mode and you are not ready to fix them for some reason (the mechanism for fixing them takes error reports from the queue as well as from the disk error logs), you can switch to one of the other run-levels (preferably 3 or 4) to get them logged. You will get one of the "successfully logged" messages for each error that occurred. When all are logged, you can switch back to the single-user mode. When you do that, the reminder message is output on the console terminal.

PANICs and Firmware-Detected Errors

If the error occurs in a critical operating system path such as swap input/output, the operating system will PANIC after the reports from the logging mechanism and disk driver are printed on the console but before the report can be logged. If an error is detected by the firmware, the error is reported; however, the firmware is incapable of performing the more sophisticated processing needed to log the error. In these cases, you, the system administrator, MUST write down the necessary information printed out in the report. When you get the system back up, you must use the auxiliary mechanism for manually adding a report to the disk error queue. The auxiliary mechanism is provided by the **/etc/hdeadd** command.

The following is an example of a bad block firmware error message.

```
id 0 CRC error at disk address 00010211
```

The disk address output (00010211 in the example) must be converted from the 8-character hexadecimal address to decimal cylinder, track, and sector numbers for use as arguments in the **hdeadd** and **hdefix** commands. The format of the disk address and how to convert the hexadecimal address to decimal cylinder, track, and sector numbers are shown in Figure 5-1. Based on the hexadecimal address **00010211**, the **-B** option of the **hdeadd** and **hdefix** commands would specify cylinder 1, track 2, and sector 17 (**-B 1 2 17**).

	PHYSICAL CYLINDER NUMBER HIGH (pcnh)		PHYSICAL CYLINDER NUMBER LOW (pcnl)		PHYSICAL HEAD NUMBER (phn)		PHYSICAL SECTOR NUMBER (psn)	
HEX ADDRESS	0	0	0	1	0	2	1	1
BINARY	0000	0000	0000	0001	0000	0010	0001	0001
DECIMAL	1 CYLINDER				2 TRACK		17 SECTOR	

Figure 5-1. Sample Disk Address Conversion

As an example of a bad block causing a PANIC, assume that physical block number 463 is in your swap space. Block number 463 has recently become a bad block (though you don't know that yet), and the operating system writes into it data that cannot be read with its current pattern of biases. When the operating system tries to read that block as part of swapping in the corresponding process, the disk driver determines that the block is unreadable, reports it, and fails the corresponding read job. The swapper runs next, finds out, and causes the PANIC. All process activity is precluded at this point, including the disk error daemon.

On the console, you get the following messages.

```
WARNING: unreadable CRC hard disk error: maj/min = 17/0

         block # = 463


WARNING:
hard disk: cannot access sector 13, head 0, cylinder 5, on drive 0

PANIC: i/o error in swap
```

Copy down (or get a printout) the 17/0 and 463 numbers. When you get the system back up, immediately go to the single-user mode to minimize the chances of getting another swap PANIC. Then the safest procedure is to go to one of the unused run-levels and enter the following **/etc/hdeadd** command.

```
hdeadd -a -D 17 0 -b 463
```

This causes the bad block to be reported to the logging mechanism in the operating system. If the current time is Fri Jul 13 02:01:00 1984, the full printout would be as follows.

```
# hdeadd -a -D 17 0 -b 463<CR>
hdeadd: logging the following error report:
        disk maj=17 min=0
        blkaddr=463, timestamp=Fri Jul 13 02:01:00 1984
        readtype=1, severity=2, badrtcnt=0, bitwidth=0
WARNING: unreadable CRC hard disk error: maj/min = 17/0

         block # = 463


Disk Error Daemon: successfully logged error for block 463 on disk maj=17 min=0
#
```

If you want to double-check that it is right, you can run the **hdelogger** command to get the following report (assuming this is the only error in the log).

```
# hdelogger -f<CR>

Disk Error Log: Full Report for maj=17 min=0
        log created:  Sun Jul  1 12:13:14 1984
       last changed: Fri Jul 13 02:01:04 1984
        entry count: 1
    phys blkno  cnt     first occurrence       last occurrence
 0:         463    1   Jul 13 02:01:00 1984   Jul 13 02:01:00 1984
TOTAL: 1 errors logged
#
```

If the firmware detected the error, there is a distinct possibility that you cannot boot from your hard disk. In these circumstances, call your service representative for help.

The Special Case of a Bad Error Log Block

Though unlikely, the new bad block could be the block for the disk error log. Obviously, if you cannot access it, you cannot log that fact in it. But another auxiliary mechanism is provided as part of the **/etc/hdefix** command that adds new bad blocks to the defect map. That command is described next.

Fixing Bad Blocks

To fix a bad block, a quiescent machine state is needed. Specifically, the machine must be shut down to the single-user run mode. You must see that all extra processes have died and that only the root file system is mounted. After all of these conditions are met, run the **/etc/hdefix -a** command. This command checks to see if you are in the single-user run-level before proceeding, though it does not check for the other conditions.

If run with just the **-a** option, the **/etc/hdefix** command scans the disk error log. When run this way, the command also looks in the disk error queue for unlogged error reports. If it finds any queued reports, it processes them when it is processing the disk. Thus, if an error is reported while you are in the single-user mode for other reasons, you need not switch run-levels to get it processed.

The **hdefix** command updates the map as appropriate (remember that bad surrogate blocks get replaced, not mapped) and removes any reports for the block (there may be more than one) from the disk error log. What the block is used for is also identified. If the block seems to be in a file system, the file system is marked bad.

If any block (or surrogate of such a block) in the normally accessible portion of the disk was processed, the **hdefix** command forces an immediate reboot. The style of reboot that is forced causes the root file system to be checked while coming back up. In addition, any other file system that was marked bad is checked before it is mounted.

If you need to manually specify blocks to be fixed, there are additional arguments that can be given to this command for specifying the disk and block(s) to fix. For instance, in the swap PANIC example, the **hdefix** command could have been used directly when first in the single-user mode. The command line would have been as follows.

```
hdefix -a -D 17 0 -b 463
```

However, when a block number is specified on the command line, **hdefix** ignores the current contents of the error log and the error queue. If there happened to be a report in the log for the block being fixed, the report would still be in the log when you were done. Whereas, if the fix list is taken from the log and queue, the log is cleaned up.

Dealing With Data Loss

Although the useful life of your disk hardware has been greatly extended through the bad block handling feature, remember that the data in a bad block is lost, and its surrogate is zeroed. Bad block handling only attempts to restore sanity to the structure of the file system. You must be prepared to restore files or file systems that are important to you. Under rare circumstances, you might also need to reformat your disk if you lose the Volume Table of Contents (VTOC) block. You may need to restore the special code for booting the system (it is not in a file system).

This may sound harsh, but a little reflection shows it is not so bad. You need backups to protect you from yourself when you inadvertently remove the wrong file or files. More than one person has managed to type **rm *** while in / as **root** (super-user).

Chapter 6

SYSTEM ADMINISTRATION COMMANDS

PAGE

Chapter 6

SYSTEM ADMINISTRATION COMMANDS

COMMAND SUMMARY

The System Administration Utilities provide 18 UNIX System V commands. Also documented in this guide are 43 UNIX System V commands that are provided with the basic machine (Essential Utilities). The 43 commands provided with the basic machine are identified by a circumflex (^) in the following listing. Commands identified by an asterisk (*) are not normally executed directly from a terminal but are called as part of another process. Manual page section information is shown in parentheses for each command. Manual page section information is described in Chapter1, "INTRODUCTION." All 61 of these commands are as follows.

bcheckrc(1M)^*
brc(1M)^*
checkall(1M)
chroot(1M)
ckauto(1M)^*
clri(1M)^
crash(1M)
cron(1M)^
dcopy(1M)
dd(1)^
devnm(1M)^
df(1M)^
dfsck(1M)
drvinstall(1M)^
du(1)^
errdump(1M)^
ff(1M)
fmtflop(1M)^
fmthard(1M)^
fsck(1M)^
fsdb(1M)
fsstat(1M)^
fuser(1M)
getmajor(1M)^
getty(1M)^*
grpck(1M)
hdeadd(1M)^
hdefix(1M)^
hdelogger(1M)^
id(1)^
init(1M)^
killall(1M)^
labelit(1M)^
ldsysdump(1)
link(1M)
mkboot(1M)^
mkfs(1M)^
mknod(1M)^
mkunix(1M)^
mount(1M)^
mountall(1M)^
mvdir(1)
ncheck(1M)
newboot(1M)^
newgrp(1)^
prtvtoc(1M)^
pwck(1M)
rc0(1M)^*
rc2(1M)^*
setclk(1M)^
setmnt(1M)^
shutdown(1M)^
su(1)^
sync(1)^
sysdef(1M)
telinit(1M)^
umount(1M)^
umountall(1M)^
unlink(1M)
volcopy(1M)
whodo(1M)

To use these commands, you should be logged in on the system as **root**. Also note that many System Administration commands require that the user be a member of the "system administration group" (sys or adm). Thus, owner and group permissions are associated with the following names: **root**, **sys**, **adm**, and **bin**. Most of these commands are located in the **/etc** directory. Each of these commands is described in this chapter. Since the **etc** directory is not normally specified in usr PATH variable (except for **root**), the complete path name for the commands are identified in the command descriptions.

The descriptions include examples of how you use each command. A summary description of each of these commands is provided in Figure 6-1. Commands that are provided as part of the Essential Utilities are preceded with a circumflex (^) in Figure 6-1. Commands are listed in Figure 6-1 in alphabetical order by command name (not path name).

HOW COMMANDS ARE DESCRIBED

A common format is used to describe each of the commands. This format is as follows:

- **General:** The purpose of the command is defined. Any unique or special information about the command is also provided.
- **Command Format:** The basic command line format (syntax) is defined and the various arguments and options described.
- **Sample Command Use:** Example command line entries and system responses are provided to show you how to use the command.

In the command format descriptions the following symbology and conventions are used to define the command syntax.

- The basic command is shown in bold type. For example: **command** is in bold type.
- Arguments that you must supply to the command are shown in a special type. For example: **command** *argument*
- Command options and arguments that do not have to be supplied are enclosed in brackets ([ ]). For example:
  command [*optional arguments*]
- The pipe symbol (|) is used to separate arguments when one of several forms of an argument can be used for a given argument field. The pipe symbol can be thought of as an exclusive OR function in this context. For example:
  command [*argument1* | *argument2*]

This chapter contains sample displays which will help you understand described commands. The sample displays in this chapter and the displays on your terminal screen may differ slightly due to improvements in the product after this document was finalized. Therefore, use the displays in this chapter as samples of the type of data available. However, the data displayed on your terminal screen accurately reflects the software on your computer.

In the sample command descriptions, user inputs and 3B2 Computer responses are shown as follows.

```
This style of type is used to show system generated
responses displayed on your screen.

This style of bold type is used to show inputs
entered from your keyboard that are displayed on your
screen.

These bracket symbols, < > identify inputs from the
keyboard that are not displayed on your screen, such
as: <CR> carriage return, <CTRL d> control-d, <ESC g>
escape-g, passwords, and tabs.

This style of italic type is used for notes that
provide you with additional information.
```

Manual pages for these and other commands are provided in the *3B2 Computer System Administration Reference Manual,* the *3B2 Computer User Reference Manual,* and the *3B2 Computer Programmers Reference Manual.*

COMMAND	DESCRIPTION
^/etc/bcheckrc	Shell procedure that is executed via entries in **/etc/inittab** by the **/etc/init** command. Checks the status of the root file system and initiates a file system check if bad status is reported.
^/etc/brc	Shell procedure that is executed via entries in the **/etc/inittab** by the **/etc/init** command. Clears the mounted file system table (**/etc/mnttab**) and puts the root file system into the mount table.
/etc/checkall	The checkall procedure is a prototype and must be modified to suit local requirements. The procedure is intended to provide a fast file system check.
/etc/chroot	Executes a command relative to a new **root** path. The meaning of any initial slashes (/) in the path names is changed for the command.
^/etc/ckauto	Checks if the **UNIX** System was reconfigured during initialization. A return code of 1 means reconfigured; a return code of 0 means no reconfiguration. The command is used in the **/etc/rc.d/autoconfig** shell script.

Figure 6-1. Command Summary (Sheet 1 of 11)

COMMAND	DESCRIPTION
^**/etc/clri**	Clears file-system i-node. The command is primarily used to remove a file which does not appear in a directory.
/etc/crash	The command is used to examine an operating system core image.
^**/etc/cron**	The **cron** program is a daemon that runs in multi-user mode to execute commands at specific times. Times and actions are specified in **/usr/spool/cron/crontabs/***logname* files. The **cron** program is initiated by an entry in the **/etc/inittab** file.
/etc/dcopy	Copies and reorganizes an unmounted file system for optimum access time. This process is called "file system reorganization."
^**/bin/dd**	Converts and copies data. The source data file is copied to the standard output or to a file and is converted in the process as specified by a conversion argument. Typical conversions include EBCDIC to ASCII, ASCII to EBCDIC, uppercase to lowercase, lowercase to uppercase, and several other conversion options.

Figure 6-1. Command Summary (Sheet 2 of 11)

COMMAND	DESCRIPTION
^/etc/devnm	Identifies the special file device name associated with the specified files of a mounted system. Full path names must be used as arguments to the command.
^/bin/df	Reports the number of free disk blocks and information nodes (i-nodes) available for on-line (mounted) file systems.
/etc/dfsck	Dual file system check is used to check two different disk drives or disk partitions at the same time. Do not use this command to check the root file system.
^/etc/drvinstall	Installs or uninstalls (removes) a software driver.
^/bin/du	Reports the disk usage as the number of blocks used for a specified path. When no arguments are given, the current path is used.
^/etc/errdump	Displays the error log maintained in Nonvolatile RAM (NVRAM).
/etc/ff	Lists file names and statistics for a file system.

Figure 6-1. Command Summary (Sheet 3 of 11)

COMMAND	DESCRIPTION
^/etc/fmtflop	Formats and verifies floppy disks.
^/etc/fmthard	Writes the Volume Table Of Contents (VTOC) to a hard disk. The VTOC defines the disk partitions and how the partitions are accessed.
^/etc/fsck	Audits and interactively repairs inconsistent conditions in a file system.
/etc/fsdb	File system debugger.
^/etc/fsstat	Reports file system status. The command is used to determine if the file system needs checking before it is mounted.
/etc/fuser	Lists the process identifications of processes that are using a file or a file structure.
^/etc/getmajor	Outputs all slot/major numbers in the equipped device table for the requested device.

Figure 6-1. Command Summary (Sheet 4 of 11)

COMMAND	DESCRIPTION
^/etc/getty	This command is used by the system to set the modes of a terminal as part of the login process. You DO NOT execute this command by typing getty. Getty is normally invoked by **init** as the first step in logging users in on the system.
/etc/grpck	Verifies all entries in the group file. Verification includes the number of fields, group name, group identification checks, and whether all login names appear in the password file. The default group file is **/etc/group**.
^/etc/hdeadd	Adds/deletes reports to/from the hard disk error log.
^/etc/hdefix	Reports or changes the bad block mapping for a hard disk.
^/etc/hdelogger	As a daemon, the command logs hard disk errors reported by the disk drivers. As a command entered at a terminal, the command reports the errors logged.

Figure 6-1. Command Summary (Sheet 5 of 11)

COMMAND	DESCRIPTION
^**/usr/bin/id**	Outputs the user and group identifications and the names of the invoking process. If the effective and real identifications are different, both are output.
^**/etc/init**	Initializes process control. It is the last step in a system boot procedure.
^**/etc/killall**	Terminates all active processes which are not related to the shut down procedure. (See **/etc/shutdown**.)
^**/etc/labelit**	Provides initial labels for unmounted disk or tape file systems.
/etc/ldsysdump	Loads multiple system dumps from floppy disk to a single file on hard disk. Hard disk files can be examined using **/etc/crash**.
/etc/link	Creates a new link to an existing file. (See **/etc/unlink**.)
^**/etc/mkboot**	Converts an **/etc/master.d/***file* to a bootable file compatible with the self-configuration boot program.

Figure 6-1. Command Summary (Sheet 6 of 11)

COMMAND	DESCRIPTION
^**/etc/mkfs**	Constructs a file system.
^**/etc/mknod**	Constructs (builds) a special file. The "make node" command makes a directory entry and corresponding i-node for a special file.
^**/etc/mkunix**	Makes a new UNIX Operating System. Copies the in-core version to a specified file.
^**/etc/mount**	This command is used to mount a removable file system on a specified directory path. (See **/etc/umount**.)
^**/etc/mountall**	This command is a prototype shell script used to mount one or more removable file systems. (See **/etc/umountall**.)
/etc/mvdir	An executable file (shell script) that renames (moves) directories.
/etc/ncheck	Displays the information node (i-node) numbers and path names for a file system. The file system is specified by the device file (/dev/dsk/c1d0s0, /dev/dsk/c1d0s2, /dev/rdsk/c1d0s0, etc). Either the raw or block device can be specified.

Figure 6-1. Command Summary (Sheet 7 of 11)

COMMAND	DESCRIPTION
^**/etc/newboot**	Loads **/lib/lboot** (or **/lib/olboot**) and **/etc/mboot** onto the device boot partition. Verifies that each given file will fit in the specified entry and calls **/bin/dd** to move it in.
^**/bin/newgrp**	Logs in to a new group.
^**/etc/prtvtoc**	Prints the volume table of contents for a device. The raw hard disk device must be specified.
/etc/pwck	Scans the password file (/etc/passwd) and notes any inconsistencies. The checks include validation of the number of fields, login name, user identification, group identification, and whether the login directory and optional program name exists.
^**/etc/rc0**	Executes commands to stop the operating system. Leaves the system in a state where it is safe to turn off the power or to go to firmware. Executes files in **/etc/shutdown.d** directory.

Figure 6-1. Command Summary (Sheet 8 of 11)

COMMAND	DESCRIPTION
^**/etc/rc2**	Executes commands to take the system to the multi-user mode. Starts all system daemons before the terminal lines are enabled for the multi-user mode (run-level 2). Executes files in **/etc/rc.d** directory.
^**/etc/setclk**	Sets the system interval time from the hardware time-of-day clock. Normal execution of the command is at system initialization via the **/etc/inittab** file.
^**/etc/setmnt**	Establishes a mounted file system table (mnttab) in directory /etc.
^**/etc/shutdown**	Terminates all currently running processes in an orderly and cautious manner. The command initiates the specified run level (default is single-user).
^**/bin/su**	The "switch user" command enables a user to become another user without logging off.
^**/bin/sync**	Updates the super block. Flushes all previously unwritten system buffers out to disk. When the system is to be stopped, sync is used to insure file system integrity.

Figure 6-1. Command Summary (Sheet 9 of 11)

COMMAND	DESCRIPTION
/etc/sysdef	Outputs system configuration information in a tabular format.
^/etc/telinit	Telinit arguments direct (tell) the **init** command what actions are to be done. The **telinit** command can only be run by **root** or a member of the **sys** group.
^/etc/umount	Unmounts removable file system. (See **/etc/mount**.)
^/etc/umountall	This command is a prototype shell script used to unmount one or more removable file systems. (See **/etc/mountall**.)
/etc/unlink	Removes a link to a file.

Figure 6-1. Command Summary (Sheet 10 of 11)

COMMAND	DESCRIPTION
/etc/volcopy	Volume copy is used to copy an entire file system with label checking from one raw (character) device partition to another raw (character) device partition. No compression is done as with **/etc/dcopy**. The copy is a literal copy.
/etc/whodo	An executable file (shell script) that combines the **who** and **ps** command to report what each logged-in user is doing.

Figure 6-1. Command Summary (Sheet 11 of 11)

COMMAND DESCRIPTIONS

/etc/bcheckrc — Prepare for Multi-User Mode

General

The **bcheckrc** command is a shell script (Figure 6-2) used to check the sanity of the **root** file system. The root file system (**/dev/dsk/c1d0s0**) is checked if the sanity check returns a status code not equal to zero. The **bcheckrc** command is executed at system initializtion by an entry in the **/etc/inittab** file.

```
# file system check the root file system if necessary

rootfs=`/etc/devnm /  |  grep '[ <tab>]/$'  |  ( read a b;  echo ${a} )`

msg=`/etc/fsstat ${rootfs} 2>&1`

if [ $? -ne 0 ]
then
        echo "
       ${msg}
       The root file system (${rootfs}) is being checked automatically."
       /etc/fsck -y -D -b ${rootfs}
fi
```

Figure 6-2. Typical /etc/bcheckrc File

Command Format

The format of the **bcheckrc** command consists of only the command name. Normally, the command is executed by an entry in the **/etc/inittab** file and is not entered at a data terminal. Hence, sample command line entries and system responses are not provided for this command.

/etc/brc — Clears Mounted File System Table

General

The **brc** command is a shell script (Figure 6-3) used to clear the mount table and then create the **root** file system mount table entry (**/etc/mnttab**). The **brc** command is executed at system initializtion by an entry in the **/etc/inittab** file.

```
# put root into mount table
/etc/devnm / | grep -v swap | /etc/setmnt
```

Figure 6-3. Typical /etc/brc File

Command Format

The format of the **brc** command consists of only the command name. Normally, the command is executed by an entry in the **/etc/inittab** file and is not entered at a data terminal. Hence, sample command line entries and system responses are not provided for this command.

/etc/checkall — Fast File System Check

General

The **checkall** command is a shell script that is used to run **/etc/fsck** on all file systems. This shell script provides the system administrator a shortcut for checking all file systems. The system must be in the single-user mode (run-level S or 1) to check the various file systems. The script provided in **/etc/checkall** is a sample and should be modified to suit your situation. Hence, sample command line entries and system responses are not provided for this command.

Command Format

Based on the sample **/etc/checkall** script the command format is as follows.

/etc/checkall *[fsck options]*

Refer to the sample shell script provided in **/etc/checkall** for additional information (Figure 6-4).

```
#
# /etc/checkall
#
# Check every filesystem.
#

rootfs=`/etc/devnm /  |  grep '[          ]/$'  |  ( read a b;  echo ${a} )`

fstab=`
        while read dev fs readonly dummy
        do
                case "${dev}" in
                '#' | '')
                        continue ;;
                esac
                echo "${dev}"
        done < /etc/fstab `

exec /etc/fsck $* ${rootfs} ${fstab}
```

Figure 6-4. Sample /etc/checkall File

/etc/chroot — Change root Directory for a Command

General

The **chroot** command is used to execute a command relative to a new **root** path. The meaning of any initial slashes (/) in the path names is changed for the command. Only **root** can use this command.

Command Format

The general format of the **chroot** command is as follows.

chroot *newroot command*

The *command* is executed relative to the *newroot* path name.

Sample Command Use

The following shows the command line entries and system responses for establishing **/bin** as the root directory (/) and executing **/bin/sh** in that environment. The standard directories for commands are **/bin** and **/usr/bin**. The **date** command is used to show that in the new environment these directories do not exist. The control-d is used to exit the shell (**/bin/sh**).

```
# chroot /bin /sh<CR>
# date<CR>
date: not found
# <CTRL d>
# date<CR>
Sun Aug 12 10:42:37 EDT 1984
#
```

/etc/ckauto — Check If System Reconfiguration Occurred at Boot

General

The **ckauto** command is used to check for a system reconfiguration at boot time (system initialization). Normal the command is executed when the system is powered up via the **/etc/rc.d/autoconfig** shell script. The **ckauto** command is also executed at system initialization by the **/etc/bzapunix** shell script. The **/etc/bzapunix** shell script is called by an entry in the **/etc/inittab** file.

Command Format

The format of the **ckauto** command consists of only the command name. Normally, the command is executed by the **/etc/rc.d/autoconfig** shell script. This shell script is called when **/etc/rc2** is executed. Hence, sample command line entries and system responses are not provided for this command. Refer to the **/etc/rc.d/autoconfig** for an example use of the command.

/etc/clri — Clear I-Node

General

The **clri** command is used to clear (zero) one or more information nodes (i-nodes) on a specified file system. The primary use of the command is to remove a file that does not have a directory. The command should be used with extreme care. When an i-node is cleared using the **clri** command, the associated data blocks are reported as missing when **/etc/fsck** is run on the file system.

Command Format

The general format of the **clri** command is as follows.

clri *device i-nodes*

The *device* argument specifies the special device file of the file system containing the *i-nodes* to be cleared. One or more i-nodes can be specified by the *i-nodes* argument.

Sample Command Use

The following shows the command line entries and system responses associated with clearing an information node (i-node). In this example, a file system on floppy disk is used to show the effects of removing an i-node. The file system is first checked using the **/etc/fsck** command. The **/etc/ncheck** command is used to list the i-nodes and files. The **clri** command is used to clear i-nodes 31 and 25. The file system is then checked to show the effect of clearing the i-nodes.

```
# fsck /dev/SA/diskette1<CR>

  /dev/SA/diskette1
  File System: rar Volume: 2032

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  24 files 584 blocks 822 free
# ncheck /dev/SA/diskette1<CR>
/dev/SA/diskette1:
3       /.profile
4       /.news_time
12      /305-323/.
13      /305-323/ch6.package
14      /305-323/disk.stats
15      /305-323/help
16      /305-323/trademarks
17      /305-323/ch1.general
18      /305-323/ch2.install
19      /305-323/ch3.files
20      /305-323/ch4.functions
21      /305-323/ch5.fsck
22      /305-323/cmds
23      /305-323/appendix.d
24      /305-323/gettydefs.fig
25      /305-323/chx.advice
26      /305-323/appendix.c
27      /305-323/toc
28      /305-323/runstates
29      /305-323/appendix.a
30      /305-323/appendix.b
31      /305-323/key
#
```

Continued

Continued from previous screen

```
# clri /dev/SA/diskette1 31 25<CR>
clearing 31
clearing 25
# fsck /dev/SA/diskette1<CR>

  /dev/SA/diskette1
  File System: rar Volume: 2032

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  UNALLOCATED  I=25  OWNER=root MODE=0
  SIZE=0 MTIME=Dec 31 19:00 1969
  NAME=/305-323/chx.advice (EMPTY) -- REMOVED
  UNALLOCATED  I=31  OWNER=root MODE=0
  SIZE=0 MTIME=Dec 31 19:00 1969
  NAME=/305-323/key (EMPTY) -- REMOVED
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  FREE INODE COUNT WRONG IN SUPERBLK
  FIX? y<CR>

  ** Phase 5 - Check Free List
  18 BLK(S) MISSING
  BAD FREE LIST
  SALVAGE? y<CR>

  ** Phase 6 - Salvage Free List
  22 files 548 blocks 858 free
  ***** FILE SYSTEM WAS MODIFIED *****
#
```

/etc/crash — Examine System Images

General

The **crash** command is used to interactively examine a system memory image.

Command Format

The general format of the **crash** command is as follows.

crash [*system*] [*namelist*]

The default *system* core image is the **/dev/mem** file. When examining system dumps on floppy disk, **/dev/SA/diskette1** is specified as the *system* memory image. The *namelist* is the text file used to boot the machine. The default *namelist* is **/unix**.

Refer to the **crash**(1M) manual pages for a description of the various commands provided by **crash**.

Sample Command Use

The use of the **crash** command to examine a system memory image is complex. Refer to the *AT&T 3B2 Computer Crash Analysis Guide* for a complete description of how to use **crash**.

/etc/cron — Clock Daemon

General

The **cron** command is always running in the multi-user mode (run-level 2). Note that once executed, the **cron** command runs continuously. This type of command is called a daemon. The **cron** command is executed by the **/etc/rc2** program as the result of the **cron** entry in the **/etc/rc.d** directory. The key to using **cron** is in making entries into the applicable **/usr/spool/cron/crontabs/***logname* files.

Providing that the user's logname is in the **/usr/lib/cron/cron.allow** file, a users can establish their own crontabs file using the **crontab** command. As **root**, you can either use the **crontab**(1) command or edit the appropriate file under **/usr/spool/cron/crontabs** to establish the appropriate entries. Refer to the **crontab**(1) command manual page for additional information.

The **cron** daemon examines the contents of the files in the **/usr/spool/cron/crontabs** directory for changes during process initialization and when a file changes.

CRON Table Format

The format of a **/usr/spool/cron/crontabs/***logname* file consists of six fields. The first five fields specify when the command line in the sixth field is to be executed. These fields are as follows:

- Minute of the hour (0-59)
- Hour of the day (0-23)
- Day of the month (1-31)
- Month of the year (1-12)
- Day of the week (0-6, where 0=Sunday)
- Command line.

The first five fields can be entered in terms of:

- A number in the acceptable range for the field
- Two numbers separated by a minus to indicate an inclusive range
- A list of numbers separated by commas to indicate each number
- An asterisk to indicate all legal values in the range for the field.

The sixth field specifies the command line. A percent sign (%) in this field is translated to a new-line character. The command field is executed by the shell up to a percent sign or the end of a line. Additional lines that do not follow the format of the first five fields are made available to the command in the sixth field as standard input.

Sample Command Use

The following is a typical **/usr/spool/cron/crontabs/root** file. This file runs:

- **calendar** every day at 01:00 AM
- **uudemon.hour** every hour at the 11 and 41 minute marks
- **uudemon.cleanup** every day at 11:45 PM
- **uudemon.poll** every hour at the 1 and 30 minute marks.

```
# cat /usr/spool/cron/crontabs/root<CR>
0 1 * * * /usr/bin/calendar -
41,11 * * * * /usr/lib/uucp/uudemon.hour > /dev/null
45 23 * * * ulimit 5000; /bin/su uucp -c "/usr/lib/uucp/uudemon.cleanup" > /dev/null 2>&1
1,30 * * * * /usr/lib/uucp/uudemon.poll > /dev/null
#
```

/etc/dcopy — Reorganize File System for Optimum Access Time

General

The **dcopy** command is used to copy and reorganize an *unmounted* file system. The **dcopy** command produces the best results when the source file system is copied from the raw device partition to a block device of another device partition. The destination partition must be large enough to accommodate the file system being reorganized. File system reorganization involves compressing the directories by removing empty entries and spacing consecutive blocks of a file by the optimal rotational gap (9). The **dcopy** command also provides options to control the placement of files and subdirectories. Note that the Simple Administration compress function (**sysadm compress**) calls the **/etc/cmpress** shell script. The **/etc/cmpress** shell script only removes empty entries and does not reorganize the placement of directories and files in the file system. As files and directories are created and removed, the file system becomes randomly organized. File system reorganization is necessary to maintain an efficient file system.

The **dcopy** provides a 10 second **DEL if wrong** pause when the command is executed. If no action is taken, command execution continues after 10 seconds. The command is aborted by typing a DEL (DEL key). A copy of a file system made using the **dcopy** command must be checked using **fsck** before the file system can be mounted as a read/write file system. Executing **fsck** sets the status of the file system to OK thus enabling the file system to be mounted as a read/write file system. Refer to Chapter 3, "ADMINISTRATIVE TASKS," for additional information on file system reorganization. The source file system (partition) should be checked before executing the **dcopy** command.

Command Format

The general format of the **dcopy** command is as follows.

dcopy [**-s***X*] [**-a***days*] [**-d**] [**-v**] [-f*fsize*:isize] *source destination*

The various options and arguments of the **dcopy** command are as follows.

-a*days*	This option causes files not accessed in the specified number of *days* to be placed after the free blocks of the destination file system. When the **-a** option is not specified, the *days* defaults to 7. When the **-a** option is specified without specifying the number of days, no movement occurs.
-d	This option causes the order of the directory entries in the destination file system to be the same as the source file system. When the **-d** option is not specified (the default case), subdirectories are moved to the beginning of directories.

-f *fsize:isize* This option specifies the size of the destination file system and the size of the information node (i-node) list. When the **-f** option is not specified, the sizes of the source file system are used. When the **-f** option is specified with only the block size (*fsize*), then the i-node list is defined as one-fourth of the block size.

-s*X* This option specifies the number of blocks per cylinder and the rotational gap for the device. The format of this option for the various configurations of disk drives is as follows.

DEVICE	-s *blocks/cylinder:gap*
72M FUJITSU Hard Disk	**-s198:9**
72M MICROPOLIS Hard Disk	**-s144:9**
72M PRIAM Hard Disk	**-s198:9**
72M WREN* II Hard Disk	**-s162:9**
30M Hard Disk	**-s90:9**
10M Hard Disk	**-s72:9**
Floppy Disk	**-s18:1**

-v This option causes the **dcopy** command to report the number of files processed and the sizes of the source and destination free lists.

* Trademark of Control Data Corporation

source This argument specifies the special device file for the source file system. This should be a raw (character) device file (partition).

destination This argument specifies the special device file for the destination file system (partition). This should be a block device file (partition) for a file system of suitable size to hold the source file system.

Sample Command Use

Using Hard Disk Intermediate Storage

The following command line entry and system responses show the use of the **dcopy** command to reorganize a file system maintained on hard disk. In this example the intermediate hard disk partition (partition c1d1sa) could also be used as a backup to the file system on partition c1d1s8. The file system on partition c1d1s8 of the second hard disk is copied to partition c1d1sa on the same hard disk using the **dcopy** command. The default options of the **dcopy** command are used; only the verbose (**-v**) option is specified in the example. Disk partitions c1d1s8 and c1d1sa are equally sized partitions. After copying partition c1d1s8 to partition c1d1sa, **fsck** is run on partition c1d1sa. Partition c1d1sa is then copied back to partition c1d1s8 using the **volcopy** command.

At the start of this example, the system is in run-level S (single user) with partitions c1d1s8 and c1d1sa of the second hard disk unmounted.

```
# fsck -D /dev/rdsk/c1d1s8<CR>

  /dev/rdsk/c1d1s8
  File System: usr2 Volume: 2032

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  111 files 874 blocks 14480 free
# dcopy -v /dev/rdsk/c1d1s8 /dev/dsk/c1d1sa<CR>
From: /dev/rdsk/c1d1s8, to: /dev/dsk/c1d1sa? (DEL if wrong)
old filesize = 7800, old inode size = 123
old stepsize = 9, old cylinder size = 90
new filesize = 7800, new inode size = 123
new stepsize = 9, new cylinder size = 90
Available mem 466944, got 63488 for inodes (that's 992 inodes)
Pass 1: Reorganizing file system
Pass 2: Fixing inums in directories
Pass 3: Remake freelist
Files:          111
Free blocks in:         7240
Free blocks out:        7240
Complete
# fsck -D /dev/rdsk/c1d1sa<CR>

  /dev/rdsk/c1d1sa
  File System: usr2 Volume: 2032

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  FILE SYSTEM STATE SET TO OKAY
  111 files 874 blocks 14480 free
  *** FILE SYSTEM WAS MODIFIED ***
```

Continued

Continued from previous screen

```
# volcopy usr2 /dev/rdsk/c1d1sa  - /dev/rdsk/c1d1s8 -<CR>
/dev/rdsk/c1d1s8 less than 48 hours older than /dev/rdsk/c1d1sa
To filesystem dated:  Sat Feb  9 05:57:17 1985
Type 'y' to override:     y<CR>
From: /dev/rdsk/c1d1sa, to: /dev/rdsk/c1d1s8? (DEL if wrong)
  END: 15600 blocks.
volcopy: cannot access /etc/log/filesave.log
#
# fsck -D /dev/rdsk/c1d1s8<CR>

  /dev/rdsk/c1d1s8
  File System: usr2 Volume: 2032

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  111 files 874 blocks 14480 free
#
```

Using Cartridge Tape Intermediate Storage

An example of *dcopying* to a cartridge tape is not provided. The use of cartridge tape as an intermediate storage device for the reorganization of a file system requires that the cartridge tape be used as a block device. Because of the time required to **dcopy** to a cartridge tape and then copy the information back to the hard disk, it is more efficient to use a hard disk partition as the intermediate storage device. This is true even if you must first make a partition available for use by copying the existing data to tape using **/etc/ctccpio**. Hence, it is recommended that cartridge tape NOT BE used as an intermediate storage device for file system reorganization.

/bin/dd — Convert and Copy a File

General

The **dd** command is used to copy and convert data from a specified input file to a specified output. The command acts like a filter in that the input (source) file is not changed (converted), only the output data is converted to the specified form. The **dd** command is especially suited to handle input and output on devices (disk and tape) because it permits the reading and writing of arbitrary block sizes.

Command Format

The general format of the **dd** command is as follows.

dd *option=value* . . .

The *option=value* argument form is used to specify the source, destination, and conversion options. The following summarizes the recognized forms of the argument.

if=*file*	Defines the input (source) file name. When omitted, the standard input is used.
of=*file*	Defines the output (destination) file name. When omitted, the standard output is used.
ibs=*n*	Defines the input block size. Default block size is 1024 bytes.
obs=*n*	Defines the output block size. Default block size is 1024 bytes.
bs=*n*	Defines both the input and output block size. Default block size is 1024 bytes.

cbs=*n*	Defines the conversion buffer size. This argument is only used when converting from ASCII to EBCDIC.
skip=*n*	Defines the number of input blocks to skip before copying data to the output.
seek=*n*	Defines the number of blocks to seek before writing to the output file of data.
count=*n*	Defines the number of input blocks to be copied to the output.
conv=ascii	Specifies conversion from EBCDIC to ASCII.
conv=ebcdic	Specifies conversion from ASCII to EBCDIC.
conv=ibm	Specifies conversion from ASCII to EBCDIC. This conversion uses a slightly different mapping of characters than does the **ebcdic** conversion.
conv=lcase	Specifies conversion from uppercase to lowercase letters.
conv=ucase	Specifies conversion from lowercase to uppercase letters.
conv=swab	Specifies a byte conversion in which every pair of bytes is swapped.
conv=noerror	Specifies that processing is to continue when errors are detected.
conv=sync	Specifies that every input block is to be padded to the block size specified by the **ibs** argument. Default input block size is 1024 bytes.
conv=...,...	Specifies a list of conversions. Selected conversions are listed with each stated conversion separated from the others by a comma.

Arguments in the previous listing that specify a number of bytes can be entered as a number followed by a designator to specify multiplication by 1024, 512, or 2. A pair of numbers separated by an **x** is used to mean a product. These designators are as follows.

*n***k** Multiply *n* by 1024.

*n***b** Multiply *n* by 512.

*n***w** Multiply *n* by 2.

*n***x***m* Multiply *n* by *m*.

Sample Command Use

The following command line entries and system responses show how you use the **dd** command to convert a file containing a mix of uppercase and lowercase letters to a file containing only lowercase letters. The **cat** command is used to display the contents of the files. Note that the **dd** command reports the number of whole and partial blocks that are input and output. In this example, the number of bytes (characters) in the **file1** and **file2** is 546. Therefore, for 512 bytes per block (**bs=512**), the files contain one whole block and a partial block. The **dd** command reports one whole block and one partial block as **1+1 blocks**. The first number is the number of whole blocks. In the following example the input block size is defined as 256 bytes, and the output block size is defined as 512 bytes to further illustrate how the **dd** command reports the number of blocks. The number of input blocks (2+1) is with respect to 256-byte blocks. The number of output blocks (2+1) is with respect to 512-byte blocks.

```
# cat file1<CR>
ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZ
!@#$%^&*()_+~ !@#$%^&*()_+~
QWERTYUIOP{!ASDFGHJKL:"}QWERTYUIOP{!ASDFGHJKL:"}
ZXCVBNM<>?1234567890-='ZXCVBNM<>?1234567890-='
ABcdeFGHIJklMNOpqRSTURwXYZABcdeFGHIJklMNOpqRSTURwXYZ
ABcdeFGHIJklMNOpqRSTURwXYZABcdeFGHIJklMNOpqRSTURwXYZ
ABcdeFGHIJklMNOpqRSTURwXYZABcdeFGHIJklMNOpqRSTURwXYZ
ABcdeFGHIJklMNOpqRSTURwXYZABcdeFGHIJklMNOpqRSTURwXYZ
ABcdeFGHIJklMNOpqRSTURwXYZABcdeFGHIJklMNOpqRSTURwXYZ
ABcdeFGHIJklMNOpqRSTURwXYZABcdeFGHIJklMNOpqRSTURwXYZ
01234567890123456789012345678901234567890123456789
$ dd if=file1 of=file2 ibs=256 obs=512 conv=lcase<CR>
2+1 blocks in
1+1 blocks out
# cat file2<CR>
abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz
!@#$%^&*()_+~ !@#$%^&*()_+~
qwertyuiop{!asdfghjkl:"}qwertyuiop{!asdfghjkl:"}
zxcvbnm<>?1234567890-='zxcvbnm<>?1234567890-='
abcdefghijklmnopqrsturwxyzabcdefghijklmnopqrsturwxyz
abcdefghijklmnopqrsturwxyzabcdefghijklmnopqrsturwxyz
abcdefghijklmnopqrsturwxyzabcdefghijklmnopqrsturwxyz
abcdefghijklmnopqrsturwxyzabcdefghijklmnopqrsturwxyz
abcdefghijklmnopqrsturwxyzabcdefghijklmnopqrsturwxyz
abcdefghijklmnopqrsturwxyzabcdefghijklmnopqrsturwxyz
01234567890123456789012345678901234567890123456789
#
```

Other uses of the **dd** command are for duplicating floppy disks and verifying floppy disks. Refer to Chapter 3, "ADMINISTRATIVE TASKS," for examples of verifying and duplicating floppy disks using the **dd** command.

/etc/devnm — Identify Device Name

General

The **devnm** command is used to identify the special device file associated with a file or with directory names. The file and directory names must be expressed as complete path names to the **devnm** command. The **devnm** command is used by **/etc/rc.d/MOUNTFILESYS** with **grep** and **/etc/setmnt** to construct a mount table entry for the **root** device. Normally, the **devnm** command is used in shell script.

Command Format

The general format of the **devnm** command is as follows.

devnm [*names*]

The *names* argument specifies one or more files of a mounted file system. The file names must be expressed as complete path names.

Sample Command Use

The following command line entries and system responses show how to determine the special device file associated with several mounted file system names.

```
# devnm /<CR>
/dev/dsk/c1d0s0 /
# devnm /usr/bin<CR>
/dev/dsk/c1d0s2 /usr/bin
# devnm /usr/lib/cron<CR>
/dev/dsk/c1d0s2 /usr/lib/cron
#
```

/bin/df — Output Number of Free Disk Blocks

General

The **df** command is used to output the number of **free** disk blocks and **free** i-nodes available for an on-line (mounted) file system. The number of free disk blocks and i-nodes tell you how much space is available on a file system. Refer to the **du** command for reporting the number of disk blocks used in a directory.

Disk space is allocated and reported in a unit of measurement called **blocks**. Status reporting commands define a **block** of data as 512 bytes (characters). This is true for 1024-byte and 512-byte block file systems. Note that a character (byte) is 8-bits. Logical blocks are the true block size used to construct a file system. The logical block size used on this system is 1024 bytes. Most file system commands expect arguments specifying blocks in terms of 512-byte blocks.

When a file is created, an **i-node** is allocated for it. The directory entry is made that contains the name and i-node number identifying the file. The i-node number is a pointer to the file. The i-node number is used as an index into a table that contains a description of the file. This file description is called an **i-node**. The **i-node** contains the following information about the file:

- User and group identification.
- Access permissions.
- Physical disk or tape addresses for the file contents. A maximum of 13 device addresses can be entered.
- Size of the file. The maximum file size is limited by the amount of available disk space.
- Creation date, date last used, and date of last change.

- Number of links to the file (number of times the file appears in the directory).
- Type identifier (directory, ordinary file, or special file).

Note that you must be concerned with both the number of free blocks and the number of free i-nodes when evaluating the capability of storing more data on a file system. When creating large numbers of small files, you can run out of i-nodes before you actually run out of free data blocks. When this occurs, you must either delete existing files (purge your files of obsolete data) or down-load files to floppy disk.

Command Format

The general format of the **df** command is as follows.

df [**-t**] [**-f**] [*file system(s)*]

The **-t** argument causes the total number of allocated blocks to be reported.

The **-f** argument causes only the number of free blocks to be reported.

The *file system(s)* argument identifies the file system or systems to be examined. If no argument is provided, all mounted file systems are reported. The file system can be identified by either the device name (for example **/dev/dsk/c1d0s2**) or the mounted directory path name (for example, **/usr**).

Sample Command Use

The following sample command line entry and system responses show how you can examine all mounted file systems. The simplest form of the **df** command is used in which no arguments are specified.

```
# df<CR>
/usr       (/dev/dsk/c1d0s2):    17338 blocks    3471 i-nodes
/          (/dev/dsk/c1d0s0):     1668 blocks    1015 i-nodes
#
```

The following command line entry and system responses show how you can examine a particular mounted file system. In this example, the number of allocated blocks and i-nodes are also reported.

```
# df -t /usr<CR>
/usr       (/dev/dsk/c1d0s2):    17338 blocks    3471 i-nodes
                  total:     43830 blocks    5472 i-nodes
#
```

/etc/dfsck — Dual File System Consistency Check and Interactive Repair

General

The **dfsck** command is used to check two file systems on different disk drives or different disk partitions at the same time. Do not use **dfsck** to check the **root** file system. Refer to Chapter 4, "FILE SYSTEM CHECKING AND REPAIR," for more information.

Command Format

The general format of the **dfsck** command is as follows.

dfsck [*options*] *fsdevice* - [*options*] *fsdevice*

The various *options* of the **dfsck** command are as follows.

-D	Directories are checked for bad blocks. This option is used to check file systems for damage after a system crash.
-f	Specifies that a fast file system check be done. Only Phase 1 (check blocks and sizes) and Phase 5 (check file system free list) are executed for a fast check. Phase 6 (reconstruct free list) is run only if necessary.
-n	Specifies a "no" response for all questions.
-q	Specifies a "quiet" file system check. Output messages from the process are suppressed.

-s*X* Specifies an unconditional reconstruction of the free list. Following the reconstruction of the free list, the system should be rebooted so that the in-core copy of the superblock is updated. The *X* argument specifies the number of blocks-per-cylinder and the number of blocks to skip (rotational gap). The format of the argument is as follows. The default values are the values specified when the file system was created. The format for various configurations of disk and cartridge tape drives is as follows.

DEVICE	-s *blocks/cylinder:gap*
72M FUJITSU Hard Disk	**-s198:9**
72M MICROPOLIS Hard disk	**-s144:9**
72M PRIAM Hard Disk	**-s198:9**
72M WREN* II Hard Disk	**-s162:9**
30M Hard Disk	**-s90:9**
23M Cartridge Tape	**-s31:1**
10M Hard Disk	**-s72:9**
Floppy Disk	**-s18:1**

-S*X* Specifies a conditional reconstruction of the free list. The format of the *X* argument is the same as described for the **-s** option.

* Trademark of Control Data Corporation

-t *file*	Specifies a scratch file for use when the file system check requires additional memory. If this option is not specified, the process asks for a file name when more memory is needed.
-y	Specifies a "yes" response for all questions.

Sample Command Use

The following command line entry and system responses show how to check two different file systems on different disk drives at the same time. The disk partitions **/dev/SA/diskette1** and **/dev/dsk/c1d0s2** are each checked and the free list rebuilt (**-s** option). In this example, the system is already in the single-user run level (only the **root** file system is mounted).

```
# dfsck -s /dev/SA/diskette1 -s /dev/dsk/c1d0s2<CR>
1 will identify /dev/SA/diskette1
2 will identify /dev/dsk/c1d0s2
Precede every answer with 1 or 2
as in 'ly' for /dev/SA/diskette1

1 /dev/SA/diskette1
1 /dev/SA/diskette1      File System:  Volume:


2 /dev/dsk/c1d0s2
2 /dev/dsk/c1d0s2    File System: usr Volume: 2032

2 /dev/dsk/c1d0s2    ** Phase 1 - Check Blocks and Sizes
1 /dev/SA/diskette1 ** Phase 1 - Check Blocks and Sizes
1 /dev/SA/diskette1 ** Phase 2 - Check Pathnames
1 /dev/SA/diskette1 ** Phase 3 - Check Connectivity
1 /dev/SA/diskette1 ** Phase 4 - Check Reference Counts
1 /dev/SA/diskette1 ** Phase 5 - Check Free List
1 /dev/SA/diskette1 ** Phase 6 - Salvage Free List
1 /dev/SA/diskette1 2 files 2 blocks 1394 free
1 /dev/SA/diskette1 ******FILE SYSTEM WAS MODIFIED *****
2 /dev/dsk/c1d0s2    ** Phase 2 - Check Pathnames
2 /dev/dsk/c1d0s2    ** Phase 3 - Check Connectivity
2 /dev/dsk/c1d0s2    ** Phase 4 - Check Reference Counts
2 /dev/dsk/c1d0s2    ** Phase 5 - Check Free List
2 /dev/dsk/c1d0s2    ** Phase 6 - Salvage Free List
2 /dev/dsk/c1d0s2    499 files 7394 blocks 1816 free
2 /dev/dsk/c1d0s2    ******FILE SYSTEM WAS MODIFIED *****
>>> DFSCK DONE <<<
#
```

/etc/drvinstall — Install or Uninstall a Driver

General

The **drvinstall** command is used to add or delete drivers. The primary use of this command is in the INSTALL and UNINSTALL shell scripts used by the **sysadm installpkg** and **sysadm removepkg** commands. For installation, the **drvinstall** command creates specially formatted files for use by the self-configuration boot program from specified *object*, *master*, and *system* input files. If the driver is a software driver, the **drvinstall** process assigns the first available major number to the driver. Major numbers for software drivers range from 48 to 127, inclusive. Major numbers outside this range are reserved for hardware devices and integral drivers. The assigned major numbers are identified in the *master* file.

Command Format

The general format of the **drvinstall** command is as follows.

drvinstall *options*

The various *options* of the **drvinstall** command are as follows.

-b	Inhibits the generation of the bootable object file.
-d *object*	Specifies the path name of the *object* file to be used. If this option is omitted, the **/boot** directory is used.
-f	This option is used with the **-u** option to disable the dependency check when removing a driver.
-m *master*	Specifies the path name of the *master* file to be used. If this option is omitted, the **/etc/master.d** directory is used.
-n	Inhibits any edit of the *system* file.

-o *directory*	Specifies the path name of the bootable file. If this option is omitted, the **boot** directory is used.
-s *system*	Specifies the path name of the *system* file to be used. If this option is omitted, the **/etc/system** file is used.
-v *version*	Specifies the version number of the **drvinstall** command that is compatible with the *master* file being used. This option must be specified. The version number to be used is **1.0**.
-u	Removes (uninstalls) a driver. Either the **-d** or the **-m** option must be specified with the **-u** option. Removing a driver involves (1) Removing the bootable *object* file, (2) Replacing the major number in the *master* file with a dash if the driver is a software driver, and (3) Deleting the INCLUDE statement from the *system* file.
-x	Enables the debugging output.

Sample Command Use

The following command line entry is typical for the installation of a hardware driver. The command runs silently.

```
# drvinstall -m /etc/master.d/ports -o /usr/src/uts/3b2/io/ports.o -v1.0<CR>
#
```

The following command line entry and system response are typical for the installation of a software driver. The command returns the major number assigned to the driver.

```
# drvinstall -m /etc/master.d/sxt -o /usr/src/uts/3b2/io/sxt.o -v1.0<CR>
48
#
```

/bin/du — Output Disk Usage Summary

General

The **du** command is used to output the number of disk blocks **used** for one or more directories. The number of disk blocks used tells you how much space has been taken-up by the data stored in the directory. Refer to the **df** command for reporting the number of available disk blocks and i-nodes on a file system(s). Also, refer to the **df** command for an explanation of disk blocks and i-nodes.

Command Format

The general format of the **du** command is as follows.

du [**-ars**] [*name(s)*]

The **-a** argument causes the command to generate a report for each file under the *name(s)* argument.

The **-r** argument causes the command to report those directories and files that cannot be accessed.

The **-s** argument causes the command to report only a grand total (sum) of the number of blocks of data used under the *name(s)* argument.

The *name(s)* argument identifies the files or directories that are to be examined. When a directory is identified, all paths under that directory are examined and the number of blocks used by each file determined. Total block counts are output at each directory level. If the **-s** option is used, only a grand total is output.

Sample Command Use

The following command line entry and system responses show how you can determine the number of blocks of data used in a particular file system by specifying the mount point as an argument to the **du** command. Only a grand total of the number of blocks used for the file system is reported (**-s** argument).

```
# du -s /usr<CR>
5556        /usr
#
```

The following command line entry and system responses show how you can determine the number of blocks of data used in a particular directory. A report is made for each file under the specified directory path name. The last line in the report is the total number of blocks used by the specified directory.

```
# du -a /f1/house/bills<CR>
3        /f1/house/bills/water
2        /f1/house/bills/gas
1        /f1/house/bills/electric
1        /f1/house/bills/telephone
8        /f1/house/bills
#
```

/etc/errdump — Error Dump

General

The **errdump** command is used to display the error log maintained in Nonvolatile Random Access Memory (NVRAM). The output of errdump can be directed to a file. Therefore, you can use a floppy disk configured as a file system to keep a historical record of error dumps. Note that previously logged panic messages stored in NVRAM may become garbled following a system reconfiguration.

The following information is output by **errdump**.

nvram status	The sanity of nonvolatile random access memory is reported.
csr	The control and status register.
psw	The program status word (register 11).
r3—r8	General user register 3 through register 8.
oap	Argument pointer (register 10).
opc	Program counter (register 15).
osp	Stack pointer (register 12).
ofp	Frame pointer (register 9).
isp	Interrupt stack pointer (register 14).
pcbp	Process control block pointer (register 13).
fltcr	Memory management unit fault code register.
fltar	Memory management unit address register.

srama and **sramb**
: Pseudo registers of the integral memory management unit.

Panic Log
: The last five panic messages are saved.

Command Format

The format of the **errdump** command consists of only the command name. No options are provided for the command. Register contents are expressed in hexadecimal format. The "0x" prefix is used in register outputs to indicate hexadecimal.

Sample Command Use

The following is a sample **errdump** showing some of the problems with invalid data and garbled output associated with the panic log. The last five panic error messages are saved at the end of the error dump. Panic message number [0] is typical. Panic message number [1] is a garbled output. Message [1] is valid; however, the contents of memory points to the wrong error data. Panic messages number [2] and [3] are invalid data. When invalid data is read, the contents of the memory for the message are dumped in hexadecimal format. Panic message [4] indicates that there is no fifth message.

```
# errdump<CR>
nvram status:   sane

csr:    0x0e48  (led) (floppy) (unassigned) (clock) (uart)

psw:    rsvd CSH_F_D QIE CSH_D OE NZVC TE IPL CM PM R I ISC TM FT
(hex)     0       0   0     0  0    0  0   0  0  0 1 0   5  0  3

r3:     0x00001186
r4:     0xffff9186
r5:     0x40041368
r6:     0x400ca220
r7:     0x400c674c
r8:     0x00000041
oap:    0xc0020220
opc:    0x40004add
osp:    0xc0020248
ofp:    0xc0020248
isp:    0x40080000
pcbp:   0xc0000000

fltar:  0xd0080afc
fltcr:  reqacc  xlevel  ftype
        0xb     0x0     0x3

        srama           sramb
[0]     0x02072800      0x0000011f
[1]     0x02073100      0x00000030
[2]     0x0208b060      0x00000074
[3]     0x0208b400      0x00000015

        Panic log

[0]     Wed Aug  8 15:47:50 1984
        KERNEL BUS TIMEOUT

[1]     Wed Dec 31 18:59:59 1969
         D,LxtV

[2]     (0xffffffff,0xffffffff,0xffffffff,0xffffffff)

[3]     (0x00000000,0xffffffff,0xff1f0000,0x00000000)

[4]     Wed Dec 31 19:00:00 1969


#
```

The following sample **errdump** output is typical for valid NVRAM contents. Note that the last five panic error messages are saved at the end of the error dump. These panic messages are numbered 0 through 4 in the panic log.

```
# errdump<CR>
nvram status:   sane

csr:    0x0e48  (led) (floppy) (unassigned) (clock) (uart)

psw:    rsvd CSH_F_D QIE CSH_D OE NZVC TE IPL CM PM R I ISC TM FT
(hex)      0       0   0     0  0    0  0   0  0  0 1 0   5  0  3

r3:     0x00001186
r4:     0xffff9186
r5:     0x40041368
r6:     0x400ca220
r7:     0x400c674c
r8:     0x00000041
oap:    0xc0020220
opc:    0x40004add
osp:    0xc0020248
ofp:    0xc0020248
isp:    0x40080000
pcbp:   0xc0000000

fltar:  0xd0080afc
fltcr:  reqacc  xlevel  ftype
        0xb     0x0     0x3

        srama           sramb
[0]     0x02072800      0x0000011f
[1]     0x02073100      0x00000030
[2]     0x0208b060      0x00000074
[3]     0x0208b400      0x00000015

        Panic log

[0]     Fri Aug 10 16:04:26 1984
        KERNEL MMU FAULT (13)

[1]     Wed Aug  8 15:47:50 1984
        KERNEL BUS TIMEOUT

[2]     Wed Aug  8 14:17:41 1984
        i/o error in swap

[3]     Fri Jul  6 16:04:26 1984
        SYSTEM PARITY ERROR INTERRUPT (in trap)

[4]     Wed Dec 31 19:00:00 1969

#
```

/etc/ff — List File System Names and Statistics

General

The **ff** command is used to read the information node (i-node) list and directories of the specified special device file of a file system and output certain data. The default output consists of a list of file names (full path names) and their associated information node (i-node) numbers. Options are provided to output selected information.

Command Format

The general format of the **ff** command is as follows.

/etc/ff [*options*] *device*

The various *options* are as follows.

-I	Inhibits the output of the i-node number after each path name.
-l	Outputs a list of all multiple linked files.
-p *prefix*	Outputs path names prefixed with *prefix*. The default prefix is dot (.).
-s	Outputs the file size in bytes after each path name.
-u	Outputs the owner name after each path name.
-a *days*	Outputs item if the i-node has been accessed in the specified number of *days*.
-m *days*	Outputs item if the i-node has been modified in the specified number of *days*.
-c *days*	Outputs item if the i-node has been changed in the specified number of *days*.
-n *file*	Outputs item if the i-node has been modified more recently than *file*.
-i *list*	Outputs only the information for the i-nodes specified in the *list*.

Sample Command Use

The following command line entry and system responses show the use of the **ff** command to output a list of path names and associated i-node numbers for a file system on floppy disk. This example shows the default output of the command (no options specified).

```
# ff /dev/SA/diskette1<CR>
ff: /dev/SA/diskette1: 14 files selected
.        2
./.profile      3
./305-323       47
./305-323/appendix.c    48
./305-323/appendix.b    51
./305-323/appendix.a    52
./305-323/trademarks    53
./305-323/ch1.general   54
./305-323/ch2.install   55
./305-323/ch3.files     56
./305-323/ch4.functions 62
./305-323/ch5.fsck      65
./305-323/toc   66
./305-323/ch6.package   67
#
```

The following command line entry and system responses show the use of the **ff** command to output the path names and associated character sizes (bytes) for a file system maintained on floppy disk.

```
# ff -s -I /dev/SA/diskette1<CR>
ff: /dev/SA/diskette1: 15 files selected
.        144
./.profile       1191
./305-323        208
./305-323/appendix.c     635
./toc    1
./305-323/appendix.b     1698
./305-323/appendix.a     1241
./305-323/trademarks     720
./305-323/ch1.general    2827
./305-323/ch2.install    8175
./305-323/ch3.files      23817
./305-323/ch4.functions  62762
./305-323/ch5.fsck       45723
./305-323/toc    686
./305-323/ch6.package    28458
#
```

/etc/fmtflop — Physically Format Floppy Disks

General

The **fmtflop** command is used to format floppy disks. The floppy disk must be inserted into the floppy disk drive BEFORE the **fmtflop** command is executed.

> ***Caution: Floppy disks formatted in one drive and used in a different drive can be a problem when the head alignment of one or both drives is out of tolerance.***

Command Format

The general format of the **fmtflop** command is as follows.

fmtflop [**-v**] *raw_device*

The **-v** option is used to format and verify that the floppy disk is properly formatted. The *raw_device* must be specified. The integral floppy disk is specified as **/dev/rSA/diskette1**.

Sample Command Use

The following shows the successful use of the **fmtflop** command to format and verify a floppy disk.

Insert the floppy disk into the drive before executing the command.

```
# fmtflop -v /dev/rSA/diskette1<CR>
#
```

Note: The command has no output when successful.

The following shows the unsuccessful use of the **fmtflop** command to format and verify a floppy disk. If this happens on several attempts to format the same floppy disk, the floppy disk medium is bad.

Insert the floppy disk into the drive before executing the command.

```
# fmtflop -v /dev/rSA/diskette1<CR>

NOTICE:
Floppy Access Error: Consult the Error Message Section
of the System Administration Utilities Guide
Error returned is 6,
fmtflop: failure on verifying track number 0, on side 0
```

Reinsert the floppy disk into the drive and try to format the floppy disk.

```
# fmtflop -v /dev/rSA/diskette1<CR>

NOTICE:
Floppy Access Error: Consult the Error Message Section
of the System Administration Utilities Guide
Error returned is 6,
fmtflop: failure on verifying track number 0, on side 0
#
```

Floppy disk medium is defective.

/etc/fmthard — Write Hard Disk VTOC

General

The **fmthard** command is used to write the Volume Table Of Contents (VTOC) to a hard disk. The command is executed from a terminal only when configuring a second hard disk (or other than root disk).

Command Format

The general format of the **fmthard** command is as follows.

fmthard [**-i**] [**-m**] [**-n** *volume*] [**-q**] [**-s** *data*] [**-v**] *device*

The various options and arguments of the **fmthard** command are as follows.

-i	Outputs the default file name used to set up the volume table of contents.
-m	Specifies the creation of file systems on the defined mountable partitions.
-n *volume*	Specifies an eight-character maximum *volume* name for the disk.
-q	Check hard disk version. An exit code of 1 is returned for a hard disk format using a Volume Table Of Contents (VTOC). An exit code of 0 is returned for a non-VTOC hard disk.
-s *data*	Specifies the data file to be used for partition information.
-v	Specifies the verbose mode. More detailed messages are output for the execution of the command when the verbose mode is specified.

device Specifies a raw device partition of the target device. Any partition can be specified.

Partition Terminology

The **–s** option specifies the file to be used for partition information. Each disk partition is defined by a separate line in the *data* file. White space (spaces or tabs) is used to separate each of the items (fields) in a line. The format of each line is as follows.

partition id flag start size

The meanings of these terms are as follows.

partition Specifies the disk partition number. Valid numbers are 0 through 15, inclusive.

id The partition identification (TAG) is a number identifying the use of the partition. Reserved codes are as follows:

NAME	NUMBER
0	UNASSIGNED
1	BOOT
2	ROOT
3	SWAP
4	USR
5	BACKUP

flag The partition identification is a number identifying the partition as unmountable (UNMT) or read only (RONLY). Reserved codes are as follows:

NUMBER	MEANING
00	MOUNTABLE, READ AND WRITE
01	NOT MOUNTABLE
10	MOUNTABLE, READ ONLY

start Identifies the starting sector number of the partition.

size Identifies the number of sectors in the partition.

Sample Command Use

The following shows the command line entries and system responses for writing a volume table of contents to a hard disk. In this example the default second disk partitioning (**/etc/vtoc/hd3dft**) is specified. The **/etc/vtoc/hd3dft** is output to show the default partitioning. After writing this VTOC to the hard disk and making the file system, the installed VTOC is output using the **/etc/prtvtoc** command. This example is intended to only show how the VTOC is written to the hard disk. The task of partitioning a device is not complete until the file systems are created, labeled, and mount table entries created, as required.

```
# cat /etc/vtoc/hd3dft<CR>
*
* CDC Wren 32M hard disk VTOC defaults
*
* Dimensions:
*   512 bytes/sector
*    18 sectors/track
*     5 tracks/cylinder
*   697 cylinders
*   695 user accessible cylinders
*
* Flags:
*       01      Not mountable
*       10      Read-only
*
* Partition     ID      Flag    Start Sector    Size in Sectors
   00           0       01      0               0
   01           0       01      0               0
   02           0       01      0               0
   03           0       01      0               0
   04           0       01      0               0
   05           0       01      0               0
   06           0       01      0               62550
   07           0       01      0               90
   08           0       00      90              62460
   09           0       01      0               0
   10           0       01      0               0
   11           0       01      0               0
   12           0       01      0               0
   13           0       01      0               0
   14           0       01      0               0
   15           0       01      0               0
# fmthard -i -m /dev/rdsk/c1d1s0<CR>
fmthard:  New volume table of contents now in place.
+(fmthard) mkfs /dev/rdsk/c1d1s8 62460 9 90
Mkfs: /dev/rdsk/c1d1s8?
(DEL if wrong)
bytes per logical block = 1024
total logical blocks = 31230
total inodes = 7792
gap (physical blocks) = 9
cylinder size (physical blocks) = 90
mkfs: Available blocks = 30740
# prtvtoc -s /dev/rdsk/c1d1s0<CR>
*   Partition  Tag  Flags  First Sector  Sector Count   Mount Directory
        6      0    01          0          62550
        7      0    01          0             90
        8      0    00         90          62460

#
```

/etc/fsck — File System Consistency Check and Interactive Repair

General

The **fsck** command is used to check and repair inconsistencies in a file system. Except for the **root** file system, a file system must be unmounted before it is checked. Refer to Chapter 4, "FILE SYSTEM CHECKING AND REPAIR," for more information.

Command Format

The following is the general format of the **fsck** command.

fsck [**-y**] [**-n**] [**-b**] [**-s***X*] [**-S***X*] [**-t** *file*] [**-q**] [**-D**] [**-f**] [*fsdevice*]

The various options of the **fsck** command are as follows.

-b Reboots the system if the file system being checked is the root (/) file system and modifications have been made by **fsck**. If minor damage is found, the file system is remounted.

-D Checks directories for bad blocks. This option is used to check file systems for damage after a system crash.

-f Specifies that a fast file system check be done. Only Phase 1 (check blocks and sizes) and Phase 5 (check file system free list) are executed for a fast check. Phase 6 (reconstruct free list) is run only if necessary.

-n Specifies a "no" response for all questions.

-q Specifies a "quiet" file system check. Output messages from the process are suppressed.

-s*X* Specifies an unconditional reconstruction of the free list. Following the reconstruction of the free list, the system should be rebooted so that the in-core copy of the superblock is updated. See the **-b** option. The *X* argument specifies the number of blocks-per-cylinder and the number of blocks to skip (rotational gap). The default values are the values specified when the file system was created. The format for various configurations of disk drives is as follows.

DEVICE	-s *blocks/cylinder:gap*
72M FUJITSU Hard Disk	**-s198:9**
72M MICROPOLIS Hard Disk	**-s144:9**
72M PRIAM Hard Disk	**-s198:9**
72M WREN* II Hard Disk	**-s162:9**
30M Hard Disk	**-s90:9**
23M Cartridge Tape	**-s31:1**
10M Hard Disk	**-s72:9**
Floppy Disk	**-s18:1**

-S*X* Specifies a conditional reconstruction of the free list. The format of the *X* argument is the same as described for the **-s** option.

-t *file* Specifies a scratch file for use when the file system check requires additional memory. If this option is not specified, the process asks for a file name when more memory is needed.

-y Specifies a "yes" response for all questions.

* Trademark of Control Data Corporation

Sample Command Use

The following command line entry and system responses show how to check the **root** file system. Refer to Chapter 3, "ADMINISTRATIVE TASKS," and Chapter 4, "FILE SYSTEM CHECKING AND REPAIR," for more information on the use of the **fsck** command. Remember that the **root** file system must be mounted to be checked; all other file systems must be unmounted.

```
# fsck /dev/dsk/c1d0s0<CR>

  /dev/dsk/c1d0s0
  File System: root Volume: root

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  289 files 6522 blocks 3220 free
#
```

/etc/fsdb — File System Debugger

The **fsdb** command is used to debug file system problems. To be able to use this command, considerable experience is required in the areas of operating system internals and file system mechanics. Contact your service representative regarding training course availability.

/etc/fsstat — Report File System Status

General

The **fsstat** command reports the status of a file system. This command is used to determine if a file system check is needed before mounting the file system. If successful, the **fsstat** command returns an exit code of 0. An exit code of 1 indicates that the file system should be checked. An exit code of 2 indicates that the file system is mounted. An exit code of 3 is used for "other" failures. The most common use of the **fsstat** command is, therefore, in shell scripts where based on the exit code status other commands are executed. (See **/etc/mountall**.)

Command Format

The general format of the **fsstat** command is as follows.

fsstat *filesystem*

The *filesystem* to be checked is expressed as the device file path name for the particular disk partition.

Sample Command Use

The following command line entries and system responses show the typical use of the **fsstat** command. The **echo $?** command is used to output the exit code of the previous **fsstat** command.

```
# fsstat /dev/dsk/c1d0s0<CR>
fsstat: root file system okay
# echo $?<CR>
0
# fsstat /dev/dsk/c1d0s2<CR>
fsstat: /dev/dsk/c1d0s2 mounted
# echo $?<CR>
2
# fsstat /dev/dsk/c1d0s1<CR>
fsstat: /dev/dsk/c1d0s1 not a valid file system
# echo $?<CR>
3
#
```

/etc/fuser — Identify Processes Using a File or File Structure

General

The **fuser** command is used to list the process identification numbers of the processes using specified files. Options are provided to kill the processes. Note that only a super-user (root) can terminate processes that belong to another user. The output of the command reports the process identification numbers associated with the user identification name. The process identification numbers are followed by a lowercase letter **c** (for current directory), **p** (for parent directory), or **r** (for **root** directory) to identify how the process is using the specified file(s).

Command Format

The two general formats of the **fuser** command are as follows. The second format is used to redefine options for another *files* argument.

/etc/fuser [**-ku**] *files*

/etc/fuser [**-ku**] *files* - [**-ku**] *files*

The **-k** option causes the **fuser** command to send a kill signal to each identified process. The **-u** option causes the **fuser** command to report the login name in parentheses after each process identification number. The options can be redefined between *files* arguments by first canceling the existing options with a lone hyphen (-) and then specifying the new options. The *files* argument specifies the file or files to be examined for process activity. The *files* argument can be an ordinary file or a device type file.

Sample Command Use

The following command line entries and system responses show how to identify all processes that are active in the **root** (/dev/dsk/c1d0s0) and **usr** (/dev/dsk/c1d0s2) file systems. The **ps** command is used to show the correlation between the output of the **fuser** command and the active system processes. Note that the output of the **fuser** command is one line for each file system. Therefore, the output format will vary as a function of the terminal line wrap.

```
# fuser -u /dev/dsk/c1d0s0 /dev/dsk/c1d0s2<CR>
/dev/dsk/c1d0s0:        0c(root)        0c(root)    2832(cec)    2897(root)
43c(root)    2914(root)
/dev/dsk/c1d0s2:     2832c(cec)    2897c(root)      43(root)    2914c(root)

# ps -ef<CR>
    UID   PID  PPID  C    STIME  TTY      TIME COMMAND
   root     0     0 40  Mar  9  ?        8:51 swapper
   root     1     0  0  Mar  9  ?        0:40 /etc/init
   root     0     0  0  Mar  9  ?        0:16 swapper
   root  2892     1  0 10:04:25 console  0:04 -sh
    cec  2832     1  0 09:46:11 contty  0:06 -sh
   root  2897  2892  0 10:04:46 console  0:08 vi fuser
   root    43     1  0  Mar  9  console  3:10 /etc/cron
    cec  2911  2832  0 10:14:06 contty  0:04 vi distinct
   root  2915  2897  0                 0:00 <defunct>
   root  2916  2897  0 10:16:59 console  0:00 sh -c ps -ef
   root  2917  2916 47 10:17:01 console  0:06 ps -ef
#
```

/etc/getmajor — Output Slot/Major Numbers of Hardware Devices

General

The **getmajor** command outputs all slot/major numbers in the equipped device table for the requested device or board code. The primary use of this command is in the INSTALL and UNINSTALL shell scripts used by the **sysadm installpkg** and **sysadm removepkg** commands.

Command Format

The general format of the **getmajor** command is as follows.

getmajor *name* | *bdcode*

The *name* argument specifies a name of the device and can be entered as either uppercase or lowercase. The *bdcode* specifies the board code. The board code is assigned as a function of hardware design.

Sample Command Use

The following command line entries and system responses show the use of the **getmajor** command to return the slot/major number for the system board (SBD) and expanded input/output board (PORTS). The following system responses indicate that slot 0 is the system board and that two ports cards are equipped in slots 1 and 2.

```
# getmajor SBD<CR>
0
# getmajor PORTS<CR>
1 2
#
```

/etc/getty — Set Terminal Type, Modes, Speed, and Line Discipline

General

The **getty** command is used as part of the login process to read the login name and to execute the **login** command with the login name as an argument. The **getty** command is the second command in a series of four commands (processes) that are used to connect a user with the system. The sequence of commands in the login process is **init**, **getty**, **login**, and **shell**. You DO NOT directly execute the **getty** command. Therefore, command line entry and system response examples are not provided in this description. The **login** command is used at the start of a terminal session to identify yourself to the system. The command is automatically executed by **getty** when you synchronize your terminal with the system.

Note that bidirectional ports used for basic networking require a different form of this process. See the **/usr/lib/uucp/uugetty** manual pages for additional information.

Command Format

Two general formats of the **getty** command are as follows.

/etc/getty [**-h**] [**-t** *timeout*] *line* [*speed* [*type* [*linedisc*]]]

/etc/getty -c *file*

When **getty** is executed WITHOUT the **-h** option, the attached line is hung up by setting the speed to zero before setting the speed to the default or specified data transmission rate. Specifying the **-h** option causes the command to execute without first hanging up the line. The **-t** *timeout* argument causes the **getty** command to exit if the line is opened for the number of seconds specified by the *timeout* argument and no data is received.

The *line* argument specifies the name of the device file to which the command is attached. For example, **/dev/tty11**.

The *speed* argument tells the command the rate at which data is to be sent over the line. The default value is 300 baud.

The *type* argument identifies the type of terminal device that is connected to the line. Acceptable options are defined in the **getty** manual pages.

The *linedisc* argument defines the line discipline used for communicating with the terminal device. The only available discipline is called **LDISC0**, which is also defined as the default option.

The **-c** option causes the **getty** command to execute in the check mode. The check mode is used to verify the contents of the **/etc/gettydefs** data. After making changes to this information, it is strongly recommended that the **getty** command be used to verify the information. Refer to the manual pages describing the **gettydefs** file for additional information. The general format of the **gettydefs** information is described in the manual pages. A typical 3B2 Computer **gettydefs** file is shown in Figure 6-5. The field separator in a **gettydefs** file is the pound symbol (#). From left to right the fields are label, initial-flags, final-flags, login-prompt, and next-label. Each of these fields is described in the **gettydefs** manual pages.

The *file* argument is used with the check option (**-c**) to specify the name of the file that is to be scanned. The file contains information used by the **getty** command to set up the speed and terminal setting for a line. Also included is information defining the login prompt. Changes to the **/etc/gettydefs** file can be made in a separate file and then the changes verified. After successful verification, the temporary file can be copied or moved to the **/etc/gettydefs** file.

```
19200# B19200 HUPCL # B19200 SANE IXANY TAB3 HUPCL #login: #9600

9600# B9600 HUPCL # B9600 SANE IXANY TAB3 HUPCL #login: #4800

4800# B4800 HUPCL # B4800 SANE IXANY TAB3 HUPCL #login: #2400

2400# B2400 HUPCL # B2400 SANE IXANY TAB3 HUPCL #login: #1200

1200# B1200 HUPCL # B1200 SANE IXANY TAB3 HUPCL #login: #300

300# B300 HUPCL # B300 SANE IXANY TAB3 HUPCL #login: #19200

console# B9600 HUPCL OPOST ONLCR # B9600 SANE IXANY TAB3 #Console Login: #console1

console1# B1200 HUPCL OPOST ONLCR # B1200 SANE IXANY TAB3 #Console Login: #console2

console2# B300 HUPCL OPOST ONLCR # B300 SANE IXANY TAB3 #Console Login: #console3

console3# B2400 HUPCL OPOST ONLCR # B2400 SANE IXANY TAB3 #Console Login: #console4

console4# B4800 HUPCL OPOST ONLCR # B4800 SANE IXANY TAB3 #Console Login: #console5

console5# B19200 HUPCL OPOST ONLCR # B19200 SANE IXANY TAB3 #Console Login: #console

contty# B9600 HUPCL OPOST ONLCR # B9600 SANE IXANY TAB3 #login: #contty1

contty1# B1200 HUPCL OPOST ONLCR # B1200 SANE IXANY TAB3 #login: #contty2

contty2# B300 HUPCL OPOST ONLCR # B300 SANE IXANY TAB3 #login: #contty3

contty3# B2400 HUPCL OPOST ONLCR # B2400 SANE IXANY TAB3 #login: #contty4

contty4# B4800 HUPCL OPOST ONLCR # B4800 SANE IXANY TAB3 #login: #contty5

contty5# B19200 HUPCL OPOST ONLCR # B19200 SANE IXANY TAB3 #login: #contty

pty# B9600 HUPCL OPOST ONLCR # B9600 SANE IXANY TAB3 #PC login: #pty

4800H# B4800 # B4800 SANE IXANY TAB3 HUPCL #login: #9600H

9600H# B9600 # B9600 SANE IXANY TAB3 HUPCL #login: #19200H

19200H# B19200 # B19200 SANE IXANY TAB3 HUPCL #login: #2400H

2400H# B2400 # B2400 SANE IXANY TAB3 HUPCL #login: #1200H

1200H# B1200 # B1200 SANE IXANY TAB3 HUPCL #login: #300H

300H# B300 # B300 SANE IXANY TAB3 HUPCL #login: #4800H
```

Figure 6-5. Typical gettydefs File

/etc/grpck — Group Password Check

General

The **grpck** command is used to check the **/etc/group** file for inconsistencies.

Command Format

The general format of the **grpck** command is as follows.

/etc/grpck [*file*]

The *file* argument is used to specify a different file than the default **/etc/group** file.

Sample Command Use

The following command line entry and system responses show the typical output of the **grpck** command.

```
# grpck<CR>


other::1:
        Null login name
#
```

/etc/hdeadd — Add/Delete Reports To/From Hard Disk Error Log

General

The **hdeadd** command is used to:

- Print the list of equipped disks
- Manually add or delete disk error reports to/from the hard disk error log.

Command Format

The general format of the **hdeadd** command is as follows.

hdeadd -a [*options*]
hdeadd -d [*options*]
hdeadd -e [[**-D**] *major minor*]
hdeadd -f *filename*
hdeadd -r [[**-D**] *major minor filename*]
hdeadd -s [[**-D**] *major minor filename*]

The various arguments and options of the **hdeadd** command are as follows.

-a	Adds new bad blocks to the hard disk error log.
-b *blockno*	This is the normal form of an argument specifying the physical disk block number. The operating system uses this format to report bad blocks.
-B *cyl trk sec*	This is an alternate argument specifying the physical disk block number in terms of the physical cylinder number, track number, and sector number.

-d Deletes reports from the hard disk error log.

-D *major minor*
Specifies the *major* device number and the *minor* device number of the disk.

-e Outputs a list of major and minor device numbers for equipped disks.

-f Reads hard disk error commands from the specified file name. Each line of text in the file is one command, starting with **-a** or **-d** argument.

-F This is an optional argument specifying the starting time for the interval being purged. Default is zero. The arguments to this option are the same as the **date** command. Time is specified by two-digit numbers for the month, day, hour, minute, and year. This option is only used for testing the bad block handling feature.

-r Restores the bad block map from the file specified by the *filename* argument.

-s Saves the bad block map in the file specified by the *filename* argument.

-T This is an optional argument specifying the "to" time for the interval being purged. Default is the current time. The arguments to this option are the same as the **date** command. Time is specified by two-digit numbers for the month, day, hour, minute, and year. This option is only used for testing the bad block handling feature.

Sample Command Use

The following shows the command line entries and system responses for the addition of a new bad block to the hard disk error log. In this example, block 7203 is added to the log.

```
# hdeadd -a -D 17 0 -b 7203<CR>
hdeadd: logging following error report:
        disk maj=17 min=0
        blkaddr=7203, timestamp=Sun Aug 12 18:00:51 1984
        readtype=1, severity=2, badrtcnt=0, bitwidth=0

WARNING: unreadable CRC hard disk error: maj/min = 17/0

         block = 7203
# Disk Error Daemon: successfully logged error for block 7203 on disk maj=17 min=0
```

Note: The prompt (#) is in response to the hdeadd command and may be output at any place in the WARNING message. The WARNING message is the result of the bad block being added to the log.

The following command line entry and system response shows the use of the **hdeadd** command to output a list of major and minor device numbers for the equipped disks.

```
# hdeadd -e -D 17 0<CR>
maj=17 min=0 is an equipped disk
#
```

The following command line entry and system response shows the use of the **hdeadd** command to save the bad block map in a file.

```
# hdeadd -s -D 17 0 /tmp/hde170<CR>
save successful
#
```

/etc/hdefix — Report/Change Bad Block Mapping

General

The **hdefix** command is used to report or change hard disk bad block mapping. You must be logged in as **root** to use this command. The machine should be shut down to run-level 1 to change the hard disk bad block mapping.

Command Format

The general formats of the **hdefix** command are as follows.

hdefix -p [[**-D**] *major minor*]
hdefix -a [*major minor* [*blocknospec ...*]]
hdefix -F [**-D**] *major minor* [*blocknospec ...*]]
hdefix -r [[**-D**] *major minor filename*]
hdefix -p [[**-D**] *major minor filename*]

The various options of the **hdefix** command are as follows.

-a Adds new bad blocks to the defect map.

-D Specifies the *major* device number and the *minor* device number of the disk.

-F Forces bad blocks to be removed from the defect map. This option is only used for testing the bad block handling feature.

-p Outputs a report of all mapped blocks.

-r Restores the bad block map from the file specified by the *filename* argument.

-s Saves the bad block map in the file specified by the *filename* argument.

The *blocknospec* has the following forms.

-b *blockno* This is the normal form of an argument specifying the physical disk block number. This is the format the operating system uses to report bad blocks.

-B *cly trk sec* This is an alternate argument specifying the physical disk block number in terms of the physical cylinder number, track number, and sector number.

Sample Command Use

The following shows the command line entries and system responses for reporting the currently mapped bad blocks.

```
# hdefix -p -D 17 0<CR>

Basic physical description of disk maj=17 min=0:
sector size=512, sectors per track=18 tracks per cylinder=5
number of cylinders=697, block number range: 0 thru 62729
defect map has 64 slots
its active slots are:
slot#   from blk#     to blk#
    0        2910           3
    1        7212           4
    2        7297           5
    3        7298           6
    4       20971           7
    5       27286           8
    6       27287           9
    7       29330          10
    8       42520          11
    9       62612          12
its surrogate region description is:
        start blk#: 3
        size (in blks): 86
        next blk#: 13
physical description is at blk# 0
error log is at blk# 89
logical start is at blk# 90
#
```

The following shows the command line entry and system responses for adding a block to the defect map. The system is already in the single-user mode at the beginning of the example.

```
# hdefix -a -D 17 0 -b 7203<CR>
hdefix: once fixing bad blocks is started, signals are ignored
starting
hdefix: processing 1 bad block on disk maj=17 min=0
processing block 7203
    new bad block (e.g., not already mapped)
    block in partitioned portion of disk
    block in partition 0
        it is block 993 in the partition
        partition has a file system containing the bad block
        marking file system dirtied by bad block handling
        the bad block was a file block
    block in partition 6
        it is block 7113 in the partition
    assigning new surrogate image block for it
finished with this disk
hdefix: causing system reboot
system being rebooted


SELF-CHECK

DIAGNOSTICS PASSED
```

Continued

```
Continued from the previous screen

UNIX System V Release 2.0.3 3B2 Version 2
unix
Copyright (c) 1984 AT&T.
All Rights Reserved

        fsstat: root file system needs checking
        The root file system (/dev/dsk/c1d0s0) is being checked automatically.

  /dev/dsk/c1d0s0
  File System: root Volume: 2032

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  FILE SYSTEM STATE SET TO OKAY
  526 files 10458 blocks 1854 free

  *** ROOT FILE SYSTEM WAS MODIFIED ***
  *** ROOT REMOUNTED ***

The system is coming up.  Please wait.
AT&T 3B2 SYSTEM CONFIGURATION:

Memory size: 2 Megabytes
System Peripherals:

        Device Name         Subdevices

        SBD
                          Floppy Disk
                          30 Megabyte Disk
                          30 Megabyte Disk
        PORTS
        PORTS
        CTC
The system is ready.

Console Login:
```

/etc/hdelogger — Hard Disk Error Logger

General

The **hdelogger** command when executed from a terminal by **root**, reports the hard disk errors that have been logged. When executed by the **init** process, **hdelogger** runs as a daemon that adds new errors to the hard disk error log as reported by the hard disk drivers.

Command Format

The general format of the **hdelogger** command is as follows.

hdelogger [**-s**][**-f**][**-D** *major minor*]

When run as a normal command to report the status of hard disk errors, the following options apply. A summary report for each equipped hard disk is the default output when no options are specified.

-s	Specifies the generation of a summary report.
-f	Specifies the generation of a full report.
-D	Restricts the report to the specified hard disk.

Sample Command Use

The following command line entry and system responses show the generation of a summary report for a specific hard disk (major=17, minor=0).

```
# hdelogger -s -D 17 0<CR>
Disk maj=17 min=0: 1 errors logged
#
```

The following command line entry and system responses show the generation of a full report for a specific hard disk (major=17, minor=0).

```
# hdelogger -f -D 17 0<CR>

Disk Error Log: Full Report for maj=17 min=0
        log created:  Fri Aug  3 22:42:51 1984
        last changed: Sun Sep 20 07:23:35 1984
        entry count: 1
    phys blkno  cnt     first occurrence       last occurrence
 0:       7203    1   Sep 20 07:23:31 1984   Sep 20 07:23:31 1984
TOTAL: 1 errors logged
#
```

/usr/bin/id — Output User and Group Identification

General

The **id** command displays your user and group identification numbers and names. The command is generally used in shell scripts to determine who is executing the program.

Command Format

The format of the **id** command consists of only the name of the command. The **id** command has no arguments.

Sample Command Use

The following is an example use of the **id** command. The response shown indicates that **root** executed the **id** command.

```
# id<CR>
uid=0(root) gid=1(other)
#
```

The **uid** is the user identification number, and the **gid** is the group identification number. In both cases, the format of the identification is the number followed by the name in parentheses.

/etc/init — Process Control Initialization

General

The **init** command is used to create processes from entries in the **/etc/inittab** file. Refer to the **inittab**(4) manual pages for information on the structure of this file.

Command Format

The general format of the **init** command is as follows.

init [**0** | **1** | **2** | **3** | **4** | **5** | **6** | **s** | **S** | **q** | **Q** | **a** | **b** | **c**]

The various options specify the system run-level that is to be initialized. The 3B2 Computer run levels are summarized in Appendix B.

Sample Command Use

The following command line entries and system responses show a typical use of the **init** command to go from the single-user mode to the multi-user mode of operation. At the beginning of the example the system is in the single-user mode of operation.

```
# init 2<CR>

INIT: New run level: 2
The system is coming up. Please wait.
AT&T 3B2 SYSTEM CONFIGURATION:

Memory size: 2 Megabytes
System Peripherals:

        Device Name         Subdevices

        SBD
                          Floppy Disk
                          30 Megabyte Disk
                          30 Megabyte Disk
        PORTS
        PORTS
        CTC
The system is ready.

Console Login:
```

/etc/killall — Kill All Active Processes

General

The **killall** command is used to stop all active processes with open files so that mounted file systems can be unmounted. The **/etc/shutdown** program uses the **killall** command to stop all active processes not related to the shutdown process.

Command Format

The general format of the **killall** command is as follows.

killall [*signal*]

The optional *signal* argument specifies the signal that is to be sent to the active processes. The default signal is the kill signal (**9**). The terminate signal (**15**) is another signal that can be used to stop a process.

Sample Command Use

The following command line entries and system responses show how to use the **killall** command to unbusy a file system so that the file system can be unmounted. The **sync** command is used to write the system buffer to disk before unmounting the file system.

```
# mount<CR>
/ on /dev/dsk/c1d0s0 read/write on Tue Aug 14 15:29:50 1984
/usr on /dev/dsk/c1d0s2 read/write on Tue Aug 14 15:33:25 1984
# umount /dev/dsk/c1d0s2<CR>
umount: /dev/dsk/c1d0s2 busy
# killall<CR>
# sync;sync;sync<CR>
# umount /dev/dsk/c1d0s2<CR>
# mount<CR>
/ on /dev/dsk/c1d0s0 read/write on Tue Aug 14 15:29:50 1984
#
```

/etc/labelit — Provide Initial Labels for Unmounted File System

General

The **labelit** command is used to read or write file system labels. The file system can be mounted or unmounted.

Command Format

The general format of the **labelit** command is as follows.

/etc/labelit *file* [*fsname volume*] [**-n**]

The command arguments and options are as follows.

file — The *file* argument specifies the full path name of the special character or block device file that represents the physical disk section or magnetic tape.

fsname — The *fsname* specifies the mounted name of the file system. For example, **root** and **usr** are names of mounted file systems. These names correspond to the device files **/dev/dsk/c1d0s0** and **/dev/dsk/c1d0s2**, respectively. The file system name is limited to six or fewer characters.

volume — The *volume* name is used to identify the version or issue. The name is limited to six or fewer characters.

-n — The **-n** option is used for initial labeling of NEW magnetic tapes. The previous contents of the tape is destroyed.

Sample Command Use

The following command line entries and system responses show how to use the **labelit** command to read the existing label of a file system.

```
# labelit /dev/dsk/c1d0s0<CR>
Current fsname: root, Current volname: root, Blocks: 9900, Inodes: 1232
FS Units: 1Kb, Date last mounted: Mon Mar 12 11:29:40 1984
# labelit /dev/dsk/c1d0s2<CR>
Current fsname: usr, Current volname: usr, Blocks: 44730, Inodes: 5584
FS Units: 1Kb, Date last mounted: Mon Mar 12 11:34:40 1984
```

The following command line entries and system responses show how to use the **labelit** command to write the file system and volume name of a file system label. The **/etc/mkfs** command is used to make a file system on a floppy disk. The **labelit** command is then used to write the file system name and the volume name.

```
# mkfs /dev/SA/diskette1 1422:192 1 18<CR>
Mkfs: /dev/SA/diskette1?
(DEL if wrong)
bytes per logical block = 1024
total logical blocks = 711
total inodes = 192
gap (physical blocks) = 1
cylinder size (physical blocks) = 18
mkfs: Available blocks = 696
# labelit /dev/SA/diskette1 rar 2032<CR>
Current fsname: , Current volname: , Blocks: 1422, Inodes: 192
FS Units: 1Kb, Date last mounted: Wed Aug 15 05:51:27 1984
NEW fsname = rar, NEW volname = 2032 -- DEL if wrong!!

#
```

/etc/ldsysdump — Load Multiple Floppy System Dumps

General

The **ldsysdump** command is used to combine multiple system dump floppy disks into a single file on the hard disk. The resulting file can be examined by using **/etc/crash**.

Command Format

The general format of the **ldsysdump** command is as follows.

ldsysdump *file*

The *file* argument specifies the name of the hard disk file used for the output of the command. This argument can be expressed as a complete path name. The **ulimit** may need to be increased to a size (in blocks) that accommodates the summation of the system dump floppy disks. The **ulimit** command with no arguments outputs the current value of the variable. The **ulimit 5000** command sets the limit to 5000 blocks.

Sample Command Use

The following command line entries and system responses are typical for the use of the **ldsysdump** command.

```
# ldsysdump /tmp/dump<CR>

Insert first sysdump floppy.
Enter 'c' to continue, 'q' to quit: c<CR>

Loading sysdump
.....................................................................
.....................................

Insert next sysdump floppy.
Enter 'c' to continue, 'q' to quit: c<CR>

Loading more sysdump
.....................................................................
.....................................

Insert next sysdump floppy.
Enter 'c' to continue, 'q' to quit: c<CR>

Loading more sysdump
.....................................................................
.....................................

Insert next sysdump floppy.
Enter 'c' to continue, 'q' to quit: q<CR>

3 Sysdump files coalesced, 4096 (512 byte) blocks

#
```

/etc/link — Execute Link System Call

General

The **link** command is used to create a file name that points to another file. Linked files and directories can be removed by the **/etc/unlink** command; however, it is recommended that the **/bin/rm** and **/bin/rmdir** commands be used instead of the **unlink** command. The difference between **/bin/ln** and **/etc/link** commands is that **link** does exactly what it is told to do. The **/bin/ln** command can be executed by anyone, file permissions permitting. The **link** command can be executed only by **root**.

Files (or directories) that are linked means that one file has multiple names. Files (or directories) cannot be linked across file systems. Removing a file name that is linked to another file name has no effect on the other file name or its contents.

Command Format

The general format of the **link** command is as follows.

/etc/link *oldfile newfile*

The *newfile* argument specifies the complete path name of the file that is to point to an existing file (*oldfile*).

Sample Command Use

The following command line entries and system responses show how the **link** command is used to create another file name for an existing file. The **ls -l** command is used to show the result of the **link** command.

```
# ls -l examples<CR>
total 1
-rw-r--r--   1 root     other          53 Sep  6 15:49 file1
# ls -ld examples<CR>
drwxr-xr-x   2 root     other          64 Sep  6 15:48 examples
# link examples/file1 examples/file2<CR>
# ls -l examples<CR>
total 2
-rw-r--r--   2 root     other          53 Sep  6 15:49 file1
-rw-r--r--   2 root     other          53 Sep  6 15:49 file2
# ls -ld examples<CR>
drwxr-xr-x   2 root     other          64 Sep  6 15:52 examples
# date >> examples/file2<CR>
# ls -l examples<CR>
total 2
-rw-r--r--   2 root     other          82 Sep  6 16:32 file1
-rw-r--r--   2 root     other          82 Sep  6 16:32 file2
#
```

/etc/mkboot — Convert a.out File to Boot Image

***Caution: Execute a separate** mkboot **command for each driver or module. DO NOT execute** mkboot **with multiple modules (drivers) specified. The internal buffers used by the command are not cleared between modules when multiple modules are specified. Executing a separate** mkboot **command for each driver produces repeatable results for the purpose of binary comparisons.*

General

The **mkboot** command is used to create an object file in a format compatible with the self-configuration boot program. Each object file must have a corresponding **master** file. The **master** files are in the **/etc/master.d** directory. The new bootable file is written to the **/boot** directory and is given the same name, in uppercase letters and without the ".o" suffix, as the object file. To conserve space on the system, the ".o" files are not kept. Therefore, the uppercase name can always be used instead of the "name.o" arguments when the ".o" file does not exist.

Command Format

The general format of the **mkboot** command is as follows.

mkboot [**-m** *master*] [**-d** *directory*] **-k** *kernel.o* | *driver.o*

The various arguments to the **mkboot** command are as follows.

-m *master*	This optional argument specifies the directory containing the master files to be used for each object file. The default master directory is **/etc/master.d**.
-d *directory*	The destination directory to be used for the new object file is specified by the *directory* argument. The default output directory is **/boot**.

-k *kernel.o* — The name of the object file for the UNIX Operating System is specified by the *kernel.o* argument. The default object file name is the *master* file name. This name is KERNEL.

driver.o — The name of the object file for a module or driver is specified by the *driver.o* argument.

Sample Command Use

The use of the **mkboot** command is the second step in changing tunable system parameters and generating a new **/unix** configuration. The applicable **/etc/master.d** files are edited to modify the tunable parameters. Then, the current directory is changed to **/boot**, and the **mkboot** command executed for each of the applicable files. (See Caution.) The new operating system configuration is generated by taking the system to the firmware mode and booting **/etc/system**.

The following command line entries and system responses show the use of the **mkboot** command to generate a new bootable KERNEL. In this example, the **/etc/master.d/kernel** file has been edited to modify certain parameters. Refer to Chapter 3, "ADMINISTRATIVE TASKS," for information on system reconfiguration.

```
# cd /boot<CR>
# ls -l KERNEL<CR>
-rwxr-xr-x   1 root     other     152286 Aug 22 16:50 KERNEL
# mkboot -k KERNEL<CR>
# ls -l KERNEL<CR>
-rwxr-xr-x   1 root     other     152286 Sep  6 11:23 KERNEL
#
```

/etc/mkfs — Construct a File System

General

The **mkfs** command is used to construct (create) a file system on a specified device partition. The size of the file system depends on the device partition that is specified. Refer to Appendix A, "DEVICE DEFAULT PARTITIONING," for the number of blocks associated with the various configurations of hard disk, floppy disk, and cartridge tape devices used as part of the 3B2 Computer.

Each file system should contain a **lost+found** directory in the root directory of the file system. The **lost+found** directory is used by **/etc/fsck**(1M) to reconnect unreferenced directories and files. Unreferenced information nodes (i-nodes) are relinked to available i-nodes in the **lost+found** directory. Therefore, each **lost+found** directory must be populated with enough i-nodes to support the size of the file system in the event of file system corruption. Refer to the "POPULATING A LOST+FOUND DIRECTORY" task description in chapter 3 for additional information.

Command Format

Two forms of the **mkfs** command are provided. The following form of the command is used to construct a file system on the specified device per the directions provided by the remainder of the command line.

/etc/mkfs *device blocks*[*:i-nodes*] [*gap blocks/cyl*]

The following form of the **mkfs** command is used to construct a file system on the specified devices per instructions provided in a prototype file.

/etc/mkfs *device proto* [*gap blocks/cyl*]

The various arguments of the **mkfs** command are as follows. Refer to Appendix A, "DEVICE DEFAULT PARTITIONING," for additional information on file system sizes, device files, and starting cylinders for the various types of devices used as part of the 3B2 Computer.

device The *device* argument specifies the name of the special device file. For example, **/dev/SA/diskette1** and **/dev/dsk/c1d0s2** are two special block device files for the floppy disk and hard disk, respectively.

blocks The *blocks* argument specifies the number of physical disk blocks (512 bytes per block) allocated for the file system.

i-nodes The *i-nodes* argument specifies the number of information nodes to allocate. The information nodes are allocated in groups of 16 (modulo 16). If a number is specified that is not a multiple of 16, the next lower multiple of 16 is allocated. If the **i-node** argument is omitted, the default is the number of logical blocks (1024 byte blocks) divided by eight. The maximum number of information nodes that can be configured for a file system is 65488.

gap The *gap* argument specifies the rotation gap for the device. The default *gap* is 9 for the hard disk, 1 for the floppy disk, and 1 for the cartridge tape.

block/cyl The *blocks/cyl* argument specifies the number of blocks per cylinder for the disk device. The blocks per cylinder for the various configurations of disk drives are as follows.

DEVICE	BLOCKS PER CYLINDER
72M FUJITSU Hard Disk	198
72M MICROPOLIS Hard Disk	144
72M PRIAM Hard Disk	198
72M WREN* II Hard Disk	162
30M Hard Disk	90
23M Cartridge Tape	31
10M Hard Disk	72
Floppy Disk	18

proto The *proto* argument specifies the file (complete path name) that contains instructions for the creation of a file system. Refer to the **mkfs**(1) manual page for a complete description of the characteristics of this file.

* Trademark of Control Data Corporation

Sample Command Use

The following command line entries and system responses show how to make a file system on the floppy disk. The maximum number of blocks (512 byte blocks) for a formatted floppy disk is 1422 blocks. The rotational gap is 1; the number of blocks per cylinder is 18. The typical number of information nodes (i-nodes) for a floppy disk is 192.

```
# mkfs /dev/SA/diskette1 1422:192 1 18<CR>
Mkfs: /dev/SA/diskette1?
(DEL if wrong)
bytes per logical block = 1024
total logical blocks = 711
total inodes = 192
gap (physical blocks) = 1
cylinder size (physical blocks) = 18
mkfs: Available blocks = 696
# labelit /dev/rSA/diskette1 rar 2032<CR>
Current fsname: , Current volname: , Blocks: 1422, Inodes: 192
FS Units: 1Kb, Date last mounted: Mon Mar 11 06:10:46 1985
NEW fsname = rar, NEW volname = 2032 -- DEL if wrong!!
#
```

The following command line entries and system responses show how to make a file system on the floppy disk. This example shows that the number of i-nodes are rounded down to a modulo 16 number of 192 from a specified value of 200.

```
# mkfs /dev/SA/diskettel 1422:200 1 18<CR>
Mkfs: /dev/SA/diskettel?
(DEL if wrong)
bytes per logical block = 1024
total logical blocks = 711
total inodes = 192
gap (physical blocks) = 1
cylinder size (physical blocks) = 18
mkfs: Available blocks = 696
# labelit /dev/rSA/diskettel rar 2032<CR>
Current fsname: , Current volname: , Blocks: 1422, Inodes: 192
FS Units: 1Kb, Date last mounted: Mon Mar 11 06:15:53 1985
NEW fsname = rar, NEW volname = 2032 -- DEL if wrong!!
#
```

/etc/mknod — Build a Special File

General

The **mknod** command is used to construct a directory and information node (i-node) for a special device file. The special device files are the software interface between the system and the input/output devices. Special files exist for each communication line, etc.

Command Format

The general format of the **mknod** command is as follows.

mknod *name* **b** | **c** *major minor*

mknod *name* **p**

The various options of the **mknod** are as follows.

b	Specifies the special file as a block-type device.
c	Specifies the special file as a character-type device.
major	The *major* number is the driver.
minor	The *minor* number is the physical device.
name	Specifies the *name* of the special file.
p	Specifies the special file as a first-in, first-out device. This is also known as a pipe.

Sample Command Use

The following command line entries and system responses show the use of the **mknod** command to create character-type files for the support of ports board number 3. Normally, these files are created by auto-configuration. If you had to manually create these files, the **mknod** command would be used as follows.

```
# cd /dev<CR>
# mknod tty31 c 3 0<CR>
# mknod tty32 c 3 1<CR>
# mknod tty33 c 3 2<CR>
# mknod tty34 c 3 3<CR>
# chgrp root tty3*<CR>
# ls -l tty3*<CR>
crw-r--r--   1 root     root       3,  0 Sep  5 11:24 /dev/tty31
crw-r--r--   1 root     root       3,  1 Sep  5 11:24 /dev/tty32
crw-r--r--   1 root     root       3,  2 Sep  5 11:25 /dev/tty33
crw-r--r--   1 root     root       3,  3 Sep  5 11:25 /dev/tty34
crw-r--r--   1 root     root       3,  4 Sep  5 11:25 /dev/tty35
#
```

/etc/mkunix — Make a New UNIX Operating System

General

The **mkunix** command is used to create a bootable kernel. The in-core version of the operating system is copied to a specified file. Normally, **mkunix** is executed following an auto-configuration boot of a new system configuration. This is done automatically as part of self-configuration.

Command Format

The general format of the **mkunix** command is as follows.

mkunix [*namelist*] [**-o** *newlist*]

The *namelist* optional argument specifies the boot program. The default is the path name specified by the BOOT variable in **/etc/system**. The *newlist* optional argument specifies the name of the object file. The default *newlist* is **a.out**.

Sample Command Use

The following command line entry and system responses show the default execution of the **mkunix** command.

```
# mkunix<CR>
Using /boot/KERNEL for input file
Read 1249 symbols from /boot/KERNEL
352 symbols added
# ls -l /unix /a.out<CR>
-rwxr--r--   1 root     other      190268 Aug 29 16:04 /unix
-rwxr--r--   1 root     other      190268 Sep  6 07:55 /a.out
#
```

/etc/mount — Mount a File System

General

The **mount** command is used to identify the existence of a removable file system on a special device file. The mount point is a directory under **root**. The mount point must already exist for the **mount** command to succeed. Special devices (file systems) can be mounted with read and write access or with only read access permission. When no arguments are specified, the command outputs the current mount table.

Command Format

The general format of the **mount** command is as follows.

/etc/mount [*device directory* [**-r**]]

When no arguments are specified, the command outputs the current mount table. When mounting a file system, the *device* specifies the path name of the special device file to be mounted at the specified *directory*. The **-r** option is used to mount a file system with only read permission. The file system is mounted with both read and write permissions when the **-r** option is omitted.

Sample Command Use

The following command line entries and system responses show how to output the current mount table and mount the **/usr** file system. The **/etc/umount** command is also used in this example to unmount a file system. The system is in run-level 1 for this example.

```
# mount<CR>
/ on /dev/dsk/c1d0s0 read/write on Tue Aug 14 15:29:50 1984
/usr on /dev/dsk/c1d0s2 read/write on Tue Aug 14 15:33:25 1984
# umount /dev/dsk/c1d0s2<CR>
# mount<CR>
/ on /dev/dsk/c1d0s0 read/write on Tue Aug 14 15:29:50 1984
# mount /dev/dsk/c1d0s2 /usr<CR>
# mount<CR>
/ on /dev/dsk/c1d0s0 read/write on Tue Aug 14 15:29:50 1984
/usr on /dev/dsk/c1d0s2 read/write on Wed Aug 15 05:23:15 1984
#
```

/etc/mountall — Mount File System per Table

General

The **mountall** command is used to mount file systems per a file system mount table. Before a file system is mounted, the file system state is checked using **/etc/fsstat**. If the state of the file system is "okay" the file system is mounted. If the file system state is bad the file system is checked using the **/etc/fsck** command. (See **/etc/umountall**.)

Command Format

The general format of the **mountall** command is as follows.

mountall [-] *table*

The "-" is used to read from the standard input. The *table* argument specifies the path name of one or more file system mount tables. Typically, the *table* argument specifies the **/etc/fstab** file.

Sample Command Use

Refer to the **mountall**(1M) manual page for examples of this command.

/etc/mvdir — Move Directory

General

The **mvdir** command is an executable file (shell script) used to rename or move directories WITHIN a file system. You must be logged in as **root** to use this command.

Command Format

The general format of the **mvdir** command is as follows.

mvdir *oldname newname*

The *oldname* is the name of an existing directory to be renamed. The *newname* is the new directory name that does not yet exist. The names can be expressed as full path names. Neither directory name can be a subset of the other.

Sample Command Use

The following command line entries and system responses show how to move the directory **/usr/sam/letters** to a new directory named **/usr/sam/oldletters**. The **ls** command is used to show the contents of the **/usr/sam** directory before and after the use of the **mvdir** command. The **oldletters** directory does not yet exist. Also shown is an attempt to move the **oldletters** directory outside the **/usr** file system and the resulting error message. When an attempt is made to move a directory to a different file system, the source directory is not moved and an error message is output.

```
# ls -l /usr/sam<CR>
total 1
drwxr-xr-x   2 root     other         80 Sep  7 10:44 letters
# mvdir /usr/sam/letters /usr/sam/oldletters<CR>
# ls -l /usr/sam<CR>
total 1
drwxr-xr-x   2 root     other         80 Sep  7 10:45 oldletters
# mvdir /usr/sam/oldletters /mnt/letters<CR>
Cannot link to /mnt/letters
# ls -l /usr/sam /mnt<CR>

/mnt:
total 0

/usr/sam:
total 1
drwxr-xr-x   2 root     other         80 Sep  7 10:45 oldletters
#
```

/etc/ncheck — Output File System Path Names and I-Nodes

General

The **ncheck** command is used to output a list of path names and associated information node (i-node) numbers for a specified file system. The default output is a list of i-nodes and path names for the file systems listed in the **/etc/checklist** file. (As the system is delivered, the **/etc/checklist** file is empty.) An option is provided to report files with a set-user identification mode. The output of the command must be sorted to be meaningful. Either the raw or block device can be specified.

Command Format

The general format of the **ncheck** command is as follows.

/etc/ncheck [**-i** *numbers*][**-a**][**-s**][*fsdevice*]

The various options and arguments of the **ncheck** command are as follows.

-a	This option causes the dot (.) and dot-dot (..) files to be included in the output.
-i *numbers*	This option causes only the files identified by the listed i-node *numbers* to be output.
-s	This option causes only the files with a set-user identification mode (set-UID) to be output.
fsdevice	The *fsdevice* argument is an optional argument specifying the file system special device file to be used as source. When this argument is unspecified, the **ncheck** command uses the root file system device (**/dev/dsk/c1d0s0**).

Sample Command Use

The following command line entries and system responses show how to use the **ncheck** command to identify all files in the **/usr** file system (**/dev/dsk/c1d0s2**) with a set-user identification. This form of the command (-s option) is used to discover violations of security. In the following example, the **/rar/bin/sh** item should be investigated.

```
# ncheck -s /dev/dsk/c1d0s2<CR>
/dev/dsk/c1d0s2:
107     /bin/nkill
204     /bin/at
210     /bin/crontab
214     /bin/shl
900     /bin/sadp
903     /bin/timex
1153    /bin/ct
1154    /bin/cu
1155    /bin/uucp
1157    /bin/uuname
1159    /bin/uustat
1161    /bin/uux
1197    /bin/cancel
1198    /bin/disable
1199    /bin/enable
1200    /bin/lp
1201    /bin/lpstat
154     /lib/mv_dir
940     /lib/expreserve
859     /lib/exrecover
1203    /lib/accept
1204    /lib/lpadmin
1205    /lib/lpmove
1206    /lib/lpsched
1207    /lib/lpshut
1210    /lib/reject
153     /lib/mailx/rmmail
906     /lib/sa/sadc
1167    /lib/uucp/uucico
1174    /lib/uucp/uusched
1175    /lib/uucp/uuxqt
1589    /rar/bin/sh
#
```

The following command line entries and system responses show how to generate a list of file names from a list of information node (i-node) numbers. When a target file system is not specified, the command defaults to the **root** file system (/dev/dsk/c1d0s0).

```
# ncheck -i 100 200 101<CR>
/dev/idsk00:
200     /bin/tail
100     /etc/vtoc/hd6dft
101     /etc/vtoc/hd7dft
# ncheck -i 100 200 101 /dev/dsk/c1d0s2<CR>
/dev/dsk/c1d0s2:
100     /bin/checkfsys
101     /bin/cut
200     /bin/uniq
#
```

/etc/newboot — Make a Bootable Device

General

The **newboot** command is used to copy "lboot" or "olboot" and "mboot" programs to a device boot partition. This enables other programs on the device to be executed (booted) from the firmware mode. The **newboot** command verifies that each boot program will fit in the specified partition and calls the **dd** command to copy the data. Either the raw (character) or block device can be specified.

Command Format

The general format of the **newboot** command is as follows.

newboot *lboot mboot device*

The boot programs specified by the *lboot* and *mboot* arguments can be of your own invention, but are more often those provided in the **lib** directory. The *device* argument is the special device partition (raw or block device) specifying the boot block partition. Refer to Appendix A, "DEVICE DEFAULT PARTITIONING," for boot partition device-dependent information. The most common command formats for the floppy, hard disk, and cartridge tape drives are as follows. The question mark (?) represents the controller number for the cartridge tape drive.

```
Floppy Disk:      newboot /lib/olboot /lib/mboot /dev/rdsk/c0d0s7

Hard Disk 1:      newboot /lib/lboot /lib/mboot /dev/rdsk/c1d0s7

Cartridge Tape:   newboot /lib/lboot /lib/mboot /dev/rdsk/c?d0s7
```

Sample Command Use

The following shows the command line entries and system responses associated with making a "DIAGNOSTIC" floppy disk. The **newboot** command is used to copy the boot programs to the boot partition on the floppy disk (c0d0s7). Refer to "MAKING A BOOTABLE DEVICE" task description in Chapter 3, "ADMINISTRATIVE TASKS," for a complete description of the procedure.

```
# fmtflop -v /dev/rdsk/c0d0s6<CR>
# mkfs /dev/dsk/c0d0s5 1404:192 1 18<CR>
MKfs: /dev/dsk/c0d0s5?
(DEL if wrong)
bytes per logical block = 1024
total logical blocks = 702
total inodes = 192
gap (physical blocks) = 1
cylinder size (physical blocks) = 18
mkfs: Available blocks = 687
# labelit /dev/rdsk/c0d0s5 dgn 2032<CR>
Current fsname: , Current volname: , Blocks: 1404, Inodes: 160
FS Units: 1Kb, Date last mounted: Fri Mar  8 12:06:43 1985
NEW fsname = dgn, NEW volname = 2032 -- DEL if wrong!!
# newboot /lib/lboot /lib/mboot /dev/rdsk/c0d0s7<CR>
newboot: confirm request to write boot programs to /dev/rdsk/c0d0s7: y<CR>
# mount /dev/dsk/c0d0s5 /mnt<CR>
mount: warning! <dgn> mounted as </mnt>
# cd<CR>
# find dgn dgmon filledt  -print | cpio -pdumv /mnt<CR>
/mnt/dgn/edt_data
/mnt/dgn/SBD
/mnt/dgn/X.SBD
/mnt/dgn/PORTS
/mnt/dgn/X.PORTS
/mnt/dgn/CTC
/mnt/dgn/X.CTC
/mnt/dgn/ISC
/mnt/dgn/X.ISC
/mnt/dgn/OLDedt_data
/mnt/dgmon
/mnt/filledt
540 blocks
# umount /dev/dsk/c0d0s5<CR>
#
```

/bin/newgrp — Log In to a New Group

General

The **newgrp** command changes the group identification of its caller. A new shell is spawned with a different group identification. You will remain logged in and your current directory will remain unchanged. Only variables in the new environment are known in the new shell. Any variables that were previously exported are no longer marked as such. Calculations of access permissions to files are now done with respect to the identification of the new group.

Command Format

The general format of the **newgrp** command is as follows.

newgrp [-] [*group*]

The various arguments are as follows.

-	Causes the environment to be changed to the one that would exist if you had actually logged in again.
group	The name of the new group. If you do not specify the group, the **newgrp** command will change the group identification to the group in the password file. In effect, you will be changing the group identification back to your original group.

A password is demanded if the group has a password and you do not. Your logname must be in the **/etc/group** as being a member of the new group. If you are **root**, none of this is applicable.

Sample Command Use

The following example shows how to use the **newgrp** command. In this example the group name is shown by creating a file and listing the group name for the file. The group name of **other** is then changed to **root**. This example assumes that your logname has been added to the **/etc/group** file for the **root** group name.

```
$ >file<CR>
$ ls -g file<CR>
-rw-r--r--   1 other          0 Aug 14 13:20 file
$ newgrp - root<CR>

     Your profile is executed. Depending on
     your profile, additional output messages
     may be displayed. Also additional input
     may be required.

$ >newfile<CR>
$ ls -g newfile<CR>
-rw-r--r--   1 root           0 Aug 14 13:20 file
$
```

After executing this example, you are a member of the group *root*. As a normal user, you can now access files or execute commands (or shell programs) that are part of the group *root* as long as the owner of the file has set the access permission bits to allow group access.

The following example shows a failed attempt to change group identification. In this example, the calling logname is NOT listed for the requested group in the **/etc/group** file.

```
$ newgrp - root<CR>
newgrp: Sorry

      Your profile is executed. Depending on
      your profile, additional output messages
      may be displayed. Also additional input
      may be required.

$
```

In these examples, additional output may be seen depending on the contents of the invoking user **.profile** file. Refer to the **newgrp** manual page for more information.

/etc/prtvtoc — Print Volume Table of Contents

General

The **prtvtoc** command is used to output the Volume Table Of Contents (VTOC) for the specified hard disk. Following a full restore of the system, the **prtvtoc** can be used to check the disk partitioning.

Command Format

The general format of the **prtvtoc** command is as follows.

prtvtoc [**-f**][**-h**][**-s**][**-t** *fstab*] *raw_device*

The various options and arguments of the **prtvtoc** command are as follows. The default output is the entire volume table of contants for the device with the **/etc/fstab** file providing the mount directory names. The following options can be used in various combinations to alter the default output of the command.

-f	Outputs ONLY the "FREE" information for a device. The information includes FREE_START, FREE_SIZE, FREE_COUNT, and FREE_PART.
-h	Outputs the volume table of contents WITHOUT comments or header information.
-s	Outputs the volume table of contents WITHOUT comments and WITH column headers.
-t *fstab*	Outputs the volume table of contents using the file system table specified by the *fstab* argument for mount point name information. The default file system table is **/etc/fstab**.

The raw device file is specified by the *raw_device* argument. The command accepts only a single argument. The output of the command contains the following information.

PARTITION Identifies the disk partition (section). A maximum of 16 partitions are identified (0 through 15).

TAG The partition identification (TAG) is a number identifying the use of the partition. Reserved codes are as follows:

NUMBER	NAME
0	UNASSIGNED
1	BOOT
2	ROOT
3	SWAP
4	USR
5	BACKUP

FLAG The FLAG is a number code identifying how the partition is to be mounted.

NUMBER	NAME
00	MOUNTABLE, READ AND WRITE
01	NOT MOUNTABLE
10	MOUNTABLE, READ ONLY

SECTOR START

Identifies the starting sector (block) partition number. Blocks are counted starting from 0 to the end of the disk.

SIZE IN SECTORS

Identifies the number of sectors (blocks) in the partition.

MOUNT DIRECTORY

Identifies the mount point of any mounted partitions, except for the root file system.

Sample Command Use

The following command line entries and system responses are typical for the use of the **prtvtoc** command. Any section (partition) can be specified to output the volume table of contents for the device. The values shown are for a 30-megabyte hard disk following a full restore procedure.

```
# prtvtoc /dev/rdsk/c1d0s0<CR>
* /dev/rdsk/c1d0s0 partition map
*
* Dimensions:
*     512 bytes/cylinder
*      18 sectors/track
*       5 tracks/cylinder
*     697 cylinders
*     695 accessible cylinders
*
* Flags:
*   1: unmountable
*  10: read-only
*
* Unallocated space:
*       Start              Size
*       62460                90
*
*   Partition  Tag  Flags  First Sector  Sector Count   Mount Directory
        0       2    00        6120          12510
        1       3    01         100           6020
        2       4    00       18630          43830        /usr
        6       0    01           0          62460
        7       0    01           0            100
# prtvtoc /dev/rdsk/c1d0s6<CR>
* /dev/rdsk/c1d0s0 partition map
*
* Dimensions:
*     512 bytes/cylinder
*      18 sectors/track
*       5 tracks/cylinder
*     697 cylinders
*     695 accessible cylinders
*
* Flags:
*   1: unmountable
*  10: read-only
*
* Unallocated space:
*       Start              Size
*       62460                90
*
*   Partition  Tag  Flags  First Sector  Sector Count   Mount Directory
        0       2    00        6120          12510
        1       3    01         100           6020
        2       4    00       18630          43830        /usr
        6       0    01           0          62460
        7       0    01           0            100
#
```

The following command line entry and system responses show the use of the **prtvtoc** command using the **-f** option. Any section (partition) can be specified to output the volume table of contents for the device. The values shown are for a 30-megabyte hard disk following a full restore procedure.

```
# prtvtoc -f /dev/rdsk/c1d0s0<CR>
FREE_START=62460 FREE_SIZE=90 FREE_COUNT=1 FREE_PART=34589abcdef
#
```

/etc/pwck — Password Check

General

The **pwck** command is used to check the **/etc/passwd** file for inconsistencies.

Command Format

The general format of the **pwck** command is as follows.

pwck [*file*]

The *file* argument is used to specify a different file than the default **/etc/passwd** file.

Sample Command Use

The following command line entry and system responses are typical for the use of the **pwck** command.

```
# pwck<CR>

sys:Locked;:3:3:0000-Admin(0000):/usr/src:
        Login directory not found

rje:Locked;:18:18:0000-rje(0000):/usr/rje:
        Login directory not found

powerdown::0:0:general system administration:/usr/admin:/bin/rsh
        Logname too long/short

checkfsys::0:0:check diskette file system:/usr/admin:/bin/rsh
        Logname too long/short

mountfsys::0:0:mount diskette file system:/usr/admin:/bin/rsh
        Logname too long/short

umountfsys::0:0:unmount diskette file system:/usr/admin:/bin/rsh
        Logname too long/short
#
```

/etc/rc0 — Execute Commands to Stop the System

General

The **rc0** command is a shell script (Figure 6-6) used to put the system in the run-level 0. Normally, the command is executed by an entry in the **/etc/inittab** file and is not entered at a data terminal. Hence, sample command line entries and system responses are not provided for this command.

```
# "Run Commands" for init state 0
# Leaves the system in a state where it is safe to turn off
#  the power or go to firmware.

stty sane 2>/dev/null
echo 'The system is coming down.  Please wait.'
for f in /etc/shutdown.d/*
{
        if [ -f ${f} ]
        then
                /bin/sh ${f}
        fi
}
trap "" 15
kill -15 -1
sleep 10
/etc/killall  9
sleep 10
sync
/etc/umountall
stty sane 2>/dev/null
ps=`ps -fe`
psc=`echo "${ps}"  |  wc -l`
if [ ${psc} -gt 7 ]
then
        echo 'The shutdown did not complete properly.
Too many processes are still running.'
        echo "${ps}"
fi
sync;  sync
echo '\nThe system is down.'
sync
```

Figure 6-6. Typical /etc/rc0 File

The **rc0** command executes files in the **/etc/shutdown.d** directory. As the system is delivered, the only function done by the **rc0** command and associated **/etc/shutdown.d** files is displaying the "System services are now being stopped." message. Additional power-down functions can be added by adding shell script files in the **/etc/shutdown.d** directory. Refer to the **/etc/rc0** descriptions in Chapter 2, "ADMINISTRATIVE DIRECTORIES AND FILES," for additional information.

Command Format

The format of the **rc0** command consists of only the command name. The command is normally executed by an entry in the **/etc/inittab** file and not from a data terminal.

/etc/rc2 — Execute Commands for Single-User Mode

General

The **rc2** command is a shell script (Figure 6-7) used to put the system in the run-level 1. Normally, the command is executed by an entry in the **/etc/inittab** file and is not entered at a data terminal. Hence, sample command line entries and system responses are not provided for this command.

```
# "Run Commands" executed when the system is changing to
# init state 2, traditionally called "multi-user".

 . /etc/TIMEZONE

 # Pickup start-up packages for mounts, daemons, services, etc.
 stty sane 2>/dev/null
 echo 'The system is coming up.  Please wait.'
 for f in /etc/rc.d/*
 {
         if [ -f ${f} ]
         then
                 /bin/sh ${f}
         fi
 }
 echo 'The system is ready.'
```

Figure 6-7. Typical /etc/rc2 File

The **rc2** command executes files in the **/etc/rc.d** directory. The functions done by the **rc2** command and associated **/etc/rc.d** files include:

- Setting and exporting the TIMEZONE variable.
- Setting-up and mounting the root (/) and user (/usr) file systems.
- Cleaning up (remaking) the **/tmp** and **/usr/tmp** directories.
- Loading the network interface and ports cards with program data and starting the associated processes.
- Starts the **cron** daemon by executing **/etc/cron**.
- Cleans up (deletes) uucp locks (LCK*), status (STST*), and temporary (TM*) files in the **/usr/spool/uucp** directory.

Other functions may be added, as required, to support the addition of hardware and software features. Refer to the **/etc/rc2** description in Chapter 2, "ADMINISTRATIVE DIRECTORIES AND FILES," for additional information.

Command Format

The format of the **rc2** command consists of only the command name. The command is normally executed by an entry in the **/etc/inittab** file and not from a data terminal.

/etc/setclk — Set System Time From Hardware Clock

General

The **setclk** command is used to set the internal system time from the hardware time-of-day clock. The command is normally executed by an entry in the **/etc/inittab** file when the run level is changed. Note that the year is maintained in Nonvolatile Random Access Memory (NVRAM). When **setclk** is run, the user is prompted for the time, date, day-of-week, hours off Greenwich Mean Time, and whether or not daylight savings is in effect if the hardware clock is insane (or not set). If the hardware clock is sane (set), **setclk** runs silently.

Command Format

The format of the command consists of only the command name. Since the command is normally executed by **/etc/inittab** and not from a data terminal, command line entries and system responses are not provided.

/etc/setmnt — Establish Mounted File System Table

General

The **setmnt** command is used to make the file system mount table that identifies the mounted file systems. The **setmnt** command reads standard input and creates a line entry in the **/etc/mnttab** file.

Command Format

The **/etc/setmnt** command does not use any options. Each **/etc/mnttab** line entry is created from **/etc/setmnt** input lines that are in the form of the special device file name followed by the path name of the directory (node) for the file system. The command can be used to establish a new file system mount table. When input is taken from a terminal, the input must be ended with a control-d. Normally, the **/etc/mount** and **/etc/umount** commands use the **setmnt** command to modify the mount table.

Sample Command Use

The following command line entries and system responses show the modification of the mount table using the **/etc/setmnt** command. Normally, this command is called by the **/etc/mount** and **/etc/umount** commands. The execution from a data terminal is for the purpose of this example.

```
# mount<CR>
/ on /dev/dsk/c1d0s0 read/write on Tue Aug 14 07:07:16 1984
/usr on /dev/dsk/c1d0s2 read/write on Tue Aug 14 07:23:11 1984
/mnt on /dev/SA/diskette1 read/write on Tue Aug 14 07:51:47 1984
# setmnt<CR>
<CTRL d>
# mount<CR>
# setmnt<CR>
c1d0s0 /<CR>
c1d0s2 /usr<CR>
c0d0s6 /mnt<CR>
<CTRL d>
# mount<CR>
/ on /dev/dsk/c1d0s0 read/write on Tue Aug 14 09:36:01 1984
/usr on /dev/dsk/c1d0s2 read/write on Tue Aug 14 09:36:01 1984
/mnt on /dev/SA/diskette1 read/write on Tue Aug 14 09:36:01 1984
#
```

/etc/shutdown — Orderly Terminate All Processing

General

The **shutdown** command is used to change the operating state of the system. The various run levels of the system are summarized in Appendix B, "RUN LEVELS." The **shutdown** command provides for the orderly transition between run levels. You must be logged in as **root** to use this command and the current directory must be /. When the **shutdown** command is executed during a scheduled backup time interval, the capability to either abort the shutdown process and return to the command level or to continue the shutdown process is provided. Refer to the "FILE SYSTEM BACKUP" task description in Chapter 3, "ADMINISTRATIVE TASKS," for additional information on planning and scheduling backups.

Command Format

The general format of the **shutdown** command is as follows.

shutdown [**-y**] [**-g***grace*] [**-i***state*]

When no arguments are specified, the **shutdown** command takes the system to the single-user mode of operation. The various arguments of the **shutdown** command are as follows.

-y	Specifies a "yes" response for shutdown continuation.
-g*grace*	Specifies the number of seconds before all "user" processes are terminated. The default grace period is 60 seconds.
-i*state*	Specifies the system run-level to initialize. Appendix B, "RUN LEVELS," summarizes the system run-levels (states). The default *state* is the single-user mode (1, s, or S).

Sample Command Use

The following shows the command line entry and system responses for a shutdown from the multi-user to single-user state. The example provided is with one user and root logged in. Note that only the broadcast messages are displayed at the user's terminal (contty or tty?? ports). The output shown is at the console terminal.

```
# shutdown -y -i1<CR>

Shutdown started.    Thr Sep  6 05:15:43 EDT 1984

Broadcast Message from root (console) Thr Sep  6 05:15:47...
The system will be shut down in 60 seconds.
Please log off now.


Broadcast Message from root (console) Thr Sep  6 05:16:48...
THE SYSTEM IS BEING SHUT DOWN NOW ! ! !
Log off now or risk your files being damaged.


#
INIT: New run level: 1

Shutdown started.    Thr Sep  6 05:17:58 EDT 1984

Broadcast Message from root (console) Thr Sep  6 05:17:59...
THE SYSTEM IS BEING SHUT DOWN NOW ! ! !
Log off now or risk your files being damaged.


The system is coming down.  Please wait.
System services are new being stopped.

The system is down.

****      SYSCON CHANGED TO /dev/console      ****

INIT: New run level: S

INIT: SINGLE USER MODE
#
```

/bin/su — Become Super-User or Another User

General

The **su** command enables a user that is logged in on the system to relogin as another user without logging off or changing directory paths. The login environment can be retained or changed to that of the other login name.

The purpose of switching to another login is associated with the need to access directories and files owned by another user that cannot be accessed from your login. In other words, the appropriate group and other access permissions are denied.

Command Format

The general format of the **su** command is as follows.

su [**-**] [*login name* [*arguments*]]

The presence of the **-** flag causes the **.profile** associated with the specified *login name* to be executed. This changes your environment to that of the new user. The *login name* specifies the name of another user. The *arguments* specify the shell command(s) to be executed.

Sample Command Use

The following sequence of command line entries and system responses are typical of this command.

```
$ su - fred<CR>
Password:<password><CR>
$
```

In this example, you have in effect logged in as **fred**. The - flag caused the **.profile** associated with the user **fred** to be executed. You can return to your login by entering a control-d (CTRL d).

/bin/sync — Update Super Block

General

The **sync** command is used to update the super block of the various file systems. The **sync** command is used in various programs to write the system memory (system buffers) to disk. The **fsck**, **df**, and **shutdown** are some of the programs that use **sync** command.

Command Format

The format of the **sync** command consists of only the command name. No arguments are provided.

Sample Command Use

Refer to **/etc/shutdown** for an example of the use of the **sync** command. A typical **etc/shutdown** file is provided in Chapter 2, "ADMINISTRATIVE FILES AND DIRECTORIES."

/etc/sysdef — Output System Definition

General

The **sysdef** command is used to output system configuration information in a tabular format. All hardware devices, device local bus addresses, device unit counts, and all tunable parameters are listed.

Command Format

The general format of the **sysdef** command is as follows.

sysdef [*opsys*] [*master.d*]

When no arguments are specified, the **sysdef** command outputs the configuration information for the current UNIX Operating System. The *opsys* argument specifies the operating system boot file. The *master.d* argument specifies the system parameter files.

Sample Command Use

The following shows a typical default output of the **sysdef** command.

```
# sysdef<CR>
*
* 3B2 Configuration
*

  Boot program: /boot/KERNEL
  Time stamp:   Fri Aug  3 21:40:25 1984
*
* Devices
*
  ports board=1
  ports board=2
  hdelog
  iuart
  idisk
  mem
  tty
  sxt
  prf
*
* Loadable Objects
*
  shm
  sem
  msg
  ipc
*
* System Configuration
*
  rootdev         idisk  minor=0
  swapdev         idisk  minor=1 swplo=0 nswap=6020
  pipedev         idisk  minor=0
```

Continued

Continued from previous screen

```
*
* Tunable Parameters
*
   250  buffers in buffer cache (NBUF)
    30  entries in callout table (NCALL)
   100  inodes  (NINODE)
   100  entries in file table (NFILE)
    25  entries in mount table (NMOUNT)
    60  entries in proc table (NPROC)
    58  entries in shared text table (NTEXT)
    80  clist buffers (NCLIST)
    25  processes per user id (MAXUP)
    80  entries in swap map table (SMAPSIZ)
    80  entries in core map table (CMAPSIZ)
   128  hash slots for buffer cache (NHBUF)
     8  buffers for physical I/O (NPBUF)
    10  auto update time limit in seconds (NAUTOUP)
   256  pages per process (MAXMEM)
*
* Utsname Tunables
*
   2.0.3  release (REL)
    unix  node name (NODE)
    unix  system name (SYS)
       2  version (VER)
```

Continued

Continued from previous screen

```
*
* IPC Messages
*
   100 entries in msg map (MSGMAP)
  8192  max message size (MSGMAX)
 16384  max bytes on queue (MSGMNB)
    10  message queue identifiers (MSGMNI)
     8  message segment size (MSGSSZ)
    40  system message headers (MSGTQL)
  1024  message segments (MSGSEG)
*
* IPC Semaphores
*
    10  entries in semaphore map (SEMMAP)
    10  semaphore identifiers (SEMMNI)
    60  semaphores in system (SEMMNS)
    30  undo structures in system (SEMMNU)
    25  max semaphores per id (SEMMSL)
    10  max operations per semop call (SEMOPM)
    10  max undo entries per process (SEMUME)
 32767  semaphore maximum value (SEMVMX)
 16384  adjust on exit max value (SEMAEM)
*
* IPC Shared Memory
*
  8192  max shared memory segment size (SHMMAX)
     1  min shared memory segment size (SHMMIN)
     8  shared memory identifiers (SHMMNI)
     4  max attached shm segments per process (SHMSEG)
    32  max in use shared memory (SHMALL)
*
* File and Record Locking
*
   100  records configured on system (FLCKREC)
    20  file headers configured on system (FLCKFIL)
#
```

/etc/telinit — Tells Init What Actions to Perform

General

The **telinit** command is used to pass signals to the **/etc/init** daemon.

Command Format

The general format of the **telinit** command is as follows

telinit [**0** | **1** | **2** | **3** | **4** | **5** | **6** | **s** | **S** | **q** | **Q** | **a**| **b** | **c**]

The various options specify the system run level that is to be initialized. The 3B2 Computer run levels are summarized in Appendix B.

Sample Command Use

The following command line entries and system responses show how to use **telinit** to start a process. The **/etc/inittab** file has been edited to add the following line.

h1:a:wait:/bin/date > /dev/console 2>&1

This process (**date** command) is run via a **telinit a** command. After the command example is run, no prompt is returned. Entering a RETURN at the end of the example would result in a prompt (#).

```
# telinit a<CR>
Tue Aug 14 08:37:58 EDT 1984
<CR>
#
```

The following command line entries and system responses show that the pseudo run-levels **a**, **b**, and **c** are not run-levels that can be entered. The **init** process cannot enter run-level **a**, **b**, or **c**. A daemon-type process started by an **a**, **b**, or **c** continues to run when **init** changes states. A request to start the execution of a process associated with **a**, **b**, or **c** does not change the current run-level. Only the "date" of the run-level is changed. Remember from the previous example that the **/bin/date** is run when initializing run-level **a**.

```
# date<CR>
Tue Aug 14 08:40:23 EDT 1984
# who -r<CR>
   .       run-level 2  Aug 10 05:32    2    0    S
# init a<CR>
Tue Aug 14 08:40:48 EDT 1984
<CR>
# who -r<CR>
   .       run-level 2  Aug 14 08:40    2    0    S
# date<CR>
Tue Aug 14 08:41:14 EDT 1984
#
```

/etc/umount — Unmount a File System

General

The **umount** command is used to unmount a mounted file system. All files must be closed in the file system that is to be unmounted. The present working directory must be outside of the file system to be unmounted. The **/etc/killall** command is used to terminate all processes with open files so that a file system can be unmounted.

Command Format

The general format of the **umount** command is as follows.

umount *device*

Sample Command Use

The following command line entries and system responses show how to unmount a file system. In this example, the **/mnt** file system is mounted on an integral floppy disk and is then unmounted. The mount table is displayed before and after these operations.

```
# mount /dev/SA/diskette1 /mnt<CR>
# mount<CR>
/ on /dev/dsk/c1d0s0 read/write on Tue Aug 14 07:07:16 1984
/usr on /dev/dsk/c1d0s2 read/write on Tue Aug 14 07:23:11 1984
/mnt on /dev/SA/diskette1 read/write on Tue Aug 14 07:51:47 1984
# umount /dev/SA/diskette1<CR>
# mount<CR>
/ on /dev/dsk/c1d0s0 read/write on Tue Aug 14 07:07:16 1984
/usr on /dev/dsk/c1d0s2 read/write on Tue Aug 14 07:23:11 1984
#
```

/etc/umountall — Unmount All File Systems Except root

General

The **umountall** command is a shell script used to unmount all currently mounted file systems, except for the **root** file system.

Command Format

The general format of the **umountall** command is as follows.

umountall [**-k**]

The **-k** option causes **/etc/fuser**(1M) to send a kill signal to all processes that have files open in each file system before unmounting the file systems. The **umountall** will fail without the **-k** option for file systems that are busy. No messages are output if the file systems are unmountable. Any error messages that are output are associated with the **/etc/umount** command.

Sample Command Use

Refer to the **umountall**(1M) manual page for examples of this command. This command is used in the **/etc/rc0** script.

/etc/unlink — Execute Unlink System Call

General

The **unlink** command is used to remove a file name that points to another file. The most common use of the **unlink** command is in the device files. Links are removed by the **/etc/unlink** command. The **unlink** can be executed only by **root**. By executing an inappropriate **unlink** command, it is possible to trash a file system. Therefore, it is recommended that the **/bin/rm** and **/bin/rmdir** commands be used to remove files and directories when possible.

Command Format

The general format of the **unlink** command is as follows.

unlink *name*

The *name* argument specifies the directory or file to be unlinked (removed).

Sample Command Use

The following command line entries and system responses show the creation of **file1**, the linking of **file1** to **file2** using the **link** command, and the removal of **file1** using the **unlink** command. The **/bin/rm** command would accomplish the same result as the **unlink** command is this example. The **/bin/ls** is used to list the files.

```
# >file1<CR>
# link file1 file2<CR>
# ls -l file1 file2<CR>
-rw-r--r--   2 root     other          0 Sep  5 19:41 file1
-rw-r--r--   2 root     other          0 Sep  5 19:41 file2
# unlink file1<CR>
# ls -l file*<CR>
-rw-r--r--   1 root     other          0 Sep  5 19:41 file2
#
```

/etc/volcopy — Make a Literal Copy of a File System

General

The **volcopy** command is used to make a literal copy of a file system. Volume copy is used to copy an entire file system with label checking from one raw (character) device partition to another raw (character) device partition. No compression or file system reorganization is done as with **/etc/dcopy** command; the volume copy is a literal copy. The source file system (partition) should be checked before copying. The destination file system (partition) should be checked after copying.

A record of file system volume copies is maintained in **/etc/log/filesave.log** file. The **/etc/log/filesave.log** file must be created before you first use the **volcopy** command if you want to record this information. The **volcopy** will not create this log file, the program only makes log entries to the file if it already exists. Thus, if you do not want a record of volume copies either do not create the file or delete the file if it exits.

Command Format

The general format of the **volcopy** command is as follows.

volcopy [*options*] *fsname srcdevice volname1 destdevice volname2*

The various options and arguments of the **volcopy** command are as follows.

-a	This option invokes a verification sequence that requires a positive operator response instead of the standard 10-second delay before copy execution.
-s	This option invokes a "DEL if wrong" verification sequence that requires a positive operator response in 10 seconds to prevent the copy execution.

fsname This argument specifies the mount point (mounted name) of the file system being copied. A maximum of six characters are used. If more than six characters are specified, only the first six characters are used. Typical *fsname* arguments are **root**, **usr**, and **init**.

srcdevice This argument specifies the source device file. The *srcdevice* is a physical disk section (partition) or physical tape section. Typical *srcdevice* arguments are **/dev/dsk/c1d0s0** (root), **/dev/dsk/c1d0s2** (usr), and **/dev/dsk/c1d0s8** (/usr2). The character device (raw device) is also used as source (/dev/rdsk/c1d0s0). The *srcdevice* and the *destdevice* must be of the same type (character or block).

volname1 This argument specifies the physical volume name of the source partition. A maximum of six characters are used. If more than six characters are specified, only the first six characters are used. If a hyphen (-) is specified, the existing source volume name is used.

destdevice This argument specifies the physical destination device file. The *destdevice* is a physical disk section (partition) or physical tape section. Typical *destdevice* arguments are **/dev/dsk/c1d1s0** (root), **/dev/dsk/c1d1s2** (usr), and **/dev/dsk/c1d1s8** (/usr2). The character device (raw device) can also used as the destination device (/dev/rdsk/c1d0s0). The *srcdevice* and the *destdevice* must be of the same type (character or block).

volname2 This argument specifies the physical volume name of the destination partition. A maximum of six characters is used. If more than six characters is specified, only the first six characters are used. If a hyphen (-) is specified, the existing source volume name is used.

Sample Command Use

The following command line entries and system responses show the use of **volcopy** to copy the **usr2** file system on partition c1d1s8 to partition c1d1sa. The **/etc/log/filesave.log** HAS NOT BEEN created in this example. The system is in the single-usr mode (run-level S) at the start of the example. The source file system partition is checked at the beginning of the example and the destination file system partition is checked at the end of the example using the **/etc/fsck**(1M) command. The **/etc/labelit**(1M) command is used to show the partition labeling. Since partition c1d1sa is not labeled, the label checking function of the **volcopy** command outputs a message noting the difference.

```
# fsck -D /dev/rdsk/c1d1s8<CR>

  /dev/rdsk/c1d1s8
  File System: usr2 Volume: 2032

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  127 files 728 blocks 14626 free
# labelit /dev/rdsk/c1d1s8<CR>
Current fsname: usr2, Current volname: 2032, Blocks: 15600, Inodes: 1936
FS Units: 1Kb, Date last mounted: Fri Mar 22 13:13:52 1985
# labelit /dev/rdsk/c1d1sa<CR>
Current fsname: , Current volname: , Blocks: 15600, Inodes: 1936
FS Units: 1Kb, Date last mounted: Mon Mar 25 05:40:38 1985
# volcopy -a usr2 /dev/rdsk/c1d1s8 - /dev/rdsk/c1d1sa -<CR>
warning! from fs(usr2) differs from to fs()
To filesystem dated:  Mon Mar 25 05:36:45 1985
From: /dev/rdsk/c1d1s8, to: /dev/rdsk/c1d1sa? (y or n) y<CR>
  END: 15600 blocks.
volcopy: cannot access /etc/log/filesave.log
# fsck -D /dev/rdsk/c1d1s8<CR>

  /dev/rdsk/c1d1sa
  File System: usr2 Volume:

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  127 files 728 blocks 14626 free
#
```

The following command line entries and system responses show the use of **volcopy** to copy the **usr2** file system from partition c1d1s8 to c1d1sa. The **-s** option requires a "DEL if wrong" response within 10 seconds to prevent the copy execution. The destination partition label matches the source partition label because of the previous example. The **/etc/log/filesave.log** is created in this example. The source file system partition is checked at the beginning of the example and the destination file system partition is checked at the end of the example using the **/etc/fsck**(1M) command.

```
# > /etc/log/filesave.log<CR>
# labelit /dev/rdsk/c1d1s8<CR>
Current fsname: usr2, Current volname: 2032, Blocks: 15600, Inodes: 1936
FS Units: 1Kb, Date last mounted: Mon Mar 25 05:37:31 1985
# labelit /dev/rdsk/c1d1sa<CR>
Current fsname: usr2, Current volname: , Blocks: 15600, Inodes: 1936
FS Units: 1Kb, Date last mounted: Mon Mar 25 05:37:31 1985
# fsck -D /dev/rdsk/c1d1s8<CR>

  /dev/rdsk/c1d1s8
  File System: usr2 Volume:

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  127 files 728 blocks 14626 free
# volcopy usr2 /dev/dsk/c1d1s8 - /dev/dsk/c1d1sa -<CR>
/dev/dsk/c1d1s8 less than 48 hours older than /dev/dsk/c1d1sa
To filesystem dated:  Fri Mar 22 13:13:52 1985
Type 'y' to override:     y<CR>
From: /dev/rdsk/c1d1s8, to: /dev/rdsk/c1d1sa? (DEL if wrong)
  END: 1368 blocks.
# cat /etc/logfilesave.log<CR>
/dev/rdsk/c1d1s0;usr2;2032 -> /dev/rdsk/c1d1s2;usr2; on Mon Mar 25 06:43:17 1985
# fsck -D /dev/rdsk/c1d1s8<CR>

  /dev/rdsk/c1d1sa
  File System: usr2 Volume: 2032

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  127 files 728 blocks 14626 free
#
```

/etc/whodo — Output Who Is Doing What

General

The **whodo** command is used to output what each logged-in user is doing. The execution of the **whodo** command depends on the existence of the **/etc/ps_data** file. The **/etc/ps_data** file is created by the execution of the **/etc/ps** command. If the **whodo** command is executed before a **bin/ps** command, the **whodo** command reports a failure to open the **/etc/ps_data** file. This situation normally occurs after a reboot of the system.

Command Format

The format of the command consists of only the command name with no arguments.

Sample Command Use

The following shows the typical output for the **whodo** command.

```
# whodo<CR>

Tue Aug 14 08:07:39 1984
unix                                  This is the node name.

console  root      7:23
    console    55    0:06 sh
    console   238    0:00 whodo

ttyl1    rar       7:35
    ttyl1     116    0:06 sh
    ttyl1     237    0:09 vi

tty12    cec       7:24
    tty12      87    0:00 sh
    tty12      88    0:03 cu
    tty12      89    0:17 cu
    tty12      58    0:05 sh

tty24    mtp       8:02
    tty24      62    0:04 sh
    tty24     234    0:01 ed
#
```

The following shows the execution of the **whodo** command that fails because of a nonexistent **/etc/ps_data** file. The **/bin/ps** command is then executed to create the **/etc/ps_data** file. After the creation of the **/etc/ps_data** file, the **whodo** command executes normally.

```
# whodo<CR>
Wed Sep  5 18:09:55 1984
unix
whodo: open error of /etc/ps_data: No such file or directory
# ps<CR>
   PID TTY      TIME COMMAND
    64 console  0:05 sh
   761 console  0:57 vi
   813 console  0:00 sh
   814 console  0:00 ps
# whodo<CR>
Wed Sep  5 18:12:59 1984
unix

console  root     13:49
    console    64    0:05 sh
    console   761    1:00 vi
      ?       818    0:00 <defunct>
    console   819    0:00 sh
    console   820    0:00 whodo
#
```

Appendix A

DEVICE DEFAULT PARTITIONING

PAGE

Appendix A

DEVICE DEFAULT PARTITIONING

DEVICE NAMES AND DESIGNATORS

Pre-Release 2.0.

The standard disk names used to identify the integral floppy disk and integral hard disk drives before Release 2.0 are **/dev/idsk00** through **/dev/idsk07** for the hard disk and **/dev/ifdsk00** through **/dev/ifdsk07** for the floppy disk. The 00 through 07 identifies the disk partitions.

The Pre-Release 2.0 disk names are linked to the new Release 2.0.3 disk names. These links provide a transition to the new disk designators for systems that are upgrading to UNIX System V, Release 2.0.

Release 2.0.3

Standard device names are used to identify the floppy disk, hard disk, and cartridge tape drives. The term "integral" is used to define devices driven by a controller on the system board. The first floppy disk drive and the first two hard disk drives are integral devices by this definition.

All disk device files use **/dev/dsk** for the block device, and **/dev/rdsk** for the raw (character) device. For the integral floppy disk drive, the controller/drive designator is "**c0d0**" for both the raw and block device. For the second floppy disk drive, the controller/drive designator is "**c?d1**" for both the raw and block device. The question mark (?) indicates the number of the controller. The second floppy disk drive is driven by the first Cartridge Tape Controller card. The major number for this controller depends on the slot used for the card. For the integral hard disk drives, the controller/drive designators are "**c1d0**" and "**c1d1**" for both the raw and the block device. For the first cartridge tape drive, the controller/drive designators are "**c?d0**" for both the raw and block device. The question mark (?) indicates the number of the controller. The partition or section designator is appended to the drive control/drive designator.

Release 2.0 and later releases expand the number of possible hard disk partitions from eight to sixteen (0 through f, hexadecimal). The other devices remain at a maximum of 8 partitions (0 through 7). The partitioning for the floppy disk and cartridge tape are fixed by the processes used to format the media. The hard disk partitioning can be configured to best support the system application.

Certain of the device partitions are used for specific functions. These functions are required when a device is a bootable device. For example:

- Partition 0 is used for root (/).
- Partition 1 is used for swap when the device is configured as the root device for the UNIX Operating System.
- Partition 2 is used for the **/usr** file system.
- Partition 6 specifies the entire device.
- Partition 7 specifies the boot area of a device.
- Partition 8 is used as the first hard disk area for user login directories (the default file system name is **/usr2**).
- Partition 9 is used as the second hard disk area for user login directories (the default file system name is **/usr3**).

Device partitions should fall on cylinder boundaries to obtain the best possible file system performance. For a root device, boot and swap partitions (partitions 7 and 1) are special in this regard. The number of blocks assigned to the boot and swap partitions are collectively chosen to cause the next partition values to fall on cylinder boundaries. (The next partitions are normally used as file systems.) This approach eliminates wasted space that would result from the strict assignment of partition values based on a modulo cylinder size for each partition of the root device. The otherwise wasted space in the boot block is used in the swap area without degrading the system performance.

The following table describes typical 3B2 Computer controller/drive/section equipment configurations.

DEVICE	CONTROLLER	DRIVE	SECTION	DESCRIPTION
c0d0s6	0	0	6	Specifies the section 6 (s6) of the integral floppy disk drive (d0) connected to controller 0 (c0).
c1d0s6	1	0	6	Specifies the section 6 (s6) of the first integral hard disk drive (d0) connected to controller 1 (c1).
c1d1s6	1	1	6	Specifies the section 6 (s6) of the second integral hard disk drive (d1) connected to controller 1 (c1).
c3d0s6	3	0	6	Specifies the section 6 (s6) of the first cartridge tape drive (d0) connected to controller 3 (c3). This controller is a CTC card in slot 2.
c3d1s6	3	1	6	Specifies the section 6 (s6) of the second floppy disk (d1) connected to controller 3 (c3). This controller is a CTC card in slot 2.
c4d0s6	4	0	6	Specifies the section 6 (s6) of the first cartridge tape drive (d0) connected to controller 4 (c4). This controller is a CTC card in slot 3.

HARD DISK DEFAULT PARTITIONS

The following tables define the use, size, and number of information nodes for the 10-, 30-, and 72-megabyte hard disks. The rotational gap (Gap) and blocks per cylinder (CYL) are defined for each device. Default Volume Table Of Contents (VTOC) partitioning is used in these tables. The number of information nodes (i-nodes) is based on the default value used by the **/etc/mkfs** command. The number of i-nodes is calculated based on the number of logical blocks (1024-byte blocks) divided by 8 and then rounded down to a modulo 16 value. Therefore, the values shown in the following tables are the result of a Full Restore procedure in which default partitioning is specified. The controller/device/section identifiers are applicable to both the raw and block devices.

SINGLE HARD DISK DEFAULT PARTITIONING					
CONFIGURATION	**PARTITION**	**USE**	**SECTOR START**	**SIZE***	**I-NODES**
SEAGATE	c1d0s0	root	3600	8928	1116
10-Megabyte	c1d0s1	swap	100	3500	----
Hard Disk	c1d0s2	usr	12528	9360	1170
(Gap=9)	c1d0s6	entire disk	0	21888	----
(Cyl=72)	c1d0s7	boot	0	100	----
WREN	c1d0s0	root	6120	12510	1552
30-Megabyte	c1d0s1	swap	100	6020	----
Hard Disk	c1d0s2	usr	18630	43920	5490
(Gap=9)	c1d0s6	entire disk	0	62550	----
(Cyl=90)	c1d0s7	boot	0	100	----
FUJITSU	c1d0s0	root	10296	12672	1584
72-Megabyte	c1d0s1	swap	100	10196	----
Hard Disks	c1d0s2	usr	22968	125928	15741
(Gap=9)	c1d0s6	entire disk	0	148896	----
(CYL=198)	c1d0s7	boot	0	100	----
MICROPOLIS	c1d0s0	root	10224	12672	1584
72-Megabyte	c1d0s1	swap	100	10124	----
Hard Disks	c1d0s2	usr	22896	124272	15534
(Gap=9)	c1d0s6	entire disk	0	147168	----
(CYL=144)	c1d0s7	boot	0	100	----
PRIAM	c1d0s0	root	10296	12672	1584
72-Megabyte	c1d0s1	swap	100	10196	----
Hard Disk	c1c0s2	usr	22968	126126	15766
(Gap=9)	c1d0s6	entire disk	0	149094	----
(CYL=198)	c1d0s7	boot	0	100	----
WREN II	c1d0s0	root	10206	12636	1580
72-Megabyte	c1d0s1	swap	100	10106	----
Hard Disk	c1d0s2	usr	22842	126684	15836
(Gap=9)	c1d0s6	entire disk	0	149526	----
(CYL=162)	c1d0s7	boot	0	100	----

* Size is reported in 512-byte blocks

DUAL HARD DISK DEFAULT PARTITIONING					
CONFIGURATION	**PARTITION**	**USE**	**SECTOR START**	**SIZE***	**I-NODES**
SEAGATE	c1d0s0	root	3600	8928	1116
10-Megabyte	c1d0s1	swap	100	3500	----
Hard Disk	c1d0s6	entire disk	0	21888	----
(Gap=9)	c1d0s7	boot	0	100	----
(CYL=72)	c1d0s8	usr2	12528	9360	1170
	c1d1s2	usr	72	21816	2727
	c1d1s6	entire disk	0	21888	----
	c1d1s7	boot	0	72	----
WREN	c1d0s0	root	6120	12510	1564
30-Megabyte	c1d0s1	swap	100	6020	----
Hard Disk	c1d0s6	entire disk	0	62550	----
(Gap=9)	c1d0s7	boot	0	100	----
(CYL=90)	c1d0s8	usr2	18630	43920	5490
	c1d1s2	usr	90	62460	7808
	c1d1s6	entire disk	0	62550	----
	c1d1s7	boot	0	90	----
FUJITSU	c1d0s0	root	10296	12672	1584
72-Megabyte	c1d0s1	swap	100	10196	----
Hard Disk	c1d0s6	entire disk	0	148896	----
(Gap=9)	c1d0s7	boot	0	100	----
(CYL=198)	c1d0s8	usr2	22968	125928	15741
	c1d1s2	usr	198	148698	18588
	c1d1s6	entire disk0		148896	----
	c1d1s7	boot	0	198	----

* Size is reported in 512-byte blocks

DUAL HARD DISK DEFAULT PARTITIONING					
MICROPOLIS	**c1d0s0**	**root**	**10224**	**12672**	**1584**
72-Megabyte	**c1d0s1**	**swap**	**100**		**10124**
Hard Disk	c1d0s6	entire disk	0	147168	----
(Gap=9)	c1d0s7	boot	0	100	----
(CYL=144)	c1d0s8	usr2	22896	124272	15534
	c1d1s2	usr	144	147024	18373
	c1d1s6	entire disk	0	147168	----
	c1d1s7	boot	0	144	----
PRIAM	c1d0s0	root	10296	12672	1584
72-Megabyte	c1d0s1	swap	100	10196	----
Hard Disk	c1d0s6	entire disk	0	149094	----
(Gap=9)	c1d0s7	boot	0	100	----
(CYL=198)	c1d0s8	usr2	22968	126126	15766
	c1d1s2	usr	198	148896	18812
	c1d1s6	entire disk	0	149094	----
	c1d1s7	boot	0	198	----
WREN II	c1d0s0	root	10206	12636	1580
72-Megabyte	c1d0s1	swap	100	10106	----
Hard Disk	c1d0s6	entire disk	0	149526	----
(Gap=9)	c1d0s7	boot	0	100	----
(CYL=162)	c1d0s8	usr2	22842	126684	15836
	c1d1s2	usr	162	149364	18671
	c1d1s6	entire disk	0	149526	----
	c1d1s7	boot	0	162	----

* Size is reported in 512-byte blocks

CARTRIDGE TAPE PARTITIONS

The following table defines the partition, use, size, and number of blocks for the various controller (c), drive (d), and section (s) identifiers for the 23-megabyte cartridge tape. These identifiers are applicable to both the raw and block devices. Note that Volume Table Of Contents (VTOC) partitioning is fixed for the cartridge tape by the tape formatting process. The rotational gap (Gap) and blocks per cylinder (CYL) are defined for the device. The controller number (?) depends on the slot in which the card is equipped.

CARTRIDGE TAPE DEFAULT PARTITIONING

CONFIGURATION	PARTITION	USE	SECTOR START	SIZE*
CIPHER	c?d0s0	root	5272	8928
23-Megabyte	c?d0s1	swap	126	5146
Cartridge	c?d0s2	usr	14200	31341
Tape	c?d0s3	usr	2	45539
(Gap=1)	c?d0s6	entire tape	0	45541
(CYL=31)	c?d0s7	boot	0	126

* Size is reported in 512-byte blocks.

FLOPPY DISK PARTITIONS

The following table defines the floppy disk partitions in terms of use, starting sector, and total number of blocks for the various controller (c), drive (d), and section (s) identifiers for the floppy disk. These identifiers are applicable to both the raw and block devices. Note that Volume Table Of Contents (VTOC) partitioning is not applicable to the floppy disk drive. The raw and block device partitions for the entire floppy disk (partition 6) are linked to **/dev/rSA/diskette1** and **/dev/SA/diskette1**, respectively. The use of these names when specifying the entire floppy disk is preferred over the use of the controller, drive, and section identifiers to avoid accidentally writing to a different device or partition.

FLOPPY DISK DRIVE (blocks per cylinder = 18) (rotational gap = 1)			
DISK PARTITION	**USE**	**SECTOR START**	**TOTAL SECTORS***
c0d0s0	root	378 (21*18)	1044
c0d0s1	usr	452 (34*18)	810
c0d0s2	usr	810 (45*18)	612
c0d0s3	usr	1008 (56*18)	414
c0d0s4	usr	2106 (67*18)	216
c0d0s5	usr	1 (1*18)	1404
c0d0s6	entire disk	0	1422
c0d0s7	boot	0	18

* Blocks (sectors) are reported in 512-byte blocks.

Appendix B

RUN LEVELS

RUN LEVEL	DESCRIPTION
0	Powerdown state.
1, s, or S	Single-user mode is used to install/remove software utilities, run file system backups/restorals, and to check file systems. Only the **root** file system is mounted on entering the single-user mode. It is necessary to mount the **/usr** file system when doing a software installation/removal or file system backup/restoral.
2	Multi-user mode is the normal operating mode for the system. The **root** (/) file system and file systems identified in the **/etc/fstab** are mounted in this mode. When the system is powered up it is put in this mode.

RUN STATE	DESCRIPTION
3	User defined run state.
4	User defined run state.
5	Firmware mode is used to display the equipped device table (**edt**), run diagnostics (**dgmon**), make a floppy key (**newkey**), change the firmware password (**passwd**), dump the system image to floppy disk (**sysdump**), and display the system generic version (**version**). The firmware mode is also used for the execution of the DEbug MONitor (DEMON). DEMON is used for the testing hardware and firmware.
6	Return to firmware and reboot the operating system. The system comes up in multi-user mode (run-level 2).
a, b, or c	Pseudo run-states that are normally processed by the **telinit** command. These entries in **inittab** are not run-levels; **init** cannot enter run-level **a**, **b**, or **c**. A process started by an **a**, **b**, or **c** continues to run when **init** changes states. A request to start the execution of a process associated with **a**, **b**, or **c** does not change the current run-level.

Appendix C

ERROR MESSAGES

PAGE

Appendix C

ERROR MESSAGES

GENERAL

Any time a problem occurs during powerup or during normal operation of your AT&T 3B2 Computer, one or more error messages are displayed on the console terminal. Most of the time the computer corrects the trouble and normal operation continues. Occasionally, some action must be performed by the user. If an error message occurs, refer to one of the following tables for the required action.

- **UNIX** SYSTEM ERROR MESSAGES
- **UNIX** SYSTEM CALLS ERROR MESSAGES
- FIRMWARE ERROR MESSAGES
- DIAGNOSTIC MONITOR PROGRAM ERROR MESSAGES
- EDT COMPLETION ERROR MESSAGES

- BOOT ERROR MESSAGES
- PUMP ERROR MESSAGES.

If one of these classes of error messages is displayed but is not in the tables, record the error message and contact your service representative for assistance.

Note: Error Messages from the network interface portion of the **UNIX** System kernel are not included in the tables.

UNIX SYSTEM ERROR MESSAGES

General

UNIX System error messages are divided into three severity classes: **NOTICE**, **WARNING**, and **PANIC**. When an error message is displayed, its severity class is displayed as the first part of the message. The following **UNIX** System error message tables are divided into severity classes. A description of each severity class is given with each table.

NOTICE Messages

NOTICE error messages provide information on the system status. These messages can sometimes help you to anticipate problems before they occur. These error messages are defined alphabetically in the following table.

UNIX SYSTEM ERROR MESSAGES
(Prefaced by NOTICE)

ERROR MESSAGE	DESCRIPTION/ACTION
bad block on floppy drive, slice #	An out of range block number was specified. Run **fsck** on the file system. Call your service representative if the problem persists.
bad block on integral hard disk drive #, partition #	An out of range block number was specified. Take the system to single-user mode and run **fsck** on the file system. Call your service representative if the problem persists.
bad count on floppy drive, slice #	Bad count in super block. Run **fsck** on the file system. Call your service representative if the problem persists.

UNIX SYSTEM ERROR MESSAGES (Prefaced by NOTICE)	
ERROR MESSAGE	**DESCRIPTION/ACTION**
bad count on integral hard disk drive #, partition #	Bad count in super block. Take the system to single-user mode and run **fsck** on the file system. Call your service representative if the problem persists.
Bad free count on floppy drive, slice #	Free list count is inconsistant. Run **fsck** on the file system. Call your service representative if the problem persists.
Bad free count on integral hard disk drive #, partition #	Free list count is inconsistant. Take the system to single-user mode and run **fsck** on the file system. Call your service representative if the problem persists.
bn = # er = #,#	Device error occurred during a read/write. Log that the error message occurred. Call your service representative if the problem persists.
Can't allocate message buffer.	System is out of message buffers. Either try again later, or send fewer inter-process communication messages.
CTC Access Error: Consult the Error Section of the 3B2 Computer Cartridge Tape Utilities Guide (error num=#)	Refer to the Cartridge Tape Utilities Guide for the specified error number.
/dev/swap doesn't match swapdev; changing it on fs	The system was booted for the first time from a new device. This is an advisory message and may be ignored.

UNIX SYSTEM ERROR MESSAGES (Prefaced by NOTICE)	
ERROR MESSAGE	**DESCRIPTION/ACTION**
File table overflow	The system file access table has overflowed. Reconfigure the system with larger NFILE tunable parameter if this is a frequent problem. Call your service representative if the problem continues.
Floppy Access Error: Consult the Error Message Section of the System Administration Utilities Guide	Make sure floppy disk is in the drive and the door is closed. Remove write protect tab or mount file system with -r (read only) option. The floppy disk requires reformatting, or its file system runs off the end of the disk. Reconfigure the floppy disk file system to be within the boundaries of the floppy disk.
iaddress > 2^24	Block number in an i-node was found to be greater than permissible when updating the file control block. Take the system to single-user mode and run **fsck** on the file systems. Call your service representative if you cannot resolve the problem.
no space on floppy drive, slice #	Copy less data to the partition or repartition if applicable. Cleanup affected file system indicated by the partition number.
no space on integral hard disk drive #, partition #	Copy less data to the partition or repartition if applicable. Cleanup affected file system indicated by the partition number.

UNIX SYSTEM ERROR MESSAGES (Prefaced by NOTICE)	
ERROR MESSAGE	**DESCRIPTION/ACTION**
Out of inodes on floppy drive, slice #	Copy less data to the partition or repartition if applicable. Cleanup affected file system indicated by the partition number.
Out of inodes on integral hard disk drive #, partition #	Copy less data to the partition or repartition if applicable. Cleanup affected file system indicated by the partition number.
proc on q	No action is necessary. The system tried to put a process on the run queue that was already on the queue. Call your service representative if the problem persists.
READ CLOCK -- TOO MANY TRIES	An attempt to read the system clock has failed 10 times. Reset the NVRAM using the floppy key (firmware password is defaulted to *mcp*). Check the battery. If error condition persist, there may be problems with the system clock hardware. Run diagnostics on the Time-of-Day Clock. If error condition persists, contact your service representative.
stray interrupt at #	Unknown interrupt signal has appeared. Message may be ignored. If it recurs repeatedly, contact your service representative.
swap space running out: needed # blocks	The system had to do more work than normal to swap out a process. If this occurs frequently, try to run fewer simultaneous processes. The 3B2 Computer may hang in these circumstances.

WARNING Messages

WARNING error messages indicate that the **UNIX** System may stop functioning if corrective action is not taken. These error messages are defined alphabetically in the following table.

UNIX SYSTEM ERROR MESSAGES
(Prefaced by WARNING)

ERROR MESSAGE	DESCRIPTION/ACTION
floppy disk: Bad address returned by VTOP	Log that the error message occurred. A bad address was passed to the driver ioctl, or a Virtual TO Physical (VTOP) address conversion failed.
floppy disk timeout: request flushed	Make sure floppy disk is in the drive and the door is closed. Contact your service representative if the problem recurs.
hard disk: Bad sanity word in VTOC on drive #	The volume table of contents (VTOC) is either bad or the wrong version. Restore the hard disk from the core floppy disks. Select the full restore option.
hard disk: Bad sanity word on drive #	Hard disk defect table is bad. Call your service representative.
hard disk: cannot access sector: #, track: #, cylinder: #, on drv #	Log that the error message occurred. No further action is required for this message. The **hdelogger** will identify any problems. Other messages requiring action may be output if the **hdelogger** logs errors. Refer to Chapter 5, "BAD BLOCK HANDLING FEATURE," for additional information.

UNIX SYSTEM ERROR MESSAGES (Prefaced by WARNING)	
ERROR MESSAGE	**DESCRIPTION/ACTION**
hard disk: Cannot read defect map on drive #	Log that the error message occurred. The hard disk defect map is bad. Call your service representative to resolve the problem.
hard disk: cannot read sector 0 on drive #	Log that the error message occurred. The hard disk defect table is bad. Call your service representative to resolve the problem.
hard disk: Cannot read the VTOC on drive #	Restore the hard disk from the core floppy disks. Select the full restore option. Call your service representative if your cannot resolve the problem.
hard disk: cannot recal drive #	Log that the error message occurred. Hardware problem is indicated. Call your service representative to resolve the problem.
hard disk: Drive # is in the 1.0 layout. It can not be used until the conversion is made to the current layout.	Restore the hard disk from the core floppy disks. Select the upgrade option.
hard disk: Drive # not equipped	Log that the error message occurred. Hardware problem is indicated. Reboot the system. If after reboot the problem still exists, call your service representative.
hard disk: drive # out of service	Log that the error message occurred. Hardware problem is indicated. Reboot the system. If after reboot the problem still exists, call your service representative.

UNIX SYSTEM ERROR MESSAGES (Prefaced by WARNING)	
ERROR MESSAGE	**DESCRIPTION/ACTION**
hard disk: partition # on drive # is marked read only	Trying to write to a read only partition. Check your command/program or mount the partition as read/write.
hard disk: too little space allocated in driver for defect table on drive #	The current UNIX Operating System cannot work with the disk configuration. Call your service representative to resolve the problem.
HDE queue full, following report not logged	Log that the error message occurred. Save the error message output and manually add the reports to the disk error log. Refer to Chapter 5, "BAD BLOCK HANDLING FEATURE," for additional information. Call your service representative if condition persists.
hdeeqd: major(ddev) = # (>=cdevcnt)	Log that the error message occurred. The **hdelogger** found a bad block and logged it. Refer to Chapter 5, "BAD BLOCK HANDLING FEATURE," for additional information.
inode table overflow	If recurring problem, reconfigure the system with a bigger information-node (i-node) table (NINODE).
mfree map overflow #. Lost # items at #	If recurring problem, reconfigure the system with a larger core map size (CMAPSIZ).
No swap space for exec args	Run fewer simultaneous processes or repartition the disk with increased swap space.

UNIX SYSTEM ERROR MESSAGES (Prefaced by WARNING)	
ERROR MESSAGE	**DESCRIPTION/ACTION**
out of swap space: needed # blocks	A process was left in memory because there was no room to swap it out. It will be swapped out if room becomes available. More room can be made by killing some processes. Run fewer simultaneous processes to avoid this problem. The 3B2 Computer may hang in these circumstances.
out of text	If recurring problem, reconfigure the system with increased text table limits (NTEXT).
PORTS: EXPRESS QUEUE OVERLOAD: One entry lost	Log that the error message occurred. The PORTS board may be insane. Reboot the system. Contact your service representative if the problem recurs.
PORTS: FAULT - opcode= #, board #, subdev = #, bytecnt = #, buff address = #	An invalid PORTS opcode was encountered. The PORTS board may be insane. "Pump" the associated PORTS board. If problem still exists, reboot the system. Contact your service representative if the problem recurs.
PORTS: QFAULT - opcode= #, board #, subdev = #, bytecnt = #, buff address = #	The PORTS job queue is invalid. The PORTS board may be insane. "Pump" the associated PORTS board. If problem still exists, reboot the system. Contact your service representative if the problem recurs.
PORTS: SYSGEN failure on board #	Log that the error message occurred. Reboot the system. Contact your service representative if the problem recurs.
PORTS: timeout on drain board (#), port (#)	Log that the error message occurred. Reboot the system. Contact your service representative if the problem recurs.

UNIX SYSTEM ERROR MESSAGES (Prefaced by WARNING)	
ERROR MESSAGE	**DESCRIPTION/ACTION**
PORTS: unknown completion code: #	Log that the error message occurred. Possible hardware problem. Reboot the system. Contact your service representative if the problem recurs.
PORTS: Unknown pump command: #	Log that the error message occurred. Reboot the system. Contact your service representative if the problem recurs.
too few HDE equipped disk slots Bad block handling skipped for maj/min = #/#	Log that the error message occurred. Reboot the system. Contact your service representative if the problem recurs.
unreadable CRC hard disk error: maj/min = #/# block = #	Log that the error message occurred. Reboot the system. The **hdelogger** logged a disk access error. Refer to Chapter 5, "BAD BLOCK HANDLING FEATURE," for additional information.
. . . on bad dev #(8)	Log that the error message occurred. A file system problem is indicated. Reboot the system. Contact your service representative if the problem recurs.

PANIC Messages

PANIC error messages indicate a problem severe enough that the **UNIX** Operating System must stop. The cause is usually a hardware problem or a minor problem in the kernel software. Some file systems may be corrupted, but the **UNIX** System checks for this when it is restarted. As with most sophisticated computer systems, "PANICs" will occasionally occur, and should not cause much concern. If a particular PANIC error message occurs repeatedly (or predictably), you should contact your service representative. These error messages are defined alphabetically in the following table.

UNIX SYSTEM ERROR MESSAGES
(Prefaced by PANIC)

ERROR MESSAGE	DESCRIPTION/ACTION
blkdev	The system description file may be incorrect. After the panic completes, take a crash dump if desired. Reboot the UNIX System. If panics occur frequently, contact your service representative.
cannot expand TEXT with swap	A request for text growth was rejected since text cannot be expanded. After the panic completes, take a crash dump if desired. Reboot the UNIX System. If panics occur frequently, contact your service representative.
cannot mount root	The root file system was corrupted or nonexistent when trying to boot. After the panic completes, take a crash dump if desired. Reboot the UNIX System. If reboot fails, then do a partial restore from the core floppy disks. If panics occur frequently, contact your service representative.

UNIX SYSTEM ERROR MESSAGES (Prefaced by PANIC)	
ERROR MESSAGE	**DESCRIPTION/ACTION**
data size error in swapin	The size of the swapped-in process data section is not the same size that was swapped-out. After the panic completes, take a crash dump if desired. Reboot the UNIX System. If panics occur frequently, contact your service representative.
devtab	The list header for the chain of buffers attached to the block-type device cannot be found. After the panic completes, take a crash dump if desired. Reboot the UNIX System. If panics occur frequently, contact your service representative.
floppy disk: Bad address returned from VTOP	A Virtual TO Physical (VTOP) address conversion failed. After the panic completes, take a crash dump if desired. Reboot the UNIX System. If panics occur frequently, contact your service representative.
hard disk: Bad address returned from VTOP	A Virtual TO Physical (VTOP) address conversion failed. After the panic completes, take a crash dump if desired. Reboot the UNIX System. If panics occur frequently, contact your service representative.
i/o error in swap	An access error occurred on the swap device. After the panic completes, take a crash dump if desired. Reboot the UNIX System. If panics occur frequently, contact your service representative.

UNIX SYSTEM ERROR MESSAGES (Prefaced by PANIC)	
ERROR MESSAGE	**DESCRIPTION/ACTION**
Illegal SIT counter selected	An illegal Sanity Interval Timer (SIT) counter was selected. After the panic completes, take a crash dump if desired. Reboot the UNIX System. If panics occur frequently, contact your service representative.
KERNEL BUS TIMEOUT	A bus request by the system was not fulfilled within the allotted time. After the panic completes, take a crash dump if desired. Reboot the UNIX System. If panics occur frequently, contact your service representative.
KERNEL DATA ALIGNMENT ERROR	The system software attempted to execute an instruction using an odd number as a pointer to a 16-bit quantity or a number that is not a multiple of 4 as a pointer to a 32-bit quantity. After the panic completes, take a crash dump if desired. Reboot the UNIX System. If panics occur frequently, contact your service representative.
KERNEL MMU FAULT . . .	The Memory Management Unit (MMU) acknowledged a fault during the execution of an instruction while in kernel mode. After the panic completes, take a crash dump if desired. Reboot the UNIX System. If panics occur frequently, contact your service representative.
KERNEL MMU FAULT #	A bus request by the system was not fulfilled within the allotted time. After the panic completes, take a crash dump if desired. Reboot the UNIX System. If panics occur frequently, contact your service representative.

UNIX SYSTEM ERROR MESSAGES (Prefaced by PANIC)	
ERROR MESSAGE	**DESCRIPTION/ACTION**
KERNEL MODE . . . FAULT	The processor unexpectedly registered the error given by . . . After the panic completes, take a crash dump if desired. Reboot the UNIX System. If panics occur frequently, contact your service representative.
KERNEL MODE FAULT, FT=#, ISC=#	The processor unexpectedly registered an error, identified by Fault Type (FT) and Internal State Code (ISC). After the panic completes, take a crash dump if desired. Reboot the UNIX System. If panics occur frequently, contact your service representative.
kernel process stack exception	A stack reference caused a memory fault. After the panic completes, take a crash dump if desired. Reboot the UNIX System. If panics occur frequently, contact your service representative.
no fs	No file system or no super-block was found. After the panic completes, take a crash dump if desired. Reboot the UNIX System. If panics occur frequently, contact your service representative.
no imt	No indirect mount point was found in the mount table. After the panic completes, take a crash dump if desired. Reboot the UNIX System. If panics occur frequently, contact your service representative.

UNIX SYSTEM ERROR MESSAGES (Prefaced by PANIC)	
ERROR MESSAGE	**DESCRIPTION/ACTION**
no procs	A process table entry was not found during a fork when an entry is available. After the panic completes, take a crash dump if desired. Reboot the UNIX System. If panics occur frequently, contact your service representative.
not a valid root	The root file system magic number is incorrect or the root device is improperly specified. After the panic completes, take a crash dump if desired. Reboot the UNIX System. If panics occur frequently, contact your service representative.
procdup() problem	An inconsistency has occurred between the parent process and the child process text size. After the panic completes, take a crash dump if desired. Reboot the UNIX System. If panics occur frequently, contact your service representative.
process exception, user = 0x#	Accessing a process control area caused a memory fault. After the panic completes, take a crash dump if desired. Reboot the UNIX System. If panics occur frequently, contact your service representative.
shmslp: swap # size # count # valid #	Log that the error message occurred. Reboot the system. Contact your service representative if the problem recurs.

UNIX SYSTEM ERROR MESSAGES (Prefaced by PANIC)	
ERROR MESSAGE	**DESCRIPTION/ACTION**
swapin lost text	The shared text table entry pointer is zero or an attempt was made to link to a text owner process. After the panic completes, take a crash dump if desired. Reboot the UNIX System. If panics occur frequently, contact your service representative.
SYSTEM ALIGNMENT ERROR INTERRUPT	The system software attempted to execute an instruction using an odd number as a pointer to a 16-bit quantity or a number that is not a multiple of 4 as a pointer to a 32-bit quantity. After the panic completes, take a crash dump if desired. Reboot the UNIX System. If panics occur frequently, contact your service representative.
SYSTEM BUS TIME OUT INTERRUPT	A bus request by the system was not fulfilled within the allotted time. After the panic completes, take a crash dump if desired. Reboot the UNIX System. If panics occur frequently, contact your service representative.
SYSTEM PARITY ERROR INTERRUPT SYSTEM PARITY ERROR INTERRUPT (in trap)	A memory parity error occurred. If this occurs repeatedly, a hardware problem exists. After the panic completes, take a crash dump if desired. Reboot the UNIX System. If panics occur frequently, contact your service representative.

UNIX SYSTEM ERROR MESSAGES (Prefaced by PANIC)	
ERROR MESSAGE	**DESCRIPTION/ACTION**
text size error in swapin	The size of the swapped-in process text is not the same size as the swapped-out text. After the panic completes, take a crash dump if desired. Reboot the UNIX System. If panics occur frequently, contact your service representative.
Timeout table overflow	The queue for timeout requests has overflowed while attempting to add another entry. After the panic completes, take a crash dump if desired. Reboot the UNIX System. If panics occur frequently, contact your service representative.
total size error in swapin	The computed size of the entire process swapped-in is not the same as that swapped-out. After the panic completes, take a crash dump if desired. Reboot the UNIX System. If panics occur frequently, contact your service representative.
trap recursion	A trap occurred from within the system trap handler. After the panic completes, take a crash dump if desired. Reboot the UNIX System. If panics occur frequently, contact your service representative.
unknown level in cmn_err (level=#, msg= . . .)	The common error software was invoked to process an error, but given an invalid severity level. This problem is secondary; the original problem is given by the msg. After the panic completes, take a crash dump if desired. Reboot the UNIX System. If panics occur frequently, contact your service representative.

UNIX SYSTEM ERROR MESSAGES (Prefaced by PANIC)	
ERROR MESSAGE	**DESCRIPTION/ACTION**
xalloc lost text	A pointer to the process text table points to a bad address. After the panic completes, take a crash dump if desired. Reboot the UNIX System. If panics occur frequently, contact your service representative.
xswap() current process 0x#	The swap-out process has been called with a process table address pointing to its own entry. After the panic completes, take a crash dump if desired. Reboot the UNIX System. If panics occur frequently, contact your service representative.
xswap error	An illegal operation was passed to the xswap function. After the panic completes, take a crash dump if desired. Reboot the UNIX System. If panics occur frequently, contact your service representative.

UNIX SYSTEM CALLS ERROR MESSAGES

A system call that is unsuccessful returns the calling process an impossible value. This impossible value is almost always a -1. When a system call is successful, the calling process is returned a value of 0. Anytime a system call is unsuccessful, an external variable called *errno* is set to one of the numbers in the following table.

When a process is returned a -1 value, the *errno* variable will contain the number corresponding to the reason of the failure. The *errno* variable is only valid immediately after a system call failure because it is not cleared on successful system calls. These error numbers are defined in the */usr/include/sys/errno.h* header file.

UNIX SYSTEM CALLS ERROR MESSAGES		
ERROR		DESCRIPTION
NUMBER	NAME	
1	EPERM	**Not Owner** Typically this error indicates an attempt to modify a file in some way forbidden except to its owner or super-user. It is also returned for attempts by ordinary users to do things allowed only to the super-user.
2	ENOENT	**No such file or directory** This error occurs when a file name is specified and the file should exist but does not, or when one of the directories in a path name does not exist.
3	ESRCH	**No such process** No process can be found corresponding to that specified by *pid* in *kill* or *ptrace*.
4	EINTR	**Interrupted system call** An asynchronous signal (such as interrupt or quit), which the user has elected to catch, occurred during a system call. If execution is resumed after processing the signal, it will appear as if the interrupted system call returned this error condition.
5	EIO	**I/O error** Some physical I/O error. This error may in some cases occur on a call following the one to which it actually applies.
6	ENXIO	**No such device or address** I/O on a special file refers to a subdevice which does not exist, or beyond the limits of the device. It may also occur when, for example, a tape drive is not on-line or no disk pack is loaded on a drive.
7	E2BIG	**Arg list too long** An argument list longer than 5,120 bytes is presented to a member of the *exec* family.

UNIX SYSTEM CALLS ERROR MESSAGES		
ERROR		DESCRIPTION
NUMBER	NAME	
8	ENOEXEC	**Exec format error** A request is made to execute a file which, although it has the appropriate permissions, does not start with a valid magic number (see *a.out*(4)).
9	EBADF	**Bad file number** Either a file descriptor refers to no open file, or a read (respectively write) request is made to a file which is open only for writing (respectively reading).
10	ECHILD	**No child processes** A *wait* was executed by a process that had no existing or unwaited-for child processes.
11	EAGAIN	**No more processes** A *fork* failed because the system's process table is full or the user is not allowed to create any more processes.
12	ENOMEM	**Not enough space** During an *exec*, *brk*, or *sbrk*, a program asks for more space than the system is able to supply. This is not a temporary condition; the maximum space size is a system parameter. The error may also occur if the arrangement of text, data, and stack segments requires too many segmentation registers, or if there is not enough swap space during a *fork*.
13	EACCES	**Permission denied** An attempt was made to access a file in a way forbidden by the protection system.
14	EFAULT	**Bad address** The system encountered a hardware fault in attempting to use an argument of a system call.
15	ENOTBLK	**Block device required** A non-block file was mentioned where a block device was required, e.g., in *mount*.

UNIX SYSTEM CALLS ERROR MESSAGES		
ERROR		DESCRIPTION
NUMBER	NAME	
16	EBUSY	**Mount device busy** An attempt to mount a device that was already mounted or an attempt was made to dismount a device on which there is an active file (open file, current directory, mounted-on file, active text segment). It will also occur if an attempt is made to enable accounting when it is already enabled.
17	EEXIST	**File exists** An existing file was mentioned in an inappropriate context, e.g., *link*.
18	EXDEV	**Cross-device link** A link to a file on another device was attempted.
19	ENODEV	**No such device** An attempt was made to apply an inappropriate system call to a device; e.g., read a write-only device.
20	ENOTDIR	**Not a directory** A non-directory was specified where a directory is required, for example in a path prefix or as an argument to *chdir*(2).
21	EISDIR	**Is a directory** An attempt to write on a directory.
22	EINVAL	**Invalid argument** Some invalid argument (e.g., dismounting a non-mounted device; mentioning an undefined signal in *signal*, or *kill*; reading or writing a file for which *lseek* has generated a negative pointer). Also set by the math functions described in the (3M) manual pages of the *Programmer Reference Manual*.

UNIX SYSTEM CALLS ERROR MESSAGES		
ERROR		DESCRIPTION
NUMBER	NAME	
23	ENFILE	**File table overflow** The system's table of open files is full, and temporarily no more *opens* can be accepted.
24	EMFILE	**Too many open files** No process may have more than 20 file descriptors open at a time.
25	ENOTTY	**Not a typewriter**
26	ETXTBSY	**Text file busy** An attempt to execute a pure-procedure program which is currently open for writing (or reading). Also an attempt to open for writing a pure-procedure program that is being executed.
27	EFBIG	**File too large** The size of a file exceeded the maximum file size (1,082,201,088 bytes) or ULIMIT; see *ulimit*(2).
28	ENOSPC	**No space left on device** During a *write* to an ordinary file, there is no free space left on the device.
29	ESPIPE	**Illegal seek** An *lseek* was issued to a pipe.
30	EROFS	**Read-only file system** An attempt to modify a file or directory was made on a device mounted read-only.
31	EMLINK	**Too many links** An attempt to make more than the maximum number of links (1000) to a file.

UNIX SYSTEM CALLS ERROR MESSAGES		
ERROR		DESCRIPTION
NUMBER	NAME	
32	EPIPE	**Broken pipe** A write on a pipe for which there is no process to read the data. This condition normally generates a signal; the error is returned if the signal is ignored.
33	EDOM	**Math argument** The argument of a function in the math package (3M) is out of the domain of the function.
34	ERANGE	**Result too large** The value of a function in the math package (3M) is not representable within machine precision.
35	ENOMSG	**No message of desired type** An attempt was made to receive a message of a type that does not exist on the spcified message queue; See *msgop*(2).
36	EIDRM	**Identifier removed** This error is returned to processes that resume execution due to the removal of an identifier from the file system's name space (see *msgctl*(2), *semctl*(2), and *shmctl*(2)).
45	EDEADLK	**Deadlock** A deadlock situation was detected and avoided.

DIAGNOSTIC MONITOR PROGRAM ERROR MESSAGES

The DiaGnostic MONitor (DGMON) program provides the user the ability to execute test phases on the 3B2 Computer. If a problem occurs while using the diagnostic monitor program, an error message is displayed on the console terminal. These error messages are prefaced by **DIAGNOSTIC MONITOR ERROR #**. These error messages are defined numerically in the following table.

DIAGNOSTIC MONITOR PROGRAM ERROR MESSAGES

NUMBER	MESSAGE	DESCRIPTION/ACTION
1-00	FILE SYSTEM IS INACCESSIBLE CONTROL WILL RETURN TO MAINTENANCE CONTROL PROGRAM.	Retry request. If it fails again, there is a possible problem with the protected disk cylinder where diagnostics reside.
1-01	UNKNOWN ID CODE (dev code) FOR DEVICE IN SLOT (slot #) NO DIAGNOSTIC TESTS RUN FOR THIS SLOT. CHECK EDT.	Retry request. If it fails again, the device is not recognized because installation is incomplete or the device reports a bad ID code.
1-02	CANNOT FIND FILE: (file name) DIAGNOSTIC REQUEST ABORTED.	Retry request.
1-03	CANNOT LOAD FILE: (file name) DIAGNOSTIC REQUEST ABORTED.	Retry request.
1-04	UNEXPECTED DIAGNOSTIC EXCEPTION. DIAGNOSTIC REQUEST ABORTED.	Retry request.
1-05	UNEXPECTED DIAGNOSTIC INTERRUPT. DIAGNOSTIC REQUEST ABORTED.	Retry request.

DIAGNOSTIC MONITOR PROGRAM ERROR MESSAGES		
NUMBER	**MESSAGE**	**DESCRIPTION/ACTION**
1-06	NON-EXISTENT UNIT: (device name) THE EQUIPPED UNIT TYPES ARE: (list of device names)	Retry request.
1-07	INVALID UNIT NUMBER FOR (device name), THE EQUIPPED UNITS ARE: (list of device names) RETRY REQUEST.	Retry request.
1-08	(echo of input string) UNRECOGNIZABLE DIAGNOSTIC REQUEST. CHECK REQUEST SYNTAX AND RE-ENTER.	Retry request.
1-09	INVALID REPEAT VALUE RE-ENTER REQUEST USING VALUE BETWEEN 1 AND 65536	Retry request.
1-10	INVALID PHASE(S) REQUESTED. CHECK REQUESTED PHASE TABLE AND RETRY.	Retry request.
1-11	REDUNDANT DIAGNOSTIC REQUEST OPTION. RE-ENTER REQUEST	Retry request.
1-12	SOAK AND UCL ARE INCOMPATIBLE DIAGNOSTIC OPTIONS. RE-ENTER REQUEST, OMITTING ONE.	Retry request.

DIAGNOSTIC MONITOR PROGRAM ERROR MESSAGES		
NUMBER	**MESSAGE**	**DESCRIPTION/ACTION**
1-13	UNIT OR UNIT TYPE NEEDED FOR PHASE OPTION REQUEST. RE-ENTER REQUEST.	Retry request.
1-14	USE UNIT TYPE ONLY FOR PHASE DISPLAY REQUEST. RE-ENTER REQUEST.	Retry request.

EQUIPPED DEVICE TABLE COMPLETION ERROR MESSAGES

The **filledt** program is a disk-based routine that runs at the firmware level. The **filledt** program completes the system equipped device table. These error messages are output if trouble occurs while **filledt** is executing. All errors except 1-08 and 1-09 are suppressed during autoboot. When manually booting the system, no errors are suppressed. Each error message is prefaced by **EDT COMPLETION ERROR** #. The EDT completion error messages are defined numerically in the following table.

EDT COMPLETION (FILLEDT) ERROR MESSAGES

NUMBER	MESSAGE	DESCRIPTION/ACTION
1-00	FILE SYSTEM IS INACCESSIBLE. CONTROL WILL RETURN TO MAINTENANCE CONTROL PROGRAM.	Retry request. If it fails again, there is a possible problem with the protected disk cylinder where diagnostics reside.
1-01	ERROR OCCURRED DURING SYSTEM CONFIGURATION. CONSOLE LOCATION PROCEEDING. CHECK EDT.	Check equipped device table, device entry garbled.
1-02	CANNOT FIND FILE: (file name) REQUEST ABORTED.	Retry request. Contact your service representative.
1-03	CANNOT LOAD FILE: (file name) REQUEST ABORTED.	Retry request. Contact your service representative.
1-04	UNEXPECTED EXCEPTION REQUEST ABORTED.	Retry request. Contact your service representative.

EDT COMPLETION (FILLEDT) ERROR MESSAGES		
NUMBER	**MESSAGE**	**DESCRIPTION/ACTION**
1-05	UNEXPECTED INTERRUPT REQUEST ABORTED.	Retry request. Contact your service representative.
1-06	SYSGEN FAILED FOR (device name) IN SLOT (slot #) EQUIPPED DEVICE TABLE COMPLETION WILL CONTINUE. CHECK EDT.	The peripheral device is not responding to configuration requests. Retry request. Contact your service representative.
1-07	DSD FAILED FOR (device name) IN SLOT (slot #) EQUIPPED DEVICE TABLE COMPLETION WILL CONTINUE. CHECK EDT.	The peripheral device is not responding to configuration requests. Retry request. Contact your service representative.
1-08	UNKNOWN ID CODE (id code) IN SLOT (slot #) EQUIPPED DEVICE TABLE COMPLETION WILL CONTINUE. CHECK EDT.	Device installation is incomplete or the device has failed. Retry request. Contact your service representative if installation is not in process.
1-09	UNKNOWN SUBDEVICE ID CODE FOR DEVICE (device name) IN SLOT (slot #) EQUIPPED DEVICE TABLE COMPLETION WILL CONTINUE. CHECK EDT.	Device installation is incomplete or the device has failed. Retry request. Contact your service representative if installation is not in process.
1-10	EDT EXCEEDS ALLOCATED SPACE AND CANNOT BE COMPLETED. REDUCE SYSTEM CONFIGURATION.	Remove some devices. Retry request.

FIRMWARE ERROR MESSAGES

The firmware mode is the state the 3B2 Computer must be in for the user to interface with several software programs. If a problem occurs while in this state, a firmware error message is displayed on the console terminal. These error messages are prefaced by **FW ERROR #**. These error messages are defined numerically in the following table.

FIRMWARE ERROR MESSAGES		
NUMBER	**MESSAGE**	**DESCRIPTION/ACTION**
1-01	NVRAM SANITY FAILURE	Try system power-up again. If same message appears, check the battery.
1-02	DISK SANITY FAILURE	Contact your service representative.
1-03	UNEXPECTED FAULT	Contact your service representative.
1-04	UNEXPECTED INTERRUPT	Contact your service representative.
1-05	SELF-CONFIGURATION FAILURE UNEXPECTED INTERRUPT	Contact your service representative.
1-06	BOOT FAILURE	If floppy boot, ensure that correct floppy is in the drive. This message may also be a result of incomplete diagnostic execution. If so, reboot the system. If the same message appears, contact your service representative.

FIRMWARE ERROR MESSAGES		
NUMBER	**MESSAGE**	**DESCRIPTION/ACTION**
1-07	FLOPPY KEY CREATE FAILURE	Ensure formatted floppy disk is in the drive when "go" is entered.
1-08	MEMORY TEST FAILURE	A failure occurred during pattern tests of the first 256K bytes of the system board memory. Power-cycle the machine to repeat the tests. Contact your service representative if the failure recurs.
1-09	DISK FORMAT NOT COMPATIBLE WITH SYSTEM	Message will appear during upgrade to Release 2.0. A 1.0 disk format was detected by firmware. Proceed with the Release 2.0 upgrade procedure. If an upgrade is not in progress, contact your service representative.

FIRMWARE ERROR MESSAGES		
NUMBER	**MESSAGE**	**DESCRIPTION/ACTION**
—	id # CRC error at disk address X if CRC error at disk address X	The # is a decimal number (0 or 1). The hard disk is **id**; the integral floppy disk is **if**. The **X** is an 8-character hexadecimal word specifying the physical cylinder number high (pcnh), the physical cylinder number low (pcnl), the physical head number (phn), and the physical sector number (psn). Refer to Chapter 5, "BAD BLOCK HANDLING FEATURE," for complete information. If the system will boot (run UNIX Operating System), add the identified defect to the defect map using the **hdeadd** and **hdefix** command. If you cannot resolve the problem, call your service representative.
—	PERIPHERAL I/O READ(WRITE) ERROR AT BLOCK #, SUBDEVICE #, SLOT #	Read/write failure detected on a hard disk connected through an XDC. Refer to Chapter 5, "BAD BLOCK HANDLING FEATURE," for complete information. If the system will boot (run UNIX Operating System), add the identified defect to the defect map using the **hdeadd** and **hdefix** command. If you cannot resolve the problem, call your service representative.

BOOT ERROR MESSAGES

Boot firmware provides the user the ability to execute a number of disk resident programs. These programs include the diagnostic monitor, the **UNIX** Operating System, and the utilities. If a problem occurs while attempting to execute one of these programs, an error message is displayed at the console terminal.

Boot PANIC message results in a second message being printed and self-configuration entering an endless loop. The only escape from this loop is to reset the machine. All boot error messages are listed below with a short description and corrective action. In all cases, contact your service representative if you can not resolve the problem.

BOOT ERROR MESSAGES (Prefaced by PANIC)	
ERROR MESSAGE	**DESCRIPTION/ACTION**
PANIC: <name>	Symbol <name> could not be resolved. Determine where symbol <name> should be defined, recompile and reboot.
PANIC: cannot chdir(/)	Cannot change directory to root (/). Action depends upon previously printed message.
PANIC: cannot mount root	Cannot mount the root file system. Action depends upon previous message printed.
PANIC: error_action() failed	Action to be taken on error is undefined. Indicates self-configuration has been corrupted.
PANIC: file table overflow	Exceeded maximum number of open files allowed (20). Indicates self-configuration has been corrupted.
PANIC: flexname too long	One of the object files being loaded contains a symbol which is more than 256 characters in length. This is a flexname size limit imposed by self-configuration. Either shorten symbol name or change flexname size allowed by self-configuration.
PANIC: Illegal error action	Recovery action to be taken is undefined. Indicates self-configuration has been corrupted.
PANIC: inode table overflow	Exceeded maximum number of inode table entries (23). Indicates self-configuration has been corrupted.

BOOT ERROR MESSAGES (Prefaced by PANIC)	
ERROR MESSAGE	**DESCRIPTION/ACTION**
PANIC: inode locked	Requested inode already in use. Indicates self-configuration has been corrupted.
PANIC: MAXCNTL exceeded	Maximum number of controllers allowed per device (16) exceeded. Indicates an illegal hardware configuration. Correct hardware configuration then reboot.
PANIC: memory overflow	Self-configuration would be overwritten. The text plus data size of modules being loaded exceeds the amount of memory available (from beginning of mainstore to start of self-configuration text). Decrease number of modules being loaded, or move origin of self-configuration.
PANIC: No memory for EXCLUDE list	Unable to allocate memory for EXCLUDE list. Decrease number of modules being loaded, or move origin of self-configuration.
PANIC: no memory for FILE buffer	Unable to allocate memory for file header. Decrease number of modules being loaded, or move origin of self-configuration.
PANIC: No memory for io_init[], io_start[] or pwr_clr[]	Unable to allocate memory for kernel data structures. Decrease number of modules being loaded, or move origin of self-configuration.
PANIC: No memory for loadmap	Unable to allocate memory for kernel loadmap. Decrease number of modules being loaded, or move origin of self-configuration.

BOOT ERROR MESSAGES (Prefaced by PANIC)	
ERROR MESSAGE	**DESCRIPTION/ACTION**
PANIC: No memory for parameter checking	Unable to allocate memory for parameter checking. Decrease number of modules being loaded, or move origin of self-configuration.
PANIC: No memory for sys3bconfig structure	Unable to allocate memory for sys3bconfig structure. Move origin of self-configuration.
PANIC: No memory for Xreloc	Unable to allocate memory for relocation entry. Decrease number of modules being loaded, or move origin of self-configuration.
PANIC: No memory for Xsymbol	Unable to allocate memory for internal symbol table. Decrease number of modules being loaded, or move origin of self-configuration.
PANIC: out of free blocks	All available buffers (25) in use. Indicates self-configuration has been corrupted.
PANIC: textSIZE	Actual text size of all object modules to be loaded plus size of interrupt routines not equal to calculated size. Indicates self-configuration has been corrupted.
PANIC: Undefined expression element	Expression element unknown. A master file has an invalid expression. See master(4) for valid expression element syntax. Check all master files for expression syntax.
PANIC: Unknown error number	Error code undefined. Indicates self-configuration has been corrupted.

BOOT ERROR MESSAGES (Prefaced by PANIC)	
ERROR MESSAGE	**DESCRIPTION/ACTION**
PANIC: Unsupported relocation type	An object file has an invalid relocation type. The valid types are R_DIR32, and R_DIR32S. Recompile with correct sgs.

The following boot error messages are warning and error messages printed by self-configuration.

BOOT ERROR MESSAGES	
ERROR MESSAGE	**DESCRIPTION/ACTION**
<driver>: character string initializer truncated	This is a warning message. Attempting to initialize a string variable for <driver> but found string too long for variable. Variable initialized to truncated string. This may cause unusual side effects.
<driver>: dependent driver <name> is EXCLUDED	<Driver> has dependencies upon driver <name>, but driver <name> is marked to be excluded. <Driver> will not be loaded. Remove driver <name> from the EXCLUDE line of the system file, or add <driver> to the EXCLUDE line. If <driver> is added to EXCLUDE line, remove it from INCLUDE line if there.
<driver>: dependent driver <name> not available	<Driver> has dependencies upon driver <name>, but the object file for driver <name> is not found in boot directory. <Driver> will not be loaded. Place mkbooted object file for driver <name> in **/boot** directory, or add <driver> to EXCLUDE line of system file. If <driver> is on INCLUDE line, remove it from that line.
<driver>: device not equipped for dependent driver <name>	<Driver> has dependencies upon driver <name>, but hardware not equipped for driver <name>. <Driver> will not be loaded. Either add hardware for driver <name>, or add <driver> to EXCLUDE line of the system file. If <driver> is added to the EXCLUDE line, then remove it from the INCLUDE if it exists there.

BOOT ERROR MESSAGES	
ERROR MESSAGE	**DESCRIPTION/ACTION**
<driver>: illegal character string initialization: zero assumed	This is a warning message. Attempting to initialize a string variable for <driver> but found illegal character string. Variable initialized to zero. This may cause unusual side effects.
<driver>: routine <name>: unknown id: RNULL assumed	The routine <name> which is referenced by <driver> could not be found. It is resolved to rnull(), which may have unusual side effects.
<file> does not exist	Unable to find <file>.
<file>: not object file and not ascii text file	If <file> refers to the boot program, then this message means that <file> is not an object file. If <file> refers to a system file, that <file> was found to be non-ascii.
<name>: already allocated	Self-configuration attempted to allocate space for variable <name> but found it was already allocated. The variable is from a master file variable list. This warning message may produce unusual side effects.
<name>: already defined	Self-configuration expects to define the symbol <name>, but found it already defined. This is a warning message, but could result in unusual side effects.
<name>: Bad file number	Invalid file descriptor.
<name>: data initializer #C(expression) unknown; zero assumed	The master file for module <name> contains a reference to a master file entry "number of controllers" (expression) which cannot be found. Correct master file <name>, mkboot driver <name>, and then reboot.

BOOT ERROR MESSAGES	
ERROR MESSAGE	**DESCRIPTION/ACTION**
<name>: data initializer #D(expression) unknown; zero assumed	The master file for module <name> contains a reference to a master file entry "number of devices" (expression) which cannot be found. Correct master file <name>, mkboot driver <name>, and then reboot.
<name>: data initializer #M(expression) unknown; zero assumed	The master file for module <name> contains a reference to a module major number (expression) which cannot be found. Correct master file <name>, mkboot driver <name>, and then reboot.
<name>: data initializer &expression cannot be resolved	The master file for module <name> contains a reference to the address of a symbol (expression) which cannot be found. Correct master file <name>, mkboot driver <name>, and then reboot.
<name>: data initializer #expression unknown; zero assumed	The master file for module <name> contains a reference to the size of a symbol (expression) which cannot be found. Correct master file <name>, mkboot driver <name>, and then reboot.
<name>: data initializer expression unknown; zero assumed	The master file for module <name> contains a reference to a parameter (expression) which cannot be found. Correct master file <name>, mkboot driver <name>, and then reboot.
<name>: File too large	The size of a file exceeded the maximum file size.

BOOT ERROR MESSAGES	
ERROR MESSAGE	**DESCRIPTION/ACTION**
<name>: flagged as ONCE only; #C set to 1	The master file for driver <name> is marked as only specify once, but the number of controllers is greater than one. The number of controllers is set to one. Correct master file <name>, mkboot driver <name>, and then reboot.
<name>: I/O error	Some physical input/output error has occurred while reading file <name>.
<name>: Invalid argument	Some invalid argument was passed.
<name>: invalid object file	Object file <name> not valid for this machine.
<name>: No drivers	Unable to find valid loadable driver object files in boot directory <name>. Check path of boot directory, and that the boot directory contains mkbooted driver object files. Then reboot.
<name>: no section headers	The file header of boot module <name> indicates number of sections in object is zero. Recompile <name>, mkboot <name>, and then reboot.
<name>: No such device	An attempt was made to apply an inappropriate system call to a device; e.g. read a write-only device.
<name>: No such file or directory	The file <name> does not exist. This error occurs when a file is specified and the file should exist but doesn't, or when one of the directories in a path name does not exist.
<name>: no symbols	No symbols were found in the file specified for boot.

BOOT ERROR MESSAGES	
ERROR MESSAGE	**DESCRIPTION/ACTION**
<name>: Not a directory	<Name> is not a directory. A non-directory was specified where a directory is required, for example in a path prefix.
<name>: not MAC32 magic	Object module <name> contains an incorrect magic number. Recompile <name> with correct sgs, mkboot <name>, and then reboot.
<name>: previously allocated	Self-configuration attempted to allocate space for variable <name> but found it had been previously allocated by self-configuration. The variable is from a master file variable list. This warning message may produce unusual side effects.
<name>: previously defined	Self-configuration expects to define the symbol <name>, but found it already defined. This is a warning message, but could result in unusual side effects.
<name>: required driver is EXCLUDED	The driver <name> is marked as being required in its master file, but is EXCLUDED in the system file. Unknown results may occur. Illegal to EXCLUDE a required driver. Remove <name> from the EXCLUDE line of the system file and reboot.
<name>: routine <name>() not found	The routine <name> was not found in the boot program <name>. The value of routine <name> is set to zero. While this is only a warning message it may have unusual side effects.

BOOT ERROR MESSAGES	
ERROR MESSAGE	**DESCRIPTION/ACTION**
<name>: Special device cannot be used	File <name> is a special device (character, block, or FIFO).
<name>: Too many open files	No process may have more than 20 file descriptors open at a time.
<name>: truncated read	Read of file <name> failed.
<name>: truncated string table	While reading string table of file <name>, end-of-file was encountered prematurely.
Driver <name>: major number greater than 127	A master file for software driver <name> contains a major number greater than 127. Correct major number in master file <name>. Then reboot.
Driver <name>: missing section .text, .data or .bss	Driver object module <name> is missing .text, .data, or .bss section header. Recompile driver <name>. Then reboot.
Driver <name>: not a valid object file	Driver <name> contains bad magic number. Recompile driver <name>. Then reboot.
Driver <name>: not processed by mkboot(1M)	Driver object file <name> was not processed by mkboot command. Run mkboot on driver file <name>. Then reboot.
EXCLUDE: <name>; driver is INCLUDED	Driver <name> to be excluded is also to be included. Remove <name> from one or the other in the system file. Then reboot.
External symbol <name> is undefined: set to zero	The external symbol <name> cannot be resolved. Its value is set to zero.

BOOT ERROR MESSAGES	
ERROR MESSAGE	**DESCRIPTION/ACTION**
I/O ERROR id= block= count= jstat= erstat= xerstat=	A disk read job failed. The message contains the buffer header pointer, disk block number, byte count, job status returned by disk subsystem, and failing status codes returned by the disk subsystem. Diagnose disk subsystem and repair. Then reboot.
INCLUDE: <name>; device not equipped	Hardware not equipped for driver <name> to be included. This is a warning and the driver <name> will not be loaded. Either add hardware for device <name>, add EXCLUDE statement to the system file for driver <name>, or remove driver from boot directory. Then reboot.
INCLUDE: <name>; driver is EXCLUDED	Driver <name> appears on both the INCLUDE and EXCLUDE lines of the system file. Remove <name> from one or the other in the system file. Then reboot.
INCLUDE: <name>; driver not found	Driver <name> is marked to be included, but unable to find its object file in the boot directory. If driver <name> is to be included, then run mkboot on <name> object file and reboot. If driver <name> was not to be included, then remove it from the INCLUDE line in the system file. Then reboot.
Device <name> previously configured at board code <n>	Device <name> has been moved. It was previously located in slot <n>. This is a warning message indicating a change in configuration was detected.

BOOT ERROR MESSAGES	
ERROR MESSAGE	**DESCRIPTION/ACTION**
Device <name> (board code <n>) not configured	Device <name> located in slot <n> was not installed at the time the absolute boot image was created. Therefore, it will not be usable when this absolute boot image is used.
Driver not found for <name> device (board code <n>)	A driver for device <name> was not found in the boot directory. The device is located in slot <n>. This is a warning message. Add driver for device <name> and reboot.
No drivers available, absolute BOOT program must be used	The driver linked-list could not be built. Therefore a self-configuration cannot be done and a absolute boot program must be used. Boot the absolute boot program.
No memory for driver linked-list	Unable to allocate memory to build the driver linked-list.
No memory for kernel optional header	Unable to allocate memory to build the kernel optional header.
No memory for driver symbol table processing	Unable to allocate memory to process a driver symbol table.
No memory for symbol table	Unable to allocate memory for kernel symbol table processing.
No section loaded at virtual address zero: interrupt vectors are inaccessible	This warning message indicates that nothing was loaded at virtual address zero. Therefore, the gate tables and interrupt vectors will not be accessible in virtual addressing modes.

BOOT ERROR MESSAGES	
ERROR MESSAGE	**DESCRIPTION/ACTION**
Section <name>(<file>) loaded below MAINSTORE address	The section <name> from file <file> has an origin below the value of MAINSTORE.
Section <name>(<file>) loaded beyond end of MAINSTORE	The section <name> from file <file> has an origin beyond the end of physical memory.
Section <name>(<file>) overlaps boot program	The section <name> from file <file> overlaid portions of self-configuration.
Section <name>(<file>) overlaps <name>(<file>)	The section <name> from file <file> overlays the section <name> from file <file>.
VTOC does not exist or is damaged.	Cannot find the volume table of contents on disk.
VTOC read failed.	Unable to read the volume table of contents on disk.

The following boot error messages are from self-configuration checking for multiply defined parameters in the master files for all drivers.

BOOT ERROR MESSAGES	
ERROR MESSAGE	**DESCRIPTION/ACTION**
Parameter <name> multiply defined	A parameter <name> is found to be defined more than once with different values. If the conflict cannot be resolved, then the parameter value will be set to zero.
<driver>: <name> = <n>	Driver <driver> defines parameter <name> to be <n>.
<driver>: <name> = <n> (<driver> EXCLUDED, parameter ignored)	Driver <driver> defines parameter <name> to be <n>, but driver <driver> is to be EXCLUDED. Therefore, this definition will be ignored.
<driver>: <name> = <n> (set to zero)	Unable to resolve conflict of parameter <name>, its value is being set to zero. Driver <driver> defines parameter <name> to be <n>. Correct parameter definitions of the master files listed. Execute **mkboot** drivers for any changed master files, and then reboot.
<driver>: <name> = <string>	Driver <name> defines parameter <name> to be <string>.
<driver>: <name> = <string> (<driver> EXCLUDED, parameter ignored)	Driver <driver> defines parameter <name> to be <string>, but driver <driver> is to be EXCLUDED. Therefore, its definition will be ignored.

BOOT ERROR MESSAGES	
ERROR MESSAGE	**DESCRIPTION/ACTION**
<driver>: <name> = <string> (set to zero)	Unable to resolve conflict of parameter <name>, its value is being set to zero. Driver <driver> defines parameter <name> to be <string>. Correct parameter definitions of the master files listed. Mkboot drivers for any master files which were changed, and then reboot.

The following boot error messages are generated by self-configuration when parsing the system file.

BOOT ERROR MESSAGES	
ERROR MESSAGE	**DESCRIPTION/ACTION**
System: line <n>: cannot boot directory	The file specified for booting is a directory.
System: line <n>: cannot boot special device	The file specified on line <n> of the system file, the file to boot, is a special device and cannot be loaded.
System: line <n>: cannot boot special file	The file specified on line <n> of the system file, the file to boot, is a special file and cannot be loaded.
System: line <n>: count must be numeric	The count value on system file line <n> is not a numeric value.
System: line <n>: file not BLOCK or CHAR special	The device on line <n> of the system file is not a block or character special device.
System: line <n>: line too long	Line <n> of the system file is longer than 256 characters.
System: line <n>: major/minor must be numeric	The major and/or minor numbers on line <n> of the system file are not numeric values.
System: line <n>: must be numeric	The numbers on line <n> are not numeric values.

BOOT ERROR MESSAGES	
ERROR MESSAGE	**DESCRIPTION/ACTION**
System: line <n>: no such file	The file specified on line <n> of the system file, the file to boot, cannot be accessed.
System: line <n>: path too long	The path of the program to boot on system file line <n> contains more than 100 characters.
System: line <n>: syntax error	A syntax error was found on line <n> of the system file.

The following boot error and warning messages result from input to system file prompts by self-configuration.

BOOT ERROR MESSAGES	
ERROR MESSAGE	**DESCRIPTION/ACTION**
System: cannot boot directory	The file specified for booting is a directory.
System: cannot boot special device	The file specified for boot is a special device file.
System: cannot boot special file	The file specified for boot is a special file.
System: count must be numeric	The count value is not numeric.
System: file not BLOCK or CHAR special	The device specified is not a character or block special device file.
System: line too long	The input line contains more than 256 characters.
System: major/minor must be numeric	The major and/or minor numbers are not numeric.
System: must be numeric	The value entered is not numeric.
System: no such file	The file specified for boot cannot be accessed.
System: path too long	The path for the boot file contains more than 100 characters.
System: syntax error	The input line contains syntax errors.

PUMP ERROR MESSAGES

Pump is a user-level command that downloads firmware to feature cards mounted in the 3B2 Computer backplane slots during the power-up sequence. Pump error messages appear on the console terminal when a phase in the autopump sequence fails. Although these errors are not fatal to the entire system, the affected card is not operational. Therefore, normal services provided by the device are not accessible.

PUMP ERROR MESSAGES	
ERROR MESSAGE (NOTES 1 and 2)	**ACTION**
Pump: "/dev/devname" returned a CIO FAULT during "phase"	Turn power off using power switch. After powerdown sequence has completed, turn power on again. If error occurs again, contact your service representative. The UNIX System may panic soon.
Pump: "/dev/devname" returned a CIO Invalid Queue Entry during "phase"	"
Pump: "/dev/devname" did not respond during "phase"	"
Pump: "/dev/devname" did not respond during "phase"	"
Pump: A timeout has occurred on "/dev/devname" during "phase"	"
Pump: There was no return for "dev/devname" during "phase"	"
Pump Error: 208-ioctl call	"

Note 1: The term "/dev/devname" refers to "/dev/ttyAB, /dev/NI, etc." where:
A = Feature Card slot on backplane
B = Port on Expansion Port Feature Card
NI = Network Interface Feature Card

Note 2: The term "phase" refers to one of the following phases:

Reset = Reset of the Feature Card so that pumping can occur
Download = Pumping to firmware of Feature Card
Sysgen = Initialization of Feature Card to known state
Force call to function = Calling the starting address of firmware that was downloaded.

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W