Issue 1
October 1984

AT&T 3B2 Computer Utilities

Select Code
305-408

Comcode
403778327

NOTE

Two UTILITIES binders are provided with each 3B2 Computer. One is intended for filing the *System Administration Utilities Guide*. The other is intended for filing all provided and optional Utilities Guides.

All the Utilities Guides come with tab separators and the pages are prepunched for easy filing in the binders.

The comment forms in this binder may be used for any comments concerning the 3B2 Computer Utilities Guides, Software Information Bulletins (SIB), or Option Manuals. When commenting on a document, please enter the document title and the six-digit select code on the comment form.

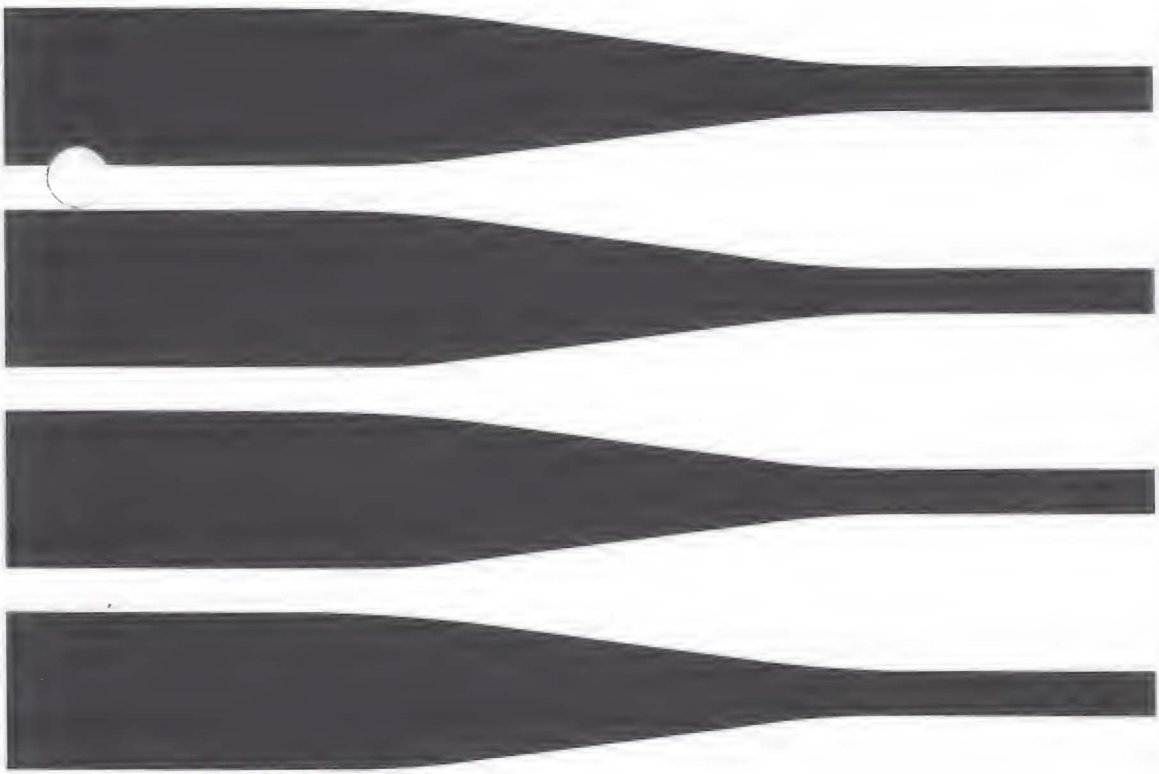
Additional copies of this binder may be ordered. Follow the ordering procedures specified in the *Documentation Catalog* provided with your 3B2 Computer.

Issue 1
November 1984

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

Select Code
305-422

Comcode
403778384



TRADEMARKS

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

- DEC, PDP, UNIBUS, and MASSBUS are trademarks of Digital Equipment Corporation.
- DIABLO is a registered trademark of Xerox Corporation.
- DOCUMENTER'S WORKBENCH is a trademark of AT&T Technologies.
- HP is a trademark of Hewlett-Packard, Inc.
- TEKTRONIX is a registered trademark of Tektronic, Inc.
- TELETYPE is a trademark of AT&T Teletype Corporation.
- UNIX is a trademark of AT&T Bell Laboratories.
- Versatec is a registered trademark of Versatec Corporation.

NOTICE

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

CONTENTS

Chapter 1

INTRODUCTION

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 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.

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 and certain other commands provided on the hard disk that are used for the administration of the system.

- Appendix A, "MANUAL PAGES," provides the UNIX* System V manual pages for the commands defined in Chapter 6, "SYSTEM ADMINISTRATION COMMANDS." Also provided are manual pages for other system administration-type commands, special files, and system maintenance procedures.

- Appendix B, "DISK PARTITIONS," shows the structure of the disk subsystem.

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

- Appendix D, "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 Bell Laboratories

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** can be 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 Appendix A, "MANUAL PAGES." Also provided in Appendix A are Section 7 (special files) and Section 8 (system maintenance procedures) manual pages. Refer to the *3B2 Computer Programmers 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.

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
- 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 Section 4 of the UNIX System V manual pages for additional information on the formats of system files. These manual pages are in the *3B2 Computer Programmers Reference Manual.*

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 of the 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 are important in the administration of the 3B2 Computer.

- /etc/checklist
- /etc/fstab
- /etc/gettydefs
- /etc/group
- /etc/inittab
- /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/sulog
- /usr/lib/cron Directory

- /usr/lib/help/HELPLOG
- /usr/lib/spell/spellhist
- /usr/news Directory
- /usr/options Directory.

Each of these files is briefly described in this subchapter.

/etc/checklist

The **/etc/checklist** file is used to define a default list of file system devices to be checked for consistency by the **fsck**(1M) 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. Remember that with the exception of **root**, a file system must be unmounted to be checked. Therefore, the **checklist** file is a convenience for use when in the single-user mode of operation with only the **root** file system mounted. As the system is delivered, this 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/umountall** 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/c1d0s2 /usr
```

Figure 2-2. Typical /etc/fstab File

/etc/gettydefs

The **/etc/gettydefs** file contains information that is used by the **getty**(1M) command to set the speed and terminal settings for a 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 complete information.
Figure 2-3 shows a typical 3B2 Computer **/etc/gettydefs** file. Note that it is common practice to add the node 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

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

group name The first field defines the group name. The group name is from three to six characters. The first character is alphabetic. The rest of 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 can 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. The group identification field is a number between 0 and 60,000. 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-state 2 is multi-user mode. Appendix C, "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
pl:s1234:powerfail:/etc/led -f
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
l2:2:respawn:/etc/getty tty12 1200
l3:2:off:/etc/getty tty13 9600
l4:2:respawn:/usr/lib/uucp/uugetty -r -t 60 tty14 1200H
l5:2:off:/etc/getty tty15 9600
```

Figure 2-5. Typical /etc/inittab 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 three to six characters. The first character is alphabetic. The rest of 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 can 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 60,000. Group 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 60,000. 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 2-6 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 2-6. 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 modify this file to set options for the **root** login. For example, the following can be added to the **/etc/profile** for the **root** login to cause the erase character to backup and to set the TERM variable.

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

Figure 2-7 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
        echo "UNIX System V Release `uname -r` `uname -m` Version `uname -v`"
        uname -n
        echo 'Copyright (c) 1984 AT&T Technologies, Inc.\nAll Rights Reserved\n'

        # 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 2-7. 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 when **/etc/rc0** is run. 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 can be done by adding a file to the **/etc/shutdown.d** directory. Figure 2-8 shows a typical 3B2 Computer **/etc/rc0** file.

```
# "Run Commands" for init stat 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 2-8. 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 when **/etc/rc2** is run. The **/etc/rc.d** executable files include:

.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 **/unix** 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**.

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

uucp When basic networking is added to the system, the **uucp** file is added to this directory. The **uucp** file cleans up (deletes) uucp locks (LCK*), status (STST*), and temporary (TM*) files under the **/usr/spool/uucp** directory structure.

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 2-9 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 2-9. 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 on floppy disks. The following files are included.

except A list of the directories and files that *should not* be copied as part of a 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/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 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 2-10 shows a typical 3B2 Computer **/etc/shutdown** file.

```
# Sequence performed to change the init state 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 2-10. 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
```

Figure 2-10. 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 0"  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 2-10. 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
fi
case "${initstate}" in
s | S )
        . /etc/rc0
esac
/etc/init ${initstate}
```

Figure 2-10. 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 2-11 shows a typical **/etc/TIMEZONE** file.

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

TZ=*TTT#SSS*

The fields of the **TZ** variable are as follows.

TTT	The three-character abbreviation for the local time zone.
#	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	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 2-11. 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. Each time **login** is run this file is updated. 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 command line limits 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 cleanup 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/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 cleanup 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 command line limits 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 2-12 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 2-12. 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 cleanup 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 command line limits 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 2-13 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 2-13. 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 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 cleanup 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 command line limits 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 2-14 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 2-14. 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 *3B2 Computer Spell Utilities Guide* for information on adding words to the dictionary and cleaning up the **spellhist** file.

/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. In run-state 2 (multi-user mode), **/etc/cron** checks the **/usr/spool/cron/crontabs** files for changes once every minute. 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 through 59.
hour	The hour field is a one- or two-digit number in the range of 0 through 24.
day	The day field is the numerical day of the month in the range of 1 through 31.
month	The month field is the numerical month of the year in the range of 1 through 12.
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 indicates 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 2-15 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 tasks that need 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 2-15. Typical /usr/spool/cron/crontabs/root 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 2-16 shows a typical **/usr/options** directory. Not all possible utilities are listed (installed). The example shown in Figure 2-16 was taken from a system with a 32-megabyte hard disk. All of the utilities identified in Figure 2-16 cannot be loaded on a 10-megabyte hard disk at the same time. The full name of the utilities are contained in the */usr/options* files. The contents of the files identified in Figure 2-16 are shown in Figure 2-17.

```
-r-xr-xr-x   1 bin      bin            22 Sep  7 15:17 calc.name
-r-xr-xr-x   1 bin      bin            23 Sep 10 17:03 cc.name
-r--r--r--   1 bin      bin            34 Aug 10 15:30 crypt.name
-r-xr-xr-x   1 bin      bin            40 Aug 10 15:32 dfm.name
-r--r--r--   1 bin      bin            19 Aug 21 12:44 dmd.name
-r--r--r--   1 bin      bin            38 Aug 21 14:23 dmdapd.name
-r--r--r--   1 bin      bin            55 Aug 21 14:47 dmdtxt.name
-r-xr-xr-x   1 bin      bin            30 Apr 26 11:28 dwb.name
-r--r--r--   1 bin      bin            18 Aug 10 15:30 ed.name
-r-xr-xr-x   1 bin      bin            39 Aug  3 11:10 esg.name
-r-xr-xr-x   1 bin      bin            19 Aug 10 15:48 graph.name
-r-xr-xr-x   1 bin      bin            15 Aug 10 15:51 help.name
-r-xr-xr-x   1 bin      bin            38 Aug 10 15:53 ipc.name
-r--r--r--   1 bin      bin            22 Aug 11 01:34 kersrc.name
-r-xr-xr-x   1 bin      bin            32 Aug 10 15:55 lp.name
-r-xr-xr-x   1 bin      bin            35 Aug 10 15:57 perf.name
-r-xr-xr-x   1 bin      bin            30 Aug 10 15:58 sccs.name
-r-xr-xr-x   1 bin      bin            30 Aug 10 16:04 sgs.name
-r-xr-xr-x   1 bin      bin            28 Sep  7 15:17 shell.name
-r-xr-xr-x   1 bin      bin            16 Aug 10 16:07 spell.name
-r-xr-xr-x   1 bin      bin            32 Aug 10 16:08 sysadm.name
-r-xr-xr-x   1 bin      bin            27 Aug 10 16:10 term.name
-r--r--r--   1 bin      bin            31 Aug 10 16:11 terminf.name
-r-xr-xr-x   1 bin      bin            27 Sep  7 15:17 usrenv.name
-rwxr-xr-x   1 root     sys            27 Aug 10 15:18 uucp.name
```

Figure 2-16. Typical /usr/options Directory

```
Calculation Utilities
C Programming Language
Security Administration 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
Graphics Utilities
HELP Utilities
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
```

Figure 2-17. Typical /usr/options File Contents

Chapter 3

ADMINISTRATIVE TASKS

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 *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 Disks

- Running Diagnostics
- Setting Time-of-Day/Date
- Rebuilding File System Free List
- Monitoring Disk Space
- Copying/Moving Directories
- Installing and Removing Utilities Packages
- Determining System Status After Trouble
- File System Checking and Repair
- Reloading UNIX Operating System
- File System Backup
- File System Restoral from Backup
- System Reconfiguration
- Establishing/Changing the System and Node Names
- Security Administration
- Forgotten **root** Password Recovery
- Forgotten FIRMWARE Password Recovery.

Most of these tasks are done on an as required basis. Certain of these tasks should be performed on a routine (scheduled) basis.

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 on the machine at the time are "kicked-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 performed 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 shutdown 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 system administration tasks are performed 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 can be 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 can 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

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 is approximately 35 minutes (approximately 3.5 minutes per floppy). The steps in formatting floppy disks are as follows.

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

2. Execute the **fmtflop** command, specifying the verify option. The raw (character) device partition for the entire floppy disk must be specified. For the integral floppy disk use **/dev/rdiskette**.

3. Remove the floppy disk from the floppy disk drive when complete (floppy disk light is off).

The following shows the typical command line entries and system responses for formatting and verifying a floppy disk in the UNIX Operating System. 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/rdiskette<CR>
#
```

Note: The command has no output when successful

DUPLICATING FLOPPY DISKS

General

The contents of an existing floppy disk is copied to another formatted floppy disk by using the **dd** command. The contents of the floppy disk to be duplicated is first copied to a temporary file space. Either the character (**/dev/rdiskette**) or the block (**/dev/diskette**) 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 1422 blocks must be available (free) in the **root** file system to use **/tmp** as the temporary file space. Any file system can be used for the temporary file space. The **/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 sufficient free space is not available in the **root** file system.

Copy Procedure

The time required to duplicate a floppy disk is approximately 12 minutes (6 minutes for each transfer).

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/rdiskette**, any login will work. As the system is delivered, only **root** has read/write permission.)

2. Insert the source floppy disk into the integral floppy disk drive.

3. Enter the following command:

```
dd if=/dev/rdiskette of=/tmp/tmpfile
```

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

4. When the light on the floppy is off, remove the source floppy disk. Insert a formatted destination floppy disk into the integral 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**.

5. Enter the following command:

```
dd if=/tmp/tmpfile of=/dev/rdiskette
```

6. When the light 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.

```
# dd if=/dev/rdiskette of=/tmp/tmpfile<CR>
1422+0 blocks in
1422+0 blocks out

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

# dd if=/tmp/tmpfile of=/dev/rdiskette<CR>
1422+0 blocks in
1422+0 blocks out
#
```

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/rdiskette**) or the block (**/dev/diskette**) 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 is the number of bytes per track (9 times 512 bytes). Specifying a block size that matches the number of blocks per track 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.

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/rdiskette**, 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 integral floppy disk drive.

3. Enter the following command:

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

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

4. When the light 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 that has 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).

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.

```
mkfs /dev/diskette 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.

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 ten 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/diskette 1422:192 1 18<CR>
Mkfs: /dev/diskette?
(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/diskette rar rar2.0<CR>
Current fsname: , Current volname: , Blocks: 1422, Inodes: 192
FS Units: 1Kb, Date last mounted: Wed Aug 22 17:41:07 1984
NEW fsname = rar, NEW volname = rar2.0 -- DEL if wrong!!
# mkdir /rar<CR>
# mount /dev/diskette /rar<CR>
# mount<CR>
/ on /dev/dsk/cld0s0 read/write on Wed Aug 22 16:52:10 1984
/usr on /dev/dsk/cld0s2 read/write on Wed Aug 22 16:53:58 1984
/rar on /dev/diskette read/write on Wed Aug 22 17:48:07 1984
# umount /dev/diskette<CR>
# mount /dev/diskette /install<CR>
mount: warning! <rar> mounted as </insta>
# mount<CR>
/ on /dev/dsk/cld0s0 read/write on Wed Aug 22 16:52:10 1984
/usr on /dev/dsk/cld0s2 read/write on Wed Aug 22 16:53:58 1984
/install on /dev/diskette read/write on Wed Aug 22 17:50:46 1984
# umount /dev/diskette<CR>
#
```

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 assistance.

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 power-up 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 power-up 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" discussion 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

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

Continued

Continued from previous screen

```
        3B2 DIAGNDSTIC MDNITDR
DGMDN > h<CR>
        3B2
     DIAGNDSTIC
      CDMMANDS   DPTIONS                                    DESCRIPTION
      ==========  =======                                    ===========
         DGN     [DEVICE [DEVICE # ¦ REP=? ¦ PH=?-? ¦    DIAGNDSE DEVICES(S)
                                     UCL ¦ SOAK ]]

        H(ELP)   (NDNE)                                     PRINT HELP MENU

        L(IST)   DEVICE                                     LIST DEVICE PHASE TABLE

        Q(UIT)   (NONE)                                     EXIT DGMDN

        S(HDW)   (NDNE)                                     SHDW EDT

DGMDN > s<CR>

Current System Configuration

System Board memory size: 2 megabyte(s)

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

Press any key to continue<CR>


D1 - device name = PDRTS      ,occurrence = D, slot = 01, ID code = 0x03
     boot device = n, board width = single, work width = 2 byte(s),
     req Q size = DxD3, comp Q size =Dx23, console ability = y, pump file = y
     subdevices(s)

DONE

DGMDN > 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

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

UNIX System V Release 2.0 3B2 Version 1
unix
Copyright (c) 1984 AT&T Technologies, Inc.
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: 1.1

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  506 files 10092 blocks 2220 free
The system is coming up.  Please wait
AT&T 3B2 SYSTEM CONFIGURATION:

Memory size: 2048 Kilobytes
System Peripherals:

         SBD                FD5
                  HD30
         PORTS    PORTSThe 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 **1** 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 **1** command is used to display these tables. To display the diagnostic phases for a specific device, enter: **1***(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 ports board 1.

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 1
Runs all NORMAL phases once on ports board 1. 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 2 ucl
Runs all NORMAL phases once on ports board 2. 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=1-4
Runs phases 1 (CPU #2 Normal DGN), 2 (CPU #3 Normal DGN), 3 (CPU #4 Normal DGN), and 4 (Memory Management #1 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 ports board 1 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

While in the Diagnostic Monitor you can 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 with the **soak** or **ucl** option and a specific number of repetitions.

   > ***Note:*** The phase number for the individual diagnostic phases is found by using the **l** command.

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 fail, you may want to run those phases again with the **soak** or **ucl** option and a specific number of repetitions.

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, proceed to "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, proceed to "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

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

UNIX System V Release 2.0 3B2 Version 1
unix
Copyright (c) 1984 AT&T Technologies, Inc.
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: 1.1

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  506 files 10092 blocks 2220 free
The system is coming up.  Please wait.
AT&T 3B2 SYSTEM CONFIGURATION:

Memory size: 2048 Kilobytes
System Peripherals:

         SBD                FD5
                  HD30
         PORTS    PORTSThe system is ready.

# Console Login:
```

Sample Diagnostic Execution

The following examples are provided to show you what to expect when running diagnostics. The actual 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 13, 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=13<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 9, 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=9<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<CR>
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 19). 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.

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 -i1<CR>

Shutdown started.

     Series of messages
     ending with the following.

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

  /dev/dsk/c1d0s2
  File System: usr Volume: 1.1

  ** 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 *****
```

Continued

Continued from the previous screen

```
  /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 (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 3B2 Version 1
unix
Copyright (c) 1984 AT&T Technologies, Inc.
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: 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 (Ignored)
  ** Phase 6 - Salvage Free List
  507 files 10158 blocks 2154 free
The system coming up.  Please wait.
AT&T 3B2 SYSTEM CONFIGURATION:

Memory size: 2048 Kilobytes
System Peripherals:

         SBD                FD5
                 HD30
         PORTS      PORTSThe system is ready.

# Console Login: root<CR>
# Password: <password><CR>
#
```

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 1422 blocks, the entire contents of a floppy disk cannot be transferred to that file system. 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 32-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 cleanup the file system. Users should be informed of diminishing system resources and requested to clean up their directories. If after such a clean up, 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 approximately 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. Cleanup 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 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/diskette**). 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 *3B2 Computer Owner/Operator Manual*. Any special instructions for the installation or removal of software are documented in the Software Information Bulletins.

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 what 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 D.), 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 in running the UNIX Operating System to use this command. If the system can not 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 runs from firmware and does not depend on the integrity of a file 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.

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:   0x40041a9c

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)

#
```

System Dump (sysdump)

The **sysdump** program is a firmware program that writes the system image to a floppy disk. Three formatted floppy disks are needed to dump the contents of a 2-megabyte memory. Existing floppy disks can be used; however, the contents of the floppy disks are overwritten. The time required to take a system dump is approximately 2:15 minutes per floppy disk.

The following command line entries and system responses show how to execute **sysdump**.

```
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
```

FILE SYSTEM CHECKING AND REPAIR

The ability to check a file system and repair any damage is described in greater detail 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 floppy disks. These floppy disks are the **3B2 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. The partial restore overwrites only the core system files. 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. For a partial restore, only the input/output terminal configuration and the password file must be reconstructed. Since all files and directories ARE NOT copied to a Simple Administration backup, certain files must be restored as a separate task. For example, all changes made to the device files (**/dev** directory) must be reconstructed for the support of such options as Input/Output Expansion, Basic Networking, and Line Printer Spooling. Also, the access permissions of these files must be changed if they need to be different from what is provided by the standard software load.

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 of the steps, then reload the operating system. If you do not understand the reload procedure, contact your service representative for help.

Partial Restore Procedure

The "user" files are not affected by a partial restoral. Certain system files are overwritten. For example, the terminal configuration and password file 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. Depending on the reason for doing the partial restore, certain system files should be saved to make the job of restoring the system configuration easier. The following steps are for a PARTIAL restoral of the system.

1. If possible, save the following system files by copying them to another name.

```
cd /etc
cp passwd opasswd
cp inittab oinittab
cp system osystem
cd /etc/master.d
cp kernel okernel
cd /usr/lib/cron
cp at.allow oat.allow
cp cron.allow ocron.allow
cd /usr/spool/cron/crontabs
cp adm oadm
cp root oroot
cp sys osys
```

2. Change directory to root (/), and take the system to the firmware mode (run-level 5).

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

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

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

6. Follow the displayed instructions to remove and insert the **3B2 Core System** floppy disks. When the last **3B2 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.

8. Recover the applicable system files from the copies and reconfigure the system as necessary. Refer to the "SYSTEM RECONFIGURATION" discussion in this chapter to reconfigure the system.

The following command line entries and system responses show a typical partial restore procedure. The sample procedure starts by saving certain system files. The system is then shut down and a partial restore started. At the end of this sample procedure, the system must be set up and the applicable system files restored.

```
# cd /etc<CR>
# cp passwd opasswd<CR>
# cp inittab oinittab<CR>
# cp system osystem<CR>
# cd /etc/master.d<CR>
# cp kernel okernel<CR>
# cd /usr/lib/cron<CR>
# cp at.allow oat.allow<CR>
# cp cron.allow ocron.allow<CR>
# cd /usr/spool/cron/crontabs<CR>
# cp adm oadm<CR>
# cp root oroot<CR>
# cp sys osys<CR>
# cd<CR>
# shutdown -y -g0 -i5<CR>

Shutdown started.    Wed Sep 19 17:49:10 EDT 1984

Broadcast message from root (console) Wed Sep 19, 17:49:13...
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
```

Continued

Continued from the previous screen

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

Option Number   Slot     Name
-------------------------------------
       0          0     FD5
       1          0     HD30

Enter Load Oevice Optlon Number [1 (H030)]: 0<CR>


UNIX System V Release 2.0 3B2 Version 1
unix
Copyright (c) 1984 AT&T Technologies, Inc.
All Rights Reserved

3B2 UNIX Release 1.1 Installation

1  Full Restore
2  Partial Restore
3  Release 1.0 Upgrade

Enter "help" for additional information.

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

-- Partial Restore --

This will replace the core system files on the hard disk with those
originally distributed. Other files will not be 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: 1.1

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  526 files 10520 blocks 1792 free

  /dev/rdsk/c1d0s2
  File System: usr Volume: 1.1

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  4833 files 40044 blocks 3098 free

Installing the initial core system files. This should take no more than
seven minutes.
1236 blocks
1 blocks
                                                                Continued
```

Continued from the previous screen

```
The system is restarting itself from the hard disk. This should
take approximately thirty seconds. The installation procedure
will then continue automatically.


SELF-CHECK

UNIX System V Release 2.0 3B2 Version 1
unix
Copyright (c) 1984 AT&T Technologies, Inc.
All Rights Reserved


  /dev/dsk/c1d0s0
  File System: root Volume: 1.1

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  561 files 10456 blocks 1856 free

  /dev/dsk/c1d0s2
  File System: usr Volume: 1.1

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  4833 files 40044 blocks 3098 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
seven minutes.
1157 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
seven minutes.
1237 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
seven minutes.
1128 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
seven minutes.
934 blocks

You may now remove the last 3B2 Core System floppy.
55 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 about one minute.
```

Continued

```
Continued from the previous screen

SELF-CHECK

DIAGNOSTICS  PASSED


UNIX System V Release 2.0 3B2 Version 1
unix
Copyright (c) 1984 AT&T Technologies, Inc.
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/c1d0s2
  File System: usr Volume: 1.1

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  4836 files 40044 blocks 3102 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.

Console Login:
```

Full Restore Procedure

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.

1. Take the system to the firmware mode (run-level 5).

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

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

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

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

6. 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.

The following command line entries and system responses show a typical full restore procedure. The system is in the firmware mode at the start of the example.

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

Option Number   Slot     Name
-------------------------------------
       0          0     FD5
       1          0     HD30

Enter Load Oevice Option Number [1 (H030)]: 0<CR>


UNIX System V Release 2.0 3B2 Version 1
unix
Copyright (c) 1984 AT&T Technologies, Inc.
All Rights Reserved

3B2 UNIX Release 1.1 Installation

1  Full Restore
2  Partial Restore
3  Release 1.0 Upgrade

Enter "help" for additional information.

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

-- Full Restore --

This will destroy EVERYTHING on the hard disk and install a
3B2 Release 1.1 core UNIX system.

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

Continued

Continued from the previous screen

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

How many blocks for the "root" partition?
[ (range 8928 through 40490) quit help] (default 12510): 12510<CR>

How many blocks for the "swap" partition?
[ (range 3500 through 40490) quit help] (default 6020): 6020<CR>

How many blocks for the "usr" partition?
[ (range 9360 through 43830) quit help] (default 43830): 43830<CR>

Type "go" to proceed, "again" to start over [go again quit help ]: go<CR>

Setting up the initial hard disk as specified; this should take no more
than five minutes.

Installing the initial core system files. This should take no more than
seven minutes.
1237 blocks
1 blocks
```

Continued

Continued from the previous screen

```
The system is restarting itself from the hard disk. This should
take approximately thirty seconds. The installation procedure
will then continue automatically.


SELF-CHECK

UNIX System V Release 2.0 3B2 Version 1
unix
Copyright (c) 1984 AT&T Technologies, Inc.
All Rights Reserved


  /dev/dsk/cld0s0
  File System: root Volume: 1.1

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  146 files 1366 blocks 10946 free

  /dev/dsk/cld0s2
  File System: usr Volume: 1.1

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  2 files 2 blocks 43140 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
seven minutes.
1157 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
seven minutes.
1237 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
seven minutes.
1125 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
seven minutes.
933 blocks

You may now remove the last 3B2 Core System floppy.
56 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 about one minute.
```

Continued

```
Continued from the previous screen

SELF-CHECK

DIAGNOSTICS  PASSED


UNIX System V Release 2.0 3B2 Version 1
unix
Copyright (c) 19B4 AT&T Technologies, Inc.
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 unsable unless I tell you otherwise.
I will now start checking file systems.


  /dev/dsk/c1d0s2
  File System: usr Volume: 1.1

  ** Phase 1 - Check Blocks and Sizes
  ** Phase 2 - Check Pathnames
  ** Phase 3 - Check Connectivity
  ** Phase 4 - Check Reference Counts
  ** Phase 5 - Check Free List
  17B files 1456 blocks 416B6 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 lead you through
those things that should be done the "first time" the machine is used.

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 can not 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. Avoid the trap of building an extensive historical library.

Simple Administration procedures are provided to do "complete" file system backups (volume backup) and incremental file system backups to floppy disks. A Simple Administration procedure is provided for restoring backup data to the hard disk. File system restoral from backups is described later in this chapter.

The capability to copy selected directories and files to floppy disks is provided by using the **find** and **cpio** commands. The directories and files are read back to the hard disk by using the **cpio** command.

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.

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. The backup plan should be reevaluated as the use of the machine changes.

Volume Backup

General

The volume backup provided by the Simple Administration backup facilities copies CERTAIN directories and files of a specified mounted file system to floppy disks. Not all directories and files are copied by the Simple Administration backup mechanism. The contents of the **/etc/save.d/except** file specifies files and directories that ARE NOT copied as part of a backup. Also note that changes made to the **/dev** directory are not part of a Simple Administration backup. A volume backup must be done before an incremental backup can be done on the file system. A volume backup can be done at any time.

Estimating Number of Floppy Disks and Time Required for Backup

You must have enough formatted floppy disks on-hand to contain the directories and files being backed up. As a rule-of-thumb, the number of floppy disks needed for a volume backup is equal to the the number of data blocks used in the file system divided by 1300, rounded to a whole number. For example, a file system that occupies 13294 blocks requires 10 floppy disks for a volume backup. This is shown in the following formula.

$$Number\ of\ Disks = \frac{blocks\ used}{1300}$$

$$Number\ of\ Disks = \frac{13294}{1300} = 10$$

The time required to do a volume backup of a file system is approximately 6 minutes per floppy disk. The 13294 block file system is estimated to require 10 floppy disks and will thus take about one hour to backup.

System Volume Backup Procedure

The following steps are necessary to backup the **root** and **usr** file systems.

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

2. Determine the number of floppy disks required to backup each file system. Use the **du** command get the number of blocks used in each file system.

3. If enough formatted floppy disks are not available, obtain and format the required number of floppy disks.

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

5. Run **fsck** on the file systems to be backed-up.

6. Mount the **usr** file system using the following command. Also, mount the file system to be backed-up if it is not already mounted.

   ```
   mount /dev/dsk/c1d0s2 /usr
   ```

   Note that the **usr** files system contains the Simple Administration commands and, therefore, must be mounted to execute the **sysadm** commands.

7. Execute the Simple Administration backup command (**sysadm backup**) and follow the displayed instructions. Label each floppy disk used for the backup. Include a sequence number as part of the label (Part 1, Part 2, etc).

8. Unmount the **usr** file system and return the system to the normal operating configuration using the following commands. The root file system (**/dev/dsk/c1d0s0**) should be file system mounted before returning the system to the normal operating configuration.

```
umount /dev/dsk/c1d0s2
init 2
```

9. At your discretion, verify the backup floppy disks. See the "VERIFYING FLOPPY DISKS" procedure described in this chapter. Store the backup floppy disks in a safe place.

Sample File System Volume Backup

The following shows the typical command line entries and system responses for doing a volume backup of the **usr** file system. The system is in the single-user mode (run-level S) with the **usr** file system mounted at the start of the sample. The **usr** file system was checked (**fsck**) before mounting the file system.

```
# du -s /usr<CR>
13098    /usr
# sysadm backup<CR>

Running subcommand 'backup' from menu 'filemgmt',
FILE MANAGEMENT


Available file systems:
/ /usr ALL
Enter file system(s) you want to backup [q, ?]: /usr<CR>
Select complete or incremental backup [c, i, q, ?]: c<CR>
Print each file name as it is copied? [q,?] [y, n, q, ?]: n<CR>

Before inserting the first part into the drive, mark it as follows:

        Complete Backup of /usr,
        Sat. 09/08/84, 05:08:31 AM
                part 1

Insert the medium in the rdiskette drive.  Press <RETURN> when ready. [q] <CR>
.................................................
Reached end of medium on output.
Remove diskette.

Before inserting the next part into the drive, mark it:

        Complete Backup of /usr,
        Sat. 09/08/84, 05:08:31 AM
                part 2

Insert the diskette.  Press <RETURN> when ready. [q] <CR>
.................................................
.....................
Reached end of medium on output.
                    ¦
                    ¦
 Process continues until all data is copied
 to the floppy disks
                    ¦
                    ¦
Reached end of medium on output.
Remove diskette.

Before inserting the next part into the drive, mark it:

        Complete Backup of /usr,
        Sat. 09/0B/B4, 05:0B:31 AM
                part n

Insert the diskette.  Press <RETURN> when ready. [q] <CR>
.........................................
13089 blocks

Complete backup of  /usr  finished.
You may now remove the medium.
#
```

Incremental Backup

General

The incremental backup provided by the Simple Administration backup facilities copies most files that have changed since the last backup to one or more floppy disks. The contents of the **/etc/save.d/except** file specifies files and directories that ARE NOT copied as part of a backup. Also note that changes made to the **/dev** directory are not part of a Simple Administration backup. A volume backup must be done before an incremental backup can be done on the file system. The incremental file system backup is fast in comparison to the time required to do a volume backup. Typically, the time required to do an incremental backup is less than six minutes per file system.

A volume backup is the first backup that is done on a file system. Once a volume 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 volume backup is the focal point of YOUR system backup plan. 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 volume backups are done once each month, over 40 floppy disks could be required for backing-up the **/usr** file system.

- How often do you want to take the time to do a volume 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, volume backup can be more efficient than incremental backups for file systems that access (change) a large percentage of the data between system backups.

- 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 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 backup 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 systems to be backed-up.
4. Mount the **usr** file system using the following command.

```
mount /dev/dsk/c1d0s2 /usr
```

   Note that the **usr** files system contains the Simple Administration commands and, therefore, must be mounted to execute the **sysadm** commands. Also, mount the file system to be backed-up if it is not already mounted.

5. Execute the Simple Administration backup command (**sysadm backup**) and follow the displayed instructions. Label the floppy disk as indicated in the displayed instructions.
6. Unmount the **usr** file system and return the system to the normal operating configuration using the following commands. The root file system (**/dev/dsk/c1d0s0**) should be the only system mounted before returning the system to the normal operating configuration.

```
umount /dev/dsk/c1d0s2
init 2
```

7. At your discretion, verify the backup floppy disks. See the "VERIFYING FLOPPY DISKS" procedure. 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. The system is in the single-user mode (run-level S) with the **/usr** file system mounted at the start of the sample. The **/usr** file system was checked (**fsck**) before mounting the file system.

```
# sysadm backup<CR>

Running subcommand 'backup' from menu 'filemgmt',
FILE MANAGEMENT


Available file systems:
/ /usr ALL
Enter file system(s) you want to backup [q, ?]: /usr<CR>
Select complete or incremental backup [c, i, q, ?]: i<CR>
Print each file name as it is copied? [q, ?] [y, n, q, ?]: y<CR>

8efore inserting the diskette into the drive, mark it as follows:

        Incremental 8ackup of /usr,
        Sat. 09/08/84, 05:08:31 AM to
        Sat. 09/08/84, 07:04:21 AM"
                Part 1
Insert the medium in the rdiskette drive.  Press <RETURN> when ready. [q] <CR>
/usr/rar/305-323/chx.functions
83 blocks

Incremental backup of  /usr  has finished.
You may now remove the medium.
#
```

Selected Backup

General

Specific directories and files can be quickly saved on a single floppy disk by using the **find** and **cpio** commands. The amount of data copied to a single floppy disk can not exceed 1422 blocks (512 bytes per block). Before you copy the selected directory(ies) or file(s), check the number of blocks involved. This method of backup can be used to save the **/dev** directory.

Selected Backup Procedure

The following are the steps necessary to copy selected information to a floppy disk.

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 write permission.)

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

3. Check that the number of data blocks involved is less than 1422 blocks (**du** command).

4. Label a formatted floppy disk to indicate the directory or file. Insert a formatted floppy disk in the floppy disk drive.

5. Copy the selected data to the floppy disk using the **find** and **cpio** commands.

6. Verify that you can copy the data back to the system by using the **cpio** command. Copy the data to the **/tmp** directory.

Sample Selected Backup

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**(1) command and the **cpio**(1) command manual pages for an explanation of the various options.

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

The following shows the typical command line entries and system responses associated with copying the information from the integral floppy disk to the integral hard disk. The command used copies the data to the current directory. In this example, the data is copied to the **/tmp** directory. Refer to the **cpio** manual pages for a complete explanation of the various options. The contents of the **/tmp** and **/usr/rar/305-323** directories are listed to compare the source data with the copied data.

```
# cd /tmp<CR>
# cpio -icdumv < /dev/diskette<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 17 08:32 ch1.general
-rw-r--r--   1 rar      other       7819 Feb 17 08:32 ch2.install
-rw-r--r--   1 rar      other      18340 Feb 17 08:32 ch3.files
-rw-r--r--   1 rar      other      17627 Feb 17 08:32 ch4.advice
-rw-r--r--   1 rar      other      45725 Feb 17 08:32 ch5.fsck
-rw-r--r--   1 rar      other      19294 Feb 17 08:32 ch6.package
-rw-r--r--   1 rar      other      13392 Feb 17 08:32 chx.functions
-rw-r--r--   1 rar      other        751 Feb 17 08:32 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
#
```

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 following discussion is based on the use of the Simple Administration restore facilities (**sysadm restore**). The procedure for loading the contents of a "selected backup" floppy disk was described in that discussion. Briefly, change directory to the appropriate directory and execute a **cpio -icdumv** command to take input from the floppy disk drive. See the discussion on "Selected Backup."

File System Restore Procedure

The following steps are necessary to restore a portion or all of a file system from a Simple Administration prepared 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 **usr** file system using the following command.

   mount /dev/dsk/c1d0s2 /usr

   Note that the **usr** files system contains the Simple Administration commands and, therefore, must be mounted to execute the **sysadm** commands.

4. Execute the Simple Administration restore command (**sysadm restore**) and follow the displayed instructions. When restoring from a volume backup, all floppy disks of that series must be loaded. Even if you intend to restore only a single file, all floppy disks of a volume backup must be loaded in sequence. In all cases, the first floppy disk of a series must be the first floppy disk read by the restore process.

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

```
umount /dev/dsk/c1d0s2
init 2
```

6. Store the backup floppy disks in a safe place.

Sample File System Restoral

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

```
# sysadm restore<CR>

Running subcommand 'restore' from menu 'filemgmt',
FILE MANAGEMENT

Select:
        1. restore a single files
        2. restore a directories of files
        3. restore all files
        4. list all the files
Enter a number [q,?]: 1<CR>
Insert the medium in the rdiskette drive.  Press <RETURN> when ready. [q] <CR>
Enter full path name of file(s) to be restored [q, ?]:
         /usr/rar/305-323/chx.functions<CR>
Do you want to rename the file as it is copied in? [y, n, q]: y<CR>
WARNING:
        Be very careful when you rename a file.  Files incorrectly named
        by typing errors are difficult to find and repair.
        Remember that only the first 14 characters of each part of the
        file name (i.e. the characters between the "/"s) are significant.
You will be asked to rename each file in turn.  An empty response skips that
file.  An answer of "." (period) restores the file with its original name.
Rename </usr/rar/305-323/chx.functions>
/usr/rar/305-323/oldfile<CR>
83 blocks
Restoration complete.
You may now remove the medium from the rdiskette drive.
Select:
        1. restore a single files
        2. restore a directories of files
        3. restore all files
        4. list all the files
Enter a number [q,?]: q<CR>

#
```

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 can 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 in case 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

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 great deal 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 can be enhanced or degraded by changing tunable parameters. Raising or lowering a parameter too far can cause strange things to happen in the operating system. It is recommended that you copy the bootable*** */unix* ***to*** */oldunix* ***so that you will have a bootable operating system in case 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 discussed here unless stated in the definition. Caution should be used when changing these variables since any small change directly affects 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 configurations 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 deficiency.

Figure 3-1 shows the starting tunable parameter values for systems equipped with different ranges of Random Access Memory (RAM). The model is based on a system supporting from two to six users. The parameters shown in Figure 3-1 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 package. The message (MSG...), semaphore (SEM...), and shared memory (SHM...) parameters are added with the optional Inter-Process Communications 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 Figure 3-1. A dash (—) is used in the size information of Figure 3-1 to indicate parameters that 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	30	100	250	CALCULATED	1076
NCALL	30	30	30	30	16
NINODE	100	120	200	100	92
NFILE	100	120	200	100	12
NMOUNT	5	5	5	10	20
NPROC	60	60	80	60	72
NTEXT	30	40	80	30	16
NCLIST	120	150	150	120	72
MAXUP	25	25	25	25	—
SMAPSIZ	70	70	85	70	8
CMAPSIZ	70	70	85	70	8
NHBUF	8	32	64	64	12
NPBUF	4	4	4	4	52
NAUTOUP	10	10	10	10	—
MAXMEM	128	128	256	CALCULATED	—
REL	2.0	2.0	2.0	2.0	—
NODE	unix	unix	unix	unix	—
SYS	unix	unix	unix	unix	—
VER	1	1	1	1	—
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. Recommended Parameter Values for 2 to 6 Users

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 how many system buffers to allocate. Real-time response improves as more buffers are allocated. The UNIX System buffers form a data cache. 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 until an upper limit is reached. Cache hits reduce the number of disk accesses. After the upper limit is reached, the hit ratio tends to fall as the number of buffers is increased. The entries are normally in the range of 30 to 256. 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.

NCALL Specifies how many call-out table entries to allocate. Each entry represents a function to be invoked at a later time by the clock handler. 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.

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 how many 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 entries are normally in the range of 100 to 200. Each NINODE entry contains 92 bytes. The value for NINODE pertains directly to the NFILE value. (NINODE is equal to or greater than NFILE). When the i-node table overflows, the following warning message is output on the system console.

```
WARNING: inode table overflow
```

NFILE Specifies how many open file table entries to allocate. Each entry represents an open file. The entry is normally in the range of 100 to 200. Each entry contains 12 bytes. The NFILE entry relates directly to the NINODE entry. (NFILE is less than or equal to NINODE). The NFILE control structure operates in the same manner as the NINODE structure. 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). These standard files must be closed to use more than the 17 files. Caution must be used if you do this (it is not recommended that this be done).

NMOUNT Specifies how many mount table entries to allocate. 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 recommended value for NMOUNT is 5. Since the mount table is searched linearly, this value should be as low as possible.

NPROC Specifies how many process table entries to allocate. Each table entry represents an active process. The 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 in the range of 50 to 200. Each entry contains 72 bytes.

NTEXT Specifies how many text table entries to allocate. Each entry represents an active read-only shared text segment. Every executing program has a text table entry. Although, multiple copies of sharable text share a single entry. The NTEXT value is normally in the range of 25 to 50. The value of NTEXT should never be set larger than NPROC minus 2 (NTEXT is equal to or less than NPROC minus 2). If NTEXT is larger than NPROC minus 2, the additional text table entires are never used. 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 how many 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.

MAXUP Specifies how many concurrent processes a non-superuser 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 on the same user identification, the default limit would be reached very quickly.

SMAPSIZ Specifies how many entries to allocate to 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, their sizes, and the amount of memory and swap space available. The entries are normally in the range 70 to 100. 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 and when 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. New programs that are not completely debugged can also cause strange system swap activities to occur because of large main memory requirements. Check the SMAPSIZ and CMAPSIZ,

one of their freelists may have become fragmented.

CMAPSIZ Specifies how many entries to allocate to the core map 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, their sizes, and the amount of system memory. The entries are normally in the range of 50 to 100. 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 can 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 how many "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. Recommended values are 8 to 64. **This value must be a power of 2.** Each entry contains 12 bytes. The NHBUF value must be chosen so that the value NBUF divided by NHBUF is approximately equal to 4.

NPBUF Specifies how many physical input/output buffers to allocate. One input/output buffer is needed for each physical read or write active. The value is normally in the range of 4 to 10. Each entry contains 52 bytes. The default value is 4.

NAUTOUP The NAUTOUP entry specifies the time limit 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 recommended value is 10. Specifying a smaller limit increases system reliability by writing the buffers to disk more frequently and decreases system performance. Specifying a larger limit increases system performance at the expense of reliability.

MAXMEM Specifies the maximum amount of memory that a user can have per process. The size is 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. The value of MAXMEM is limited to the available main memory less the size of the operating system. This value is calculated.

REL Specifies the UNIX System release. This value is 2.0.

NODE Specifies the node name of the system. The default node name is **unix**. Refer to the "ESTABLISHING/CHANGING THE SYSTEM AND NODE NAMES" discussion in this chapter for additional information.

SYS Specifies the system name. The default system name is **unix**. Refer to the "ESTABLISHING/CHANGING THE SYSTEM AND NODE NAMES" discussion in this chapter for additional information.

VER Specifies the version. The default version is 1.

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 inter-process communication messages. 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 inter-process communication semaphores. 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 inter-process communication shared memory. 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 that you will have a bootable operating system in the event that 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 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. This can result in the creation of an invalid boot file.

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**. (See Caution 2.)

7. Take the system to the firmware mode and boot the **/etc/system** file. (See Note.)

Note: An alternate boot procedure is to **/bin/touch** the **/etc/system** file and then execute **shutdown -i6**.

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 random access memory. The new size of random access memory is now 2-megabytes. Because of the additional memory, the following parameters are being increased.

NBUF changed from 100 to 250 (See Note.)

> ***Note:*** The NBUF parameter value is calculated by the system reconfiguration program based on the amount of equipped RAM.

NINODE changed from 120 to 200

NFILE changed from 120 to 200

NPROC changed from 60 to 80

NCLIST changed from 120 to 150

SMAPSIZ changed from 70 to 85

CMAPSIZ changed from 70 to 85

NHBUF changed from 32 to 64

MAXMEM 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 down.


SELF-CHECK

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

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

Continued

Continued from the previous screen

```
CONFIGURATION SUMMARY
=====================
     ----driver---- #devices major
     XT                1     49
     PRF               1     29
     SXT               1     48
     TTY               1     20
     PORTS             2      1,   2
     MEM               1     19
     IUART             1      0
     IDISK             1     17
     HDELOG            1     16

     ----module---
     SEM
     SHM
     MSG
     IPC


LOAD MAP
========
     --------section--------   --------address--------
     UNIX(boott)              p 2004000
     UNIX(bootd)              p 2004a58
     UNIX(.gate)              p 2005800   v       0
     UNIX(.text)              p 2006000   v 40000000
     UNIX(.data)              p 2027000   v 40040000
     UNIX(.bss)               p 202b800   v 40080000
```

All of load map not shown.

```
     HDELOG(.bss)             p 2084810   v 400d9010
     END


UNIX System V Release 2.0 3B2 Version 1
unix
Copyright (c) 1984 AT&T Technologies, Inc.
All Rights Reserved
The system is coming up.  Please wait.
Generating a new /unix
AT&T 3B2 SYSTEM CONFIGURATION:

Memory size: 2048 Kilobytes
System Peripherals:

         SBD                FD5
                  HD30
         PORTS    PORTSThe 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 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 *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 3-2 shows the default values used for NVRAM and system reconfiguration. The format used in Figure 3-2 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	1	3B2
sysadm nodename	ABcd5678	ABcd5678	2.0	1	3B2
Floppy Key	unix	unix	2.0	1	3B2
Reconfiguration	unix	unix	2.0	1	3B2

Figure 3-2. 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 1 3B2
# uname -S abcdefghijk<CR>
uname: name must be <= 8 letters
# uname -S ABcd5678<CR>
# uname -a <CR>
ABcd5678 ABcd5678 2.0 1 3B2
#
```

Figure 3-2 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 1 3B2
#
```

Figure 3-2 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 3-2 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 1 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. For this reason, 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.

- 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 as 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 3-3 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 3-3. 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 change can be made to a new password.

- 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** 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:RTKESmMOE2m.E,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). 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:RTKESmMOE2m.E: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 arguments 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/sh1
/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 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.

Two methods are offered for the recovery of the ability to log in as **root**. The first method is the most drastic in terms of losing data in the root (/) file system. The second method requires that you make a floppy disk while you can log in as **root**.

Root Recovery Method 1

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. If you cannot login as a conventional user, go to Step 3. If you can login as a conventional user, power-up the system. Login and copy the **/etc/password** and the **/etc/inittab** files to your login directory.

2. Depress the power switch to STANDBY. Wait for the power down sequence to complete.

3. Power-up the system. When the word DIAGNOSTICS is output, depress the RESET switch. This action takes the system to the firmware mode.

4. Enter the firmware password (**mcp** is the default password).

5. Reload the UNIX Operating System from floppy disks (**3B2 Core System** floppy disks 1 through 5), selecting the partial restore option. Refer to the "RELOADING **UNIX** OPERATING SYSTEM" procedure described in this chapter.

6. Log in as **root** and restore the root (/) file system from backups, as required. Refer to the "FILE SYSTEM RESTORAL FROM BACKUP" procedure described in this chapter.

7. If you were able to copy the **/etc/passwd** and the **/etc/inittab** files, restore the files from the copies. (Move the copies to the original files.) When complete, go to Step 9.

8. If you were not able to copy the **/etc/passwd** and the **/etc/inittab** files in Step 1, copy the **/etc/opasswd** file to **/etc/passwd**. The terminal configuration (**/etc/inittab**) must be restored if the configuration is different from the last backup.

9. Execute **passwd** to change the root password.

10. Store the backup and **3B2 Core System** floppy disks in a safe place.

The differences between the **/etc/passwd** file as it exits at the end of this procedure, compared to what it was at the beginning of the procedure is as follows.

a. If you copied the **opasswd** file to the **passwd** file, the new password file DOES NOT contain the last change that had been made to the file. If the last change was the addition of a user, that user needs to be added to the password file. The important point is that you are one change off.

b. You can log in as **root**.

Root Recovery Method 2

General

The second method for recovery of the **root** login begins when you first get your machine, before you have assigned a password to **root.** If you have already assigned a password, edit the password file to temporarily remove the password field for the **root** login. Remember to execute **passwd** at the end of this procedure to reassign the password. The steps are as follows.

1. Do a volume backup of the **root** file system using the Simple Administration **sysadm backup**. (At your discretion, you can **touch /etc/save.d/timestamp/:root:**, and save the time of doing the backup.)

2. Change the date on the /etc/passwd file using the **touch** command (**touch /etc/passwd**).

3. Do an incremental backup of the **root** (/) file system using the Simple Administration **sysadm backup**. The only file copied is the **/etc/passwd** file.

4. Label this floppy disk FORGOTTEN ROOT PASSWORD. Lock-up this incremental backup floppy disk for future use in the event of a forgotten **root** password.

Fast Recovery

The use of the FORGOTTEN ROOT PASSWORD floppy disk to quickly open the **root** login is as follows. This procedure takes approximately 15 minutes and requires the use of the Simple Administration **sysadm** command.

1. If you cannot login as a conventional user, go to Step 2. If you can login as a conventional user, power-up the system. Login and copy the **/etc/password** and the **/etc/inittab** files to your login directory.

2. At the console terminal, log in as **sysadm**.

3. Execute the file restoral of **/etc/passwd** using the **sysadm restore** command in the **filemgmt** menu. When you are instructed to insert a floppy disk, use the FORGOTTEN ROOT PASSWORD floppy disk. Do not rename the file.

4. When the restoration is complete, log in as **root**.

5. If you were able to copy the **/etc/passwd** file, restore the file from the copy. (Move the copy to the original file.) When the move is complete, go to Step 7.

6. If you were not able to copy the **/etc/passwd** file, copy the **/etc/opasswd** file to **/etc/passwd** file.

7. Change the **root** password by executing **passwd**.

8. Lock-up the FORGOTTEN ROOT PASSWORD floppy disk.

The differences between the **/etc/passwd** file as it exits at the end of this procedure, compared to what it was at the beginning of the procedure is as follows.

a. If you copied the **/etc/opasswd** file to the **passwd** file, the new password file DOES NOT contain the last change that had been made to the file. If the last change was the addition of a user, that user needs to be added to the password file. The important point is that you are one change off if you use the **/etc/opasswd** file.

b. You can log in as **root**.

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 using the power switch.
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 is 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 3B2 Version 1
unix
Copyright (c) 1984 AT&T Technologies, Inc.
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: 2048 Kilobytes
System Peripherals:
         SBD                FD5
                HD30
         PORTSThe 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 power-up.

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 3B2 Computer is powered up. If file system status checks indicate the need to check a file system, the **fsck** program is automatically executed. This precautionary measure helps to ensure a reliable environment for file storage on disk. If an inconsistency is discovered, corrective action must be taken.

The 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 perform 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 performed 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 one 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 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 prior to 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 performing 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 discovered, **fsck** reports the inconsistency for the user to chose a corrective action.

The following paragraphs discuss 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 performed 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 actually check these sizes, **fsck** can check for them being within reasonable bounds. All

other checks of the file system depend on the correctness of these sizes.

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 by the actual 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 by the actual 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 an actual link count for each i-node.

If the stored link count is not zero and the actual link count is zero, it means that no directory entry appears for the i-node. If the stored and actual 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 actual link count is zero, **fsck** can, under user control, link the disconnected file to the **lost+found** directory. If the stored and actual link counts are not zero and unequal, **fsck** can replace the stored link count by the actual 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 can occur by using a file system with blocks claimed by both the free-block list and by other parts of the file system.

A large number of duplicate blocks in an i-node may be due to 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 due to 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 the number of blocks actually used not matching that indicated by 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 performed by computing from the size field the number of blocks that should be associated with the i-node and comparing it to the actual 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 actual number of blocks does not match the computed number of blocks, **fsck** warns of a possible file-size error. This is only a warning

because the 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 discussion of detection and correction of the inconsistencies associated with indirect blocks, see the discussions 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's 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 a list count outside of range, block numbers outside of range, and blocks already associated with the file system.

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

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** performs successive phases over each file system performing cleanup, checking blocks and sizes, pathnames, connectivity, reference counts, and the free-block list (possibly rebuilding it).

When an inconsistency is detected, **fsck** reports the error condition to the 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 discussed under "Initialization."

Initialization

Before a file system check can be performed, 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**. See the **fsck**(1M) manual pages in Appendix A, "MANUAL PAGES," for further details.

Bad –t option

The **–t** option is not followed by a file name. The **fsck** program terminates on this error condition. See the **fsck**(1M) manual pages in Appendix A, "MANUAL PAGES," for further details.

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. See the **fsck**(1M) manual pages in Appendix A, "MANUAL PAGES," for further details.

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. See the **fsck**(1M) manual pages in Appendix A, "MANUAL PAGES," for further details.

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 assistance. 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 assistance. 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 assistance. 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 assistance.

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	Terminate program.

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 assistance.

Possible responses to CONTINUE prompt are:

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

CAN NOT WRITE: BLK B (CONTINUE)

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	Terminate program.

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	Terminate the program.

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	Terminate the program.

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	Terminate the program.

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	Terminate the program.

POSSIBLE FILE SIZE ERROR I=I

The i-node *I* size does not match the actual 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 assistance. 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	Terminate the program.

DUPS/BAD IN ROOT INODE (CONTINUE)

Phase 1 or Phase 1B have 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	Terminate the program.

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 have 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 have 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**. See **fsck**(1M) manual pages in Appendix A, "MANUAL PAGES," for further details.

SORRY. NO SPACE IN lost+found DIRECTORY

There is no space to add another entry to the **lost+found** directory in the root directory of the file system; **fsck** ignores the request to link a directory in **lost+found**. This invokes the UNREF error condition in Phase 4. Clean out unnecessary entries in **lost+found** or make **lost+found** larger. See **fsck**(1M) manual pages in Appendix A, "MANUAL PAGES," for further details.

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 have 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 have 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 actual count of the free i-nodes 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 actual 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	Terminate the program.

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	Terminate the program.

BAD FREEBLK COUNT

The count of free blocks in a free-list block is greater than 50 or less than 0. This error condition 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 actual count of free blocks 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 actual count.
NO	Ignore this error condition.

BAD FREE LIST (SALVAGE)

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

Possible responses to SALVAGE prompt are:

YES	Replace actual free-block list with a new free-block list. The new free-block list will be ordered 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. See **fsck**(1M) manual pages in Appendix A, "MANUAL PAGES," for further details.

Cleanup

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

X files Y blocks Z free

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

******** BOOT 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 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 with whatever backup procedures you deem appropriate for your situation. 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 due to new bad blocks.

What Is a Bad Block?

A bad block is a disk block that cannot reliably store data. This is found out 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 will need to 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 approximately 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 very 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 will even remap 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 feature to be used with minimal knowledge for the vast majority of cases that will occur. 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 32-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 discuss what was going on behind the scenes.

Disk Identification

The mechanisms used for bad block handling are designed to be general purpose mechanisms. 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 are 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 definitely 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 discussed 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 32-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 will then print on the console.

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, it will report the error but it 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 is 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 assistance.

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 discussed next.

Fixing Bad Blocks

To fix a bad block a very quiescent machine 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 files 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 will have to be checked prior to it being mountable.

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

Chapter 6

SYSTEM ADMINISTRATION COMMANDS

COMMAND SUMMARY

The System Administration Utilities provide 16 UNIX System V commands. Also documented in this guide are 40 UNIX System V commands that are provided with the basic machine (Essential Utilities). The 40 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. This section information is described later in this chapter. All 56 of these commands are as follows.

bcheckrc(1M)^*	ff(1M)	killall(1M)^	pwck(1M)
brc(1M)^*	fmtflop(1M)^	labelit(1M)^	rc0(1M)^*
checkall(1M)	fsck(1M)^	ldsysdump(1)	rc2(1M)^*
chroot(1M)	fsdb(1M)	link(1M)	setclk(1M)^
clri(1M)^	fsstat(1M)^	mkboot(1M)^	setmnt(1M)^
crash(1M)	fuser(1M)	mkfs(1M)^	shutdown(1M)^
cron(1M)^	getmajor(1M)^	mknod(1M)^	su(1)^
dd(1)^	getty(1M)^*	mkunix(1M)^	sync(1)^
devnm(1M)^	grpck(1M)	mount(1M)^	sysdef(1M)
df(1M)^	hdeadd(1M)^	mountall(1M)^	telinit(1M)^
dfsck(1M)	hdefix(1M)^	mvdir(1)	umount(1M)^
drvinstall(1M)^	hdelogger(1M)^	ncheck(1M)	umountall(1M)^
du(1)^	id(1)^	newgrp(1)^	unlink(1M)
errdump(1M)^	init(1M)^	prtvtoc(1M)^	whodo(1M)

To use these commands, you must 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**. The majority 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 (with the exception of **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 discussed.
- **Sample Command Use:** Example command line entries and system responses are provided to show you how to use the command.

In the command format discussions 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*]

In the sample command discussions, 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.
```

Refer to Appendix A, "MANUAL PAGES," the *3B2 Computer User Reference Manual,* or the *3B2 Computer Programmers Reference Manual* as applicable, for UNIX System V manual pages supporting the commands described in this guide.

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/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.

Figure 6-1. Command Summary (Sheet 1 of 9)

COMMAND	DESCRIPTION
^**/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.
^**/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.
^**/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.

Figure 6-1. Command Summary (Sheet 2 of 9)

COMMAND	DESCRIPTION
^/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.
^/etc/fmtflop	Formats and verifies floppy disks.
^/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.

Figure 6-1. Command Summary (Sheet 3 of 9)

COMMAND	DESCRIPTION
^**/etc/getmajor**	Outputs all slot/major numbers in the equipped device table for the requested device.
^**/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 4 of 9)

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.
^**/etc/mkfs**	Constructs a file system.

Figure 6-1. Command Summary (Sheet 5 of 9)

COMMAND	DESCRIPTION
^**/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.
^**/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.

Figure 6-1. Command SUMMARY (Sheet 6 of 9)

COMMAND	DESCRIPTION
/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.
^/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.

Figure 6-1. Command Summary (Sheet 7 of 9)

COMMAND	DESCRIPTION
^**/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.
/etc/sysdef	Outputs system configuration information in a tabular format.
^**/etc/telinit**	Telinit arguments direct (tell) the **init** command what actions are to be performed. 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**.)

Figure 6-1. Command Summary (Sheet 8 of 9)

COMMAND	DESCRIPTION
/etc/unlink	Removes a link to a file.
/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 9 of 9)

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. A file system check on the root file system (**/dev/dsk/c1d0s0**) is run if the sanity check returns a status code not equal to zero. The **bcheckrc** command is executed for every change in the system run level 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. For this reason, sample command line entries and system responses are not provided.

/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 for every change in the system run level 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. For this reason, sample command line entries and system responses are not provided.

/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 of the 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. For this reason, a Sample Command Use description is not provided.

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).

```
fsck $* /dev/dsk/c1d0s0 /dev/dsk/c1d0s2
```

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/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 /e**tc/fsck** command. The /e**tc/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/diskette<CR>

  /dev/diskette
  File System: rar Volume: 1.1

  ** 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/diskette<CR>
/dev/diskette:
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/diskette 31 25<CR>
clearing 31
clearing 25
# fsck /dev/diskette<CR>

  /dev/diskette
  File System: rar Volume: 1.1

  ** 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/diskette** 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 *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 user 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** command examines the contents of the files in the **/usr/spool/cron/crontabs** directory once every minute to see if the contents have changed. If the contents of these files have changed, the **cron** command reads the new table.

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
#
```

/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 be output starting from the beginning of the 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***b** 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{lASDFGHJKL:"}QWERTYUIOP{lASDFGHJKL:"}
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{lasdfghjkl:"}qwertyuiop{lasdfghjkl:"}
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 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/cld0s0 /
# devnm /usr/bin<CR>
/dev/dsk/cld0s2 /usr/bin
# devnm /usr/lib/cron<CR>
/dev/dsk/cld0s2 /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 file systems that are configured on the basis of 1024-byte or 512-byte blocks. For the 1024-byte block file system, a divide-by-two function is performed for reporting the number of blocks. Note that a character (byte) is 8-bits.

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 Phases 1 (check blocks and sizes) and Phase 2 (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 drives are as follows.

DEVICE	**-s***blocks/cylinder:gap*
32M Hard Disk	**-s90:7**
10M Hard Disk	**-s72:7**
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 in the event that 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/diskette** 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/diskette -s /dev/dsk/c1d0s2<CR>
1 will identify /dev/diskette
2 will identify /dev/dsk/c1d0s2
Precede every answer with 1 or 2
as in '1y' for /dev/diskette

1 /dev/diskette
1 /dev/diskette     File System:  Volume:


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

2 /dev/dsk/c1d0s2   ** Phase 1 - Check Blocks and Sizes
1 /dev/diskette     ** Phase 1 - Check Blocks and Sizes
1 /dev/diskette     ** Phase 2 - Check Pathnames
1 /dev/diskette     ** Phase 3 - Check Connectivity
1 /dev/diskette     ** Phase 4 - Check Reference Counts
1 /dev/diskette     ** Phase 5 - Check Free List
1 /dev/diskette     ** Phase 6 - Salvage Free List
1 /dev/diskette     2 files 2 blocks 1394 free
1 /dev/diskette     ******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 is 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 can not 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 /fl/house/bills<CR>
3        /fl/house/bills/water
2        /fl/house/bills/gas
1        /fl/house/bills/electric
1        /fl/house/bills/telephone
8        /fl/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] was determine to be 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 is 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 is a sample **errdump** output that is representative of valid NVRAM contents. Note that the last five panic error messages are saved at the end of the error dump.

```
# 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/diskette<CR>
ff: /dev/diskette: 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/diskette<CR>
ff: /dev/diskette: 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 integral floppy disk drive BEFORE the **fmtflop** command is executed.

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/rdiskette**.

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/rdiskette<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/rdiskette<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/rdiskette<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/fsck — File System Consistency Check and Interactive Repair

General

The **fsck** command is used to check and repair inconsistencies in a file system. With the exception of 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 options 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 2 (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 with the file system was created. The format for various configurations of disk drives are as follows.

DEVICE	**-s***blocks/cylinder:gap*
32M Hard Disk	**-s90:7**
10M Hard Disk	**-s72:7**
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 in the event that 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 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 are 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. This is the format the operating system uses 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 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 the defect map. The system in 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 3b2 Version 1
unix
Copyright (c) 1984 AT&T Technologies, Inc.
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: 1.1

  ** 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 382 SYSTEM CONFIGURATION:

Memory size: 2048 Kilobptes
System Peripherals:

         SDB                FD5
                  HD30
         PORTSThe 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 C.

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 SYTEM CONFIGURATION:

Memory Size: 2048 Kilobytes
System Peripherals:

         SBD                FD5
                  HD30
         PORTSThe 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 use 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/diskette 1422:192 1 18<CR>
Mkfs: /dev/diskette?
(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/diskette rar rar2.0<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 = rar2.0 -- 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** can be executed only by **root**.

Files (or directories) that are linked means that one file has multiple names. Files (or directories) can not be lined 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. This can result in the creation of an invalid boot file.

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 is 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 disk partition. The size of the file system depends on the disk partition that is specified. Refer to Appendix B, "DISK PARTITIONS," for the number of blocks associated with the various configurations of disk drives used as part of the 3B2 Computer.

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 B, "DISK PARTITION," for additional information on file system sizes, device files, and starting cylinders for the various types of disks used as part of the 3B2 Computer.

device — The *device* argument specifies the name of the special device file. For example, **/dev/diskette** 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 divided by four. 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 7 for the hard disk and 1 for the floppy disk.

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	*-sblocks/cylinder*
32M Hard Disk	**-s90:7**
10M Hard Disk	**-s72:7**
Floppy Disk	**-s18:1**

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.

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/diskette 1422:192 1 18<CR>
Mkfs: /dev/diskette?
(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/rdiskette rar rar2.0<CR>
Current fsname: , Current volname: , Blocks: 1422, Inodes: 192
FS Units: 1Kb, Date last mounted: Wed Aug 15 06:21:50 1984
NEW fsname = rar, NEW volname = rar2.0 -- 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 files as a block-type device.
c	Specifies the special files 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 files 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 each file system is mounted, it is checked using **/etc/fsstat**. (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.

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 made to move the **oldletters** directory outside of the **/usr** file system and the resulting error message. In the case of an attempt 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-node numbers and path names for the **root** file system (/dev/dsk/c1d0s0). 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 of the 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
#
```

/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 performed 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>
UNIX System V Release 2.0 3B2 Version 1
unix
Copyright (c) 1984 AT&T Technologies, Inc.
All Rights Reserved

$ >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
UNIX System V Release 2.0 3B2 Version 1
unix
Copyright (c) 1984 AT&T Technologies, Inc.
All Rights Reserved

$
```

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 *raw_device*

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:

NAME	NUMBER
UNASSIGNED	0
BOOT	1
ROOT	2
SWAP	3
USR	4
BACKUP	5

FLAG The FLAG is a number code identifying how the partition is to be mounted.

NAME	NUMBER
MOUNTABLE, READ AND WRITE	00
NOT MOUNTABLE	01
MOUNTABLE, READ ONLY	10

SECTOR START

Identifies the starting sector (block) number of the partition. Blocks are counted starting from 0 to the end of the disk.

SIZE IN SECTORS

Identifies the number of sectors (blocks) in the partition.

Sample Command Use

The following command line entry 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 32-megabyte hard disk following a full restore procedure.

```
# prtvtoc /dev/rdsk/c1d0s0<CR>
THE VOLUME TABLE OF CONTENTS FOR DEVICE /dev/rdsk/c1d0s0
VOLUME NAME:

POSSIBLE NUMBER OF PARTITIONS:  16

PARTITION       TAG      FLAG          SECTOR START     SIZE IN SECTORS
    0            2        0               6120             12510
    1            3        1               100              6020
    2            4        0               18630            43830
    6            0        1               0                62460
    7            0        1               0                100
# prtvtoc /dev/rdsk/c1d0s6<CR>
THE VOLUME TABLE OF CONTENTS FOR DEVICE /dev/rdsk/c1d0s6
VOLUME NAME:
VERSION:  1
POSSIBLE NUMBER OF PARTITIONS:  16

PARTITION       TAG      FLAG          SECTOR START     SIZE IN SECTORS
    0            2        0               6120             12510
    1            3        1               100              6020
    2            4        0               18630            43830
    6            0        1               0                62460
    7            0        1               0                100
#
```

/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. For this reason, sample command line entries and system responses are not provided.

```
# "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** discussions 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. For this reason, sample command line entries and system responses are not provided.

```
# "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 can be added, as required, to support the addition of hardware and software features. Refer to the **/etc/rc2** discussion 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/diskette 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/diskette 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 C, "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 /.

Command Format

The general format of the **shutdown** command is as follows.

shutdown [**-y**] [**-g***grace*] [**-i***state*] [**-p***count*]

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 C, "RUN LEVELS," summarizes the system run-levels (states). The default *state* is the single-user mode (1, s, or S).
-p*count*	Specifies the number of active processes that is permitted to exist for the shutdown process to run to completion. The default number of processes is seven.

Sample Command Use

The following shows the command line entry and system responses for a shut down 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 -il<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)
    10  entries in mount table (NMOUNT)
    60  entries in proc table (NPROC)
    30  entries in shared text table (NTEXT)
   120  clist buffers (NCLIST)
    25  processes per user id (MAXUP)
    70  entries in swap map table (SMAPSIZ)
    70  entries in core map table (CMAPSIZ)
    64  hash slots for buffer cache (NHBUF)
     4  buffers for physical I/O (NPBUF)
    10  auto update time limit in seconds (NAUTOUP)
   256  pages per process (MAXMEM)
*
* Utsname Tunables
*
     2.0  release (REL)
    unix  node name (NODE)
    unix  system name (SYS)
       1  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 C.

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 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 **/etc/killall** command is used to terminate all processes with open files so that a file system can be "unbusied."

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/diskette /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/diskette read/write on Tue Aug 14 07:51:47 1984
# umount /dev/diskette<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 with the exception of 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>
#  -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/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

ttyll    rar         7:35
    ttyll      116     0:06 sh
    ttyll      237     0:09 vi

ttyl2    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
#
```

AT&T 3B2 COMPUTER
SYSTEM ADMINISTRATION
UTILITIES GUIDE

Update Issue 1

November 15, 1984

This update inventory should be placed behind the *3B2 Computer System Administration Utilities Guide* title page. This inventory identifies the pages that have been added or changed by the *3B2 Computer System Administration Utilities Guide Update* (305-404), dated November 15, 1984.

Revised Page(s)	Date
brc(1M) manual page	11/84
ckauto(1M) manual page	11/84
cron(1M) manual page	11/84
devnm(1M) manual page	11/84
errdump(1M) manual pages	11/84
fmthard(1M) manual page	11/84
fsck(1M) manual pages	11/84
id(7) manual page	11/84
if(7) manual page	11/84
intro(7) manual page	11/84
labelit(1M) manual page	11/84
led(1M) manual page	11/84
mkfs(1M) manual pages	11/84
mountall(1M) manual page	11/84
newboot(1M) manual page	11/84
prtconf(1M) manual page	11/84
prtvtoc(1M) manual page	11/84
setmnt(1M) manual page	11/84

Appendix A

MANUAL PAGES

PAGE

Appendix A

MANUAL PAGES

This appendix contains the UNIX System V Section 1 manual pages for the System Administration Utilities. Also provided are Section 7 (special files) and Section 8 (system maintenance procedures) manual pages. A permuted index to Section 1, 7, and 8 manual pages is provided as part of this appendix. Manual pages are organized in alphabetical sequence by Section. These **section** identifiers are described in Chapter 1, "INTRODUCTION."

The following listing identifies the manual pages (commands) for which there is a corresponding description in Chapter 6, "SYSTEM ADMINISTRATION COMMANDS." The circumflex (^) is used to identify system administration commands that are provided as part of the Essential Utilities. The identifiers in parentheses are the **UNIX** System V manual page *section* code. Commands identified by an asterisk (*) are not normally executed directly from a terminal but are called as part of another process.

bcheckrc(1M)^*
brc(1M)^*
checkall(1M)
chroot(1M)
clri(1M)^
crash(1M)
cron(1M)^
dd(1)^
devnm(1M)^
df(1M)^
dfsck(1M)
drvinstall(1M)^
du(1)^
errdump(1M)^
ff(1M)
fmtflop(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)
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)
whodo(1M)

Certain commands are described on a manual page along with one or more other commands. Commands that are described on a different manual page than indicated by the name of the command are as follows.

COMMAND	MANUAL PAGE
bcheckrc	brc(1M)
dfsck	fsck(1M)
grpck	pwck(1M)
labelit	volcopy(1M)
rc	brc(1M)
telinit	init(1M)
umount	mount(1M)
unlink	link(1M)

INTRODUCTION

This manual is intended to supplement the information contained in the *AT&T 3B2 Computer System User Reference Manual* and to provide an easy reference volume for those who must administer a UNIX system.

This manual is divided into three sections:

1. System Maintenance Commands and Application Programs
7. Special Files
8. System Maintenance Procedures

Throughout this volume, each reference of the form *name*(1M), *name*(7), or *name*(8), refers to entries in this manual, while all other references to entries of the form *name*(N), where *N* is a number possibly followed by a letter, refer to entry *name* in Section *N* of the *AT&T 3B2 Computer System User Reference Manual* or the *3B2 Computer System Programmer Reference Manual.*

Section 1 (*System Maintenance Commands and Application Programs*) contains a subset of commands that are initially resident on the hard disk (labeled Essential Utilities) and commands that are installable from the System Administration Utilities floppy diskette. These entries carry a sub-class designation of "1M" for cross-referencing reasons.

Section 7 (*Special Files*) discusses the characteristics of each system file that actually refers to an input/output device. The names in this section generally refer to device names for the hardware, rather than to the names of the special files themselves.

Section 8 (*System Maintenance Procedures*) discusses crash recovery and boot procedures, facility descriptions, etc.

Each section consists of a number of independent entries of a page or so each. The name of the entry appears in the upper corners of its pages. Entries within each section are alphabetized, with the exception of the introductory entry that begins each section. Some entries may describe several routines, commands, etc. In such cases, the entry appears only once, alphabetized under its "major" name.

All entries are based on a common format, not all of whose parts always appear:

The **NAME** part gives the name(s) of the entry and briefly states its purpose.

The **SYNOPSIS** part summarizes the use of the program being described. A few conventions are used, particularly in Section 1 (*Commands*):

Boldface strings are literals and are to be typed just as they appear.

Italic strings usually represent substitutable argument prototypes and program names found elsewhere in the manual. (They are underlined in the typed version of the entries.)

Square brackets [] around an argument prototype indicate that the argument is optional. When an argument prototype is given as "name" or "file", it always refers to a *file* name.

Ellipses ... are used to show that the previous argument prototype may be repeated.

A final convention is used by the commands themselves. An argument beginning with a minus −, plus +, or equal sign = is often taken to be some sort of flag argument, even if it appears in a position where a file name could appear. Therefore, it is unwise to have files whose names begin with −, +, or =.

The **DESCRIPTION** part discusses the subject at hand.

The **EXAMPLE(S)** part gives example(s) of usage, where appropriate.

The **FILES** part gives the file names that are built into the program.

The **SEE ALSO** part gives pointers to related information.

The **DIAGNOSTICS** part discusses the diagnostic indications that may be produced. Messages that are intended to be self-explanatory are not listed.

The **WARNINGS** part points out potential pitfalls.

The **BUGS** part gives known bugs and sometimes deficiencies. Occasionally, the suggested fix is also described.

A table of contents and a permutted index derived from that table precede section 1M. On each *index* line, the title of the entry to which that line refers is followed by the appropriate section number in parentheses. The *Permuted Index* is used by searching the middle column for a key word or phrase. The right column will then contain the name of the manual page that contains that command. The left column contains additional useful information about the command.

TABLE OF CONTENTS

1. System Maintenance Commands and Application Programs

7. Special Files

8. System Maintenance Procedures

PERMUTED INDEX

NAME

intro − introduction to system maintenance commands and application programs

DESCRIPTION

This section describes, in alphabetical order, commands that are used chiefly for system maintenance and administration purposes. The commands in this section should be used along with those listed in Section 1 of the *3B2 Computer System User Reference Manual* and Sections 2, 3, 4, and 5 of the *3B2 Computer System Programmer Reference Manual.* References to other manual entries not of the form *name*(1M), *name*(7) or *name*(8) refer to entries of the above manuals.

COMMAND SYNTAX

Unless otherwise noted, commands described in this section accept options and other arguments according to the following syntax:

name [*option*(*s*)] [*cmdarg*(*s*)]
where:

name	The name of an executable file.
option	− *noargletter*(*s*) or, − *argletter*<>*optarg* where <> is optional white space.
noargletter	A single letter representing an option without an argument.
argletter	A single letter representing an option requiring an argument.
optarg	Argument (character string) satisfying preceding *argletter*.
cmdarg	Path name (or other command argument) *not* beginning with − or, − by itself indicating the standard input.

SEE ALSO

getopt(1) in the *3B2 Computer System User Reference Manual.*
getopt(3C) in the *3B2 Computer System Programmer Reference Manual.*

DIAGNOSTICS

Upon termination, each command returns two bytes of status, one supplied by the system and giving the cause for termination, and (in the case of "normal" termination) one supplied by the program (see *wait*(2) and *exit*(2)). The former byte is 0 for normal termination; the latter is customarily 0 for successful execution and non-zero to indicate troubles such as erroneous parameters, bad or inaccessible data, or other inability to cope with the task at hand. It is called variously "exit code", "exit status", or "return code", and is described only where special conventions are involved.

BUGS

Regretfully, many commands do not adhere to the aforementioned syntax.

NAME

brc, bcheckrc, rc2 − system initialization shell scripts

SYNOPSIS

/etc/brc

/etc/bcheckrc

/etc/rc2

DESCRIPTION

These shell procedures are executed via entries in **/etc/inittab** by *init*(1M) when the system changes run states.

The *brc* procedure clears the mounted file system table, **/etc/mnttab** and puts root into the mount table.

The *bcheckrc* procedure checks the status of the root file system. If the root file system is found to be bad, *bcheckrc* repairs it.

The *rc2* procedure executes **/etc/TIMEZONE** and every file in the **rc.d** directory.

These shell procedures, may be used for several run-level states. The *who*(1) command may be used with the **−r** argument to get the run-level informantion.

FILES

/etc/rc.d

SEE ALSO

fsck(1M), init(1M), shutdown(1M).
who(1) in the *3B2 Computer System User Reference Manual.*

NAME

checkall – faster file system checking procedure

SYNOPSIS

/etc/checkall

DESCRIPTION

The *checkall* procedure is a prototype and must be modified to suit local conditions. The following will serve as an example:

```
# check the root file system by itself
fsck /dev/dsk/c1d0s6

# dual fsck of (integral hard disk)
dfsck /dev/rdsk/c1d0s2 – /dev/rdsk/c1d1s1
```

The *checkall* procedure takes 11 minutes.

Dfsck is a program that permits an operator to interact with two *fsck*(1M) programs at once. To aid in this, *dfsck* will print the file system name for each message to the operator. When answering a question from *dfsck*, the operator must prefix the response with a **1** or a **2** (indicating that the answer refers to the first or second file system group).

Due to the file system load balancing required for dual checking, the *dfsck*(1M) command should always be executed through the *checkall* shell procedure.

In a practical sense, the file systems are divided as follows:

```
dfsck file_systems_on_drive_0 – file_systems_on_drive_1
  . . .
```

WARNINGS

1. Do not use *dfsck* to check the *root* file system.
2. The *dfsck* procedure is useful only if the system is set up for multiple physical I/O buffers.

SEE ALSO

fsck(1M).

NAME

checkfsys – check a file system on a removable disk

SYNOPSIS

The *checkfsys* command allows the user to check for and optionally repair a damaged file system on a removable disk.

The user is asked if they wish to:

check the file system
: No repairs are attempted.

repair it interactively
: The user is informed about each instance of damage and asked if they wish to repair it.

repair it automatically
: The program applies a standard repair to each instance of damage.

This command may be controlled by passwords. See *sysadm*(1), the **admpasswd** sub-command.

The identical function is also available under the *sysadm* menu:

sysadm checkfsys

WARNING

While automatic and interactive checks are generally successful, they can occasionally lose a file or a file's name. Files with content but without names are put in the */file-system/***lost+found** directory.

If losing data is of particular concern, "check" the file system first to discover how damaged it appears to be. Then either use one of the repair mechanisms or get knowledgeable help from someone who knows how to debug and reconstruct a file system.

SEE ALSO

fsck(1M), makefsys(1M), mountfsys(1M).
sysadm(1) in the *3B2 Computer System User Reference Manual.*

NAME

chroot — change root directory for a command

SYNOPSIS

/etc/chroot newroot command

DESCRIPTION

The given command is executed *relative to the new root*. The meaning of any initial slashes (/) in path names is changed for a command and any of its children to *newroot*. Furthermore, the initial working directory is *newroot*.

Notice that:

```
chroot newroot command >x
```

will create the file x relative to the original root, not the new one.

This command is restricted to the super-user.

The new root path name is always relative to the current root: even if a *chroot* is currently in effect, the *newroot* argument is relative to the current root of the running process.

SEE ALSO

cd(1) in the *3B2 Computer System User Reference Manual.*

BUGS

One should exercise extreme caution when referencing special files in the new root file system.

NAME

ckauto – determine if the UNIX operating system was reconfigured at boot time.

SYNOPSIS

/etc/ckauto

DESCRIPTION

The *ckauto* command uses a *sys3b*(2) call to determine if the UNIX system was reconfigured during initialization. If the *sys3b* call is successful, *ckauto* exits with 1, otherwise it exists with 0. *Ckauto* is intended to be used by utilities that will do services based on the output of the *sys3b* call.

SEE ALSO

sys3b(2) in the *3B2 Computer System Programmer Reference Manual.*

NAME

clri – clear i-node

SYNOPSIS

/etc/clri file-system i-number ...

DESCRIPTION

Clri writes zeros on the 64 bytes occupied by the i-node numbered *i-number*. *File-system* must be a special file name referring to a device containing a file system. After *clri* is executed, any blocks in the affected file will show up as "missing" in an *fsck*(1M) of the *file-system*.

Read and write permission is required on the specified *file-system* device. The i-node becomes allocatable.

The primary purpose of this routine is to remove a file which for some reason appears in no directory. If it is used to *zap* an i-node which does appear in a directory, care should be taken to track down the entry and remove it. Otherwise, when the i-node is reallocated to some new file, the old entry will still point to that file. At that point removing the old entry will destroy the new file. The new entry will again point to an unallocated i-node, so the whole cycle is likely to be repeated again and again.

SEE ALSO

fsck(1M), fsdb(1M), ncheck(1M).
fs(4) in the *3B2 Computer System Programmer Reference Manual.*

BUGS

If the file is open, *clri* is likely to be ineffective.

WARNINGS

This command should only be used in emergencies and extreme care should be exercised.

NAME

crash – examine system images

SYNOPSIS

/etc/crash [system_image] [namelist]

DESCRIPTION

The *crash* command is used to interactively examine system memory images. In response to user requests, *crash* formats and prints system control structures and other miscellaneous information.

Parameters to crash are the *system_image*, the file containing the system memory image and *namelist*, the text file that was booted to produce the *system*memory*image*. This file contains the symbol table information needed for symbolic access to the system memory image.

System_image is defaulted to **/dev/mem** or the active memory of a running system. This can also be specified by '–'. The system image can also be **/dev/ifdsk06**, with the first floppy of a system dump taken with *sysdump*(8) or the path name of a file produced by *ldsysdump*(1M).

The default *namelist* is the **/unix** of the current machine. If a system image from another machine is to be used, the text file must be copied from that machine as well.

Commands

Input to *crash* is typically of the form:

command [options] [items]

where *options* modifies the format of the output and *items* indicates which items are printed. If a table is being dumped, the default is all valid table entries. List items are specified by table entry number (0 based). Default for items that are process related is the current process, i.e., the process that was running at the time of the crash.

The use of [a] denotes that "a" is optional, (a\|\b) that either "a" or "b" are acceptable, but not both.

The commands are:

? print synopsis of commands

!*cmd* escape to shell to execute a command.

buffer [format] [list of buffers]
Aliases: **b**
Print the data from a system buffer according to *format*. If no format is specified, the last format is used. Valid formats are **byte, character, decimal, directory, hex, inode, octal,** and **write.** The **write** format creates a file **buf.#** in the current directory containing the buffer data.

bufhdr [list of buffers]
Aliases: **buf, hdr**
Format and print system buffer headers.

callout Aliases: **c, call, calls, time, timeout, tout**
Print all entries in the callout table.

ds data_address
Print data or bss symbol closest to but less than *data_address*.

file [list of file table entries]
Aliases: **i, ino**
Format and print i-node table entries.

kfp [address]
Aliases: **fp, r9**
Prints stack frame pointer of process that caused panic. If *address* is specified, sets frame pointer to address.

lck Aliases: **l**
Print the active and sleep record lock tables; also verify the correctness of the record locking linked lists.

map Formats and prints coremap.

mount [list of mount table entries]
Aliases: **m, mnt**
Format and print mount table entries.

nm symbol Print address and type of specified symbol.

nvram area Format and print contents of nvram areas. Areas are: **fwnvr, unxnvr, systate,** and **errlog.**

od (symbol | [−p] address) [count [format]]
Aliases: **dump, rd**
Dump *count* data values starting at symbol value or address according to given *format*. A physical address is indicated by '−p'.

pcb [list of process table entries]

pcb −i [symbol| address [index]]
Format and print process control blocks. The **−i** indicates an interrupt pcb. *Index* is used when an array of pcbs is addressed.

proc [− [r]] [list of process table entries]
Aliases: **p, ps**
Format and print process table entries. The '−' option generates more information. The '−r' option limits output to runnable processes.

quit Aliases: **q**
Terminates the crash session.

regs Prints the MMU registers.

sdt [− segment table entry] [list of process table entries]
Prints the segment descriptor tables for sections 2 and 3. The *−segment table entry* limits printing to the specified table entry.

smap Formats and prints swap map.

sram Prints SRAM values.

stack [list of process table entries]

stack −i (symbol| address) [index]
Aliases: **k, kernel, s, stk**
Dump the process stack. The **−i** option specifed interrupt stacks. *Index* is used with arrays of interrupt pcbs.

stat Print system statistics.

sysdt [k | K | m | M]
Formats and prints kernel segment descriptor tables (sections 0 and 1). **K** or **k** restricts printing to section 0. **M** or **m** restricts printing to section 1.

tables [list of process table entries]
Aliases: **tbls**
Prints segment descripto tables for section 2 and 3.

text [list of text table entries]
Aliases: **txt, x**
Format and print text table entries.

trace [−r] [process table entries]

trace [−r] −**i** (**symbol** | **address**) [**index**]
Aliases: **t**
Trace provides a formatted dump of the specified stack. The **−i options specifies an interrupt stack.** Index is used with arrays of interrupt pcbs.

ts text-address
Prints the text symbol closest to but less than the text address.

tty [type] [−i] [list of tty table entries]
Aliases: **con, term**
Formats and prints stty tables. The **−i** option causes stty settings to be printed.

user [list of process table entries]
Aliases: **u, u_area, uarea, ublock**
Formats and prints ublocks.

var Aliases: **tunable, tunables, tune, v**
Prints tunable system parameters.

FILES

buf.#	file in current dirctory where beffer number # is written
/dev/mem	system image of currently running system
/dev/ifdsk06	used to access system image on floppy diskette

SEE ALSO
ldsysdump(1M), sysdump(8).

NAME

cron - clock daemon

SYNOPSIS

/etc/cron

DESCRIPTION

Cron executes commands at specified dates and times. Regularly scheduled commands can be specified according to instructions found in crontab files; users can submit their own crontab file via the *crontab* command. Commands which are to be executed only once may be submitted via the *at* command. Since *cron* never exits, it should only be executed once. This is best done by running *cron* from the initialization process through the file **/etc/rc2**.

Cron only examines crontab files and at command files during process initialization and when a file changes. This reduces the overhead of checking for new or changed files at regularly scheduled intervals.

FILES

/usr/lib/cron	main cron directory
/usr/lib/cron/log	accounting information
/usr/spool/cron	spool area

SEE ALSO

at(1), crontab(1), sh(1) in the *3B2 Computer System User Reference Manual.*

DIAGNOSTICS

A history of all actions taken by cron are recorded in **/usr/lib/cron/log.**

NAME

dd – convert and copy a file

SYNOPSIS

dd [option=value] ...

DESCRIPTION

Dd copies the specified input file to the specified output with possible conversions. The standard input and output are used by default. The input and output block size may be specified to take advantage of raw physical I/O.

option	*values*
if=*file*	input file name; standard input is default
of=*file*	output file name; standard output is default
ibs=*n*	input block size *n* bytes (default 512)
obs=*n*	output block size (default 512)
bs=*n*	set both input and output block size, superseding *ibs* and *obs*; also, if no conversion is specified, it is particularly efficient since no in-core copy need be done
cbs=*n*	conversion buffer size
skip=*n*	skip *n* input blocks before starting copy
seek=*n*	seek *n* blocks from beginning of output file before copying
count=*n*	copy only *n* input blocks
conv=**ascii**	convert EBCDIC to ASCII
ebcdic	convert ASCII to EBCDIC
ibm	slightly different map of ASCII to EBCDIC
lcase	map alphabetics to lower case
ucase	map alphabetics to upper case
swab	swap every pair of bytes
noerror	do not stop processing on an error
sync	pad every input block to *ibs*
... , ...	several comma-separated conversions

Where sizes are specified, a number of bytes is expected. A number may end with **k**, **b**, or **w** to specify multiplication by 1024, 512, or 2, respectively; a pair of numbers may be separated by **x** to indicate a product.

Cbs is used only if *ascii* or *ebcdic* conversion is specified. In the former case *cbs* characters are placed into the conversion buffer, converted to ASCII, and trailing blanks trimmed and new-line added before sending the line to the output. In the latter case ASCII characters are read into the conversion buffer, converted to EBCDIC, and blanks added to make up an output block of size *cbs*.

After completion, *dd* reports the number of whole and partial input and output blocks.

SEE ALSO

cp(1) in the *3B2 Computer System User Reference Manual.*

DIAGNOSTICS

f+p blocks in(out) numbers of full and partial blocks read(written)

NAME

devnm – device name

SYNOPSIS

/etc/devnm [names]

DESCRIPTION

Devnm identifies the special file associated with the mounted file system where the argument *name* resides. (As a special case, both the block device name and the swap device name are printed for the argument name / if swapping is done on the same disk section as the **root** file system.) Argument names must be full path names.

This command is most commonly used by **/etc/brc** (see *brc*(1M)) to construct a mount table entry for the **root** device.

EXAMPLE

The command:

```
/etc/devnm /usr
```

produces

```
/dev/c1d0s2 usr
```

if **/usr** is mounted on **/dev/dsk/c1d0s2**.

FILES

/dev/dsk/*
/etc/mnttab

SEE ALSO

brc(1M).

NAME

df − report number of free disk blocks

SYNOPSIS

df [−t] [−f] [file-systems **]**

DESCRIPTION

The *df* command prints out the number of free blocks and free i-nodes available for on-line file systems by examining the counts kept in the super-blocks; *file-systems* may be specified either by device name (e.g., **/dev/dsk/c1d0s2** or by mounted directory name (e.g., **/usr**). If the *file-systems* argument is unspecified, the free space on all of the mounted file systems is printed.

The *df* command uses the following options:

−t causes the total allocated block figures to be reported as well

−f only an actual count of the blocks in the free list is made (free i-nodes are not reported). With this option, *df* will report on raw devices.

FILES

/dev/dsk/*
/etc/mnttab

SEE ALSO

fs(4), mnttab(4) in the *3B2 Computer System Programmer Reference Manual.*

NAME

drvinstall – install/uninstall a driver

SYNOPSIS

/etc/drvinstall [–m master **] [–d** object **] [–s** system **] [–o** directory **] –v** version **–nbufx**

DESCRIPTION

Drvinstall accepts an *object* file, *master* file, and *system* file as inputs and creates the corresponding specially formatted files for use by the self-configuring boot.

If the file is a software driver then *drvinstall* will assign the first available major number to that driver. This major number will be inserted into the *master* file. The major numbers available for software drivers on for 3B2 computer 48-127 inclusive. The remaining major numbers are either reserved for hardware devices or used by integral drivers.

Drvinstall determines the available major numbers by scanning all existing *master* files for major numbers and then assigns the first major number from the above list.

Drvinstall will print the major number assigned, or the major number found in the *master* file if installing/uninstalling a software driver. For hardware drivers or loadable modules nothing is printed.

One or both of the **–m** or **–d** options must be specified. *Drvinstall* expects any unused field of the *master* file to be filled with a "–".

Drvinstall inserts/deletes **INCLUDE** statements in the *system* file for loadable object modules and software drivers.

The **–m** *master* argument specifies the path name of the *master* file to be used. If this flag is omitted, the **/etc/master.d** directory is used.

The **–d** *object* argument specifies the path name of the *object* file to be used. If this flag is omitted, the **/boot** directory is used.

The **–s** *system* argument specifies the path name of the *system* file to be used. If this flag is omitted, the **/etc/system** file is used.

The **–o** *directory* argument specifies the path name of the bootable file. If this flag is omitted, the **/boot** directory is used.

The **–v** *version* argument specifies the *version* number of *drvinstall* command compatible with the *master* file being used. This is a required option.

The **–u** option will uninstall a driver. Either the **–d** or **–m** option must be used in conjunction with the **-u** option. A driver dependency check is made and if a dependency is found, a warning message is issused and the command is aborted. If no dependency is found, then the following actions are taken:

1. the bootable *object* file is removed.
2. the major number is replaced with a "–" in the *master* file if a software driver.
3. the **INCLUDE** statement is deleted from the *system* file.

The **–f** option, when used with the **–u** option, disables the dependency check. This will result in the driver being uninstalled.

The –n option inhibits any edit of the *system* file.

The **–b** option inhibits generation of the bootable *object* file.

The –x option enables debugging output.

SEE ALSO

mkboot(1M).
master(4), system(4) in the *3B2 Computer System Administration Utilities Guide.*

DIAGNOSTICS

The major numbered assigned or found for a software driver is printed on **stdout**. A zero is returned for success and an non-zero is returned for failures.

NAME

du − summarize disk usage

SYNOPSIS

du [**−sar**] [names]

DESCRIPTION

Du gives the number of blocks contained in all files and (recursively) directories within each directory and file specified by the *names* argument. The block count includes the indirect blocks of the file. If *names* is missing, . is used.

The optional arguments are as follows:

−s causes only the grand total (for each of the specified *names*) to be given.

−a causes an entry to be generated for each file.

Absence of either **−s** or **−a** causes an entry to be generated for each directory only.

−r will cause *du* to generate messages about directories that cannot be be read, files that cannot be opened, etc., rather than being silent (the default).

A file with two or more links is only counted once.

BUGS

If the **−a** option is not used, non-directories given as arguments are not listed.
If there are too many distinct linked files, *du* will count the excess files more than once.
Files with holes in them will get an incorrect block count.

NAME

errdump — print error log

SYNOPSIS

errdump

DESCRIPTION

This command displays on the system console the error log contained in the system's nonvolatile ram. The display contains the previous saved system state, the last 5 panic messages and their time of occurrence, and an indication of the log's sanity.

DIAGNOSTICS

The phrase "not superuser" if invoked by other than the superuser. Superuser is defined as individuals logged in under the root directory from the console port.

EXAMPLE

The following is an example of the printout in response to the *errdump* command.

```
#
#
#
#
# errdump
nvram status:   sane

csr:     0x0648  (floppy)  (unassigned)  (clock)  (uart)

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

r3:      0x00049001
r4:      0x00000081
r5:      0x00000000
r6:      0x40091348
r7:      0x0001a13f
r8:      0x4008edd8
oap:     0x400816d8
opc:     0x400083bc
osp:     0x40081700
ofp:     0x40081700
isp:     0x40080008
pcbp:    0x40041a40

fltar:   0xc0021140
fltcr:   reqacc  xlevel  ftype
         0xa     0x0     0x0

         srama           sramb
[0]      0x02034800      0x0000011f
[1]      0x02035100      0x00000030
[2]      0x02035860      0x00000074
[3]      0x02035c00      0x00000015
```

```
        Panic log

[0]     Thu Sep 20 09:51:36 1984
        KERNEL DATA ALIGNMENT ERROR

[1]     Thu Sep 20 09:51:37 1984
        KERNEL DATA ALIGNMENT ERROR

[2]     Thu Sep 20 09:51:40 1984
        KERNEL DATA ALIGNMENT ERROR

[3]     Thu Sep 20 09:52:21 1984
        KERNEL DATA ALIGNMENT ERROR

[4]     Fri Sep 21 05:50:10 1984
        SYSTEM PARITY ERROR INTERRUPT
```

SEE ALSO

UNIX system error messages in the *3B2 Computer System Administration Utilities Guide*.

NAME

ff − list file names and statistics for a file system

SYNOPSIS

/etc/ff [options] special

DESCRIPTION

Ff reads the i-list and directories of the *special* file, assuming it to be a file system, saving i-node data for files which match the selection criteria. Output consists of the path name for each saved i-node, plus any other file information requested using the print *options* below. Output fields are positional. The output is produced in i-node order; fields are separated by tabs. The default line produced by *ff* is:

path-name i-number

With all *options* enabled, output fields would be:

path-name i-number size uid

The argument *n* in the *option* descriptions that follow is used as a decimal integer (optionally signed), where *+n* means more than *n*, *−n* means less than *n*, and *n* means exactly *n*. A day is defined as a 24 hour period.

−I Do not print the i-node number after each path name.

−l Generate a supplementary list of all path names for multiply linked files.

−p *prefix* The specified *prefix* will be added to each generated path name. The default is ..

−s Print the file size, in bytes, after each path name.

−u Print the owner's login name after each path name.

−a *n* Select if the i-node has been accessed in *n* days.

−m *n* Select if the i-node has been modified in *n* days.

−c *n* Select if the i-node has been changed in *n* days.

−n *file* Select if the i-node has been modified more recently than the argument *file*.

−i *i-node-list*
Generate names for only those i-nodes specified in *i-node-list*.

SEE ALSO

ncheck(1M).
find(1) in the *3B2 Computer System User Reference Manual.*

BUGS

Only a single path name out of any possible ones will be generated for a multiply linked i-node, unless the **−l** option is specified. When **−l** is specified, no selection criteria apply to the names generated. All possible names for every linked file on the file system will be included in the output.

On very large file systems, memory may run out before *ff* does.

NAME

fmtflop − physically format floppy disks

SYNOPSIS

fmtflop [−v] *device*

DESCRIPTION

Fmtflop physically formats the 96 tpi floppy media inserted in the floppy disk drive. The **−v** option formats and verifies that the formatted floppy disk is correct. The *device* is the path name of the floppy disk drive (e.g., **/dev/idsk/c0d0s6**).

Fmtflop formats DOUBLE SIDED media with 512 byte sectors, 9 sectors per track, and 80 tracks. Before executing *fmtflop*, the floppy media must be placed in the disk drve and the door must be closed.

SEE ALSO

if(7).

DIAGNOSTICS

An error message is returned if the format or verify fails. If this occurs, it is best to reissue the command. If the command fails the second time (especially on the same area of the disk) the floppy media is probably bad.

NAME

fmthard – populate VTOC on hard disks

SYNOPSIS

fmthard **–i** **–m** [**–s** datafile] [**–n** volumename] /dev/rdsk/c?d?s6

DESCRIPTION

Fmthard populates the VTOC (volume table of contents) on the **hard** disks.

The following options apply to fmthard:

–i just prints the default file name which would be used to set up the VTOC.

–m automatically makes a file system on each mountable partition.

–s the VTOC is populated according to a *datafile* created by the user. The *datafile* format is described below.

–n allows the disk to be given a volume name up to 8 character long.

If no options are given, the VTOC is populated according to the specification of a default file, *hdxdft,* where *x* is the drive id of the disk. For example, for the 30M disk, the default file is labeled *hd3dft.*

The *datafile* contains one line for each partition, listed in consecutive order. Each line is delimited by a newline. Comment lines may be inserted, the first character being an asterisk. Each line is composed of entries that are position dependent separated by white space and have the following format:

partition id flag Start Sector Size in Sectors

Where the entries have the following values.

partition The partition number 0-15.

id The partition id consists of a two digit hex number. The following being reserved codes: V_BOOT 0x01, V_ROOT 0x02, V_SWAP 0x03, V_USR 0x04 and V_BACKUP 0x05.

flag The flag allows a partition to be flagged as unmountable or read only, the masks being: V_UNMNT 0x01, and V_RONLY 0x10.

Start Sector Start Sector is defined as the sector number on which the partition starts.

Size in Sectors The size of the partition is specified by the number of sectors.

FILES

/etc/vtoc/*

NAME

fsck, dfsck − file system consistency check and interactive repair

SYNOPSIS

/etc/fsck [**−y**] [**−n**] [**−sX**] [**−SX**] [**−t** file] [**−q**] [**−D**] [**−f**] [**−b**] [file-systems]
/etc/dfsck [options1] filsys1 ... − [options2] filsys2 ...

DESCRIPTION

Fsck

Fsck audits and interactively repairs inconsistent conditions for UNIX system files. If the file system is consistent then the number of files, number of blocks used, and number of blocks free are reported. If the file system is inconsistent the user is prompted for concurrence before each correction is attempted. It should be noted that most corrective actions will result in some loss of data. The amount and severity of data lost may be determined from the diagnostic output. The default action for each consistency correction is to wait for the user to respond yes or **no**. If the user does not have write permission *fsck* will default to a **−n** action.

Fsck has more consistency checks than its predecessors *check*, *dcheck*, *fcheck*, and *icheck* combined.

The following options are interpreted by *fsck*.

−y Assume a yes response to all questions asked by *fsck*.

−n Assume a **no** response to all questions asked by *fsck*; do not open the file system for writing.

−s*X* Ignore the actual free list and (unconditionally) reconstruct a new one by rewriting the super-block of the file system. The file system should be unmounted while this is done; if this is not possible, care should be taken that the system is quiescent and that it is rebooted immediately afterwards. This precaution is necessary so that the old, bad, in-core copy of the superblock will not continue to be used, or written on the file system.

The **−s***X* option allows for creating an optimal free-list organization.

If *X* is not given, the values used when the file system was created are used.

−S*X* Conditionally reconstruct the free list. This option is like **−s***X* above except that the free list is rebuilt only if there were no discrepancies discovered in the file system. Using **−S** will force a **no** response to all questions asked by *fsck*. This option is useful for forcing free list reorganization on uncontaminated file systems.

−t If *fsck* cannot obtain enough memory to keep its tables, it uses a scratch file. If the **−t** option is specified, the file named in the next argument is used as the scratch file, if needed. Without the **−t** flag, *fsck* will prompt the user for the name of the scratch file. The file chosen should not be on the file system being checked, and if it is not a special file or did not already exist, it is removed when *fsck* completes.

−q Quiet *fsck*. Do not print size-check messages. Unreferenced **fifos** will silently be removed. If *fsck* requires it, counts in the superblock will be automatically fixed and the free list salvaged.

−D Directories are checked for bad blocks. Useful after system crashes.

−f Fast check. Check block and sizes and check the free list. The free list will be reconstructed if it is necessary.

-b Reboot. If the file system being checked is the root file system and modifications have been made, then either remount the root file system or reboot the system. A remount is done only if there was minor damage.

If no *file-systems* are specified, *fsck* will read a list of default file systems from the file **/etc/checklist**.

Inconsistencies checked are as follows:

1. Blocks claimed by more than one i-node or the free list.
2. Blocks claimed by an i-node or the free list outside the range of the file system.
3. Incorrect link counts.
4. Size checks:
   Incorrect number of blocks.
   Directory size not 16-byte aligned.
5. Bad i-node format.
6. Blocks not accounted for anywhere.
7. Directory checks:
   File pointing to unallocated i-node.
   I-node number out of range.
8. Super Block checks:
   More than 65536 i-nodes.
   More blocks for i-nodes than there are in the file system.
9. Bad free block list format.
10. Total free block and/or free i-node count incorrect.

Orphaned files and directories (allocated but unreferenced) are, with the user's concurrence, reconnected by placing them in the **lost+found** directory, if the files are nonempty. The user will be notified if the file or directory is empty or not. If it is empty, *fsck* will silently remove them. *Fsck* will force the reconnection of nonempty directories. The name assigned is the i-node number. The only restriction is that the directory **lost+found** must preexist in the root of the file system being checked and must have empty slots in which entries can be made. This is accomplished by making **lost+found**, copying a number of files to the directory, and then removing them (before *fsck* is executed).

Checking the raw device is almost always faster and should be used with everything but the *root* file system.

Dfsck

Dfsck allows two file system checks on two different drives simultaneously. *options1* and *options2* are used to pass options to *fsck* for the two sets of file systems. A **-** is the separator between the file system groups.

The *dfsck* program permits a user to interact with two *fsck* programs at once. To aid in this, *dfsck* will print the file system name for each message to the user. When answering a question from *dfsck*, the user must prefix the response with a **1** or a **2** (indicating that the answer refers to the first or second file system group). The following serves as an example:

/etc/dfsck /dev/dsk/c1d0s6 /dev/dsk/c1d0s5

WARNINGS

Do not use *dfsck* to check the *root* file system.

FILES

/etc/checklist	contains default list of file systems to check.
/etc/checkall	optimizing *dfsck* shell file.

SEE ALSO

checkall(1M), mkfs(1M), ncheck(1M), crash(1M).
uadmin(2), checklist(4), fs(4) in the *3B2 Computer System Programmer Reference Manual.*

BUGS

I-node numbers for . and .. in each directory should be checked for validity.

DIAGNOSTICS

The diagnostics produced by *fsck* are intended to be self-explanatory.

NAME

fsdb – file system debugger

SYNOPSIS

/etc/fsdb special [–]

DESCRIPTION

Fsdb can be used to patch up a damaged file system after a crash. It has conversions to translate block and i-numbers into their corresponding disk addresses. Also included are mnemonic offsets to access different parts of an i-node. These greatly simplify the process of correcting control block entries or descending the file system tree.

Fsdb contains several error-checking routines to verify i-node and block addresses. These can be disabled if necessary by invoking *fsdb* with the optional – argument or by the use of the **O** symbol. (*Fsdb* reads the i-size and f-size entries from the superblock of the file system as the basis for these checks.)

Numbers are considered decimal by default. Octal numbers must be prefixed with a zero. During any assignment operation, numbers are checked for a possible truncation error due to a size mismatch between source and destination.

Fsdb reads a block at a time and will therefore work with raw as well as block I/O. A buffer management routine is used to retain commonly used blocks of data in order to reduce the number of read system calls. All assignment operations result in an immediate write-through of the corresponding block.

The symbols recognized by *fsdb* are:

#	absolute address
i	convert from i-number to i-node address
b	convert to block address
d	directory slot offset
+,–	address arithmetic
q	quit
>,<	save, restore an address
=	numerical assignment
=+	incremental assignment
=–	decremental assignment
="	character string assignment
O	error checking flip flop
p	general print facilities
f	file print facility
B	byte mode
W	word mode
D	double word mode
!	escape to shell

The print facilities generate a formatted output in various styles. The current address is normalized to an appropriate boundary before printing begins. It advances with the printing and is left at the address of the last item printed. The output can be terminated at any time by typing the delete character. If a number follows the **p** symbol, that many entries are printed. A check is made to detect block boundary overflows since logically sequential blocks are generally not physically sequential. If a count of zero is used, all entries to the end of the current block are printed.

The print options available are:

i	print as i-nodes
d	print as directories
o	print as octal words
e	print as decimal words
c	print as characters
b	print as octal bytes

The **f** symbol is used to print data blocks associated with the current i-node. If followed by a number, that block of the file is printed. (Blocks are numbered from zero.) The desired print option letter follows the block number, if present, or the **f** symbol. This print facility works for small as well as large files. It checks for special devices and that the block pointers used to find the data are not zero.

Dots, tabs, and spaces may be used as function delimiters but are not necessary. A line with just a new-line character will increment the current address by the size of the data type last printed. That is, the address is set to the next byte, word, double word, directory entry or i-node, allowing the user to step through a region of a file system. Information is printed in a format appropriate to the data type. Bytes, words and double words are displayed with the octal address followed by the value in octal and decimal. A **.B** or **.D** is appended to the address for byte and double word values, respectively. Directories are printed as a directory slot offset followed by the decimal i-number and the character representation of the entry name. I-nodes are printed with labeled fields describing each element.

The following mnemonics are used for i-node examination and refer to the current working i-node:

md	mode
ln	link count
uid	user ID number
gid	group ID number
sz	file size
a#	data block numbers (0 − 12)
at	access time
mt	modification time
maj	major device number
min	minor device number

EXAMPLES

386i	prints i-number 386 in an i-node format. This now becomes the current working i-node.
ln=4	changes the link count for the working i-node to 4.
ln=+1	increments the link count by 1.
fc	prints, in ASCII, block zero of the file associated with the working i-node.
2i.fd	prints the first 32 directory entries for the root i-node of this file system.
d5i.fc	changes the current i-node to that associated with the 5th directory entry (numbered from zero) found from the above command. The first logical block of the file is then printed in ASCII.
512B.p0o	prints the superblock of this file system in octal.

2i.a0b.d7=3	changes the i-number for the seventh directory slot in the root directory to 3. This example also shows how several operations can be combined on one command line.
d7.nm="name"	changes the name field in the directory slot to the given string. Quotes are optional when used with **nm** if the first character is alphabetic.
a2b.p0d	prints the third block of the current i-node as directory entries.

SEE ALSO

fsck(1M).

dir(4), fs(4) in the *3B2 Computer System Programmer Reference Manual.*

NAME

fsstat – file system status

SYNOPSIS

/etc/fsstat file-system

DESCRIPTION

Fsstat reports on the status of *file-system*. During startup, this command is used to decide if the file system needs checking before it is mounted. It succeeds if the file system is unmounted and appears okay. For the root file system, it succeeds if it is active and not marked bad.

SEE ALSO

fs(4) in the *3B2 Computer System Programmer Reference Manual.*

DIAGNOSTICS

If successful, the command has an exit status of 0. Otherwise, the command has an exit status of 1 if the file system needs to be checked, 2 if mounted, and 3 for other failures.

NAME

fuser – identify processes using a file or file structure

SYNOPSIS

/etc/fuser [–ku] files **[–] [[–ku]** files**]**

DESCRIPTION

Fuser lists the process IDs of the processes using the *files* specified as arguments. For block special devices, all processes using any file on that device are listed. The process ID is followed by **c**, **p** or **r** if the process is using the file as its current directory, the parent of its current directory (only when in use by the system), or its root directory, respectively.

The following options may be used with *fuser*:

–u the login name, in parentheses, also follows the process ID.

–k the **SIGKILL** signal is sent to each process. Only the super-user can terminate another user's process [see *kill*(2)].

Options may be respecified between groups of files. The new set of options replaces the old set, with a lone dash canceling any options currently in force.

The process IDs are printed as a single line on the standard output, separated by spaces and terminated with a single new line. All other output is written on standard error.

FILES

/unix	for namelist
/dev/kmem	for system image
/dev/mem	also for system image

SEE ALSO

mount(1M).
ps(1) in the *3B2 Computer System User Reference Manual.*
kill(2), signal(2) in the *3B2 Computer System Programmer Reference Manual.*

NAME

getmajor — print slot/major number(s) of hardware devices

SYNOPSIS

/etc/getmajor name | bdcode

DESCRIPTION

Getmajor prints all slot/major numbers found in system equipped device table for the requested device. The argument *name* may be entered in lower or upper case.

DIAGNOSTICS

If successful a zero is returned. If *name* | *bdcode* is not found, a NULL is printed and a nonzero is returned.

NAME

getty − set terminal type, modes, speed, and line discipline

SYNOPSIS

/etc/getty [**−h**] [**−t** timeout] line [speed [type [linedisc]]]
/etc/getty −c file

DESCRIPTION

Getty is a program that is invoked by *init*(1M). It is the second process in the series, (*init-getty-login-shell*) that ultimately connects a user with the UNIX system. Initially *getty* prints the login message field for the entry it is using from **/etc/gettydefs**. *Getty* reads the user's login name and invokes the *login*(1) command with the user's name as argument. While reading the name, *getty* attempts to adapt the system to the speed and type of terminal being used.

Line is the name of a tty line in **/dev** to which *getty* is to attach itself. *Getty* uses this string as the name of a file in the **/dev** directory to open for reading and writing. Unless *getty* is invoked with the **−h** flag, *getty* will force a hangup on the line by setting the speed to zero before setting the speed to the default or specified speed. The **−t** flag plus *timeout* in seconds, specifies that *getty* should exit if the open on the line succeeds and no one types anything in the specified number of seconds. The optional second argument, *speed*, is a label to a speed and tty definition in the file **/etc/gettydefs**. This definition tells *getty* at what speed to initially run, what the login message should look like, what the initial tty settings are, and what speed to try next should the user indicate that the speed is inappropriate (by typing a <*break*> character). The default *speed* is 300 baud. The optional third argument, *type*, is a character string describing to *getty* what type of terminal is connected to the line in question. *Getty* understands the following types:

none	default
vt61	DEC vt61
vt100	DEC vt100
hp45	Hewlett-Packard 45
c100	Concept 100

The default terminal is **none**; i.e., any crt or normal terminal unknown to the system. Also, for terminal type to have any meaning, the virtual terminal handlers must be compiled into the operating system. They are available, but not compiled in the default condition. The optional fourth argument, *linedisc*, is a character string describing which line discipline to use in communicating with the terminal. Again the hooks for line disciplines are available in the operating system but there is only one presently available, the default line discipline, LDISC0.

When given no optional arguments, *getty* sets the *speed* of the interface to 300 baud, specifies that raw mode is to be used (awaken on every character), that echo is to be suppressed, either parity allowed, new-line characters will be converted to carriage return-line feed, and tab expansion performed on the standard output. It types the login message before reading the user's name a character at a time. If a null character (or framing error) is received, it is assumed to be the result of the user pushing the "break" key. This will cause *getty* to attempt the next *speed* in the series. The series that *getty* tries is determined by what it finds in **/etc/gettydefs**.

The user's name is terminated by a new-line or carriage-return character. The latter results in the system being set to treat carriage returns appropriately (see *ioctl*(2)).

The user's name is scanned to see if it contains any lower-case alphabetic characters; if not, and if the name is non-empty, the system is told to map any future upper-case characters into the corresponding lower-case characters.

Finally, *login* is called with the user's name as an argument. Additional arguments may be typed after the login name. These are passed to *login*, which will place them in the environment (see *login*(1)).

A check option is provided. When *getty* is invoked with the **−c** option and *file*, it scans the file as if it were scanning **/etc/gettydefs** and prints out the results to the standard output. If there are any unrecognized modes or improperly constructed entries, it reports these. If the entries are correct, it prints out the values of the various flags. See *ioctl*(2) to interpret the values. Note that some values are added to the flags automatically.

FILES

/etc/gettydefs
/etc/issue

SEE ALSO

init(1M), tty(7).
login(1) in the *3B2 Computer System User Reference Manual.*
ioctl(2), gettydefs(4), inittab(4) in the *3B2 Computer System Programmer Reference Manual.*
ct(1C) in the *Basic Networking Utilities.*

BUGS

While *getty* does understand simple single character quoting conventions, it is not possible to quote the special control characters that *getty* uses to determine when the end of the line has been reached, which protocol is being used, and what the erase character is. Therefore it is not possible to login via *getty* and type a #, @, /, !, _, backspace, **^U**, **^D**, or & as part of your login name or arguments. They will always be interpreted as having their special meaning as described above.

NAME

hdeadd – add/delete reports to/from Hard Disk Error (HDE) Log

SYNOPSIS

hdeadd **–a** [*aoptions*]

hdeadd **–d** [*doptions*]

hdeadd **–e** [[**–D**] major minor]

hdeadd **–f** filename

hdeadd **–r** [**–D**] major minor filename

hdeadd **–s** [**–D**] major minor filename

DESCRIPTION

This command is a bad block handling utility command. **You must be superuser to use it.** It is used to print the list of equipped disks. It is also used for manually adding or deleting disk error reports. The manual mechanism is used only when the system has reported a bad block and was in an inappropriate state for manually logging it. These include disk errors reported while in firmware mode and disk errors that cause the system to PANIC. In addition, this command has some options that are intended to be used only while testing the feature.

The following options may be used with *hdeadd*:

–a *hdeadd* allows a Hard Disk Error (HDE) report to be manually added to the HDE Log of a disk.

–d *hdeadd* allows a specific report or a range of reports to be deleted from the HDE Log of a disk.

–e prints out the list of major/minor device numbers of the equipped hard disks. If the *major* and *minor* device numbers are also provided, it determines if that specification is an equipped hard disk. The result is both printed on the standard output and is used to determine the exit status. A NORMAL (or TRUE) exit means it is an equipped disk.

–f the file specified by *filename* is assumed to contain a canned set of HDE Log manipulations. Each line of text contains one specification in the command argument form, starting with a **–a** or a option.

–s saves a copy of the HDE Log of the specified (by *major/minor* device number) disk in the file specified by *filename*.

–r restores the HDE Log of the specified disk from the file specified by *filename*.

The valid *aoptions* are *hard disk error* specifications.

The valid *doptions* are either a *hard disk error* specification or an *error range* specification.

A *hard disk error* specification includes the following values:

–D *maj min* Specifies the major device number (*maj*) and minor device number (*min*) of the disk.

–b *blockno* **Normal form:** Specifies the physical disk block number in integer counter form (e.g., treating the disk as a simple stream of blocks). Physical disk block numbering starts with zero meaning sector 0 of track 0 of cylinder 0. This is the normal form that is reported by the operating system.

−B *cyl trk sec*	**Alternate form:** Specifies the physical disk block number in in terms of its physical cylinder number (*cyl*), track number within cylinder (*trk*), and sector number within track (*sec*). This alternate form is available to cover the possibility of a non-operating system detector reporting block numbers in this hardware form.
−t *mmddhhmm*[*yy*]	**Optional:** Specifies the time of day when the error actually occurred. If omitted when adding reports, the current time is used. If omitted when deleting reports, any reports for the given block are deleted.

An *error range* specification includes the following values:

−D *maj min*	Specifies the major device number (*maj*) and minor device number (*min*) of the disk.
−F *mmddhhmm*[*yy*]	**Optional:** Specifies the "from" time for the time interval being purged. If omitted, zero (the beginning of time) is used.
−T *mmddhhmm*[*yy*]	**Optional:** Specifies the "to" time for the time interval being purged. If omitted, the end of time is used. The range comparisons include the end values of the range in the purge.

FILES

/dev/hdelog

SEE ALSO

hdefix(1M), hdelogger(1M), hdelog(7).

DIAGNOSTICS

The HDE commands exit with one of three values:

0	means NORMAL, or TRUE
1	means bad command usage or execution errors
2	means BAD BLOCKS or FALSE (but command executed successfully)

WARNINGS

You must be super-user to use *hdeadd*.

NAME

hdefix – report or change bad block mapping

SYNOPSIS

hdefix **–p** [[**–D**] major minor]

hdefix **–a** [major minor [*blocknospec* ...]]

hdefix **–F** [**–D**] major minor [*blocknospec* ...]]

hdefix **–r** [**–D**] major minor filename

hdefix **–s** [**–D**] major minor filename

DESCRIPTION

This command is a bad block handling utility command. **You must be superuser to use it.** The *hdefix* command is used to find out what blocks are currently mapped to surrogate images on the equipped hard disks and it is used the change what blocks are mapped. In addition, this command has some options that are intended to be used only while testing the feature.

When the mapping is changed, block initialization is performed. The original block is assumed to be unreadable and the new surrogate image is zeroed. Data is probably lost, so damage is expected.

If the block is associated with a file system, the file system may be damaged as a result of the mapping change. To handle this situation, the file system is marked dirty, which means *fsck*(1M) must be run before the file system can be used, and a system reboot is forced after all other bad block processing is complete. If the block is a data block of a file, that file will be corrupted, even after this recovery has finished.

The following options may be used with *hdefix*:

–p *hdefix* prints a report of media properites that includes the currently mapped bad blocks. If a specific disk is specified (by giving its *major* and *minor* device numbers), only the report for that disk is printed. If no disk is specified, a report is given for each equipped disk.

–D used to specify the major device number (*maj*) and minor device number (*min*) of the disk.

–a To map new bad blocks, the **–a** option is used. If no arguments follow the **–a** option, each equipped disk is processed, using the HDE Log on each disk to determine which blocks to map. If a specific disk is specified, only that disk is processed. If one or more block numbers are specified, those blocks are mapped, instead of using the HDE Log to get blocks to map. This is the only way to map an unreadable block containing the HDE Log.

–F forces blocks to be removed from the map without any attempt to initialize them. This is intended only for testing the Bad Block Handling feature. If no block number is specified, the last block in the map is removed and the block number is ouput on the standard output.

–s a copy of the bad block map table and the surrogates pointed to by the map is saved in the file specified by the *filename* argument.

–r the bad block map table and the surrogates pointed to by the map are restored from the file specified by the *filename* argument. The save and restore options are intended only for testing. The restore can be quite destructive unless used under very controlled conditions.

A *blocknospec* has the following forms:

−b *blockno*	Specifies the physical disk block number in integer counter form (e.g., treating the disk as a simple stream of blocks). Physical disk block numbering starts with zero meaning sector 0 of track 0 of cylinder 0.
−B *cyl trk sec*	Specifies the physical disk block number in in terms of its physical cylinder number (*cyl*), track number within cylinder (*trk*), and sector number within track (*sec*). Only one of the alternate forms of block number should be specified for a given block.

FILES

/dev/hdelog

SEE ALSO

hdeadd(1M), hdelogger(1M), hdelog(7).

DIAGNOSTICS

The HDE commands exit with one of three values:

0	means NORMAL, or TRUE
1	means bad command usage or execution errors
2	means BAD BLOCKS or FALSE (but command executed successfully)

WARNINGS

You must be super-user to use *hdefix*.

NAME

hdelogger – Hard Disk Error (HDE) status report command and Log Demon

SYNOPSIS

hdelogger [–s] [–f] [–D maj min **]**

DESCRIPTION

This command is a bad block handling utility command. **You must be super-user to use it.** The *hdelogger* command serves two purposes. When run by the **init** process (process 1 – see *init*(1M)), this command performs the functions of the Hard Disk Error (HDE) Log Demon. These functions include providing summaries of outstanding errors during system startup and shutdown transitions, along with adding new errors to HDE Logs and giving the revised status summaries as errors are reported by hard disk drivers. When run as the demon, no options are used.

When run as a normal command (process 1 is not its parent), this command provides on the spot reports of outstanding errors as recorded in the HDE Logs of equipped hard disks. The following options control report generation:

–s	Specifies that summary reports are to be generated. The summary report provides sufficient information for normal Bad Block Handling operations. This is the default.
–f	Specifies that full reports are to be generated. This is intended mainly for testing the Bad Block Handling feature, but is available in case maintenance personnel need additional detail for troubleshooting complicated problems.
–D *maj min*	Restricts the report generation to a specific hard disk. If this option is omitted, reports will be generated for all equipped hard disks.

FILES

/dev/hdelog

SEE ALSO

hdeadd(1M), hdefix(1M), hdelog(7).

DIAGNOSTICS

The HDE commands exit with one of three values:

0	means NORMAL, or TRUE
1	means bad command usage or execution errors
2	means BAD BLOCKS or FALSE (but command executed successfully)

WARNINGS

You must be super-user to use *hdelogger*.

NAME

id – print user and group IDs and names

SYNOPSIS

id

DESCRIPTION

Id writes a message on the standard output giving the user and group IDs and the corresponding names of the invoking process. If the effective and real IDs do not match, both are printed.

SEE ALSO

logname(1) in the *3B2 Computer System User Reference Manual.*
getuid(2) in the *3B2 Computer System Programmer Reference Manual.*

NAME

init, telinit – process control initialization

SYNOPSIS

/etc/init [0123456SsQq]

/etc/telinit [0123456sSQqabc]

DESCRIPTION

Init

Init is a general process spawner. Its primary role is to create processes from a script stored in the file **/etc/inittab** (see *inittab*(4)). This file usually has *init* spawn *getty*'s on each line that a user may log in on. It also controls autonomous processes required by any particular system.

Init considers the system to be in a *run-level* at any given time. A *run-level* can be viewed as a software configuration of the system where each configuration allows only a selected group of processes to exist. The processes spawned by *init* for each of these *run-levels* is defined in the *inittab* file. *Init* can be in one of eight *run-levels*, **0–6** and **S** or **s**. The *run-level* is changed by having a privileged user run **/etc/init** (which is linked to */etc/telinit*). This user-spawned *init* sends appropriate signals to the orginal *init* spawned by the operating system when the system was rebooted, telling it which *run-level* to change to.

Init is invoked inside the UNIX system as the last step in the boot procedure. The first thing *init* does is to look for **/etc/inittab** and see if there is an entry of the type *initdefault* (see *inittab*(4)). If there is, *init* uses the *run-level* specified in that entry as the initial *run-level* to enter. If this entry is not in *inittab* or *inittab* is not found, *init* requests that the user enter a *run-level* from the virtual system console, **/dev/console.** If an **S** (**s**) is entered, *init* goes into the *SINGLE USER* level. This is the only *run-level* that doesn't require the existence of a properly formatted *inittab* file. If **/etc/inittab** doesn't exist, then by default the only legal *run-level* that *init* can enter is the *SINGLE USER* level. In the *SINGLE USER* level the virtual console terminal **/dev/console** is opened for reading and writing and the command **/bin/su** is invoked immediately. To exit from the *SINGLE USER run-level* one of two options can be elected. First, if the shell is terminated (via an end-of-file), *init* will reprompt for a new *run-level*. Second, the *init* or *telinit* command can signal *init* and force it to change the *run-level* of the system.

When attempting to boot the system, failure of *init* to prompt for a new *run-level* may be due to the fact that the device **/dev/console** is linked to a device other than the physical system teletype (**/dev/contty**). If this occurs, *init* can be forced to relink **/dev/console** by typing a delete on the system teletype which is collocated with the processor.

When *init* prompts for the new *run-level*, the operator may enter only one of the digits **0** through **6** or the letters **S** or **s**. If **S** is entered *init* operates as previously described in *SINGLE USER* mode with the additional result that **/dev/console** is linked to the user's terminal line, thus making it the virtual system console. A message is generated on the physical console, **/dev/contty**, saying where the virtual terminal has been relocated.

When *init* comes up initially and whenever it switches out of *SINGLE USER* state to normal run states, it sets the *ioctl*(2) states of the virtual console, **/dev/console**, to those modes saved in the file **/etc/ioctl.console**. This file is written by *init* whenever *SINGLE USER* mode is entered. If this file does not exist when *init* wants to read it, a warning is printed and default settings are assumed.

If a **0** through **6** is entered *init* enters the corresponding *run-level*. Any other input will be rejected and the user will be re-prompted. If this is the first time *init* has entered a *run-level* other than *SINGLE USER*, *init* first scans *inittab* for special entries of the type *boot* and *bootwait*. These entries are performed, providing the *run-level* entered matches that of the entry before any normal processing of *inittab* takes place. In this way any special initialization of the operating system, such as mounting file systems, can take place before users are allowed onto the system. The *inittab* file is scanned to find all entries that are to be processed for that *run-level*.

Run-level **2** is usually defined by the user to contain all of the terminal processes and daemons that are spawned in the multi-user environment.

In a multi-user environment, the *inittab* file is usually set up so that *init* will create a process for each terminal on the system.

For terminal processes, ultimately the shell will terminate because of an end-of-file either typed explicitly or generated as the result of hanging up. When *init* receives a child death signal, telling it that a process it spawned has died, it records the fact and the reason it died in **/etc/utmp** and **/etc/wtmp** if it exists (see *who*(1)). A history of the processes spawned is kept in **/etc/wtmp** if such a file exists.

To spawn each process in the *inittab* file, *init* reads each entry and for each entry which should be respawned, it forks a child process. After it has spawned all of the processes specified by the *inittab* file, *init* waits for one of its descendant processes to die, a powerfail signal, or until *init* is signaled by *init* or *telinit* to change the system's *run-level*. When one of the above three conditions occurs, *init* re-examines the *inittab* file. New entries can be added to the *inittab* file at any time; however, *init* still waits for one of the above three conditions to occur. To provide for an instantaneous response the **init Q** or **init q** command can wake *init* to re-examine the *inittab* file.

If *init* receives a *powerfail* signal (*SIGPWR*) and is not in *SINGLE USER* mode, it scans *inittab* for special powerfail entries. These entries are invoked (if the *run-levels* permit) before any further processing takes place. In this way *init* can perform various cleanup and recording functions whenever the operating system experiences a power failure.

When *init* is requested to change *run-levels* (via *telinit*), *init* sends the warning signal (**SIGTERM**) to all processes that are undefined in the target *run-level*. *Init* waits 20 seconds before forcibly terminating these processes via the kill signal (**SIGKILL**).

Telinit

Telinit, which is linked to */etc/init*, is used to direct the actions of *init*. It takes a one-character argument and signals *init* via the *kill* system call to perform the appropriate action. The following arguments serve as directives to *init*.

0–6 tells *init* to place the system in one of the *run-levels* **0–6**.

a,b,c tells *init* to process only those **/etc/inittab** file entries having the **a**, **b** or **c** *run-level* set.

Q,q tells *init* to re-examine the **/etc/inittab** file.

s,S tells *init* to enter the single user environment. When this level change is effected, the virtual system teletype, **/dev/console**, is changed to the terminal from which the command was executed.

FILES

/etc/inittab
/etc/utmp

/etc/wtmp
/etc/ioctl.console
/dev/console
/dev/contty

SEE ALSO

getty(1M).
login(1), sh(1), who(1) in the *3B2 Computer System User Reference Manual.*
kill(2), inittab(4), utmp(4) in the *3B2 Computer System Programmer Reference Manual.*

DIAGNOSTICS

If *init* finds that it is continuously respawning an entry from **/etc/inittab** more than 10 times in 2 minutes, it will assume that there is an error in the command string, and generate an error message on the system console, and refuse to respawn this entry until either 5 minutes has elapsed or it receives a signal from a user *init* (*telinit*). This prevents *init* from eating up system resources when someone makes a typographical error in the *inittab* file or a program is removed that is referenced in the *inittab.*

WARNINGS

Telinit can only be run by someone who is super-user or a member of group sys.

NAME

killall – kill all active processes

SYNOPSIS

/etc/killall [signal]

DESCRIPTION

Killall is a procedure used by **/etc/shutdown** to kill all active processes not directly related to the shutdown procedure.

Killall is chiefly used to terminate all processes with open files so that the mounted file systems will be unbusied and can be unmounted.

Killall sends *signal* (see *kill*(1)) to all remaining processes not belonging to the above group of exclusions. If no *signal* is specified, a default of **9** is used.

FILES

/etc/shutdown

SEE ALSO

fuser(1M), shutdown(1M).
kill(1), ps(1) in the *3B2 Computer System User Reference Manual.*
signal(2) in the *3B2 Computer System Programmer Reference Manual.*

WARNINGS

The *killall* command can only be executed by the super-user.

NAME

labelit − provide labels for disk file systems

SYNOPSIS

/etc/labelit special [fsname volume [**−n**]]

DESCRIPTION

Labelit can be used to provide initial labels for unmounted disk file systems. With the optional arguments omitted, *labelit* prints current label values. The **−n** option provides for initial labeling only (this destroys previous contents).

The *fsname* argument represents the mounted name (e.g.,: **root**, **u1**, etc.) of the filsystem being copied.

The *special* should be the physical disk section (e.g.,: **/dev/dsk/c0d0s6**).

Fsname and *volumename* are recorded in the last 12 characters of the superblock (**char fsname[6], volname[6];**).

SEE ALSO

sh(1) in the *3B2 Computer System User Reference Manual.*
fs(4) in the *3B2 Computer System Programmer Reference Manual.*

NAME

ldsysdump – load multiple floppy sysdump

SYNOPSIS

ldsysdump destination_file

DESCRIPTION

The *ldsysdump* command recombines the multiple floppies taken during a system dump into a single file on the hard disk suitable for use by *crash*(1). The *destination_file* is the name of the hard disk file into which the floppy data will be loaded.

When invoked, *ldsysdump* begins an interactive procedure that prompts the user to insert the floppies to be loaded. The user has the option of quitting the session at any time. This allows only the portion of the system image needed to be dumped.

EXAMPLES

This example loads the 3 floppies produced via *sysdump* on a machine equipped with 2 MB of memory.

```
$ldsysdump /usr/tmp/cdump

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

Loading sysdump
.....
.....

Insert next sysdump floppy.
Enter 'c' to continue, 'q' to quit: c

Loading more sysdump
.....
.....

Insert next sysdump floppy.
Enter 'c' to continue, 'q' to quit: c

Loading more sysdump
.....
.....

3 Sysdump files coalesced, 4096 (512 byte) blocks
$
```

FILES

/dev/dsk/c0d0s6 device used for floppy access

SEE ALSO

crash(1M), sysdump(8).

DIAGNOSTICS

For reversible errors a message will be printed and the user will be allowed to insert a new floppy and continue the session.

WARNINGS

Since the 3B2 computer can be equipped with up to 4 MB of memory, the *destination_file* can become quite large. The file size limit must be set large enough to hold a file of this size.

NAME

led − flash green LED

SYNOPSIS

/etc/led [−f] [−o]

DESCRIPTION

Led is used to turn the green LED (light emitting diode) on. The options are as follows:

−f invokes the *sys3b*(2) system call to set the green LED to a flashing state.

−o invokes *sys3b*(2) system call to set the green LED to a solid on state.

SEE ALSO

sys3b(2) in the *3B2 Computer System Programmer Reference Manual.*

WARNINGS

This command can only be executed by the super-user.

NAME

link, unlink – exercise link and unlink system calls

SYNOPSIS

/etc/link file1 file2
/etc/unlink file

DESCRIPTION

Link and *unlink* perform their respective system calls on their arguments, abandoning all error checking.

SEE ALSO

rm(1) in the *3B2 Computer System User Reference Manual.*
link(2), unlink(2) in the *3B2 Computer System Programmer Reference Manual.*

WARNINGS

These commands may only be executed by the super-user, who (it is hoped) knows what he or she is doing.

NAME

makefsys – create a file system on a diskette

SYNOPSIS

makefsys

DESCRIPTION

This command, which may be under password control, allows the user to create a file system on a diskette.

The user is asked some questions and then the file system is created. Once created, the diskette is self-identifying.

The *makefsys* command may be put under password control by the use of **admpasswd** contained on the *sysadm*(1) manual page.

The identical function is also available under the *sysadm*(1) command:

sysadm makefsys

SEE ALSO

checkfsys(1M), mountfsys(1M).
sysadm(1) in the *3B2 Computer System User Reference Manual.*

NAME

mkboot – create an object file in proper format for self-config boot

SYNOPSIS

/etc/mkboot [**–m** master] [**–d** directory] [**–k** kernel.o] driver.o ...

DESCRIPTION

Mkboot accepts object files as input and creates the corresponding specially formatted files for use by the self-config boot. Each object file named must have a corresponding *master*(4) file in the **/etc/master.d** directory. The UNIX system kernel object file is always matched with the name *kernel*. The *master* file named for the remaining object files is derived from the object file name itself--any optional path prefix or **".o"** suffix is removed, and the lower case result is used as the *master* file name.

Each applicable *master* file is read and the configuration information associated is extracted. For each object file named, a new file is created containing this configuration information. This new file is written to the **/boot** directory and is given the corresponding device name (in capital letters, and without any **".o"** suffix) as the corresponding object file.

The options are:

-m *master* specifies the directory containing the master files to be used for each object file. If this flag is omitted, the **/etc/master.d** directory is used.

-d *directory* specifies the directory to be used for storing the each new object file. If this flag is omitted, the **/boot** directory is used.

-k *kernel.o* specifies the name of the object file for the UNIX operating system. The **master** file name used for this object file is always named *kernel*.

EXAMPLE

mkboot -m newmaster adli.o – will read the file named *adli* from the directory **newmaster** for the **adli** device configuration data, take the file **adli.o** from the current directory and create the formatted file **/boot/ADLI** containing the configuration information for the **adli**.

SEE ALSO

master(4) in the *3B2 Computer System Programmer Reference Manual.*

DIAGNOSTICS

Most messages should be self-explanatory.

Name.o: not processed; cannot open /etc/master.d/name – The file **name.o** was specified on the command line but there was no master file in the master.d directory for *name.o*

Name.o: not processed – An error has aborted processing for the named object file.

NAME

mkfs − construct a file system

SYNOPSIS

/etc/mkfs special blocks[:i-nodes] [gap blocks/cyl]
/etc/mkfs special proto [gap blocks/cyl]

DESCRIPTION

Mkfs constructs a file system by writing on the special file according to the directions found in the remainder of the command line. The command waits 10 seconds before starting to construct the file system. If the second argument is given as a string of digits, *mkfs* builds a file system with a single empty directory on it. The size of the file system is the value of *blocks* interpreted as a decimal number. This is the number of *physical* disk blocks the file system will occupy. The boot program is left uninitialized. If the optional number of i-nodes is not given, the default is the number of *logical* blocks divided by 4.

If the second argument is a file name that can be opened, *mkfs* assumes it to be a prototype file *proto*, and will take its directions from that file. The prototype file contains tokens separated by spaces or new-lines. The first token is the name of a file to be copied onto block zero as the bootstrap program. The second token is a number specifying the size of the created file system in *physical* disk blocks. Typically it will be the number of blocks on the device, perhaps diminished by space for swapping. The next token is the number of i-nodes in the file system. The maximum number of i-nodes configurable is 65500. The next set of tokens comprise the specification for the root file. File specifications consist of tokens giving the mode, the user ID, the group ID, and the initial contents of the file. The syntax of the contents field depends on the mode.

The mode token for a file is a 6-character string. The first character specifies the type of the file. (The characters **−bcd** specify regular, block special, character special and directory files respectively.) The second character of the type is either **u** or − to specify set-user-id mode or not. The third is **g** or − for the set-group-id mode. The rest of the mode is a 3 digit octal number giving the owner, group, and other read, write, execute permissions (see *chmod*(1)).

Two decimal number tokens come after the mode; they specify the user and group IDs of the owner of the file.

If the file is a regular file, the next token is a path name whence the contents and size are copied. If the file is a block or character special file, two decimal number tokens follow which give the major and minor device numbers. If the file is a directory, *mkfs* makes the entries . and .. and then reads a list of names and (recursively) files specifications for the entries in the directory. The scan is terminated with the token **$**.

A sample prototype specification follows:

```
/stand/diskboot
4872 110
d--777 3 1
usr     d--777 3 1
        sh      ---755 3 1 /bin/sh
        ken     d--755 6 1
                $
        b0      b--644 3 1 0 0
        c0      c--644 3 1 0 0
        $
$
```

In both command syntaxes, the rotational *gap* and the number of *blocks/cyl* can be specified. The following values are recommended:

Device	Gap Size	Blks/Cyl	
30M Hard Disk	7	90	
10M Hard Disk	7	72	
72M Hard Disk	7	162	(CDC Wren II)
72aM Hard Disk	7	144	(Micropolis)
72bM Hard Disk	7	198	(Priam)
72cM Hard Disk	7	198	(Fujitsu)
Floppy Disk	1	18	

The *default* will be used if the supplied *gap* and *blocks/cyl* are considered illegal values or if a short argument count occurs. The default value is 400 blocks/cyl and gap size 7.

FILES

/etc/vtoc/*

SEE ALSO

chmod(1) in the *3B2 Computer System User Reference Manual.*
dir(4), fs(4) in the *3B2 Computer System Programmer Reference Manual.*

BUGS

If a prototype is used, it is not possible to initialize a file larger than 64K bytes, nor is there a way to specify links.

NAME

mknod – build special file

SYNOPSIS

/etc/mknod name **c** | **b** major minor
/etc/mknod name **p**

DESCRIPTION

Mknod makes a directory entry and corresponding i-node for a special file. The first argument is the *name* of the entry. In the first case, the second is **b** if the special file is block-type (disks, tape) or **c** if it is character-type (other devices). The last two arguments are numbers specifying the *major* device type and the *minor* device (e.g., unit, drive, or line number), which may be either decimal or octal.

The assignment of major device numbers is specific to each system. They have to be dug out of the system source file **conf.c**.

Mknod can also be used to create fifo's (a.k.a named pipes) (second case in *SYNOPSIS* above).

SEE ALSO

mknod(2) in the *3B2 Computer System Programmer Reference Manual.*

NAME

mkunix – create a bootable kernel namelist file, merging kernel and driver symbol tables

SYNOPSIS

mkunix [namelist] [**–o** newlist]

DESCRIPTION

The *mkunix* command will create a bootable kernel namelist file from the UNIX system kernel file and the object files which were loaded by self-config. Typically, *mkunix* would be run following an auto-config boot with a new system configuration. The resulting a.out file can be used as the namelist file for *ps*, *crash*, etc. In addition, this file may be booted directly, bypassing the self-configuration phase of the boot process (see *3b2boot*(1M)). This will save on the order of 30 to 60 seconds at boot time.

Namelist (defaults to the path name specified as the **BOOT** program in the **/etc/system** file) is read to obtain the object, data, and symbol table for the basic kernel. This name, if specified, must be the same as that used in **/etc/system** for the boot line; if not, a warning diagnostic is issued since the resulting namelist file will not be accurate.

The argument *-o newlist* (defaults to **a.out**) is the new file--a bootable image of the currently running UNIX system with the composite symbol table.

SEE ALSO

crash(1M), 3b2boot(8).
ps(1) in the *3B2 Computer System User Reference Manual.*
nm(1) in the *Software Generation System Utilities.*

NAME

mount, umount − mount and dismount file system

SYNOPSIS

/etc/mount [special directory [**−r**]]
/etc/umount special

DESCRIPTION

Mount announces to the system that a removable file system is present on the device *special*. The *directory* must exist already; it becomes the name of the root of the newly mounted file system. The mount system call is used to check the validity of the file system.

These commands maintain a table of mounted devices. If invoked with no arguments, *mount* prints the table.

The optional last argument indicates that the file system is to be mounted read-only. A file system on a physically write-protected media can only be mounted read-only.

Umount announces to the system that the removable file system previously mounted on device *special* is to be removed.

FILES

/etc/mnttab mount table

SEE ALSO

setmnt(1M).
mount(2), umount(2), mnttab(4) in the *3B2 Computer System Programmer Reference Manual.*

DIAGNOSTICS

If the mount system call fails, *mount* prints an appropriate diagnostic. *Mount* issues a warning if the file system to be mounted is currently mounted under another name.

Umount fails if the special file is not mounted or if it is busy. The file system is busy if it contains an open file or some user's working directory.

NAME

mountall – mount all file systems according to a table

SYNOPSIS

/etc/mountall *file-system-table* ...

DESCRIPTION

This command is executed by the super-user, "root", to mount file systems according to a *file-system-table*. The special file name "-" reads from the standard input.

Before each file system is mounted, it is checked using *fsstat*(1M) to see if it appears mountable. If the file system does not appear to be mountable, it is checked, using *fsck*(1M), before the mount is attempted.

EXAMPLES

The following examples are equivalent:

/etc/mountall /etc/fstab /etc/fstab2

/etc/mountall - /etc/fstab2 < /etc/fstab

FILES

File-system-table format:

column 1	block special file name of file system
column 2	mount-point directory
column 3	"-r" if to be mounted readonly
column 4+	ignored

White-space separates columns. Lines beginning with "#" are comments. Empty lines are ignored.

A typical file-system-table might read:

```
/dev/dsk/c1d0s0    /
/dev/dsk/c1d0s2    /usr
```

SEE ALSO

fsck(1M), fsstat(1M), mount(1M).
sysadm(1) in the *3B2 Computer System User Reference Manual.*

DIAGNOSTICS

No messages are printed if the file systems are mountable and clean.

Error and warning messages come from *fsck*(1M), *fsstat*(1M), and *mount*(1M).

NAME

mountfsys, umountfsys − mount (unmount) a diskette file system

SYNOPSIS

mountfsys [−y] [−r]
umountfsys [−y]

DESCRIPTION

The *mountfsys* command mounts a file system that is on a removable disk so that the user can read and write on it. The options provide the following:

−r the file system is mounted read-only.

−y suppresses any questions asked the user during mounting or unmounting.

The *umountfsys* command unmounts the file system.

By default, the user is told the name of the file system on the disk and asked if the file system should be mounted. The optional **−y** argument supresses the questions and mounts or unmounts the file system immediately.

The commands *mountfsys* and *unmountfsys* can be password controlled. See *sysadm*(1), **admpasswd** sub-command.

SEE ALSO

checkfsys(1M), makefsys(1M).
sysadm(1) in the *3B2 Computer System User Reference Manual.*

WARNING

ONCE THE DISK IS MOUNTED IT MUST NOT BE REMOVED FROM THE DISK DRIVE UNTIL IT HAS BEEN UNMOUNTED!

Removing the disk while it is still mounted can cause the data on the disk to be damaged beyond repair.

The *checkfsys*(1M) command can be used to check for and optionally repair a damaged file system on a removable disk.

The identical functions are also available under the *sysadm* commands:

sysadm mountfsys
sysadm umountfsys

BUGS

The *mountfsys* command should refuse to mount a file system that has been removed without unmounting or which was mounted at the time of a system crash.

The hardware should prevent removal of a disk whenever it is opened or mounted.

It should be possible to detect that a write-protected disk and only mount it as "read-only".

A file system that has no label cannot be mounted.

NAME

mvdir – move a directory

SYNOPSIS

/etc/mvdir dirname name

DESCRIPTION

Mvdir moves directories within a file system. *Dirname* must be a directory; *name* must not exist. Neither name may be a sub-set of the other (**/x/y** cannot be moved to **/x/y/z**, nor vice versa).

SEE ALSO

mkdir(1), mv(1) in the *3B2 Computer System User Reference Manual.*

WARNINGS

Only super-user can use *mvdir*.

NAME

ncheck – generate names from i-numbers

SYNOPSIS

/etc/ncheck [–i numbers **] [–a] [–s] [** file-system **]**

DESCRIPTION

Ncheck with no argument generates a path-name vs. i-number list of all files on a set of default file systems. Names of directory files are followed by */..*

The options are as follows:

–i reduces the report to only those files whose i-numbers follow.

–a allows printing of the names **.** and **..**, which are ordinarily suppressed.

–s reduces the report to special files and files with set-user-ID mode; it is intended to discover concealed violations of security policy.

A file system may be specified.

The report should be sorted so that it is more useful.

SEE ALSO

fsck(1M).
sort(1) in the *3B2 Computer System User Reference Manual.*

DIAGNOSTICS

When the file system structure is improper, **??** denotes the "parent" of a parentless file and a path-name beginning with **...** denotes a loop.

NAME

newboot − load lboot and mboot onto the disk boot partition

SYNOPSIS

/etc/newboot [**−y**] lboot mboot boot-special
/etc/newboot [**−y**] olboot mboot boot-special

DESCRIPTION

Newboot replaces the *lboot* and *mboot* files on the given *boot-special* section of a hard disk. In the case of a floppy disk, *newboot* places *olboot* and *mboot* files on the *boot-special* section of a disk. A confirmation is required before the *boot-special* file is overwritten if **−y** is not specified.

Mboot is the 512-byte micro boot file loaded by the boot device firmware and executed to load the larger *lboot* file.

Lboot is a file containing the boot program that is loaded by *mboot* and executed to boot the UNIX system.

Olboot is a file containing the boot program on a floppy diskette that is loaded by *mboot* and executed to boot the absolute operating system.

SEE ALSO

dd(1M), mkboot(1M).

DIAGNOSTICS

Can't open file FILE FILE not found.

WARNINGS

Installing a bad *lboot* or *mboot* may make the affected disk pack unbootable. Be sure you have a good backup disk before *newboot* is run.

NAME

newgrp – log in to a new group

SYNOPSIS

newgrp [**–**] [group]

DESCRIPTION

Newgrp changes a user's group identification. The user remains logged in and the current directory is unchanged, but calculations of access permissions to files are performed with respect to the new real and effective group IDs. The user is always given a new shell, replacing the current shell, by *newgrp*, regardless of whether it terminated successfully or due to an error condition (i.e., unknown group).

Exported variables retain their values after invoking *newgrp*; however, all unexported variables are either reset to their default value or set to null. System variables (such as PS1, PS2, PATH, MAIL, and HOME), unless exported by the system or explicitly exported by the user, are reset to default values. For example, a user has a primary prompt string (**PS1**) other than **$** (default) and has not exported **PS1**. After an invocation of *newgrp* , successful or not, their **PS1** will now be set to the default prompt string **$**. Note that the shell command *export* (see *sh*(1)) is the method to export variables so that they retain their assigned value when invoking new shells.

With no arguments, *newgrp* changes the group identification back to the group specified in the user's password file entry.

If the first argument to *newgrp* is a **–**, the environment is changed to what would be expected if the user actually logged in again.

A password is demanded if the group has a password and the user does not, or if the group has a password and the user is not listed in **/etc/group** as being a member of that group.

FILES

/etc/group	system's group file
/etc/passwd	system's password file

SEE ALSO

environ(5), group(4), passwd(4), in the *3B2 Computer System Programmer Reference Manual.*
login(1), sh(1) in the *3B2 Computer System User Reference Manual.*

BUGS

There is no convenient way to enter a password into **/etc/group**. Use of group passwords is not encouraged, because, by their very nature, they encourage poor security practices. Group passwords may disappear in the future.

NAME

powerdown – stop all processes and turn off the power

SYNOPSIS

powerdown [–y | –Y]

DESCRIPTION

This command brings the system to a state where nothing is running and then turns off the power.

By default, the user is asked questions that control how much warning the other users are given. The options:

–y prevents the questions from being asked and just gives the warning messages. There is a 60 second pause between the warning messages.

–Y is the same as **–y** except it has no pause between messages. It is the fastest way to bring the system down.

Password control can be instituted on this command. See *sysadm*(1), **admpasswd** sub-command.

The identical function is also available under the *sysadm* command:

sysadm powerdown

EXAMPLES

some-long-running-command; powerdown –y

The first command is run to completion and then the machine turns off. This is useful for, say, formatting a document to the printer at the end of a day.

FILES

/etc/shutdown - does the work

SEE ALSO

shutdown(1M).
sysadm(1) in the *3B2 Computer System User Reference Manual.*

NAME

/etc/prtconf - print system configuration

SYNOPSIS

/etc/prtconf

DESCRIPTION

The *prtconf* command prints the system configuration information when executed. As an option, the configuration information can be displayed every time the system is initialized to multi-user mode. This option is turned on by copying the file */etc/prtconf* to the file */etc/prtconfig*. The information printed includes the memory and peripheral configuration.

EXAMPLES

To print the configuration of the 3B2 Computer:

/etc/prtconf

```
AT&T 3B2 SYSTEM CONFIGURATION:

Memory size: 1024 kilobytes
System Peripherals:

SBD         /*System Board*/
FD5         /*Floppy Disk Drive*/
HD30        /*30 Megabyte Hard Disk Drive*/
PORTS       /*Expansion Ports Feature Card*/
```

FILES

/etc/rc.d/sysetup

SEE ALSO

3B2 Computer System Administration Utilities Guide

BUGS

The output in response to the *prtconf* command is not aligned properly.

NAME

prtvtoc – print the volume table of contents of a block device

SYNOPSIS

prtvtoc /dev/rdsk/c?d?s6

DESCRIPTION

Prtvtoc allows the contents of the VTOC (volume table of contents) to be viewed by the user for reference or verification.

The *device* name must be a raw device in the form of */dev/rdsk/c?d?s6*.

NAME

pump – Download B16 or X86 a.out file to a peripheral board

SYNOPSIS

pump /dev/devname file

DESCRIPTION

The *pump* command will read a B16 or X86 **a.out** file's sections into a buffer according to the physical address of the section. *Pump* expects a section in the **a.out** file called ".start". Once it has found this section, *pump* will inform the peripheral to start executing at the address that it found in ".start" after it has downloaded the **a.out** file.

Error Messages

Pump error: UNIX error number – Can't get status of /dev/devname

There may be no **/dev/devname**.

Pump Error: error number – ioctl call

The ioctl call failed. The error number returned can be a UNIX system error number or, in the case of the NI, an error number of 208. Error number 208 is a timeout message. The peripheral board did not respond in time to the request made of it [this is not the only error, see *intro*(2) for a complete list].

Can't open a.out filename for reading!

The error may be that there is no such file or the permissions are such that the file cannot be read [see *chmod*(1)].

Error: Object file is not in b16 or x86 common object format

The file to be downloaded to the peripheral is not a B16 or X86 **a.out** file.

Section size is too big for the buffer

The **a.out** file may be greater that the 32K bytes that is the limit of RAM on the peripheral.

Error: No section name called .start

.I Pump needs ".start" for the starting address that the peripheral needs to execute the downloaded code.

Pump: /dev/devname returned a CIO FAULT during phase

The peripheral encountered a hardware fault during one of the phases of the pump. Phase is one of the following:

reset — This phase will cause a hardware reset on the peripheral.

downlaod — This phase will download the **a.out** file to the peripheral.

force call to function — This phase will inform the peripheral to start executing at the address found in the ".start" section.

sysgen This phase will *sysgen* the peripheral. It allows normal functioning of the peripheral to occur.

Pump: /dev/devname returned a CIO Invalid Queue Entry during phase

The peripheral did not understand the command phase that was issued by *pump*.

Pump: /dev/devname did not respond during phase

The UNIX system driver called may not have understood the command.

Pump: A timeout has occurred on /dev/devname during phase

The peripheral did not respond to one of the commands given.

Pump: There was no return code for /dev/devname during phase

The return code that was given may have been corrupted.

SEE ALSO

intro(2), a.out(4) in the *3B2 Computer System Programmer Reference Manual*.

NAME

pwck, grpck – password/group file checkers

SYNOPSIS

/etc/pwck [file]
/etc/grpck [file]

DESCRIPTION

Pwck scans the password file and notes any inconsistencies. The checks include validation of the number of fields, login name, user ID, group ID, and whether the login directory and optional program name exist. The default password file is **/etc/passwd**.

Grpck verifies all entries in the group file. This verification includes a check of the number of fields, group name, group ID, and whether all login names appear in the password file. The default group file is **/etc/group**.

FILES

/etc/group
/etc/passwd

SEE ALSO

group(4), passwd(4) in the *3B2 Computer System Programmer Reference Manual.*

DIAGNOSTICS

Group entries in **/etc/group** with no login names are flagged.

NAME

rc0 – run commands performed to stop the operating system

SYNOPSIS

/etc/rc0

DESCRIPTION

This file is executed at each system state change that needs to have the system in an inactive state. It is responsible for those actions that bring the system to a quiescent state, traditionally called "shut down".

Normally, init state 0 is taken to mean "stop the operating system", so the inittab entry might read:

```
r0:0:wait:/etc/rc0 > /dev/console 2>&1
```

The recommended sequence for */etc/rc0* is:

Stop System Services, Demons, Accounting, etc.

The */etc/rc0* command executes the files found in the directory **/etc/shutdown.d**. Each of these files terminates some system service. When new services are added that should be terminated when the system is shut down, the appropriate files are installed in **/etc/shutdown.d**.

Terminate Processes

SIGTERM signals are sent to all running processes by *killall*(1M). Processes stop themselves cleanly if sent SIGTERM.

Kill Processes

SIGKILL signals are sent to all remaining processes; no process can resist SIGKILL.

At this point the only processes left are those associated with */etc/rc0* and processes 0 and 1, which are special to the operating system.

Unmount All File Systems

Only the root file system (/) remains mounted.

If the **/etc/inittab** has not defined any other actions to be performed for init stat 0, then the operating system should have nothing to do. It should not be possible to get the system's attention. The only thing that can be done is to turn off the power or possibly get the attention of a firmware monitor.

EXAMPLES

On machines where it is possible to turn off the power under program control, or to reboot, or go to a firmware monitor, one could set up the **/etc/inittab** like this:

```
fl:056:wait:/etc/flash-power-light
r0:056:wait:/etc/rc0 >/dev/console 2>&1
po:0:wait:/etc/poweroff
fw:5:wait:/etc/firmware
rb:6:wait:/etc/reboot
```

The following are prototypical files found in **/etc/shutdown.d**.

errdemon

```
#   Terminate the error logging process
/etc/errstop
```

procacct

```
#   Terminate process accounting
/usr/lib/acct/shutacct
```

FILES

The advice in */etc/rc2*(1M) is appropriate for files in **/etc/shutdown.d**.

Files in **/etc/shutdown.d** that begin with a dot, ".", will not be executed. This feature can be used to "hide" files that are not to be executed for the time being without removing them.

SEE ALSO

killall(1M), rc2(1M), shutdown(1M).

NAME

rc2 − run commands performed for multi-user environment

SYNOPSIS

/etc/rc2

DESCRIPTION

This file is executed at each system state change that goes to one of the numbered states (0 through 6) and is responsible for those initializations that bring the system to a ready-to-use state, traditionally called "multi-user".

The actions performed by */etc/rc2* are found in files in the directory **/etc/rc.d**. These files are executed by **/bin/sh** in ascii sort sequence order. Thus that names of the files are significant if there are interdependencies between them. When functions are added that need to be initialized when the system goes multi-user, an appropriate file should be added in **/etc/rc.d**.

EXAMPLES

The following are prototypical files found in **/etc/rc.d**.

MOUNTFILESYS

```
#   Set up and mount file systems

cd /
> /etc/mnttab
/etc/devnm /  | grep −v swap | grep  −v root | /etc/setmnt

# clean up /tmp
rm −rf /tmp
mkdir /tmp

mount /dev/usr /usr
mount /dev/fs1 /fs1
```

uucp

```
#   clean-up uucp locks, status, and temporary files


rm −f /usr/spool/uucp/LCK* /usr/spool/uucp/STST.* /usr/spool/uucp/TM.
```

The file **etc/TIMEZONE** is included early in */etc/rc2*, thus establishing the default time zone for all commands that follow.

Some hints about files in **/etc/rc.d**: If your world is simple, you can probably get away with a simple naming rule such as "all important things are named with capital letters, everything else starts with lower case." If you have lots of interdependencies and orders are important, using the first character as a sequence indicator may help. Thus files starting with the following characters would be:

[0-9].	very early
[A-Z].	early
[a-n].	later
[o-z].	last

So that the files in **/etc/rc.d** might be named:

```
3.mountfilesys
B.errdemon
c.uucp
r.cron
```

Files in **/etc/rc.d** that begin with a dot, ".", will not be executed. This feature can be used to "hide" files that are not to be executed for time being without removing them.

SEE ALSO

/etc/shutdown(1M).

NAME

setclk – set clock

SYNOPSIS

setclk

DESCRIPTION

The *setclk* command checks the NVRAM only for the correct date. If the date is wrong *setclk* promplts the user to use *sysadm* **datetime** [see *sysadm*(1)] for the proper setting of the hardware clock. *Setclk* is executed through the inittab of the system initialization time.

SEE ALSO

sysadm(1) in the *3B2 Computer System User Reference Manual.*

NAME

setmnt – establish mount table

SYNOPSIS

/etc/setmnt

DESCRIPTION

Setmnt creates the **/etc/mnttab** table which is needed for both the *mount*(1M) and *umount* commands. *Setmnt* reads standard input and creates a *mnttab* entry for each line. Input lines have the format:

filesys node

where *filesys* is the name of the file system's *special file* (e.g., **/dev/dsk/c?d?s?**) and *node* is the root name of that file system. Thus *filesys* and *node* become the first two strings in the mount table entry.

FILES

/etc/mnttab

SEE ALSO

mount(1M).

BUGS

Problems may occur if *filesys* or *node* are longer than 32 characters.
Setmnt silently enforces an upper limit on the maximum number of *mnttab* entries.

NAME

shutdown — shut down system

SYNOPSIS

/etc/shutdown [−y] [−ggrace-period [−iinit-state]

DESCRIPTION

This command is executed by the super-user, "root", to change the state of the machine. By default, it brings the system to a state where only the console has access to the UNIX system. This "root-only" state is traditionally called "single-user".

The command sends a warning message and a final message before it starts actual shutdown activities. By default, the command asks for confirmation before it starts shutting down daemons and killing processes. The options are used as follows:

−y pre-answers the question so the command can be run without user intervention. By default, the time periods between the warning message and the final message and between the final message and the confirmation are 60 seconds.

−g*grace-period*
allows the super-user to specify a different number of seconds.

−i*init-state*
specifies the state that *init*(1M) is to be put in following the warnings, if any. By default, init state "s" is used.

NOTE: THIS VERSION OF SHUTDOWN IS DIFFERENT FROM PREVIOUS VERSIONS.

In the past, the *shutdown* procedure performed process killing and file system unmounts before changing init state. This proved unreliable. Now, the new init state defines what state the machine is to be in and is responsible for making it that way. Recommended definitions are:

state 0
Shut the machine down so it is safe to remove the power. Have the machine remove power if it can.

state 2
Bring machine to state traditionally called multi-user.

state 5
Stop the UNIX system and go to the firmware monitor.

state 6
Stop the UNIX system and reboot.

SEE ALSO

init(1M), rc0(1M), rc2(1M).

NAME

su − become super-user or another user

SYNOPSIS

su [−] [name [arg ...]]

DESCRIPTION

Su allows one to become another user without logging off. The default user *name* is **root** (i.e., super-user).

To use *su*, the appropriate password must be supplied (unless one is already **root**). If the password is correct, *su* will execute a new shell with the real and effective user ID set to that of the specified user. The new shell will be the optional program named in the shell field of the specified user's password file entry (see *passwd*(4)), or **/bin/sh** if none is specified (see *sh*(1)). To restore normal user ID privileges, type an **EOF** (*cntrl-d*) to the new shell.

Any additional arguments given on the command line are passed to the program invoked as the shell. When using programs like *sh*(1), an *arg* of the form **−c** *string* executes *string* via the shell and an arg of **−r** will give the user a restricted shell.

The following statements are true only if the optional program named in the shell field of the specified user's password file entry is like *sh*(1). If the first argument to *su* is a −, the environment will be changed to what would be expected if the user actually logged in as the specified user. This is done by invoking the program used as the shell with an *arg0* value whose first character is −, thus causing first the system's profile (**/etc/profile**) and then the specified user's profile (**.profile** in the new HOME directory) to be executed. Otherwise, the environment is passed along with the possible exception of **$PATH**, which is set to **/bin:/etc:/usr/bin** for **root**. Note that if the optional program used as the shell is **/bin/sh**, the user's **.profile** can check *arg0* for **−sh** or **−su** to determine if it was invoked by *login*(1) or *su*(1), respectively. If the user's program is other than **/bin/sh**, then **.profile** is invoked with an *arg0* of *-program* by both *login*(1) and *su*(1).

All attempts to become another user using *su* are logged in the log file **/usr/adm/sulog**.

EXAMPLES

To become user **bin** while retaining your previously exported environment, execute:

```
su bin
```

To become user **bin** but change the environment to what would be expected if **bin** had originally logged in, execute:

```
su - bin
```

To execute *command* with the temporary environment and permissions of user **bin**, type:

```
su - bin -c "command args"
```

FILES

/etc/passwd	system's password file
/etc/profile	system's profile
$HOME/.profile	user's profile
/usr/adm/sulog	log file

SEE ALSO

env(1), login(1), sh(1) in the *3B2 Computer System User Reference Manual.*
environ(5), passwd(4), profile(4) in the *3B2 Computer Programmer Reference Manual.*

NAME

sync – update the super block

SYNOPSIS

sync

DESCRIPTION

Sync executes the *sync* system primitive. If the system is to be stopped, *sync* must be called to insure file system integrity. It will flush all previously unwritten system buffers out to disk, thus assuring that all file modifications up to that point will be saved. See *sync*(2) for details.

SEE ALSO

sync(2) in the *3B2 Computer System Programmer Reference Manual.*

NAME

sysdef – system definition

SYNOPSIS

/etc/sysdef [opsys [master.d]]

DESCRIPTION

Sysdef analyzes the named operating system file (opsys) and extracts configuration information. The operating system file must be an "absolute" boot file [see *mkunix*(1M)]. All hardware devices, their local bus addresses, and unit count, as well as pseudo devices, system devices, and loadable modules are listed. In addition, the values of all tunable parameters are listed. The output of *sysdef* is in tabular form.

FILES

/unix	default operating system file
/etc/master.d/*	default directory containing master files

SEE ALSO

mkunix(1M).
master(4), nlist(3C) in the *3B2 Computer System Programmer Reference Manual.*

DIAGNOSTICS

internal name list overflow
if the master table contains more than an internally specified number of entries for use by *nlist*(3C).

NAME

uadmin – administrative control

SYNOPSIS

uadmin cmd fcn

DESCRIPTION

The *uadmin* command provides control for basic administrative functions. This command is tightly coupled to the system administration procedures and is not intended for general use. It may be invoked only by the super-user.

The arguments are converted to integers and passed to the *uadmin* system call.

SEE ALSO

uadmin(2) in the *3B2 Computer System Programmer Reference Manual.*

NAME

umountall – unmount all file systems except root

SYNOPSIS

etc/umountall [–k]

DESCRIPTION

The *umountall* command is executed by the super-user, "root", to unmount all currently mounted file systems except the root file system.

The **–k** option causes *fuser*(1M) to send a SIGKILL signal to all processes that have files open in each file system before it is unmounted. Without **–k** it is possible that an unmount may fail because the file system is busy.

EXAMPLES

/etc/umountall

/etc/umountall -k

DIAGNOSTICS

No messages are printed if the file systems are unmountable.

Error and warning messages come from *umount*(1M).

SEE ALSO

fuser(1M), mountall(1M), umount(1M).
sysadm(1) in the *3B2 Computer System User Reference Manual.*
signal(2) in the *3B2 Computer System Programmer Reference Manual.*

NAME

whodo – who is doing what

SYNOPSIS

/etc/whodo

DESCRIPTION

Whodo produces merged, reformatted, and dated output from the *who*(1) and *ps*(1) commands.

FILES

etc/passwd

SEE ALSO

ps(1), who(1) in the *3B2 Computer System User Reference Manual.*

NAME

intro – introduction to special files

DESCRIPTION

This section describes various special files that refer to specific hardware peripherals and UNIX system device drivers. The names of the entries are generally derived from names for the hardware, as opposed to the names of the special files themselves. Characteristics of both the hardware device and the corresponding UNIX system device driver are discussed where applicable.

Disk device file names are in the following format:

/dev/{r}dsk/c#d#s#

where **r** indicates a raw interface to the disk, the **c#** indicates the controller number, **d#** indicates the device attached to the controller and **s#** indicates the section number of the partitioned device.

SEE ALSO

Disk Partitioning in the *3B2 Computer System Administration Utilities Guide*.

NAME

console - console interface

DESCRIPTION

The console provides the operator interface to the 3B2 computer.

The file **/dev/console** refers to the system console. This special file implements the features described in termio(7).

The file **/dev/contty** refers to an asynchronous serial data line originating from the system board. This special file implements the features described in termio(7).

FILES

/dev/console
/dev/contty

SEE ALSO

termio(7).

NAME

hdelog — hard disk error log interface

DESCRIPTION

Hdelog is a special file that provides access to the disk error logging mechanism, the equipped disk table, and the disk drivers of the equipped disks for doing physical (non-partitioned) disk I/O. It is an internal interface of bad block handling and a few other disk utilities and is not intended to be used directly by users. You must be super-user to use it. This command is a bad block handling utility command. You must be super-user to use it.

FILES

/dev/hdelog

SEE ALSO

hdeadd(1M), hdefix(1M), hdelogger(1M).

NAME

id – 3B2 Computer Integral Disk Subsystem

DESCRIPTION

The 3B2 Computer integral disk subsystem supports three types of drives:

10 megabyte
30 megabyte
72 megabyte

Each drive contains only fixed media.

The 3B2 Computer may contain a maximum of two integral disk drives. The core system supports one drive and with the addition of an expansion module a second drive may be used.

Each drive has its own volume table of contents (VTOC). This table contains information about the distribution of data across the media. This distribution is specified by partitions allowing the media to be broken up into more manageable pieces. The media may be divided into as many as sixteen partitions (0x0-0xf).

All files refering to disks reside under the directory, **/dev/dsk**. Each file name is broken into three parts, the controller number, drive number, and slice (or partition) number. The controller number for the integral hard disks is 1. Since there can be up to two integral drives, the files are **c1d[0-1]s[0-f]**. There is one partition which is restricted for special use, this is partition 6 (**c1d[0-1]s6**). Partition 6 will specify the full disk. If a copy of an entire disk is being done, partition 6 must be specified. This will ensure that the VTOC and boot code (if a boot device) has been copied along with the data. If on a second disk, partitions must start after the VTOC (and boot area if a bootable device). The VTOC always resides on the second sector (sector 1 when sectors are numbered 0–n). The remaining partitions are not restricted and can be used in any manner.

The files in **/dev/dsk** access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a "raw" interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient. The names of the raw disk files are the same but they reside in the directory **/dev/rdsk**.

In raw I/O, the buffer must begin on a word boundary and transfer counts can be as small as a single byte.

FILES

/dev/dsk/c1d[1-0]s[0-f]
/dev/rdsk/c1d[1-0]s[0-f]
/etc/fmthard
/etc/vtoc/*
/etc/prtvtoc

NAME

if – 3B2 Computer Floppy Disk Subsystem

DESCRIPTION

The 3B2 Computer floppy disk subsystem consists of floppy drive(s). The media contains 1422 blocks. The files **/dev/dsk/c0d0s0**, ..., **/dev/dsk/c0d0s7** refer to sections of the floppy disk drive. This slicing allows the media to be broken up into more manageable pieces.

The origin and size of the sections on each drive are as follows:

SIZE	START CYL.	PARTITION
990	24	partition 0—cyl 24-78 (root)
810	34	partition 1—cyl 34-78
612	45	partition 2—cyl 45-78
414	56	partition 3—cyl 56-78
216	67	partition 4—cyl 67-78
1404	1	partition 5—cyl 1-78
1422	0	partition 6—cyl 0-78 (full disk)
18	0	partition 7—cyl 0 (boot)

The start address is a cylinder address, with length representing the number of blocks within the section.

The **/dev/dsk/c0d0s*** files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a "raw" interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw disk files begin with **/dev/dsk/c0d0s*** and end with a number which selects the same disk section as the corresponding **/dev/dsk/c0d0s*** file.

In raw I/O the buffer must begin on a word boundary, and transfer counts can be as small as single byte.

FILES

/dev/ifdsk*
/dev/rifdsk*
/dev/dsk/c0d0s*
/dev/dsk/c0d0s*

NAME

mem, kmem — core memory

DESCRIPTION

Mem is a special file that is an image of the core memory of the computer. It may be used, for example, to examine, and even to patch the system.

Byte addresses in *mem* are interpreted as memory addresses. References to non-existent locations cause errors to be returned.

Examining and patching device registers is likely to lead to unexpected results when read-only or write-only bits are present.

The file *kmem* is the same as *mem* except that kernel virtual memory rather than physical memory is accessed.

The I/O addresses begin at location 0x60000 of *kmem* and per-process data for the current process begins at 0x80880000.

FILES

/dev/mem
/dev/kmem

BUGS

Some of *kmem* cannot be read because of write-only addresses or unequipped memory addresses.

NAME

null – the null file

DESCRIPTION

Data written on a null special file is discarded.

Reads from a null special file always return 0 bytes.

FILES

/dev/null

NAME

port - 5 line asynchronous interface

DESCRIPTION

Each of the five lines attached to a port behaves as described in *termio*(7). Each port supports 4 RS232 lines and one parallel Centronics interface. The **c_cflag** of B50, B75, B200, EXTA, and EXTB are not available.

FILES

/dev/tty?? serial interface

SEE ALSO

termio(7).

NAME

prf – operating system profiler

DESCRIPTION

The file **prf** provides access to activity information in the operating system. Writing the file loads the measurement facility with text addresses to be monitored. Reading the file returns these addresses and a set of counters indicative of activity between adjacent text addresses.

The recording mechanism is driven by the system clock and samples the program counter at line frequency. Samples that catch the operating system are matched against the stored text addresses and increment corresponding counters for later processing.

The file **prf** is a pseudo-device with no associated hardware.

FILES

/dev/prf

SEE ALSO

config(1M), profiler(1M).

NAME

sxt — pseudo-device driver

DESCRIPTION

Sxt is a pseudo-device driver that interposes a discipline between the standard *tty* line disciplines and a real device driver. The standard disciplines manipulate *virtual tty* structures (channels) declared by the *sxt* driver. *Sxt* acts as a discipline manipulating a *real tty* structure declared by a real device driver. The *sxt* driver is currently only used by the *shl*(1) command.

Virtual ttys are named by inodes in the subdirectory **/dev/sxt** and are allocated in groups of up to eight. To allocate a group, a program should exclusively open a file with a name of the form **/dev/sxt/??0** (channel 0) and then execute a SXTIOCLINK *ioctl* call to initiate the multiplexing.

Only one channel, the *controlling* channel, can receive input from the keyboard at a time; others attempting to read will be blocked.

There are two groups of *ioctl*(2) commands supported by *sxt*. The first group contains the standard *ioctl* commands described in *termio*(7), with the addition of the following:

TIOCEXCL Set *exclusive use* mode: no further opens are permitted until the file has been closed.

TIOCNXCL Reset *exclusive use* mode: further opens are once again permitted.

The second group are directives to *sxt* itself. Some of these may only be executed on channel 0.

SXTIOCLINK Allocate a channel group and multiplex the virtual ttys onto the real tty. The argument is the number of channels to allocate. This command may only be executed on channel 0. Possible errors include:

- EINVAL The argument is out of range.
- ENOTTY The command was not issued from a real tty.
- ENXIO *linesw* is not configured with *sxt*.
- EBUSY An SXTIOCLINK command has already been issued for this real *tty*.
- ENOMEM There is no system memory available for allocating the virtual tty structures.
- EBADF Channel 0 was not opened before this call.

SXTIOCSWTCH Set the controlling channel. Possible errors include:

- EINVAL An invalid channel number was given.
- EPERM The command was not executed from channel 0.

SXTIOCWF Cause a channel to wait until it is the controlling channel. This command will return the error, *EINVAL*, if an invalid channel number is given.

SXTIOCUBLK Turn off the **loblk** control flag in the virtual tty of the indicated channel. The error *EINVAL* will be returned if an invalid number or channel 0 is given.

SXTIOCSTAT Get the status (blocked on input or output) of each channel and store in the *sxtblock* structure referenced by the argument. The error *EFAULT* will be returned if the structure cannot be written.

SXTIOCTRACE Enable tracing. Tracing information is written to the console on the 3B2 Computer. This command has no effect if tracing is not configured.

SXTIOCNOTRACE Disable tracing. This command has no effect if tracing is not configured.

FILES

/dev/sxt/??[0-7] Virtual tty devices

SEE ALSO

termio(7).
shl(1), stty(1) in the *3B2 Computer System User Reference Manual.*
ioctl(2), open(2) in the *3B2 Computer System Programmer Reference Manual.*

NAME

termio – general terminal interface

DESCRIPTION

All of the asynchronous communications ports use the same general interface, no matter what hardware is involved. The remainder of this section discusses the common features of this interface.

When a terminal file is opened, it normally causes the process to wait until a connection is established. In practice, user's programs seldom open these files; they are opened by *getty* and become a user's standard input, output, and error files. The very first terminal file opened by the process group leader of a terminal file not already associated with a process group becomes the *control terminal* for that process group. The control terminal plays a special role in handling quit and interrupt signals, as discussed below. The control terminal is inherited by a child process during a *fork*(2). A process can break this association by changing its process group using *setpgrp*(2).

A terminal associated with one of these files ordinarily operates in full-duplex mode. Characters may be typed at any time, even while output is occurring, and are only lost when the system's character input buffers become completely full, which is rare, or when the user has accumulated the maximum allowed number of input characters that have not yet been read by some program. Currently, this limit is 256 characters. When the input limit is reached, all the saved characters are thrown away without notice.

Normally, terminal input is processed in units of lines. A line is delimited by a new-line (ASCII LF) character, an end-of-file (ASCII EOT) character, or an end-of-line character. This means that a program attempting to read will be suspended until an entire line has been typed. Also, no matter how many characters are requested in the read call, at most one line will be returned. It is not, however, necessary to read a whole line at once; any number of characters may be requested in a read, even one, without losing information.

During input, erase and kill processing is normally done. By default, the character # erases the last character typed, except that it will not erase beyond the beginning of the line. By default, the character @ kills (deletes) the entire input line, and optionally outputs a new-line character. Both these characters operate on a key-stroke basis, independently of any backspacing or tabbing that may have been done. Both the erase and kill characters may be entered literally by preceding them with the escape character (\). In this case the escape character is not read. The erase and kill characters may be changed.

Certain characters have special functions on input. These functions and their default character values are summarized as follows:

INTR (Rubout or ASCII DEL) generates an *interrupt* signal which is sent to all processes with the associated control terminal. Normally, each such process is forced to terminate, but arrangements may be made either to ignore the signal or to receive a trap to an agreed-upon location; see *signal*(2).

QUIT (Control-| or ASCII FS) generates a *quit* signal. Its treatment is identical to the interrupt signal except that, unless a receiving process has made other arrangements, it will not only be terminated but a core image file (called **core**) will be created in the current working directory.

SWTCH (Control-z or ASCII SUB) is used by the job control facility, *shl*, to change the current layer to the control layer.

ERASE (#) erases the preceding character. It will not erase beyond the start of a line, as delimited by a NL, EOF, or EOL character.

KILL (@) deletes the entire line, as delimited by a NL, EOF, or EOL character.

EOF (Control-d or ASCII EOT) may be used to generate an end-of-file from a terminal. When received, all the characters waiting to be read are immediately passed to the program, without waiting for a new-line, and the EOF is discarded. Thus, if there are no characters waiting, which is to say the EOF occurred at the beginning of a line, zero characters will be passed back, which is the standard end-of-file indication.

NL (ASCII LF) is the normal line delimiter. It can not be changed or escaped.

EOL (ASCII NUL) is an additional line delimiter, like NL. It is not normally used.

STOP (Control-s or ASCII DC3) can be used to temporarily suspend output. It is useful with CRT terminals to prevent output from disappearing before it can be read. While output is suspended, STOP characters are ignored and not read.

START (Control-q or ASCII DC1) is used to resume output which has been suspended by a STOP character. While output is not suspended, START characters are ignored and not read. The start/stop characters can not be changed or escaped.

The character values for INTR, QUIT, SWTCH, ERASE, KILL, EOF, and EOL may be changed to suit individual tastes. The ERASE, KILL, and EOF characters may be escaped by a preceding \ character, in which case no special function is done.

When the carrier signal from the data-set drops, a *hang-up* signal is sent to all processes that have this terminal as the control terminal. Unless other arrangements have been made, this signal causes the processes to terminate. If the hang-up signal is ignored, any subsequent read returns with an end-of-file indication. Thus, programs that read a terminal and test for end-of-file can terminate appropriately when hung up on.

When one or more characters are written, they are transmitted to the terminal as soon as previously-written characters have finished typing. Input characters are echoed by putting them in the output queue as they arrive. If a process produces characters more rapidly than they can be typed, it will be suspended when its output queue exceeds some limit. When the queue has drained down to some threshold, the program is resumed.

Several *ioctl*(2) system calls apply to terminal files. The primary calls use the following structure, defined in **<termio.h>**:

```
#define NCC 8
struct termio {
        unsigned short c_iflag;      /* input modes */
        unsigned short c_oflag;      /* output modes */
        unsigned short c_cflag;      /* control modes */
        unsigned short c_lflag;      /* local modes */
        char           c_line;       /* line discipline */
        unsigned char  c_cc[NCC];    /* control chars */
};
```

The special control characters are defined by the array *c_cc*. The relative positions and initial values for each function are as follows:

0	VINTR	DEL
1	VQUIT	FS
2	VERASE	#
3	VKILL	@
4	VEOF	EOT
5	VEOL	NUL
6	reserved	
7	SWTCH	

The *c_iflag* field describes the basic terminal input control:

IGNBRK	0000001	Ignore break condition.
BRKINT	0000002	Signal interrupt on break.
IGNPAR	0000004	Ignore characters with parity errors.
PARMRK	0000010	Mark parity errors.
INPCK	0000020	Enable input parity check.
ISTRIP	0000040	Strip character.
INLCR	0000100	Map NL to CR on input.
IGNCR	0000200	Ignore CR.
ICRNL	0000400	Map CR to NL on input.
IUCLC	0001000	Map upper-case to lower-case on input.
IXON	0002000	Enable start/stop output control.
IXANY	0004000	Enable any character to restart output.
IXOFF	0010000	Enable start/stop input control.

If IGNBRK is set, the break condition (a character framing error with data all zeros) is ignored, that is, not put on the input queue and therefore not read by any process. Otherwise if BRKINT is set, the break condition will generate an interrupt signal and flush both the input and output queues. If IGNPAR is set, characters with other framing and parity errors are ignored.

If PARMRK is set, a character with a framing or parity error which is not ignored is read as the three-character sequence: 0377, 0, X, where X is the data of the character received in error. To avoid ambiguity in this case, if ISTRIP is not set, a valid character of 0377 is read as 0377, 0377. If PARMRK is not set, a framing or parity error which is not ignored is read as the character NUL (0).

If INPCK is set, input parity checking is enabled. If INPCK is not set, input parity checking is disabled. This allows output parity generation without input parity errors.

If ISTRIP is set, valid input characters are first stripped to 7-bits, otherwise all 8-bits are processed.

If INLCR is set, a received NL character is translated into a CR character. If IGNCR is set, a received CR character is ignored (not read). Otherwise if ICRNL is set, a received CR character is translated into a NL character.

If IUCLC is set, a received upper-case alphabetic character is translated into the corresponding lower-case character.

If IXON is set, start/stop output control is enabled. A received STOP character will suspend output and a received START character will restart output. All start/stop characters are ignored and not read. If IXANY is set, any input character, will restart output which has been suspended.

If IXOFF is set, the system will transmit START/STOP characters when the input queue is nearly empty/full.

The initial input control value is all-bits-clear.

The *c_oflag* field specifies the system treatment of output:

OPOST	0000001	Postprocess output.
OLCUC	0000002	Map lower case to upper on output.
ONLCR	0000004	Map NL to CR-NL on output.
OCRNL	0000010	Map CR to NL on output.
ONOCR	0000020	No CR output at column 0.
ONLRET	0000040	NL performs CR function.
OFILL	0000100	Use fill characters for delay.
OFDEL	0000200	Fill is DEL, else NUL.
NLDLY	0000400	Select new-line delays:
NL0	0	
NL1	0000400	
CRDLY	0003000	Select carriage-return delays:
CR0	0	
CR1	0001000	
CR2	0002000	
CR3	0003000	
TABDLY	0014000	Select horizontal-tab delays:
TAB0	0	
TAB1	0004000	
TAB2	0010000	
TAB3	0014000	Expand tabs to spaces.
BSDLY	0020000	Select backspace delays:
BS0	0	
BS1	0020000	
VTDLY	0040000	Select vertical-tab delays:
VT0	0	
VT1	0040000	
FFDLY	0100000	Select form-feed delays:
FF0	0	
FF1	0100000	

If OPOST is set, output characters are post-processed as indicated by the remaining flags, otherwise characters are transmitted without change.

If OLCUC is set, a lower-case alphabetic character is transmitted as the corresponding upper-case character. This function is often used in conjunction with IUCLC.

If ONLCR is set, the NL character is transmitted as the CR-NL character pair. If OCRNL is set, the CR character is transmitted as the NL character. If ONOCR is set, no CR character is transmitted when at column 0 (first position). If ONLRET is set, the NL character is assumed to do the carriage-return function; the column pointer will be set to 0 and the delays specified for CR will be used. Otherwise the NL character is assumed to do just the line-feed function; the column pointer will remain unchanged. The column pointer is also set to 0 if the CR character is actually transmitted.

The delay bits specify how long transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. In all cases a value of 0 indicates no delay. If OFILL is set, fill characters will be transmitted for delay instead of a timed delay. This is useful for high baud rate terminals which need only a minimal delay. If OFDEL is set, the fill character is DEL, otherwise NUL.

If a form-feed or vertical-tab delay is specified, it lasts for about 2 seconds.

New-line delay lasts about 0.10 seconds. If ONLRET is set, the carriage-return delays are used instead of the new-line delays. If OFILL is set, two fill characters will be transmitted.

Carriage-return delay type 1 is dependent on the current column position, type 2 is about 0.10 seconds, and type 3 is about 0.15 seconds. If OFILL is set, delay type 1 transmits two fill characters, and type 2, four fill characters.

Horizontal-tab delay type 1 is dependent on the current column position. Type 2 is about 0.10 seconds. Type 3 specifies that tabs are to be expanded into spaces. If OFILL is set, two fill characters will be transmitted for any delay.

Backspace delay lasts about 0.05 seconds. If OFILL is set, one fill character will be transmitted.

The actual delays depend on line speed and system load.

The initial output control value is all bits clear.

The *c_cflag* field describes the hardware control of the terminal:

CBAUD	0000017	Baud rate:
B0	0	Hang up
B600	0000010	600 baud
B1200	0000011	1200 baud
B1800	0000012	1800 baud
B2400	0000013	2400 baud
B4800	0000014	4800 baud
B9600	0000015	9600 baud
EXTA	0000016	External A
EXTB	0000017	External B
CSIZE	0000060	Character size:
CS5	0	5 bits
CS6	0000020	6 bits
CS7	0000040	7 bits
CS8	0000060	8 bits
CSTOPB	0000100	Send two stop bits, else one.
CREAD	0000200	Enable receiver.
PARENB	0000400	Parity enable.
PARODD	0001000	Odd parity, else even.
HUPCL	0002000	Hang up on last close.
CLOCAL	0004000	Local line, else dial-up.
LOBLK	0010000	Block layer output.

The CBAUD bits specify the baud rate. The zero baud rate, B0, is used to hang up the connection. If B0 is specified, the data-terminal-ready signal will not be asserted. Normally, this will disconnect the line. For any particular hardware, impossible speed changes are ignored.

The CSIZE bits specify the character size in bits for both transmission and reception. This size does not include the parity bit, if any. If CSTOPB is set, two stop bits are used, otherwise one stop bit. For example, at 110 baud, two stops bits are required.

If PARENB is set, parity generation and detection is enabled and a parity bit is added to each character. If parity is enabled, the PARODD flag specifies odd parity if set, otherwise even parity is used.

If CREAD is set, the receiver is enabled. Otherwise no characters will be received.

If HUPCL is set, the line will be disconnected when the last process with the line open closes it or terminates. That is, the data-terminal-ready signal will not be asserted.

If CLOCAL is set, the line is assumed to be a local, direct connection with no modem control. Otherwise modem control is assumed.

If LOBLK is set, the output of a job control layer will be blocked when it is not the current layer. Otherwise the output generated by that layer will be multiplexed onto the current layer.

The initial hardware control value after open is B300, CS8, CREAD, HUPCL.

The *c_lflag* field of the argument structure is used by the line discipline to control terminal functions. The basic line discipline (0) provides the following:

ISIG	0000001	Enable signals.
ICANON	0000002	Canonical input (erase and kill processing).
XCASE	0000004	Canonical upper/lower presentation.
ECHO	0000010	Enable echo.
ECHOE	0000020	Echo erase character as BS-SP-BS.
ECHOK	0000040	Echo NL after kill character.
ECHONL	0000100	Echo NL.
NOFLSH	0000200	Disable flush after interrupt or quit.

If ISIG is set, each input character is checked against the special control characters INTR, SWTCH, and QUIT. If an input character matches one of these control characters, the function associated with that character is performed. If ISIG is not set, no checking is done. Thus these special input functions are possible only if ISIG is set. These functions may be disabled individually by changing the value of the control character to an unlikely or impossible value (e.g., 0377).

If ICANON is set, canonical processing is enabled. This enables the erase and kill edit functions, and the assembly of input characters into lines delimited by NL, EOF, and EOL. If ICANON is not set, read requests are satisfied directly from the input queue. A read will not be satisfied until at least MIN characters have been received or the timeout value TIME has expired between characters. This allows fast bursts of input to be read efficiently while still allowing single character input. The MIN and TIME values are stored in the position for the EOF and EOL characters, respectively. The time value represents tenths of seconds.

If XCASE is set, and if ICANON is set, an upper-case letter is accepted on input by preceding it with a \ character, and is output preceded by a \ character. In this mode, the following escape sequences are generated on output and accepted on input:

for:	*use*:
`	\'
|	\!
~	\^
{	\(
}	\)
\	\\

For example, **A** is input as **\a**, **\n** as **\\n**, and **\N** as **\\\n**.

If ECHO is set, characters are echoed as received.

When ICANON is set, the following echo functions are possible. If ECHO and ECHOE are set, the erase character is echoed as ASCII BS SP BS, which will clear the last character from a CRT screen. If ECHOE is set and ECHO is not set, the erase character is echoed as ASCII SP BS. If ECHOK is set, the NL character will be echoed after the kill character to emphasize that the line will be deleted. Note that an escape character preceding the erase or kill character removes any special function. If ECHONL is set, the NL character will be

echoed even if ECHO is not set. This is useful for terminals set to local echo (so-called half duplex). Unless escaped, the EOF character is not echoed. Because EOT is the default EOF character, this prevents terminals that respond to EOT from hanging up.

If NOFLSH is set, the normal flush of the input and output queues associated with the quit, switch, and interrupt characters will not be done.

The initial line-discipline control value is all bits clear.

The primary *ioctl*(2) system calls have the form:

```
ioctl (fildes, command, arg)
struct termio *arg;
```

The commands using this form are:

TCGETA — Get the parameters associated with the terminal and store in the *termio* structure referenced by **arg**.

TCSETA — Set the parameters associated with the terminal from the structure referenced by **arg**. The change is immediate.

TCSETAW — Wait for the output to drain before setting the new parameters. This form should be used when changing parameters that will affect output.

TCSETAF — Wait for the output to drain, then flush the input queue and set the new parameters.

Additional *ioctl*(2) calls have the form:

```
ioctl (fildes, command, arg)
int arg;
```

The commands using this form are:

TCSBRK — Wait for the output to drain. If *arg* is 0, then send a break (zero bits for 0.25 seconds).

TCXONC — Start/stop control. If *arg* is 0, suspend output; if 1, restart suspended output.

TCFLSH — If *arg* is 0, flush the input queue; if 1, flush the output queue; if 2, flush both the input and output queues.

FILES

/dev/tty*

SEE ALSO

stty(1) in the *3B2 Computer System User Reference Manual.*
fork(2), ioctl(2), setpgrp(2), signal(2) in the *3B2 Computer System Programmer Reference Manual.*

NAME

tty – controlling terminal interface

DESCRIPTION

The file **/dev/tty** is, in each process, a synonym for the control terminal associated with the process group of that process, if any. It is useful for programs or shell sequences that wish to be sure of writing messages on the terminal no matter how output has been redirected. It can also be used for programs that demand the name of a file for output, when typed output is desired and it is tiresome to find out what terminal is currently in use.

FILES

/dev/tty
/dev/tty*

SEE ALSO

console(7), ports(7).

NAME

intro — introduction to system maintenance procedures

DESCRIPTION

This section outlines certain procedures that will be of interest to those charged with the task of system maintenance. Included are discussions on such topics as boot procedures, recovery from crashes, file backups, etc.

BUGS

No manual can take the place of good, solid experience.

NAME

3B2boot – 3B2 computer bootstrap procedures

DESCRIPTION

The bootstrap procedure on the 3B2 computer consists of three phases. The first phase is initiated by the user typing boot while in firmware. This phase will prompt for the name of the file to be booted, then the device from which to boot. (This phase is also initiated after power-up, and upon completion of diagnostics. In this case, the file to boot defaults to unix and the device to boot defaults to id.) The second phase is the execution of *mboot* which reads from the boot block of the selected boot device. *Mboot* is loaded at physical address 0x2004000. Its function is to read the *lboot* phase from the remainder of the boot partition of into the top of the first half megabyte of main memory and to initiate *lboot* execution.

Any error during the *mboot* phase will cause a return to the firmware.

The *lboot* phase has been passed the name of the file to be booted. If the name given is a directory on the init file system, *lboot* will respond with a listing of the files present in that directory. Once a valid file name has been obtained, *lboot* loads the file into memory and starts the execution of the file.

SEE ALSO

newboot(1M).
a.out(4) in the *3B2 Computer System Programmer Reference Manual.*

DIAGNOSTICS

Self-explanatory messages about bad directory entries and bad file formats.

BUGS

Lboot isn't smart enough to know which *a.out* files can be used as bootable programs. If an *a.out* is specified that is not bootable, *lboot* will load it and branch to it. What happens after that is unpredictable.

NAME

dgn - diagnose computer

SYNOPSIS

dgn
dgn [unit[=number]]
dgn [unit[=number]] [ph=a[-b]] [rep=n] [ucl] [soak]
l(ist) unit
h(elp)
s(how)

(phase range specified ph=a-b means phases a through b)

DESCRIPTION

The Diagnostic Monitor Utility allows diagnostics to be run on the 3B2 Computer via the system console. The interface and the entry into the diagnostic mode is discussed in detail in the *3B2 Computer System Administration Utilities Guide.* Diagnostics can be invoked for the entire computer, types of devices (e.g., all ports boards), a specific device (e.g., the system board), or a particular phase or range of phases for the device or device type. Each diagnostic phase and phase description can be found by listing the diagnostic phases for each computer device (eg. l sbd). The Monitor will list the 3B2 Computer physical devices with the s(how) command.

Types of diagnostic phases:

Normal-Diagnostics that run every time the computer is powered up

Demand-Diagnostics that run during soak or must be specifically requested

Interactive-Diagnostics that must be specifically requested and may cause loss of stored data or require operator intervention.

(Diagnostic Monitor will deny request for diagnostics on unequipped devices and non-existent phases)

Option definitions:

ucl -unconditional execution, run all specified phases and display all failing results

rep=n -repeat phases(s) n times

ph=a -select individual phase

ph=a-b -select phase range a through b

soak -silently and continuouslyontinuously run normal and demand diagnostics for specified range (default: all of phase table) and for specified repititions (default: continuous-stopped with keyboard entry)

EXAMPLES

dgn	(full system)
dgn ports	(all ports devices)
dgn sbd 0 ucl	(Unconditional execution)
dgn ports 0	(ports 0 diagnostic)

dgn ports 1 ucl

dgn sbd ph=3 (phase specification)

dgn sbd ph=1-5 (phase range specification)

dgn sbd soak (diagnostic system board soak)

Whenever specific phases are requested, the device to be tested must be designated.

DIAGNOSTICS

Improperly typed syntax will generate message or specify the invalid input. The "h" command will generate a listing of the correct syntax for the system board firmware.

NAME

ports – create character device files and inittab entries for ports boards automatically

SYNOPSIS

ports

DESCRIPTION

Ports will create character device files in **/dev** and add new entries in **/etc/inittab** for 4 asynchronous RS-232 ports and 1 parallel printer port. Each port will be named

tty[slot#][1-5]

If the board configuration has changed, *ports* will do the following:

Remove any tty device files for a board that no longer resides in a slot.

Remove device files of other boards such as the NI if a ports board now resides in the slot that previously held an NI. A warning message would be sent to the console that a device file was being removed.

Make new tty device files for the ports boards if needed.

Make new inittab entries for the ports boards.

If the configuration has not changed, the ports program will exit without doing anything.

Any devices, such as a printer or a modem, that are added to a ports board should link the names that are to be used for the devices to the corresponding tty device files that were created [see *ln*(1)].

EXAMPLE

A parallel printer is added to a ports board that is in slot 1. The corresponding slot would be tty15. The user should use *ln* to link an appropriate name such as lp1 to the tty device file.

ln /dev/tty15 /dev/lp1

FILES

/etc/inittab
/dev/tty[stol#][1-5]

SEE ALSO

ln(1) in the *3B2 Computer System User Reference Manual.*

WARNINGS

A warning will be issued for each device file that is removed provided it is not a tty device file. If a ports board has been removed and lp1 has been linked to a tty device file, the message would be as follows:

Warning: /dev/lp1 is being removed.

NAME

sysdump - boot option to dump system memory image to floppy disk(s)

SYNOPSIS

sysdump

DESCRIPTION

The *sysdump* command dumps the system memory image to one or more floppy disks depending on the size of memory and user request. This memory image can later be analyzed by *crash*(1M). *Sysdump* is invoked as a boot option.

When booted, *sysdump* begins an interactive procedure that prompts the user to insert the floppies to be loaded. The user has the option of quitting the session any time. This allows only the portion of the system image needed to be dumped. floppies.

The output of *sysdump* provides one input to *crash(1)*. The other input is the text file that was used to boot this system image. This is needed to provide symbolic reference to the system dump. The text file must be manually saved after the machine has been booted. If */unix* was booted then this should be dumped to floppy to accompany the system dump.

EXAMPLES

This example loads the 3 floppies produced via *sysdump* on a machine equipped with 2 MB of memory.

```
$ldsysdump /usr/tmp/cdump

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

Loading sysdump
.....
.....

Insert next sysdump floppy.
Enter 'c' to continue, 'q' to quit: c

Loading more sysdump
.....
.....

Insert next sysdump floppy.
Enter 'c' to continue, 'q' to quit: c

Loading more sysdump
.....
.....

3 Sysdump files coalesced, 4096 (512 byte) blocks
$
```

FILES

/dev/ifdsk06 device used for floppy access /unix the text file typically used to boot the machine

SEE ALSO

crash(1M), ldsysdump(1M).

DIAGNOSTICS

For reversible errors, a message will be printed and the user will be allowed to insert a new floppy and continue the session.

WARNINGS

It is critical to provide access to the text file used to boot the machine. This file must be saved.

The diskettes should be labeled clearly so they can be loaded in the proper sequence.

Appendix B

DISK PARTITIONING

PAGE

Appendix B

DISK 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 prior to 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 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

Standard disk names are used to identify the integral floppy disk and integral hard disk drives. 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 integral hard disk drive, the controller/drive designator is "**c1d0**" for both the raw and the

block device. The disk partition or section is appended to the drive control/drive designator. Release 2.0 expands the number of possible disk partitions from eight to sixteen (0 through f, hexadecimal). For example, **/dev/dsk/c0d0s6** specifies section (slice) 6 of the integral floppy disk. The entire integral floppy is also linked to **/dev/diskette** and **/dev/rdiskette** for the block and raw (character) devices, respectively. Section 0 of the block device for integral hard disk is **/dev/dsk/c1d0s0**. Section 0 of the raw device for integral hard disk is **/dev/rdsk/c1d0s0**. 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 (d0) connected to controller 0 (c0).
c1d0s6	1	0	6	Specifies the section 6 (s6) of the integral hard disk (d0) connected to controller 1 (c1).

HARD DISK PARTITIONS

The following tables define the use, starting sector, and total number of sectors (blocks) for the various controller (c), drive (d), and section (s) identifiers for the 10- and 32-megabyte hard disks. Default Volume Table Of Contents (VTOC) partitioning is used in these tables. The identifiers are applicable to both the raw and block devices.

10-MEGABYTE HARD DISK DRIVE
(blocks per cylinder = 72)
(rotational gap = 7)

DISK PARTITION	USE	SECTOR START	TOTAL SECTORS*
c1d0s0	root	3600	8928
c1d0s1	swap	100	3500
c1d0s2	usr	12528	9360
c1d0s3	unassigned		
c1d0s4	unassigned		
c1d0s5	unassigned		
c1d0s6	entire disk	0	21888
c1d0s7	boot	0	100
c1d0s8	unassigned		
c1d0s9	unassigned		
c1d0sa	unassigned		
c1d0sb	unassigned		
c1d0sc	unassigned		
c1d0sd	unassigned		
c1d0se	unassigned		
c1d0sf	unassigned		

* Blocks (sectors) are reported in 512-byte blocks.

32-MEGABYTE HARD DISK DRIVE (blocks per cylinder = 90) (rotational gap = 7)					
		UPGRADED DISK		FULLY RESTORED DISK	
DISK PARTITION	USE	SECTOR START	TOTAL SECTORS*	SECTOR START	TOTAL SECTORS*
c1d0s0	root	5220	12510	6120	12510
c1d0s1	swap	100	5120	100	6020
c1d0s2	usr	17730	44730	18630	43830
c1d0s3	unassigned				
c1d0s4	unassigned				
c1d0s5	unassigned				
c1d0s6	entire disk	0	62460	0	62460
c1d0s7	boot	0	100	0	100
c1d0s8	unassigned				
c1d0s9	unassigned				
c1d0sa	unassigned				
c1d0sb	unassigned				
c1d0sc	unassigned				
c1d0sd	unassigned				
c1d0se	unassigned				
c1d0sf	unassigned				

* Blocks (sectors) are reported in 512-byte blocks.

FLOPPY DISK PARTITIONS

The following tables define the use, starting cylinder, 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 are linked to **/dev/rdiskette** and **/dev/diskette**, 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 the hard disk.

FLOPPY DISK DRIVE (blocks per cylinder = 18) (rotational gap = 7)			
DISK PARTITION	**USE**	**SECTOR START**	**TOTAL SECTORS***
c0d0s0	root	21	1044
c0d0s1	usr	34	810
c0d0s2	usr	45	612
c0d0s3	usr	56	414
c0d0s4	usr	67	216
c0d0s5	1	1404	
c0d0s6	entire disk	0	1422
c0d0s7	boot	0	18

* Blocks (sectors) are reported in 512-byte blocks.

Appendix C

RUN LEVELS

RUN LEVEL	DESCRIPTION
0	Power-down 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/remove or file system backup/restoral.
2	Multi-user mode is the normal operating mode for the system. The **root** (/) and user (**/usr**) file systems are mounted in this mode. When the system is powered up it is put in this mode.
3	User defined run state.
4	User defined run state.

RUN STATE	DESCRIPTION
5	Firmware mode is used to 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 D

ERROR MESSAGES

PAGE

Appendix D

ERROR MESSAGES

GENERAL

Any time a problem occurs during power-up 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
- 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 troubles occur. These error messages are defined alphabetically in the following table.

UNIX SYSTEM ERROR MESSAGES (Prefaced by NOTICE)

ERROR MESSAGE	DESCRIPTION/ACTION
Device Error 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.
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.

UNIX SYSTEM ERROR MESSAGES (Prefaced by NOTICE)	
ERROR MESSAGE	**DESCRIPTION/ACTION**
Floppy Access Error: Consult 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.
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.
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.

UNIX SYSTEM ERROR MESSAGES (Prefaced by NOTICE)	
ERROR MESSAGE	**DESCRIPTION/ACTION**
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: Can't 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.
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.

UNIX SYSTEM ERROR MESSAGES (Prefaced by NOTICE)	
ERROR MESSAGE	**DESCRIPTION/ACTION**
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.
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.
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 from 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: cannot access disk, work list flushed	Make sure floppy disk is in the drive and the door is closed. Contact your service representative if the problem recurs.
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: 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.
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.

UNIX SYSTEM ERROR MESSAGES (Prefaced by WARNING)	
ERROR MESSAGE	**DESCRIPTION/ACTION**
too few HDE equipped disk slots	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.
swapin lost text	The shared text table entry pointer is zero or an attempt was made to link to 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.

UNIX SYSTEM ERROR MESSAGES (Prefaced by PANIC)	
ERROR MESSAGE	**DESCRIPTION/ACTION**
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 exits. After the panic completes, take a crash dump if desired. Reboot the UNIX System. If panics occur frequently, contact your service representative.
text size error in swapin	The size of the swapped-in process text section is not the same size of 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.

UNIX SYSTEM ERROR MESSAGES (Prefaced by PANIC)	
ERROR MESSAGE	**DESCRIPTION/ACTION**
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.
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.

UNIX SYSTEM ERROR MESSAGES (Prefaced by PANIC)	
ERROR MESSAGE	**DESCRIPTION/ACTION**
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.

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.
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.
1-05	UNEXPECTED INTERRUPT REQUEST ABORTED.	Retry request. Contact your service representative.

EDT COMPLETION (FILLEDT) ERROR MESSAGES		
NUMBER	**MESSAGE**	**DESCRIPTION/ACTION**
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.
1-07	FLOPPY KEY CREATE FAILURE	Ensure formatted floppy disk in the drive when "go" is entered.

FIRMWARE ERROR MESSAGES		
NUMBER	**MESSAGE**	**DESCRIPTION/ACTION**
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.
—	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.

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.
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.

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

P

R

S

T

U

V

W