SCALDsystem™ Manager's Reference Manual

Valid Logic Systems Incorporated
2820 Orchard Park Way
San Jose, CA 95134
(408) 945-9400, Telex 371 9004

900-00044 Rev B

MANUAL CONTENTS

This manual is a collection of documents pertinent to the administration of a SCALDsystem. The manual is organized as follows:

SCALD/VMUNIX SYSTEM MANAGER'S GUIDE

ABSTRACT

This guide describes the procedures and defines the responsibilities of a UNIX system administrator for SCALDsystem I, II, and SCALDstar systems and for the file server on SCALDsystem IV network installations; a separate section (section 12) is included at the back of this guide to define the responsibilities associated with a SCALDsystem IV work station. Among the topics covered are: making new users, rebooting the system, and backing up and restoring system and user files.

An appendix to this document includes a Valid version of the hier man pages from the UNIX Programmer's Manual and provides a brief summary of the system directory structure; a second appendix is the user's guide for the file system consistency checker fsck.

This document describes facilities, services, and documentation for the 7.25 release of the SCALDsystem software.

Copyright (C) 1984, 1985 VALID Logic Systems, Inc.

This document contains confidential proprietary information which is not to be disclosed to unauthorized persons without the written consent of an officer of Valid Logic Systems Incorporated.

TABLE OF CONTENTS

SECTION 1
GENERAL INFORMATION

1.1 INTRODUCTION

This guide describes the responsibilities of a UNIX* system administrator and the procedures to be followed in the execution of those responsibilities. The guide is not intended as a UNIX tutorial and assumes that the reader has some degree of familiarity with both UNIX and one of the supported text editors such as vi or ed (a vi tutorial is included at the back of the UNIX Programmer's Manual, an ed tutorial is included within this manual). Also, it is beyond the scope of this guide to provide detailed information for the debugging of hardware or software; your Valid Field Service representative should be contacted in the event of serious trouble.

1.2 A WORD TO THE BEGINNER

A Chinese proverb says "Tell me and I will forget. Show me and I may remember. Involve me and I will understand." To understand UNIX well enough to be an effective manager, you will have to spend a lot of time using it. Read the manual and experiment with the commands. The tutorials, especially Kernighan's UNIX for Beginners, are excellent (UNIX for Beginners is included in this reference manual). Read them through while at your terminal and try the examples. You can build a usable subset of UNIX very quickly (e.g., the use of one editor, the shell command language, and the tape and disk utilities) and acquire additional knowledge with continued exposure and study.

1.3 GETTING STARTED

The duties of a system administrator include adding users, assuring disk integrity, making tape backups of user and system files, and, probably most important, answering the questions of your users. As an administrator, you will have to be familiar with several documents. You should be very familiar with the UNIX Programmer's Manual supplied with the system (Valid manual number 900-00038). Throughout this manual, references will be made to the UNIX manual. These references will be of the form "program name (chapter)." As an example, mkdir(1) references the manual page on "mkdir," the "make a directory" program, located in Chapter 1 of the manual. Pay particular attention to the entries for chmod(1), chown(1), date(1), df(1), du(1), ed(1), find(1), kill(1), mail(1), mkdir(1), ps(1), rm(1), rmdir(1), stty(1), su(1), time(1), wall(1), who(1), and write(1) in Chapter 1; the special device names in Chapter 4; passwd(5) and group(5) in Chapter 5; and all the maintenance utilities in Chapter 8 such as backup(8), fsck(8), mkfs(8), mknod(8), newfs(8), restor(8), and sync(8).

*UNIX is a trademark of Bell Laboratories.

1.4 THE SUPER-USER

There are two types of users within the UNIX system, general users and the "super-user." The super-user has virtually unlimited powers within UNIX and can read, write, execute, or remove any user's files, can change a user's password, and can manipulate special files and system programs (most of the procedures described in this manual require the "super user privilege"). Essentially, anything that can be done in UNIX can be done by the super-user.

BECOMING THE SUPER-USER

There are two ways to become the super-user. One way is to log in as root (root is a special user-id that has automatic super-user privileges). You also can become the super-user while logged in as a general user by typing the following in response to the shell prompt:

su

A password will be demanded; enter the root password. After the root password is entered, UNIX displays the super-user prompt (#).

NOTE

The super-user's ability to manage UNIX, when used carelessly, can destroy some or all of the data within the file system and even can render the SCALDsystem unusable. Guard the root password and change it frequently.

THE SUPER-USER'S LOG

As the system administrator, you will immediately want to set up a system log. Within this log, you will make entries for all tape backup activities performed on the system. In addition, you should note all hardware and software modifications made to the system as well as any system crash or problem.

SECTION 2
ADDING AND REMOVING USERS

2.1 ADDING USERS

The procedure for adding a user is to log in as root (you must have super-user privilege) and then execute the mkusr(8) program by typing:

/etc/mkusr

The mkusr program is self prompting and requests entries for the following:

- Login name of the new user
- Group name (default group name is "user")
- Full user name (optional)
- Shell type (Bourne or C shell; Bourne shell is the default, C shell is preferred)

After the shell-type entry is made, mkusr repeats the information and asks if it is correct. Note that the user and group id numbers also will be displayed; the user id number is a unique number assigned by mkusr, and the group id number is an encoded number for the group name entry. If all of the entries are correct, enter "y" (for yes); if an error is made, enter "n" (the entry requests will be repeated).

Mkusr edits the password and group files with the information entered and creates a directory for the new user. Mkusr then copies the contents of the /u0/user directory to the new user's directory. /u0/user contains a set of user startup files (.profile, .login, and .cshrc), a startup.ged file, and a set of SCALDsystem command or "directives" files for running the other SCALD analysis tools. Once the mkusr program has been run, have the new user login and set his or her password by typing:

passwd

The passwd(1) program prompts for a passwd entry; the password entered is not echoed on the screen. A second entry is requested as a verification of the initial entry. When the password is repeated, the /etc/passwd file is updated with the encrypted password (see section 4). If a password entry is not made, the login program will not require a password when the user logs in (or when anyone enters the user's login name). It is recommended that every user has a password, and further, that all users use non-trivial passwords. Null and obvious passwords make it easy for unauthorized persons

to gain access to the system and potentially destroy your data. A brief history and discussion of UNIX password security is given in the Password Security tutorial included within this reference manual.

For systems of two or more machines or "hosts" in a network configuration, the system administrator must determine if the new user is to access only the local host or all hosts on the network. In the latter case, mkusr must be run on each host within the system to create identical password entries on each host.

NOTE

> While the mkusr program normally creates a unique user id number for each user, duplication of user id numbers can occasionally occur especially on a system with a large number of users or on networked systems. As a security precaution, the /etc/passwd file always should be checked (on each host) for possible duplications of user id numbers (two users with the same user id number have access to each other's files). A duplicated id number is changed by manually editing the /etc/passwd file (see section 4); a user must have the same user id number throughout the network.

2.2 REMOVING USERS

A user is removed from the system by manually deleting his or her corresponding entry from both the /etc/passwd and /etc/group files and, after transferring any files to be saved to another account, deleting the user's directory entry in /u0 (the -r option to rm(1) can be used to recursively remove all files and directories within a user's account).

The /etc/passwd entry for each user contains the following fields:

- name (login name, no upper-case letters)
- encrypted password
- numerical user id
- numerical group id
- comment field (typically contains user's full name)
- home directory
- shell program

Each user entry is on a separate line and each field is separated by a colon (:). The comment field is optional; the password field is initially null until the new user establishes a password. As an example, a password entry for Suzy Scald might appear as:

suzy:AfqeSlLotMzVr:3615:10:Suzy Scald:/u0/suzy:/bin/csh

The character string following the login name is the encrypted password, the "3615" is the unique user id number created by the mkusr utility, and "10" is the numerical equivalent of the default group name "user." Users who work together on a common project can be assigned to the same group, and, by using group access permission bits (see chmod(1)), can share files. Suzy's home directory is /u0/suzy, and her shell is the C shell ("/bin/csh").

The /etc/group entry contains the following fields:

group name
encrypted password entry
group id number
list of the user names within the group separated by commas

Each group begins on a separate line; the members of the group are listed by login name (separated by commas) in chronological order. A typical group entry might appear as:

user::10:garyl, zero, mjp, suzy, alanb

To remove suzy from the group, remove just her name (suzy) and the comma.

2.3 SECURITY

UNIX was not designed with security foremost in mind, and no time-sharing system can really be considered secure. There are some precautions that can be taken, however, to protect the integrity of the system:

- Insist on a 6- to 8-character password for all of your users. Never write down a password, and give your users this same advice.

- Change the super-user (root) password frequently. Keep the number of people who know this password as small as possible.

- The following command mails (to root) a list of all "set user ID" programs owned by root:

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

  Any surprises in root's mail such as files in user's directories being root set-user-id should be investigated.

SECTION 3
ADDING AND REMOVING HOSTS

3.1 ADDING NEW HOSTS

All machines on a network are identified in a "host table" (/etc/hosts) maintained at each machine by the chkhosts(8V) utility. When a new host is added to the network, every host table automatically is updated with an entry for the new host (the table is updated every half hour by cron(8) and whenever a machine is rebooted). To check the status of each machine on the network, use either shownet(1V) or the /bin/conn maintenance utility (see conn(8V)). To display the status of all machines on the system, enter either:

shownet

or

/bin/conn show

Conn displays the status and complete history of each machine on the network; shownet displays a list of only the reachable (and unreachable) machines; conn requires super-user privilege, shownet does not. Note that both programs only display the status of machines since the current machine was rebooted; if any other machine has gone down prior to bringing up the current machine, no record of the down machine is available. Note also that shownet and conn access their machine status information directly from the kernel (which is updated every 30 seconds) rather than from the host table.

If the host table ever becomes damaged, it should be repaired with chkhosts(8V). To repair the host table, first enter

/etc/chkhosts -clean

to purge all entries from the table and then enter

/etc/chkhosts

with no arguments to rebuild the table.

3.2 REMOVING HOSTS

Hosts are logically deleted from the network with the conn utility. Entering

/etc/conn -shutdown period

from the host to be deleted informs the connection manager that the host is to be shutdown in "period" seconds (i.e., no more connection packets are accepted or sent). Note that local operations from the host are still permitted.

SECTION 4
FILE MAINTENANCE

4.1 MAINTAINING USER FILES

For anyone doing serious work, tape back-ups are essential. Even a day- or week-old tape copy of work is vastly better than starting from scratch. A number of utilities exist for saving and restoring files to and from tapes. These are the UNIX programs cpio(1), tar(1), backup(8), restor(8), and, from VRM (Valid Resident Monitor described in the next section of this reference manual), XBOOT. and XRSTR.

4.2 POLICY FOR BACKING UP FILES

You will have to decide how often to back up your files, based on the number of users, how often their files change, and how many tapes are to be maintained. At VALID, we perform full weekly and incremental daily back-ups of our design clusters. A recommended policy is daily back-ups which only take an hour or so to perform and which can be done early in the morning or late at night when the system is idle. A daily back-up provides a high level of protection in case a disaster occurs. Daily back-ups can be kept in a rotation of seven tapes, one for each day of the week. Keep weekly "full" dumps forever. You also should keep tapes of an archival nature--tapes that can be read days, weeks, or months later--as records of your users' projects. These tapes optimally are stored in an off-site archive. Pick a location away from your work area, even away from your company's physical plant, and keep the tapes in a fireproof vault or similar cool, secure location.

If all of this data duplication and storage seems extreme, it is extreme. Do it anyway. By far the most valuable part of a SCALDsystem is the data that your users create, and you, as the system administrator, have been entrusted with the responsibility of protecting this data.

4.3 TAR (Tape ARchiver)

The UNIX program tar(1) operates on individual files and directories rather than entire file systems and is most effective when run by users to maintain their own project tape archives. Tar is used both to read and to write tapes. It knows about directories and can be used to save and recover entire directory trees. "tar c" is used to Create tapes, and "tar x" is used to eXtract files from tapes. The following example shows how tar is used to read the contents of a tape into a user's directory:

```
cd /u0/suzy/project1.wrk
tar xv .
```

The "v" tells tar to be verbose, that is, to print the name of each file that it reads from the tape and the file's size, both in bytes and in tape blocks, on the screen.

When using "tar c" to copy files or directories to tape, if a directory name is included (e.g., tar cv /u0/suzy/project1.wrk), the files written on tape always are read into the named directory when the files are extracted regardless of the current directory location (i.e., the files are "rooted"). The following example shows how to create unrooted files:

```
cd /u0/suzy/project1.wrk
tar cv .
```

By first selecting the directory and then taring to "." (meaning "remove the mount prefix"), files can be subsequently extracted into any directory, examined to be sure that the desired files are present, and then copied into the appropriate directory. Note that when unrooted files are extracted, file ownership automatically is transferred to the directory's owner.

NOTE

Tar operates with the existing file system rather than the raw disk. Accordingly, tar is significantly slower than the backup and restor utilities (see sections 4.4 and 4.5). Also, the print statements caused by the "v" flag further decrease the speed by an additional 30 to 50 percent and should be considered when moving large directory trees.

4.4 MAINTAINING FILE SYSTEMS

There are two utilities for maintaining file systems: backup(8V) and restor(8).

BACKING UP FILE SYSTEMS

Backup(8V) is a front end to the principal UNIX backup facility dump(8). Backup asks if a full or incremental dump is to be performed and if the root file system is to be dumped. Backup determines the names of your file systems and dumps them to tape, and asks if tapes are to be changed between file systems (backup also will instruct you to change tapes when the file system is too large to fit on a single tape). The recommended backup policy is to perform a full dump at the beginning of the week and to perform incremental dumps daily. To run backup, type:

```
/etc/backup
```

CAUTION

If a system is backed up while running in multi-user mode, all files may not be dumped (a file being written at the precise moment that dump is trying to read the file will not be dumped). Backup asks if a consistency check is to be run (via fsck) on the file systems before dumping; if the system is running multi-user, do **not** run this check as irreparable damage may result.

RESTORING INDIVIDUAL FILES

The restor(8) program reads tapes made with backup. The syntax is

restor x file ...

where x means "extract," and file ... is the name or names of files to be extracted from the tape. These filenames have all "mount" prefixes removed (e.g., /u0/libl is named "./libl" on the tape). The following restor procedure outlines the process:

1. Mount the file system tape with the files to be restored. If the file system is contained on multiple volumes, mount volume 1. Change the working directory to be the same as that of the mounted file system.

2. Type restor x file ... with the desired filenames. Restor announces if the files are found or not found, writes the files found to the disk, and rewinds the tape.

3. After the tape is rewound, restor prompts "mount the desired tape volume." This prompt is included for file systems on multiple tapes; mount the second tape and enter its volume number. On a multi-volume dump, mount the last through the first volume in that order. Restor checks to see if any of the files requested are on the mounted tape (or on a subsequent tape, hence the reverse order) and doesn't read through the tape if the requested files are not present. If you are working with a single volume dump or if the number of files to be restored is large, respond to the query with "1," and restor will read the tapes in sequential order.

4.5 RESTORING FULL FILE SYSTEMS

If you need to recover a lost user file system, you must restor the entire file system by first making a new file system with mkfs(1) and then restoring onto the new file system. Be **extremely** careful when running these programs--mkfs and restor are operations that write directly to the disk; any errors made are unrecoverable. Do not restore a file system unless you are running single user. (It's actually a pretty good idea to first backup

the file system that you're planning to restor; if a tragedy occurs, you still have something.) Note that the following discussion assumes that the /u0 directory (mounted as the file system /dev/rim0a) is to be restored; the directory and file system names will have to be altered to correspond to other file systems and configurations.

DETERMINING FILE SYSTEM SIZE

The file system size can be determined by executing the df(1) utility. For example, to determine the size of file system /dev/rim0a, enter:

df /dev/rim0a

Df will respond by printing file system information in the following form:

```
Filesystem    kbytes    used  available  capacity  Mounted on
/dev/rim0a     33843   27527       4961       85%          /u0
```

The number, 33843 in this example, is the size of the file system in thousands of bytes. The dump tape capacity must equal the "used capacity" plus approximately 15% for i-node and other storage. For the case above, approximately 31.7 megabytes of tape capacity would be required.

RESTORING THE FILE SYSTEM

To make the new file system, type:

/etc/newfs -v /dev/rim0a p7050

The newfs utility is a front end to mkfs that uses disk partition information stored in the file "/etc/disktab." The parameter "p7050" instructs newfs to use the PRIAM disk model 7050 entry in /etc/disktab while making the file system. This command essentially says, "erase this file system and make a new file system with all blocks free."

After running newfs, perform the following:

/etc/fsck /dev/rim0a	{check the new or "virgin" file system}
/etc/mount /dev/rim0a /u0	{mount the file system}
cd /u0	{change directory to /u0}
restor x	{restore file system}
cd /	{change directory to root}
/etc/umount /dev/rim0a	{unmount file system}
/etc/fsck /dev/rim0a	{check the restored file system}

SECTION 5
VRM OPERATIONS

VRM (Valid Resident Monitor) is a PROM-resident program that can be used for disk formatting, hardware and software debugging, and tape operations. VRM is a primitive utility intended primarily for Valid field service engineers and only can be used on systems with a physical maintenance console. Only the procedures for backing up the root file system and making a dump tape are described in this guide; to back-up user file systems, the UNIX utilities previously described should be used.

The VRM operations XBKUP. and XRSTR. are the means by which Valid software is installed (VRM is PROM-resident while UNIX must be loaded from disk). Accordingly, VRM is used to read in the UNIX software.

VRM is entered by shutting down the cluster system (i.e., the S-32 or file server). See "Booting UNIX" in the following section for a description of the shutdown procedure.

The VRM prompt (">") will appear after some messages. To back up the root file system, load your tape and put it on line, then type:

XBKUP.

VRM will prompt for the start and end cylinders. The starting cylinder for the system area is 0, and the ending cylinder is "AC" (172 in hexadecimal). VRM next asks if you are sure -- answer Y if you are, N if not.

Restoring disks follows the same procedure except that "XRSTR." is used instead of "XBKUP.".

SECTION 6
REBOOTING UNIX

6.1 PRECONDITIONS TO BOOTING

There are several normal and abnormal conditions that will require the rebooting of UNIX.

CRASHES

The most obvious condition in which UNIX needs to be restarted or "rebooted" is when it appears to have crashed (i.e., when it is not responding to keyboard input). Perhaps the console screen is filled with strange messages about I/O errors or unresolvable page faults. In any event, first carefully note the conditions under which the system crashed in your log. Write down which users were logged in and what they were running; note error messages, diagnostics, and in general, any anomalous behavior just before the crash. Be sure that your field service representative gets a report, along with a memory dump tape (see "Making a Tape Dump of Memory" in the section 6.3). Once all of this bookkeeping has been done, proceed to reboot.

NORMAL OPERATIONS

Whenever you use VRM operations such as XBKUP., you must reset the system, run VRM, and then reboot UNIX. Also, stand-alone operations such as restoring file systems require that you run in single-user mode, which means that you must reboot.

6.2 RESETTING THE SYSTEM -- THE SHUTDOWN PROGRAM

The shutdown(8) program provides an orderly shutdown of the cluster UNIX system. The program closes the logging devices, runs a sync(2) on the file systems to make sure they are up to date, resets the system, and automatically reboots the system (if applicable).

For example,

/etc/shutdown +15 "System coming down at 9:30 for tape dumps"

sends the specified message to all users and brings down the system in fifteen minutes. Read the shutdown(8) manual page long and hard. (You can get a copy on your screen by entering man shutdown; to find out how man(1) works, type man man.)

If the system has crashed, you do something entirely different: see "Making a Tape Dump of Memory" in the next section. When resetting is complete, you will get the VRM ">" prompt.

6.3 MAKING A TAPE DUMP OF MEMORY

CAUTION

The following procedure only works with a maintenance console! If you don't have a maintenance console, press the RESET button on the front of the CPU chassis to reboot the system.

If the system has crashed, the first thing to do is to obtain a tape dump of your UNIX system. A complete dump of memory is an indispensable tool for determining the state of the system at the time of a crash.

1. Slide the CPU chassis out of the rack and, with a screwdriver, remove the cover-mounting screws and open the cover.

2. Make sure that switches 6 and 8 on the CPU board are off (down facing the pc board). Note that the CPU board generally is installed in slot 10 of the card cage and can be readily identified by the large, 64-pin microprocessor integrated circuit installed on the board; the switches are in a bank of eight switches along the top edge of the CPU board.

3. Mount a tape (with a write-enable ring in place) in the drive and place the drive on-line.

4. Press the RESET switch on the front of the CPU chassis. The message "VRM starting - diagnostics disabled" will be displayed on the maintenance console followed by the VRM prompt (the ">" character).

NOTE

If the "VRM starting" message and prompt are not displayed (and switches 6 and 8 are off), a hardware malfunction is indicated.

5. In response to the VRM prompt, type **XREW.** to rewind the tape. When the tape is at its load point, the following status message is displayed:

   Tape status = (RFE RFC) (OL LP R)

6. Type **XDES.** and add 80000 (hex) bytes to the last (highest) memory address displayed (this address is the "limit address" and is used in subsequent steps).

7. If the limit address from the previous step is greater than 400000 (hex) bytes, perform steps 8 through 11; if the limit address is less than 400000, skip directly to step 12.

8. Type **XTW.** (write to tape) and respond to the prompts as follows (operator entry is underlined); terminate each entry with a carriage return:

   Starting bus adr = 0
   Limit bus adr = 380000
   Buffer size (0 yields default size) = 0

   NOTE

   If an error is made when entering a value (and before the carriage return is entered), the value can be changed by entering the character "Z" and repeating the entry. The entire command can be cancelled prior to the last entry by entering a CTRL-X.

   After approximately 1.5 minutes, the following status line is displayed:

   Tape status = (RFE RFC) (OL R FB)(COUNT = 4000)

   Note that if a tape error is reported, use another tape and repeat the above steps.

9. Type **XTW.** and respond as follows:

   Starting bus adr = 0
   Limit bus adr = 80000
   Buffer size (0 yields default size) = 0

   After approximately 15 seconds, the following tape status line is displayed:

   Tape status = (RFE RFC) (OL R FB)(COUNT = 4000)

10. Type **XTW.** and respond as follows:

    Starting bus adr = 400000
    Limit bus adr = 0000 (enter limit address from step 6)
    Buffer size (0 yields default size) = 0

    Depending on the amount of memory installed, up to two minutes may be required to transfer the memory contents to tape. When the transfer is complete, the following status line is displayed:

    Tape status = (RFE RFC) (OL R FB)(COUNT = 4000)

11. Type **XTW.** and respond as follows:

    Starting bus adr = 3D0000
    Limit bus adr = 3D4000
    Buffer size (0 yields default size) = 0

    Since only 4000 hex bytes are transferred, this operation occurs almost immediately; the following status line is again displayed:

    Tape status = (RFE RFC) (OL R FB)(COUNT = 4000)

    After the status line is displayed, omit step 12 and go directly to step 13.

12. Type **XTW.** and respond as follows:

    Starting bus adr = 0
    Limit bus adr = 0000 (enter limit address from step 6)
    Buffer size (0 yields default size) = 0

    Depending on the number of bytes transferred, this step can require up to two minutes to transfer the data to tape. When complete, the following status line is displayed:

    Tape status = (RFE RFC) (OL R FB)(COUNT = 4000)

13. Type **XWFM.** (write tape file mark). The tape status reported should be:

    Tape status = (RFE RFC) (FM OL R FB)

14. If any other VRM operations must be performed (e.g., XBKUP.), do them before going on to the next step.

15. Boot the system up in single-user mode by typing:

    XBOOT.

    The XBOOT. command will prompt for the following information.

    - When prompted for the device, enter P (for primary). The following status lines will be displayed:

      Disk status = (RFE) () ERR#1C
      Disk status = (RFE RFC) (C) or **(C S2)**

    - When prompted for the startup mode, enter P for Parms.

    - When prompted for (n.n) boot:, enter

      rim(0,6)vmunix

      and press the RETURN key.

16. When the single-user prompt (#) is displayed (after approximately 30 to 40 seconds), check the root file system with fsck(8) by entering

    fsck /dev/rim06

    and pressing RETURN. Correct any discrepancies (see "Using FSCK" in the following section). Checking the root file system takes approximately 30 seconds.

17. Write the kernel and configuration onto the tape by entering

    tar cv vmunix etc/configuration

    and pressing RETURN. The following messages will be displayed:

    a vmunix nnn **blocks**
    a etc/configuration nnn **blocks**

18. Restore switches 6 and 8 to their original positions and enter CTRL-D to initiate bringing the system up in multi-user mode.

19. When prompted, enter the date in one of the formats shown.

20. When asked if the file system is to be checked, respond **yes** and correct any discrepancies. After fsck is run (typically 15 to 20 minutes), the system will automatically enter the multi-user mode and will display the login message.

21. Rewind the tape, remove the tape from the drive, and remove the write-enable ring. Label the tape with the time, date, and circumstances of the crash and send the tape to your Valid field office. Be sure to enter this information in your system log in order to maintain a complete record of system operations.

SECTION 7
FILE SYSTEM CHECK PROGRAM

File systems can be damaged in a number of ways: the system can crash while I/O is in progress and can cause the devices to do unpredictable things. The system might be shut down in an abnormal manner, leaving important files open and not updated or the hardware may experience an error that causes it to fail in a mysterious manner.

Fsck (file system check program) is an intelligent, interactive program that examines file systems for damage and, in the event of damage, repairs scrambled systems. If the file system has not been damaged, fsck simply runs to completion and displays messages to give the user an idea of its progress. If there is damage, you will be asked if it should be repaired. At this point, your judgement will come into play. Knowledge of when to modify and when not to modify the file system only comes with experience, but the following responses are almost always correct:

1. If fsck asks if a disconnected i-node is to be reconnected, look at the SIZE= field provided by fsck. If this field is 0 (i.e., an empty file), answer n (for no), and when asked to CLEAR, answer y. If the SIZE= field is non-zero, answer y to reconnect. The file will be reconnected in the lost+found directory; for the root file system, this directory is /lost+found and for the user file system, this directory is /u0/lost+found. The name will be the i-number of the reconnected file.

2. If fsck asks if a bad block count in the free list is to be fixed, answer y unconditionally. Also, if asked if a bad i-node count in the superblock is to be fixed, always answer y.

3. When running fsck directly (rather than through rc) and the root file system is corrupted, the following message will be displayed:

```
***** FILE SYSTEM WAS MODIFIED *****
* INCORE COPY OF ROOT *
* SUPERBLOCK IS CORRUPT *
***** BOOT UNIX (NO SYNC!) *****
```

   In response, press the RESET switch on the front of the CPU chassis to reset the system.

Never fsck a file system while the system is running multi-user. If a user writes on a file while fsck is running, the disk can become scrambled in subtle ways that can cause fsck to further scramble the file while attempting to repair the inconsistency. Note that it is safe to run fsck while running in the single-user mode.

Fsck cannot help in the event of a really serious hardware error, but it is invaluable for detecting and correcting software-crash related inconsistencies. More advice and details of fsck's operation appear in the fsck addendum following this guide.

SECTION 8
DISK USAGE

If your UNIX system is a success, you will soon run out of disk space. At the limiting case (i.e., 0 blocks free), the system will display a "No free space" message and little else. Until you are able to get more disks for your system, you will have to police your disks rather closely:

- The df(1) command provides an accounting of free blocks and i-nodes for each file system and is an invaluable tool for keeping track of disk space.

- The du(1) command is used to monitor system sizes. Entering:

  du /u0 > /etc/du.TODAYSDATE

  once a week will help you to keep track of who the disk hogs are.

- "The force of moral suasion" may be worth something in persuading your users to police their own directories; in practice, it is not unless you have a relatively small number of users who can be taught to take care of "their" system.

SECTION 9
SYSTEM ACCOUNTING

There are two programs that keep track of connect time: ac(8) and last(1). The file /usr/adm/wtmp (see wtmp(5)) contains information about users who log onto UNIX. Although the login(1) program maintains wtmp, it does not initialize the file; you must do this explicitly by entering:

```
cp /dev/null /usr/adm/wtmp
```

The wtmp file will maintain records that contain the names, terminals, and login/logout times of all users henceforth.

However, the wtmp file grows without bound so that periodically (once a week, once every few days, or once a day according to how busy your system is) you should gather what information you need from the file and then reinitialize the file to zero length.

Two programs process the wtmp file: ac(8) and last(1). Ac sums up login times in hours. Entering:

```
/etc/ac -p
```

might cause the program to display the following list:

```
joe      7.95
root     4.42
suzy     8.07
total   20.44
```

By contrast, last gives a chronological record of logins. Entering:

```
last
```

for the same wtmp file causes a display of the form:

```
root     console   Fri Jul  8 13:11 - 17:36  (4:25)
suzy     tty1      Fri Jul  8 10:04 - 10:55  (0:51)
joe      tty3      Fri Jul  8 08:28 still logged in
```

SECTION 10
COMMUNICATING WITH YOUR USERS

Announcements about planned down-time for dumps and the like are placed in the file /etc/motd. This file automatically is displayed when a user logs in (see login(1)).

To communicate with users who are already logged in, use wall(1) (write all). Limit the use of wall to emergencies as users tire of it very quickly.

SECTION 11
THE UNIX DIRECTORY STRUCTURE

There are two file systems. One, known as the root and denoted "/," contains mainly system programs and data files. The other, denoted "/u0," is the "user" file system and contains mainly user directories along with the Graphics Editor data (/u0/editor), system testing (/u0/fe, /u0/qa), and the component libraries (/u0/lib*).

The first addendum to this guide reproduces the hier(7) manual page of the UNIX Programmer's Manual. The page has been adapted to reflect the SCALD/UNIX directory structure.

SECTION 12
SCALDsystem IV OPERATIONS

This section defines the additional responsibilities associated with administering a SCALDsystem IV work station and the file server.

12.1 THE NETWORKED SCALDsystem IV WORK STATION

The SCALDsystem IV work station is linked to the file server through the Omninet network. In most cases, the files supporting SCALDsystem operations are transferred transparently from the file server to the SCALDsystem IV work stations across the network so that each work station essentially appears to its user as a stand-alone work station.

All design data is initially developed locally on the work station and is subsequently copied or "checked in" by the user to the file server where it is archived. Once copied to the file server, the directory containing the design can be copied or "checked out" to the SCALDsystem IV for subsequent editing and analysis. By continually checking out and checking in designs, the backup and project management capabilities of the file server can be utilized. With the exception of directory transfers, SCALDsystem operations are virtually transparent to the user.

Communications with other work stations and file servers on the network are supported by the Extended File System (EFS), remote operations (rlogin and rsh), and by the checkin/checkout utilities for design data transfers. To the work station, the file server and the other work stations on the network are viewed as UNIX nodes. To access files on another node, the "/net" prefix and full pathname are used. The syntax for a remote file transfer is

/net/<nodename>/<pathname>/<filename>

where <nodename> is the name of the node (file server or work station). In the following example

cp /net/papa/u0/scald/suzy/notes /u0/garyl/ch7

"papa" is the node name, and "/u0/scald/suzy" is the pathname to the file "notes" which is copied to the file "ch7" in "/u0/garyl."

12.2 THE FILE SERVER

With the exception of the Graphics Editor which is resident on each work station, most of the SCALD analysis tools (i.e., the Compiler, Timing Verifier, Simulator, and Packager) reside on the file server and are transparently copied to the SCALDsystem IV each time they are invoked. Once the SCALD analysis tool is copied to the work station, it is executed and then erased from the local SCALDsystem IV disk. SCALD libraries also reside on the file server; individual components from the libraries are accessed

transparently over the network when required by the SCALDsystem program. Also, a number of the UNIX utilities are resident on the file server and similarly are copied, executed, and erased.

DIAL, Realfast, and Realchip are not supported locally at the work station and must be executed remotely on the file server through remote login (rlogin).

12.3 ADDING AND REMOVING USERS

To use the remote facilities and to take advantage of the file server's archival and project management capabilities, a user account must be created for each new user on both the file server and on each work station on which the user is to have login privilege.

Users are added with the **/etc/mkusr** utility as described in section 2. A new feature of the mkusr utility is the addition of command line arguments to allow the passwd entry initially created on one system within the network to be copied to the local work station. For example

/etc/mkusr suzy server1

copies the passwd entry for user "suzy" from node "server1" to the local system. The mkusr utility then creates a home directory for "suzy" in /u0/suzy with the appropriate SCALD and shell files.

Users are removed from the system by deleting their /etc/passwd and /etc/group entries and deleting their accounts from each work station and from the file server.

12.4 ADDING AND REMOVING WORK STATIONS

Initial installations (file server, Omninet "trunk" cable, and work stations) are performed by by Valid field service personnel. Add-on work stations are installed by the user as described in the "SCALDsystem IV Installation Guide."

Individual work stations are logically removed from the network with the "shutdown" option to the conn(8V) command. Once logically removed, the work station can be physically disconnected by simply unplugging the Omninet cable at the tap box. Note that stand-alone operation of the SCALD IV work station is not supported and should not be attempted. Before power is removed from a work station, all processes must by killed (see section 12.9). Additionally, if the work station is to moved, the drive heads first must be retracted or "parked" as described in section 12.9.

12.5 FILE SERVER BACKUPS

File server backups are identical to the procedures described in section 4.

12.6 SCALDsystem IV BACKUPS

There are two types or general categories of backups within the SCALDsystem IV environment: individual files and the user file system. Individual file backups normally are performed by the user using either the floppy disk copy utility "fdcopy" or tar(1). The fdcopy utility offers the additional advantage of multi-volume backups (tar is limited to only a single diskette). Incremental or full backups of the user file system are done on either the file server's magnetic tape or on diskette locally at the work station.

NOTE

The root file system ("/") on the SCALDsystem IV cannot be restored from a backup; see the section "Restoring the Root File System."

BACKING UP TO TAPE OVER THE NETWORK

The user file system on individual SCALDsystem IV work stations can be backed up on the file server's tape using the **rbackup** utility (rbackup is a front end to dump(8) that permits dumping the file system on a remote machine over the network). The syntax for the rbackup command is

rbackup <name>

where <name> is the name of the system to be backed up (if a name is not entered, the operator is prompted to enter a system name). The rbackup utility is interactive and first prompts for the type of backup to be performed (monthly, weekly, or daily) and then asks if the system to be backed up is a SCALDsystem IV. If the system is a SCALDsystem IV, rbackup dumps the user file system "/dev/cv0a" to file server tape (if the system is not a SCALDsystem IV, a file system name will be requested; the name entered must be a block-special device associated with the file system).

The advantage to backing up on tape is that all of the work stations can be backed up from a single machine; the major disadvantage is that there is no facility for restoring directly from tape to a work station.

BACKING UP TO LOCAL DISKETTE

The user file system on a SCALDsystem IV can be backed up on the local floppy disk using the backup(8V) utility described in section 4. The SCALDsystem IV version of this utility recognizes the size of a floppy (640 kbytes) and as each floppy is filled, prompts the operator to insert another diskette. The advantage of backing up to floppy disk is that a restore operation can be performed directly from floppy diskette at the SCALDsystem IV. The disadvantages to this method are:

- Number of Diskettes: A full backup of a full user file system may require up to 25 diskettes.
- Time: Each floppy can take up to 10 minutes (at 60 kbytes/minute).
- Logistics: The administrator must keep track of the diskette volumes and must service "floppy interrupts" in the loading and unloading of the diskettes themselves.

BACKING UP INDIVIDUAL FILES

Individual files are normally backed up by the user locally on diskette using either the fdcopy utility (see "SCALDsystem IV Utilities" in Chapter 13 of the reference manual) or tar(1) as described in section 4.3 (tar operates with both tape and diskette, but is limited to a single diskette).

12.7 FILE SERVER RESTORE OPERATIONS

The restoring of individual files or file systems is done with the restor(8) utility as described in section 4.4.

12.8 SCALDsystem IV RESTORE OPERATIONS

Restore operations for a system administrator fall into three categories: restoring individual files from a file system backup, restoring the full user file system, and restoring the root file system. Restoring individual files is usually necessitated by the an oversight on the part of the user (e.g., accidently removing a file or needing an earlier version of an updated file). Restoring a file system is necessitated when the system is inoperable (i.e., following a crash).

RESTORING THE ROOT FILE SYSTEM

The root file system must be restored when the SCALDsystem IV fails to boot up in the single-user mode. Since it is not possible to restore the root file system from a backup, if the root file system ever becomes damaged, it must be recreated with the special set of "help" diskettes provided with the system. The following steps outline this procedure:

1. If possible, perform an orderly shutdown of the system as described in section 12.9.

2. Place the front panel POWER switch to OFF.

3. If the SCALDsystem IV is in the auto-boot mode, disconnect the ac line cord from the back of the system chassis, remove the chassis cover, and place switch position 8 of the network address switch in toward the center of the chassis to disable the auto-boot mode; replace the chassis cover and reconnect the ac line cord.

4. Place the POWER switch to ON.

5. Insert the "HELP" program floppy diskette into the drive.

6. In response to the "PROM" prompt (">"), enter:

   xboot.

7. When prompted for the boot device, enter "f" for floppy.

8. After the "HELP" program has been loaded, enter:

   recover boot

9. In response to the message "Hit carriage return when your first floppy is loaded:", install the boot floppy and press RETURN.

10. After the "boot partition" (partition 05) has been read, enter:

    recover root

11. When prompted, install volumes 2 through 8 (in order) of the "root file system recover" diskettes. After the last diskette has been read, the SCALDsystem IV can be booted up in the single-user mode by entering:

    boot

    and responding "a" to the "boot type:" prompt.

NOTE

Note that if the single-user prompt ("#") is not displayed, a possible hardware malfunction is indicated and further repair must be referred to your Valid field service engineer.

12. When the single-user prompt is displayed, check the root file system with fsck(8) by entering

    fsck /dev/cv06

    and pressing RETURN. Correct any discrepancies (see section 7).

13. From the single-user mode, enter ^D (control-D) to initiate bringing up the system in the multi-user mode.

14. As part of the multi-user startup, fsck is automatically run to check the user file system. If the file system is intact, the system enters the multi-user mode and displays the login message; if the user file system is damaged, fsck reports the damaged area and asks if it is to be repaired before bringing up the system in multi-user mode.

NOTE

If the system does not come up in the multi-user mode (i.e., if the login message is not displayed) or if an excessively large number of damaged files are reported by fsck, the user file system must be restored as described in the next section.

RESTORING THE USER FILE SYSTEM

The procedure used to restore the user file system is determined by how the file system was backed up (tape or diskette). Independent of how the user file system was backed up, if the user file system is lost (i.e., if the system will not boot up or a large number of errors is reported by fsck), the entire file system must be restored by first making a new file system with newfs(8) and then restoring onto the newly-created file system.

CAUTION

Be extremely careful when restoring a file system -- newfs and the restore operations write directly to the disk; any errors made are unrecoverable.

Before beginning to restore the user file system, perform (if possible) an orderly shutdown of the work station, place the POWER switch to OFF, and be sure that the work station is in the single-user mode (see previous section).

1. Place the POWER switch to ON and, in response to the "PROM" prompt (">"), enter:

   xboot.

2. When prompted for the boot device, enter "d" for disk.

3. The screen will clear and display "cv(3.9) boot:" as a prompt to enter the UNIX system name. Press RETURN to initiate single-user mode boot (defaults to system name "cv(0,6)vmunix").

4. When the system is booted up, the single-user prompt ("#") will be displayed.

In response to the single-user prompt, create a new user file system by entering:

/etc/newfs -v /dev/cv0a cv

The newfs utility is a front end to mkfs(8) that uses disk partition information stored in the file "/etc/disktab." The mkfs command erases the existing file system and creates a new file system with all blocks free.

Restoring The User File System From Tape

To restore the user file system on a SCALDsystem IV from tape, the backup tape of the file system is first copied to a specific directory on the file server with dd(1) and then the file system is restored from the file server directory over the network.

At the file server:

1. Load the backup tape

2. **mkdir /u0/recover** {create the directory "recover"}

3. **cd /u0/recover** {change directory to "recover"}

4. **dd if= /dev/rmt0 of= recover bs=10k**

The above command copies the file system from the tape device (/dev/rmt0) to the directory "recover."

From the SCALDsystem IV (after running newfs):

1. **/etc/fsck /dev/cv0a** {check the new file system}
2. **/etc/mount /dev/cv0a /u0** {mount the file system}
3. **cd /u0** {change directory to /u0}
4. **/etc/restor xf /net/<server>/u0/recover**

In the above command, <server> is the name of the file server. After the "recover" directory has been restored on /u0, boot the system up in multi-user mode (enter control-D) as described in the previous section. Once the SCALDsystem IV is up and running, the "restore" directory on the file server can be deleted with the following command:

rm -rf /u0/recover

Restoring The User File System From Diskette

To restore the user file system on a SCALDsystem IV from local diskette, the restor(8) utility is used. To use this facility, the SCALDsystem IV must be in single-user mode and a new file system must have been created with newfs. At the SCALDsystem IV, perform the following:

1. **/etc/fsck /dev/cv0a** {check the new file system}
2. **/etc/mount /dev/cv0a /u0** {mount the file system}
3. **cd /u0** {change directory to /u0}
4. Insert the first volume of the diskette
5. **restor x** {restore the user file system}

NOTE

The restor program recognizes diskette size and will prompt "mount the desired volume" when the next diskette is required. When the last diskette has been restored:

6. **sync** {update the file system}
7. **cd /** {change directory to root}
8. **/etc/umount /dev/cv0a** {unmount the file system}
9. **/etc/fsck /dev/cv0a** {check the restored file system}

RESTORING INDIVIDUAL FILES FROM DISKETTE

The fdcopy utility (see the utilities section in Chapter 13 of the reference manual) and tar(1) are normally used to restore individual files or directories that have been backed up on diskette by general users. Note that fdcopy and tar operate with the existing file system structure rather than the raw disk. Accordingly, these programs are significantly slower than the backup and restor utilities previously described and are not intended to be used for backing up or restoring the system.

The restor utility, in addition to restoring full file systems, accepts file name arguments to allow the restoring of individual files or directories. When a file name or names (separated by spaces) are specified, restor searches the diskette for the named files, announces if they are found, and writes the files found to the work station's system disk. To use restor with individual files, first select the directory before running restor.

RESTORING INDIVIDUAL FILES FROM TAPE

To restore individual files from the backup tape of the user file system, the required files are restored to a specific directory on the file server and then copied across the network with the cpio(1) command from the work station using the remote shell facility.

At the file server:

1. Load the backup tape
2. **mkdir /u0/recover** {create the directory "recover"}
3. **cd /u0/recover** {change directory to "recover"}
4. **/etc/restorex** <file1> <file2> ...

   where <file1> <file2> are the names of the files to be restored.

From the SCALDsystem IV:

Enter the following command line (note that while two lines are shown, the the command line is entered on one line):

```
rsh <server> "cd /u0/recover; find . -print |
cpio -oB" | (cd /u0/<directory>; cpio -iBdum)
```

In the above command line <server> is the name of the file server and <directory> is the name of the SCALDsystem IV directory where the files are to be restored. The above command line uses the server's shell, changes the server's directory to "recover," recursively descends the "recover" directory hierarchy, copies the files out, changes the SCALDsystem IV directory, and then copies the files into the named directory.

12.9 SYSTEM SHUTDOWN

Power to the SCALDsystem IV must not be removed until the shutdown(8) utility has been run.

CAUTION

> Powering down the SCALDsystem IV without running the shutdown utility can cause data on the disk drive to be lost. As the system administrator, warn your users about turning off power from the front panel POWER switch (users should only power down the CRT monitor from the POWER switch on the rear of the monitor chassis).

The shutdown utility provides an orderly shutdown of the UNIX system by first running a sync(2) on the file system to make sure it is to date and then resetting and automatically rebooting the system. For example, entering

/etc/shutdown now

shuts the system down immediately.

PARKING THE DRIVE HEADS

When it becomes necessary to physically move a SCALDsystem IV, the heads on the SCALDsystem IV system drive must be retracted or "parked" before the system chassis is moved.

CAUTION

> Failure to park the drive heads prior to moving the system chassis can cause irreparable drive damage.

The following procedure parks the drive heads:

1. **sync;sync** {update the file system}
2. **kill 1** {kill processes that write to disk}
3. **sync** {safety measure; wait until disk activity light goes out}
4. **cv /dev/rcv06 -h** {parks the heads}

Once the heads are parked, power can be removed and the SCALDsystem IV can be moved. Note that the heads are automatically "unparked" when power is reapplied.

12.10 AUTO BOOTING

Unless the auto boot mode has been disabled, the SCALDsystem IV will automatically "boot up" in multi-user mode when power is applied.

NOTE

> When power is initially applied to the system, an interval of between 10 and 15 seconds is required for execution of the self diagnostic and approximately one minute is required for CRT monitor warm up. Also, before applying power, make sure that a diskette is NOT installed in the floppy disk drive.

As part of the auto-boot procedure, a file system check of the file systems is run (if the file systems are in order, the running of fsck is transparent) and the time is set from the file server (by the popcorn utility) before the multi-user prompt is displayed.

Note that if the root file system is damaged (i.e., corrupted), the system will repeat the boot sequence in an attempt to reboot UNIX. If the system cannot be rebooted, restore the root file system as previously described.

NAME

hier - file system hierarchy

DESCRIPTION

The following outline gives a quick tour through a representative SCALD/UNIX directory hierarchy.

```
/     root
/unix
      the kernel binary (UNIX itself)
/lost+found
      directory for connecting detached files for fsck(8)
/dev/
      devices (4)
      bisync
            device driver for RSCS (IBM only)
      console
            main console, tty(4)
      log   console log, log(4)
      rim*  disks
      rrim*
            raw disks
      tty*  terminals, tty(4)
      vms   file copy device driver (VAX only)
      ...
/bin/
      utility programs, cf /usr/bin/ (1)
      as    assembler
      cc    C compiler executive, cf /lib/ccom, /lib/cpp,
            /lib/c2
      ged   graphics editor executive, cf
            /u0/editor/lib/ged.ex
      sh    shell
      ...
/etc/
      essential data and maintenance utilities; section(8)
      configuration
            site-dependent data, configuration(5)
      cron the clock daemon, cron(8)
      dumpdates
            dump history, dump(8)
      getty
            part of login, getty(8)
      group
            group file, group(5)
      init the parent of all processes, init(8)
      motd message of the day, login(1)
      mtab mounted file table, mtab(5)
      mount
            mount(8)
      passwd
            password file, passwd(5)
```

```
        rc   shell program to bring the system up
        termcap
             description of terminal capabilities, termcap(5)
        ttys properties of terminals, ttys(5)
        wall Writetoallusers,wall(8)
        ...
   /include/
        standard #include files
        a.out.h
             object file layout, a.out(5)
        stdio.h
             standard I/O, stdio(3)
        math.h
             math definitions, see exp, floor, j0(3M)
        ...
        local/
             locally-defined header files
        sys/ system-defined header files
   /lib/
        object libraries and other stuff, cf /usr/lib/
        libc.a
             system calls, standard I/O, etc. (2,3,3S)
        ...
        cpp  C preprocessor
        ...
   /mnt/
        use this to mount extra file systems
   /tmp/
        temporary files, cf /usr/tmp/
        e*   used by ed(1)
        ctm* used by cc(1)
        ...
   /usr/
        general-pupose directory
        adm/ administrative information
             rscs/
                  RSCS accounting
        bin/ utility programs, to keep /bin/ small
        tmp/ temporaries, to keep /tmp/ small
        dict/
             word lists, etc.
             spellhist
                  history file for spell(1)
             words
                  principal word list, used by look(1)
        doc/ papers, mostly in volume 2 of this manual,
             typically in ms(7) format
             as/  assembler manual
             c    C manual
             ...
        games/
        ...
```

```
        include/
             more #include files
        lib/
             object libraries and stuff, to keep /lib/ small
             atrun
                  scheduler for at(1)
             units
                  conversion tables for units(1)
             lint/
                  utility files for lint
                  lint[12]
                       subprocesses for lint(1)
                  llib-lc
                       dummy declarations for /lib/libc.a, used
                       by lint(1)
                  llib-lm
                       dummy declarations for /lib/libc.m
                  ...
             rscs/
                  RSCS control directory
             ...
        local/
             binaries of programs written locally
        man/
             volume 1 of this manual, man(1)
             man0/
                  (general introduction)
                  intro
                       introduction to volume 1, ms(7) format
                  xx   template for manual page
             man1/
                  chapter 1
                  as.1
                  mount.1m
                  ...
             ...
             cat1/
                  preformatted pages for section 1
             ...
        new/ binaries of new versions of programs
        preserve/
             editor temporaries preserved here after
             crashes/hangups
        pub/ general user directory for programs, games, etc.
/u0/
        the user file system
        class/
             data and work files for SCALDsystem classes
        editor/
             doc/ editor help files
             softkeys/
                  keyboard templates
```

```
fe/  Valid Field Engineering work area
lib*/
     SCALD parts libraries
lost+found/
     lost+found directory for this file system.
qa/  system test and validation directory
scald/
     SCALD programs and documentation
spool/
     delayed execution files
     at/  used by at(1)
     vpd/ same as lpd/.  Used by vpr, cf lpr(1)
     lpd/ used by lpr(1)
          lock present when line printer is active
          cf*  copy of file to be printed, if necessary
          df*  daemon control file, lpd(8)
          tf*  transient control file, while lpr is
               working
     rscs/
          used by RSCS (IBM only)
          SYSTEM
               files for the system
          UNKNOWN
               files arriving for unknown users
          *    files destined for system *
     mail/
          mailboxes for mail(1)
          name mail file for user name
          name.lock
               lock file while name is receiving mail
     secretmail/
          like mail/
user/
     startup files for a typical user
wd/  initial working directory of a user; typically wd
     is the user's login name
     .profile
          set environment for sh(1), environ(5)
     .cshrc
          startup file for csh(1)
     .exrc
          startup file for ex(1)
     .login
          login file for csh users, analogue of
          .profile.
     startup.ged
          graphics editor startup file
```

SEE ALSO

ls(1), ncheck(8), find(1), grep(1)

BUGS

The position of files is subject to change without notice.

Fsck – The UNIX† File System Check Program

Revised July 28, 1983

Marshall Kirk McKusick

Computer Systems Research Group
Computer Science Division
Department of Electrical Engineering and Computer Science
University of California, Berkeley
Berkeley, CA 94720

T. J. Kowalski

Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

This document reflects the use of *fsck* with the 4.2BSD file system organization. This is a revision of the original paper written by T. J. Kowalski.

File System Check Program (*fsck*) is an interactive file system check and repair program. *Fsck* uses the redundant structural information in the UNIX file system to perform several consistency checks. If an inconsistency is detected, it is reported to the operator, who may elect to fix or ignore each inconsistency. These inconsistencies result from the permanent interruption of the file system updates, which are performed every time a file is modified. Unless there has been a hardware failure, *fsck* is able to repair corrupted file systems using procedures based upon the order in which UNIX honors these file system update requests.

The purpose of this document is to describe the normal updating of the file system, to discuss the possible causes of file system corruption, and to present the corrective actions implemented by *fsck*. Both the program and the interaction between the program and the operator are described.

†UNIX is a trademark of Bell Laboratories.

This work was done under grants from the National Science Foundation under grant MCS80-05144, and the Defense Advance Research Projects Agency (DoD) under Arpa Order No. 4031 monitored by Naval Electronic System Command under Contract No. N00039-82-C-0235.

TABLE OF CONTENTS

1. Introduction

This document reflects the use of *fsck* with the 4.2BSD file system organization. This is a revision of the original paper written by T. J. Kowalski.

When a UNIX operating system is brought up, a consistency check of the file systems should always be performed. This precautionary measure helps to insure a reliable environment for file storage on disk. If an inconsistency is discovered, corrective action must be taken. *Fsck* runs in two modes. Normally it is run non-interactively by the system after a normal boot. When running in this mode, it will only make changes to the file system that are known to always be correct. If an unexpected inconsistency is found *fsck* will exit with a non-zero exit status, leaving the system running single-user. Typically the operator then runs *fsck* interactively. When running in this mode, each problem is listed followed by a suggested corrective action. The operator must decide whether or not the suggested correction should be made.

The purpose of this memo is to dispel the mystique surrounding file system inconsistencies. It first describes the updating of the file system (the calm before the storm) and then describes file system corruption (the storm). Finally, the set of deterministic corrective actions used by *fsck* (the Coast Guard to the rescue) is presented.

2. Overview of the file system

The file system is discussed in detail in [Mckusick83]; this section gives a brief overview.

2.1. Superblock

A file system is described by its *super-block*. The super-block is built when the file system is created (*newfs*(8)) and never changes. The super-block contains the basic parameters of the file system, such as the number of data blocks it contains and a count of the maximum number of files. Because the super-block contains critical data, *newfs* replicates it to protect against catastrophic loss. The *default super block* always resides at a fixed offset from the beginning of the file system's disk partition. The *redundant super blocks* are not referenced unless a head crash or other hard disk error causes the default super-block to be unusable. The redundant blocks are sprinkled throughout the disk partition.

Within the file system are files. Certain files are distinguished as directories and contain collections of pointers to files that may themselves be directories. Every file has a descriptor associated with it called an *inode*. The inode contains information describing ownership of the file, time stamps indicating modification and access times for the file, and an array of indices pointing to the data blocks for the file. In this section, we assume that the first 12 blocks of the file are directly referenced by values stored in the inode structure itself†. The inode structure may also contain references to indirect blocks containing further data block indices. In a file system with a 4096 byte block size, a singly indirect block contains 1024 further block addresses, a doubly indirect block contains 1024 addresses of further single indirect blocks, and a triply indirect block contains 1024 addresses of further doubly indirect blocks.

In order to create files with up to 2↑32 bytes, using only two levels of indirection, the minimum size of a file system block is 4096 bytes. The size of file system blocks can be any power of two greater than or equal to 4096. The block size of the file system is maintained in the super-block, so it is possible for file systems of different block sizes to be accessible simultaneously on the same system. The block size must be decided when *newfs* creates the file system; the block size cannot be subsequently changed without rebuilding the file system.

2.2. Summary information

Associated with the super block is non replicated *summary information*. The summary information changes as the file system is modified. The summary information contains the number of blocks, fragments, inodes and directories in the file system.

2.3. Cylinder groups

The file system partitions the disk into one or more areas called *cylinder groups*. A cylinder group is comprised of one or more consecutive cylinders on a disk. Each cylinder group includes inode slots for files, a *block map* describing available blocks in the cylinder group, and summary information describing the usage of data blocks within the cylinder group. A fixed number of inodes is allocated for each cylinder group when the file system is created. The current policy is to allocate one inode for each 2048 bytes of disk space; this is expected to be far more inodes than will ever be needed.

All the cylinder group bookkeeping information could be placed at the beginning of each cylinder group. However if this approach were used, all the redundant information would be on the top platter. A single hardware failure that destroyed the top platter could cause the loss of all copies of the redundant super-blocks. Thus the cylinder group bookkeeping information begins at a floating offset from the beginning of the cylinder group. The offset for the $i+1$st cylinder group is about one track further from the beginning of the cylinder group than it was for the ith cylinder group. In this way, the redundant information spirals down into the pack; any single track, cylinder, or platter can be lost without losing all copies of the super-blocks.

†The actual number may vary from system to system, but is usually in the range 5-13.

Except for the first cylinder group, the space between the beginning of the cylinder group and the beginning of the cylinder group information stores data.

2.4. Fragments

To avoid waste in storing small files, the file system space allocator divides a single file system block into one or more *fragments*. The fragmentation of the file system is specified when the file system is created; each file system block can be optionally broken into 2, 4, or 8 addressable fragments. The lower bound on the size of these fragments is constrained by the disk sector size; typically 512 bytes is the lower bound on fragment size. The block map associated with each cylinder group records the space availability at the fragment level. Aligned fragments are examined to determine block availability.

On a file system with a block size of 4096 bytes and a fragment size of 1024 bytes, a file is represented by zero or more 4096 byte blocks of data, and possibly a single fragmented block. If a file system block must be fragmented to obtain space for a small amount of data, the remainder of the block is made available for allocation to other files. For example, consider an 11000 byte file stored on a 4096/1024 byte file system. This file uses two full size blocks and a 3072 byte fragment. If no fragments with at least 3072 bytes are available when the file is created, a full size block is split yielding the necessary 3072 byte fragment and an unused 1024 byte fragment. This remaining fragment can be allocated to another file, as needed.

2.5. Updates to the file system

Every working day hundreds of files are created, modified, and removed. Every time a file is modified, the operating system performs a series of file system updates. These updates, when written on disk, yield a consistent file system. The file system stages all modifications of critical information; modification can either be completed or cleanly backed out after a crash. Knowing the information that is first written to the file system, deterministic procedures can be developed to repair a corrupted file system. To understand this process, the order that the update requests were being honored must first be understood.

When a user program does an operation to change the file system, such as a *write*, the data to be written is copied into an internal *in-core* buffer in the kernel. Normally, the disk update is handled asynchronously; the user process is allowed to proceed even though the data has not yet been written to the disk. The data, along with the inode information reflecting the change, is eventually written out to disk. The real disk write may not happen until long after the *write* system call has returned. Thus at any given time, the file system, as it resides on the disk, lags the state of the file system represented by the in-core information.

The disk information is updated to reflect the in-core information when the buffer is required for another use, when a *sync*(2) is done (at 30 second intervals) by */etc/update*(8), or by manual operator intervention with the *sync*(8) command. If the system is halted without writing out the in-core information, the file system on the disk will be in an inconsistent state.

If all updates are done asynchronously, several serious inconsistencies can arise. One inconsistency is that a block may be claimed by two inodes. Such an inconsistency can occur when the system is halted before the pointer to the block in the old inode has been cleared in the copy of the old inode on the disk, and after the pointer to the block in the new inode has been written out to the copy of the new inode on the disk. Here, there is no deterministic method for deciding which inode should really claim the block. A similar problem can arise with a multiply claimed inode.

The problem with asynchronous inode updates can be avoided by doing all inode deallocations synchronously. Consequently, inodes and indirect blocks are written to the disk synchronously (*i.e.* the process blocks until the information is really written to disk) when they are being deallocated. Similarly inodes are kept consistent by synchronously deleting, adding, or changing directory entries.

3. Fixing corrupted file systems

A file system can become corrupted in several ways. The most common of these ways are improper shutdown procedures and hardware failures.

File systems may become corrupted during an *unclean halt*. This happens when proper shutdown procedures are not observed, physically write-protecting a mounted file system, or a mounted file system is taken off-line. The most common operator procedural failure is forgetting to *sync* the system before halting the CPU.

File systems may become further corrupted if proper startup procedures are not observed, e.g., not checking a file system for inconsistencies, and not repairing inconsistencies. Allowing a corrupted file system to be used (and, thus, to be modified further) can be disastrous.

Any piece of hardware can fail at any time. Failures can be as subtle as a bad block on a disk pack, or as blatant as a non-functional disk-controller.

3.1. Detecting and correcting corruption

Normally *fsck* is run non-interactively. In this mode it will only fix corruptions that are expected to occur from an unclean halt. These actions are a proper subset of the actions that *fsck* will take when it is running interactively. Throughout this paper we assume that *fsck* is being run interactively, and all possible errors can be encountered. When an inconsistency is discovered in this mode, *fsck* reports the inconsistency for the operator to chose a corrective action.

A quiescent‡ file system may be checked for structural integrity by performing consistency checks on the redundant data intrinsic to a file system. The redundant data is either read from the file system, or computed from other known values. The file system must be in a quiescent state when *fsck* is run, since *fsck* is a multi-pass program.

In the following sections, we discuss methods to discover inconsistencies and possible corrective actions for the cylinder group blocks, the inodes, the indirect blocks, and the data blocks containing directory entries.

3.2. Super-block checking

The most commonly corrupted item in a file system is the summary information associated with the super-block. The summary information is prone to corruption because it is modified with every change to the file system's blocks or inodes, and is usually corrupted after an unclean halt.

The super-block is checked for inconsistencies involving file-system size, number of inodes, free-block count, and the free-inode count. The file-system size must be larger than the number of blocks used by the super-block and the number of blocks used by the list of inodes. The file-system size and layout information are the most critical pieces of information for *fsck*. While there is no way to actually check these sizes, since they are statically determined by *newfs*, *fsck* can check that these sizes are within reasonable bounds. All other file system checks require that these sizes be correct. If *fsck* detects corruption in the static parameters of the default super-block, *fsck* requests the operator to specify the location of an alternate super-block.

3.3. Free block checking

Fsck checks that all the blocks marked as free in the cylinder group block maps are not claimed by any files. When all the blocks have been initially accounted for, *fsck* checks that the number of free blocks plus the number of blocks claimed by the inodes equals the total number of blocks in the file system.

‡ I.e., unmounted and not being written on.

If anything is wrong with the block allocation maps, *fsck* will rebuild them, based on the list it has computed of allocated blocks.

The summary information associated with the super-block counts the total number of free blocks within the file system. *Fsck* compares this count to the number of free blocks it found within the file system. If the two counts do not agree, then *fsck* replaces the incorrect count in the summary information by the actual free-block count.

The summary information counts the total number of free inodes within the file system. *Fsck* compares this count to the number of free inodes it found within the file system. If the two counts do not agree, then *fsck* replaces the incorrect count in the summary information by the actual free-inode count.

3.4. Checking the inode state

An individual inode is not as likely to be corrupted as the allocation information. However, because of the great number of active inodes, a few of the inodes are usually corrupted.

The list of inodes in the file system is checked sequentially starting with inode 2 (inode 0 marks unused inodes; inode 1 is saved for future generations) and progressing through the last inode in the file system. The state of each inode is checked for inconsistencies involving format and type, link count, duplicate blocks, bad blocks, and inode size.

Each inode contains a mode word. This mode word describes the type and state of the inode. Inodes must be one of six types: regular inode, directory inode, symbolic link inode, special block inode, special character inode, or socket inode. Inodes may be found in one of three allocation states: unallocated, allocated, and neither unallocated nor allocated. This last state suggests an incorrectly formated inode. An inode can get in this state if bad data is written into the inode list. The only possible corrective action is for *fsck* is to clear the inode.

3.5. Inode links

Each inode counts the total number of directory entries linked to the inode. *Fsck* verifies the link count of each inode by starting at the root of the file system, and descending through the directory structure. The actual link count for each inode is calculated during the descent.

If the stored link count is non-zero and the actual link count is zero, then no directory entry appears for the inode. If this happens, *fsck* will place the disconnected file in the *lost+found* directory. If the stored and actual link counts are non-zero and unequal, a directory entry may have been added or removed without the inode being updated. If this happens, *fsck* replaces the incorrect stored link count by the actual link count.

Each inode contains a list, or pointers to lists (indirect blocks), of all the blocks claimed by the inode. Since indirect blocks are owned by an inode, inconsistencies in indirect blocks directly affect the inode that owns it.

Fsck compares each block number claimed by an inode against a list of already allocated blocks. If another inode already claims a block number, then the block number is added to a list of *duplicate blocks*. Otherwise, the list of allocated blocks is updated to include the block number.

If there are any duplicate blocks, *fsck* will perform a partial second pass over the inode list to find the inode of the duplicated block. The second pass is needed, since without examining the files associated with these inodes for correct content, not enough information is available to determine which inode is corrupted and should be cleared. If this condition does arise (only hardware failure will cause it), then the inode with the earliest modify time is usually incorrect, and should be cleared. If this happens, *fsck* prompts the operator to clear both inodes. The operator must decide which one should be kept and which one should be cleared.

Fsck checks the range of each block number claimed by an inode. If the block number is lower than the first data block in the file system, or greater than the last data block, then the block number is a *bad block number*. Many bad blocks in an inode are usually caused by an

indirect block was not written to the file system, a condition which can only occur if there has been a hardware failure. If an inode contains bad block numbers, *fsck* prompts the operator to clear it.

3.6. Inode data size

Each inode contains a count of the number of data blocks that it contains. The number of actual data blocks is the sum of the allocated data blocks and the indirect blocks. *Fsck* computes the actual number of data blocks and compares that block count against the actual number of blocks the inode claims. If an inode contains an incorrect count *fsck* prompts the operator to fix it.

Each inode contains a thirty-two bit size field. The size is the number of data bytes in the file associated with the inode. The consistency of the byte size field is roughly checked by computing from the size field the maximum number of blocks that should be associated with the inode, and comparing that expected block count against the actual number of blocks the inode claims.

3.7. Checking the data associated with an inode

An inode can directly or indirectly reference three kinds of data blocks. All referenced blocks must be the same kind. The three types of data blocks are: plain data blocks, symbolic link data blocks, and directory data blocks. Plain data blocks and symbolic link data blocks contain the information stored in a file. Directory data blocks contain directory entries. *Fsck* can only check the validity of directory data blocks.

Each directory data block is checked for several types of inconsistencies. These inconsistencies include directory inode numbers pointing to unallocated inodes, directory inode numbers that are greater than the number of inodes in the file system, incorrect directory inode numbers for "." and "..", and directories that are not attached to the file system. If the inode number in a directory data block references an unallocated inode, then *fsck* will remove that directory entry. Again, this condition can only arise when there has been a hardware failure.

If a directory entry inode number references outside the inode list, then *fsck* will remove that directory entry. This condition occurs if bad data is written into a directory data block.

The directory inode number entry for "." must be the first entry in the directory data block. The inode number for "." must reference itself; e.g., it must equal the inode number for the directory data block. The directory inode number entry for ".." must be the second entry in the directory data block. Its value must equal the inode number for the parent of the directory entry (or the inode number of the directory data block if the directory is the root directory). If the directory inode numbers are incorrect, *fsck* will replace them with the correct values.

3.8. File system connectivity

Fsck checks the general connectivity of the file system. If directories are not linked into the file system, then *fsck* links the directory back into the file system in the *lost+found* directory. This condition only occurs when there has been a hardware failure.

Acknowledgements

I thank Bill Joy, Sam Leffler, Robert Elz and Dennis Ritchie for their suggestions and help in implementing the new file system. Thanks also to Robert Henry for his editorial input to get this document together. Finally we thank our sponsors, the National Science Foundation under grant MCS80-05144, and the Defense Advance Research Projects Agency (DoD) under Arpa Order No. 4031 monitored by Naval Electronic System Command under Contract No. N00039-82-C-0235. (Kirk McKusick, July 1983)

I would like to thank Larry A. Wehr for advice that lead to the first version of *fsck* and Rick B. Brandt for adapting *fsck* to UNIX/TS. (T. Kowalski, July 1979)

References

[Dolotta78] Dolotta, T. A., and Olsson, S. B. eds., *UNIX User's Manual, Edition 1.1* (January 1978).

[Joy83] Joy, W., Cooper, E., Fabry, R., Leffler, S., McKusick, M., and Mosher, D. *4.2BSD System Manual*, University of California at Berkeley, Computer Systems Research Group Technical Report #4, 1982.

[McKusick83] McKusick, M., Joy, W., Leffler, S., and Fabry, R. *A Fast File System for UNIX*, University of California at Berkeley, Computer Systems Research Group Technical Report #7, 1982.

[Ritchie78] Ritchie, D. M., and Thompson, K., The UNIX Time-Sharing System, *The Bell System Technical Journal* 57, 6 (July-August 1978, Part 2), pp. 1905-29.

[Thompson78] Thompson, K., UNIX Implementation, *The Bell System Technical Journal* 57, 6 (July-August 1978, Part 2), pp. 1931-46.

4. Appendix A — Fsck Error Conditions

4.1. Conventions

Fsck is a multi-pass file system check program. Each file system pass invokes a different Phase of the *fsck* program. After the initial setup, *fsck* performs successive Phases over each file system, checking blocks and sizes, path-names, connectivity, reference counts, and the map of free blocks, (possibly rebuilding it), and performs some cleanup.

Normally *fsck* is run non-interactively to *preen* the file systems after an unclean halt. While preen'ing a file system, it will only fix corruptions that are expected to occur from an unclean halt. These actions are a proper subset of the actions that *fsck* will take when it is running interactively. Throughout this appendix many errors have several options that the operator can take. When an inconsistency is detected, *fsck* reports the error condition to the operator. If a response is required, *fsck* prints a prompt message and waits for a response. When preen'ing most errors are fatal. For those that are expected, the response taken is noted. This appendix explains the meaning of each error condition, the possible responses, and the related error conditions.

The error conditions are organized by the *Phase* of the *fsck* program in which they can occur. The error conditions that may occur in more than one Phase will be discussed in initialization.

4.2. Initialization

Before a file system check can be performed, certain tables have to be set up and certain files opened. This section concerns itself with the opening of files and the initialization of tables. This section lists error conditions resulting from command line options, memory requests, opening of files, status of files, file system size checks, and creation of the scratch file. All of the initialization errors are fatal when the file system is being preen'ed.

C option?
C is not a legal option to *fsck*; legal options are −b, −y, −n, and −p. *Fsck* terminates on this error condition. See the *fsck*(8) manual entry for further detail.

cannot alloc NNN bytes for blockmap
cannot alloc NNN bytes for freemap
cannot alloc NNN bytes for statemap
cannot alloc NNN bytes for lncntp
Fsck's request for memory for its virtual memory tables failed. This should never happen. *Fsck* terminates on this error condition. See a guru.

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

Can't stat root
Fsck's request for statistics about the root directory "/" failed. This should never happen. *Fsck* terminates on this error condition. See a guru.

Can't stat *F*
Can't make sense out of name *F*
Fsck's request for statistics about the file system *F* failed. When running manually, it ignores this file system and continues checking the next file system given. Check access modes of *F*.

Can't open *F*
Fsck's request attempt to open the file system *F* failed. When running manually, it ignores this

file system and continues checking the next file system given. Check access modes of *F*.

F: (NO WRITE)
Either the −n flag was specified or *fsck*'s attempt to open the file system *F* for writing failed. When running manually, all the diagnostics are printed out, but no modifications are attempted to fix them.

file is not a block or character device; OK
You have given *fsck* a regular file name by mistake. Check the type of the file specified.

Possible responses to the OK prompt are:

YES Ignore this error condition.

NO ignore this file system and continues checking the next file system given.

One of the following messages will appear:
MAGIC NUMBER WRONG
NCG OUT OF RANGE
CPG OUT OF RANGE
NCYL DOES NOT JIVE WITH NCG*CPG
SIZE PREPOSTEROUSLY LARGE
TRASHED VALUES IN SUPER BLOCK

and will be followed by the message:
F: BAD SUPER BLOCK: *B*
USE -b OPTION TO FSCK TO SPECIFY LOCATION OF AN ALTERNATE
SUPER-BLOCK TO SUPPLY NEEDED INFORMATION; SEE fsck(8).
The super block has been corrupted. An alternative super block must be selected from among those listed by *newfs* (8) when the file system was created. For file systems with a blocksize less than 32K, specifying −b 32 is a good first choice.

INTERNAL INCONSISTENCY: *M*
Fsck's has had an internal panic, whose message is specified as *M*. This should never happen. See a guru.

CAN NOT SEEK: BLK *B* (CONTINUE)
Fsck's request for moving to a specified block number *B* in the file system failed. This should never happen. See a guru.

Possible responses to the CONTINUE prompt are:

YES attempt to continue to run the file system check. Often, however the problem will persist. This error condition will not allow a complete check of the file system. A second run of *fsck* should be made to re-check this file system. If the block was part of the virtual memory buffer cache, *fsck* will terminate with the message "Fatal I/O error".

NO terminate the program.

CAN NOT READ: BLK *B* (CONTINUE)
Fsck's request for reading a specified block number *B* in the file system failed. This should never happen. See a guru.

Possible responses to the CONTINUE prompt are:

YES attempt to continue to run the file system check. Often, however, the problem will persist. This error condition will not allow a complete check of the file system. A second run of *fsck* should be made to re-check this file system. If the block was part of the

virtual memory buffer cache, *fsck* will terminate with the message "Fatal I/O error".

NO terminate the program.

CAN NOT WRITE: BLK *B* (CONTINUE)
Fsck's request for writing a specified block number *B* in the file system failed. The disk is write-protected. See a guru.

Possible responses to the CONTINUE prompt are:

YES attempt to continue to run the file system check. Often, however, the problem will persist. This error condition will not allow a complete check of the file system. A second run of *fsck* should be made to re-check this file system. If the block was part of the virtual memory buffer cache, *fsck* will terminate with the message "Fatal I/O error".

NO terminate the program.

4.3. Phase 1 — Check Blocks and Sizes

This phase concerns itself with the inode list. This section lists error conditions resulting from checking inode types, setting up the zero-link-count table, examining inode block numbers for bad or duplicate blocks, checking inode size, and checking inode format. All errors in this phase except INCORRECT BLOCK COUNT are fatal if the file system is being preen'ed,

CG *C*: BAD MAGIC NUMBER The magic number of cylinder group *C* is wrong. This usually indicates that the cylinder group maps have been destroyed. When running manually the cylinder group is marked as needing to be reconstructed.

UNKNOWN FILE TYPE I=*I* (CLEAR) The mode word of the inode *I* indicates that the inode is not a special block inode, special character inode, socket inode, regular inode, symbolic link, or directory inode.

Possible responses to the CLEAR prompt are:

YES de-allocate inode *I* by zeroing its contents. This will always invoke the UNALLOCATED error condition in Phase 2 for each directory entry pointing to this inode.

NO ignore this error condition.

LINK COUNT TABLE OVERFLOW (CONTINUE)
An internal table for *fsck* containing allocated inodes with a link count of zero has no more room. Recompile *fsck* with a larger value of MAXLNCNT.

Possible responses to the CONTINUE prompt are:

YES continue with the program. This error condition will not allow a complete check of the file system. A second run of *fsck* should be made to re-check this file system. If another allocated inode with a zero link count is found, this error condition is repeated.

NO terminate the program.

B BAD I=*I*
Inode *I* contains block number *B* with a number lower than the number of the first data block in the file system or greater than the number of the last block in the file system. This error condition may invoke the EXCESSIVE BAD BLKS error condition in Phase 1 if inode *I* has too many block numbers outside the file system range. This error condition will always invoke the BAD/DUP error condition in Phase 2 and Phase 4.

EXCESSIVE BAD BLKS I=*I* (CONTINUE)
There is more than a tolerable number (usually 10) of blocks with a number lower than the number of the first data block in the file system or greater than the number of last block in the file system associated with inode *I*.

Possible responses to the CONTINUE prompt are:

YES ignore the rest of the blocks in this inode and continue checking with the next inode in the file system. This error condition will not allow a complete check of the file system. A second run of *fsck* should be made to re-check this file system.

NO terminate the program.

B* DUP I=*I
Inode *I* contains block number *B* which is already claimed by another inode. This error condition may invoke the **EXCESSIVE DUP BLKS** error condition in Phase 1 if inode *I* has too many block numbers claimed by other inodes. This error condition will always invoke Phase 1b and the **BAD/DUP** error condition in Phase 2 and Phase 4.

EXCESSIVE DUP BLKS I=*I* (CONTINUE)
There is more than a tolerable number (usually 10) of blocks claimed by other inodes.

Possible responses to the CONTINUE prompt are:

YES ignore the rest of the blocks in this inode and continue checking with the next inode in the file system. This error condition will not allow a complete check of the file system. A second run of *fsck* should be made to re-check this file system.

NO terminate the program.

DUP TABLE OVERFLOW (CONTINUE)
An internal table in *fsck* containing duplicate block numbers has no more room. Recompile *fsck* with a larger value of DUPTBLSIZE.

Possible responses to the CONTINUE prompt are:

YES continue with the program. This error condition will not allow a complete check of the file system. A second run of *fsck* should be made to re-check this file system. If another duplicate block is found, this error condition will repeat.

NO terminate the program.

PARTIALLY ALLOCATED INODE I=*I* (CLEAR)
Inode *I* is neither allocated nor unallocated.

Possible responses to the CLEAR prompt are:

YES de-allocate inode *I* by zeroing its contents.

NO ignore this error condition.

INCORRECT BLOCK COUNT I=*I* (*X* should be *Y*) (CORRECT)
The block count for inode *I* is *X* blocks, but should be *Y* blocks. When preen'ing the count is corrected.

Possible responses to the CORRECT prompt are:

YES replace the block count of inode *I* with *Y*.

NO ignore this error condition.

4.4. Phase 1B: Rescan for More Dups

When a duplicate block is found in the file system, the file system is rescanned to find the inode which previously claimed that block. This section lists the error condition when the duplicate block is found.

B DUP I=*I*
Inode *I* contains block number *B* that is already claimed by another inode. This error condition will always invoke the **BAD/DUP** error condition in Phase 2. You can determine which inodes have overlapping blocks by examining this error condition and the DUP error condition in Phase 1.

4.5. Phase 2 — Check Pathnames

This phase concerns itself with removing directory entries pointing to error conditioned inodes from Phase 1 and Phase 1b. This section lists error conditions resulting from root inode mode and status, directory inode pointers in range, and directory entries pointing to bad inodes. All errors in this phase are fatal if the file system is being preen'ed.

ROOT INODE UNALLOCATED. TERMINATING.
The root inode (usually inode number 2) has no allocate mode bits. This should never happen. The program will terminate.

NAME TOO LONG *F*
An excessively long path name has been found. This is usually indicative of loops in the file system name space. This can occur if the super user has made circular links to directories. The offending links must be removed (by a guru).

ROOT INODE NOT DIRECTORY (FIX)
The root inode (usually inode number 2) is not directory inode type.

Possible responses to the FIX prompt are:

YES replace the root inode's type to be a directory. If the root inode's data blocks are not directory blocks, a VERY large number of error conditions will be produced.

NO terminate the program.

DUPS/BAD IN ROOT INODE (CONTINUE)
Phase 1 or Phase 1b have found duplicate blocks or bad blocks in the root inode (usually inode number 2) for the file system.

Possible responses to the CONTINUE prompt are:

YES ignore the **DUPS/BAD** error condition in the root inode and attempt to continue to run the file system check. If the root inode is not correct, then this may result in a large number of other error conditions.

NO terminate the program.

I OUT OF RANGE I=*I* NAME=*F* (REMOVE)
A directory entry *F* has an inode number *I* which is greater than the end of the inode list.

Possible responses to the REMOVE prompt are:

YES the directory entry *F* is removed.

NO ignore this error condition.

UNALLOCATED I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F* (REMOVE)
A directory entry *F* has a directory inode *I* without allocate mode bits. The owner *O*, mode *M*, size *S*, modify time *T*, and directory name *F* are printed.

Possible responses to the REMOVE prompt are:

YES the directory entry *F* is removed.

NO ignore this error condition.

UNALLOCATED I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* FILE=*F* (REMOVE)
A directory entry *F* has an inode *I* without allocate mode bits. The owner *O*, mode *M*, size *S*, modify time *T*, and file name *F* are printed.

Possible responses to the REMOVE prompt are:

YES the directory entry *F* is removed.

NO ignore this error condition.

DUP/BAD I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F* (REMOVE)
Phase 1 or Phase 1b have found duplicate blocks or bad blocks associated with directory entry *F*, directory inode *I*. The owner *O*, mode *M*, size *S*, modify time *T*, and directory name *F* are printed.

Possible responses to the REMOVE prompt are:

YES the directory entry *F* is removed.

NO ignore this error condition.

DUP/BAD I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* FILE=*F* (REMOVE)
Phase 1 or Phase 1b have found duplicate blocks or bad blocks associated with directory entry *F*, inode *I*. The owner *O*, mode *M*, size *S*, modify time *T*, and file name *F* are printed.

Possible responses to the REMOVE prompt are:

YES the directory entry *F* is removed.

NO ignore this error condition.

ZERO LENGTH DIRECTORY I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F* (REMOVE)
A directory entry *F* has a size *S* that is zero. The owner *O*, mode *M*, size *S*, modify time *T*, and directory name *F* are printed.

Possible responses to the REMOVE prompt are:

YES the directory entry *F* is removed; this will always invoke the BAD/DUP error condition in Phase 4.

NO ignore this error condition.

DIRECTORY TOO SHORT I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F* (FIX)
A directory *F* has been found whose size *S* is less than the minimum size directory. The owner *O*, mode *M*, size *S*, modify time *T*, and directory name *F* are printed.

Possible responses to the FIX prompt are:

YES increase the size of the directory to the minimum directory size.

NO ignore this directory.

DIRECTORY CORRUPTED I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F* (SALVAGE)
A directory with an inconsistent internal state has been found.

Possible responses to the FIX prompt are:

YES throw away all entries up to the next 512-byte boundary. This rather drastic action can throw away up to 42 entries, and should be taken only after other recovery efforts have failed.

NO Skip up to the next 512-byte boundary and resume reading, but do not modify the directory.

BAD INODE NUMBER FOR '.' I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F* (FIX)
A directory *I* has been found whose inode number for '.' does does not equal *I*.

Possible responses to the FIX prompt are:

YES change the inode number for '.' to be equal to *I*.

NO leave the inode number for '.' unchanged.

MISSING '.' I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F* (FIX)
A directory *I* has been found whose first entry is unallocated.

Possible responses to the FIX prompt are:

YES make an entry for '.' with inode number equal to *I*.

NO leave the directory unchanged.

MISSING '.' I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F*
CANNOT FIX, FIRST ENTRY IN DIRECTORY CONTAINS *F*
A directory *I* has been found whose first entry is *F*. *Fsck* cannot resolve this problem. The file system should be mounted and the offending entry *F* moved elsewhere. The file system should then be unmounted and *fsck* should be run again.

MISSING '.' I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F*
CANNOT FIX, INSUFFICIENT SPACE TO ADD '.'
A directory *I* has been found whose first entry is not '.'. *Fsck* cannot resolve this problem as it should never happen. See a guru.

EXTRA '.' ENTRY I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F* (FIX)
A directory *I* has been found that has more than one entry for '.'.

Possible responses to the FIX prompt are:

YES remove the extra entry for '.'.

NO leave the directory unchanged.

BAD INODE NUMBER FOR '..' I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F* (FIX)
A directory *I* has been found whose inode number for '..' does does not equal the parent of *I*.

Possible responses to the FIX prompt are:

YES change the inode number for '..' to be equal to the parent of *I*.

NO leave the inode number for '..' unchanged.

MISSING '..' I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F* (FIX)
A directory *I* has been found whose second entry is unallocated.

Possible responses to the FIX prompt are:

YES make an entry for '..' with inode number equal to the parent of *I*.

NO leave the directory unchanged.

MISSING '..' I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F*
CANNOT FIX, SECOND ENTRY IN DIRECTORY CONTAINS *F*
A directory *I* has been found whose second entry is *F*. *Fsck* cannot resolve this problem. The file system should be mounted and the offending entry *F* moved elsewhere. The file system should then be unmounted and *fsck* should be run again.

MISSING '..' I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F*
CANNOT FIX, INSUFFICIENT SPACE TO ADD '..'
A directory *I* has been found whose second entry is not '..'. *Fsck* cannot resolve this problem as it should never happen. See a guru.

EXTRA '..' ENTRY I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F* (FIX)
A directory *I* has been found that has more than one entry for '..'.

Possible responses to the FIX prompt are:

YES remove the extra entry for '..'.

NO leave the directory unchanged.

4.6. Phase 3 — Check Connectivity

This phase concerns itself with the directory connectivity seen in Phase 2. This section lists error conditions resulting from unreferenced directories, and missing or full *lost+found* directories.

UNREF DIR I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* (RECONNECT)
The directory inode *I* was not connected to a directory entry when the file system was traversed. The owner *O*, mode *M*, size *S*, and modify time *T* of directory inode *I* are printed. When preen'ing, the directory is reconnected if its size is non-zero, otherwise it is cleared.

Possible responses to the RECONNECT prompt are:

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

NO ignore this error condition. This will always invoke the UNREF error condition in Phase 4.

SORRY. NO lost+found DIRECTORY
There is no *lost+found* directory in the root directory of the file system; *fsck* ignores the request to link a directory in *lost+found*. This will always invoke the UNREF error condition in Phase 4. Check access modes of *lost+found*. See *fsck*(8) manual entry for further detail. This error is fatal if the file system is being preen'ed.

SORRY. NO SPACE IN lost+found DIRECTORY
There is no space to add another entry to the *lost+found* directory in the root directory of the file system; *fsck* ignores the request to link a directory in *lost+found*. This will always invoke the UNREF error condition in Phase 4. Clean out unnecessary entries in *lost+found* or make

lost+found larger. See *fsck*(8) manual entry for further detail. This error is fatal if the file system is being preen'ed.

DIR I=*I1* CONNECTED. PARENT WAS I=*I2*
This is an advisory message indicating a directory inode *I1* was successfully connected to the *lost+found* directory. The parent inode *I2* of the directory inode *I1* is replaced by the inode number of the *lost+found* directory.

4.7. Phase 4 — Check Reference Counts

This phase concerns itself with the link count information seen in Phase 2 and Phase 3. This section lists error conditions resulting from unreferenced files, missing or full *lost+found* directory, incorrect link counts for files, directories, symbolic links, or special files, unreferenced files, symbolic links, and directories, bad and duplicate blocks in files, symbolic links, and directories, and incorrect total free-inode counts. All errors in this phase are correctable if the file system is being preen'ed except running out of space in the lost+found directory.

UNREF FILE I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* (RECONNECT)
Inode *I* was not connected to a directory entry when the file system was traversed. The owner *O*, mode *M*, size *S*, and modify time *T* of inode *I* are printed. When preen'ing the file is cleared if either its size or its link count is zero, otherwise it is reconnected.

Possible responses to the RECONNECT prompt are:

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

NO ignore this error condition. This will always invoke the CLEAR error condition in Phase 4.

(CLEAR)
The inode mentioned in the immediately previous error condition can not be reconnected. This cannot occur if the file system is being preen'ed, since lack of space to reconnect files is a fatal error.

Possible responses to the CLEAR prompt are:

YES de-allocate the inode mentioned in the immediately previous error condition by zeroing its contents.

NO ignore this error condition.

SORRY. NO lost+found DIRECTORY
There is no *lost+found* directory in the root directory of the file system; *fsck* ignores the request to link a file in *lost+found*. This will always invoke the CLEAR error condition in Phase 4. Check access modes of *lost+found*. This error is fatal if the file system is being preen'ed.

SORRY. NO SPACE IN lost+found DIRECTORY
There is no space to add another entry to the *lost+found* directory in the root directory of the file system; *fsck* ignores the request to link a file in *lost+found*. This will always invoke the CLEAR error condition in Phase 4. Check size and contents of *lost+found*. This error is fatal if the file system is being preen'ed.

LINK COUNT FILE I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* COUNT=*X* SHOULD BE *Y* (ADJUST)
The link count for inode *I* which is a file, is *X* but should be *Y*. The owner *O*, mode *M*, size *S*,

and modify time *T* are printed. When preen'ing the link count is adjusted.

Possible responses to the ADJUST prompt are:

YES replace the link count of file inode *I* with *Y*.

NO ignore this error condition.

LINK COUNT DIR I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* COUNT=*X* SHOULD BE *Y* (ADJUST)

The link count for inode *I* which is a directory, is *X* but should be *Y*. The owner *O*, mode *M*, size *S*, and modify time *T* of directory inode *I* are printed. When preen'ing the link count is adjusted.

Possible responses to the ADJUST prompt are:

YES replace the link count of directory inode *I* with *Y*.

NO ignore this error condition.

LINK COUNT *F* I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* COUNT=*X* SHOULD BE *Y* (ADJUST)

The link count for *F* inode *I* is *X* but should be *Y*. The name *F*, owner *O*, mode *M*, size *S*, and modify time *T* are printed. When preen'ing the link count is adjusted.

Possible responses to the ADJUST prompt are:

YES replace the link count of inode *I* with *Y*.

NO ignore this error condition.

UNREF FILE I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* (CLEAR)

Inode *I* which is a file, was not connected to a directory entry when the file system was traversed. The owner *O*, mode *M*, size *S*, and modify time *T* of inode *I* are printed. When preen'ing, this is a file that was not connected because its size or link count was zero, hence it is cleared.

Possible responses to the CLEAR prompt are:

YES de-allocate inode *I* by zeroing its contents.

NO ignore this error condition.

UNREF DIR I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* (CLEAR)

Inode *I* which is a directory, was not connected to a directory entry when the file system was traversed. The owner *O*, mode *M*, size *S*, and modify time *T* of inode *I* are printed. When preen'ing, this is a directory that was not connected because its size or link count was zero, hence it is cleared.

Possible responses to the CLEAR prompt are:

YES de-allocate inode *I* by zeroing its contents.

NO ignore this error condition.

BAD/DUP FILE I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* (CLEAR)

Phase 1 or Phase 1b have found duplicate blocks or bad blocks associated with file inode *I*. The owner *O*, mode *M*, size *S*, and modify time *T* of inode *I* are printed. This error cannot arise when the file system is being preen'ed, as it would have caused a fatal error earlier.

Possible responses to the CLEAR prompt are:

YES de-allocate inode *I* by zeroing its contents.

NO ignore this error condition.

BAD/DUP DIR I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* (CLEAR)
Phase 1 or Phase 1b have found duplicate blocks or bad blocks associated with directory inode *I*. The owner *O*, mode *M*, size *S*, and modify time *T* of inode *I* are printed. This error cannot arise when the file system is being preen'ed, as it would have caused a fatal error earlier.

Possible responses to the CLEAR prompt are:

YES de-allocate inode *I* by zeroing its contents.

NO ignore this error condition.

FREE INODE COUNT WRONG IN SUPERBLK (FIX)
The actual count of the free inodes does not match the count in the super-block of the file system. When preen'ing, the count is fixed.

Possible responses to the FIX prompt are:

YES replace the count in the super-block by the actual count.

NO ignore this error condition.

4.8. Phase 5 - Check Cyl groups

This phase concerns itself with the free-block maps. This section lists error conditions resulting from allocated blocks in the free-block maps, free blocks missing from free-block maps, and the total free-block count incorrect.

CG *C*: BAD MAGIC NUMBER
The magic number of cylinder group *C* is wrong. This usually indicates that the cylinder group maps have been destroyed. When running manually the cylinder group is marked as needing to be reconstructed. This error is fatal if the file system is being preen'ed.

EXCESSIVE BAD BLKS IN BIT MAPS (CONTINUE)
An inode contains more than a tolerable number (usually 10) of blocks claimed by other inodes or that are out of the legal range for the file system. This error is fatal if the file system is being preen'ed.

Possible responses to the CONTINUE prompt are:

YES ignore the rest of the free-block maps and continue the execution of *fsck*.

NO terminate the program.

SUMMARY INFORMATION *T* BAD
where *T* is one or more of:
(INODE FREE)
(BLOCK OFFSETS)
(FRAG SUMMARIES)
(SUPER BLOCK SUMMARIES)
The indicated summary information was found to be incorrect. This error condition will always invoke the **BAD CYLINDER GROUPS** condition in Phase 6. When preen'ing, the summary information is recomputed.

***X* BLK(S) MISSING**
X blocks unused by the file system were not found in the free-block maps. This error condition will always invoke the **BAD CYLINDER GROUPS** condition in Phase 6. When preen'ing, the block maps are rebuilt.

BAD CYLINDER GROUPS (SALVAGE)
Phase 5 has found bad blocks in the free-block maps, duplicate blocks in the free-block maps,

or blocks missing from the file system. When preen'ing, the cylinder groups are reconstructed.

Possible responses to the SALVAGE prompt are:

YES replace the actual free-block maps with a new free-block maps.

NO ignore this error condition.

FREE BLK COUNT WRONG IN SUPERBLOCK (FIX)

The actual count of free blocks does not match the count in the super-block of the file system. When preen'ing, the counts are fixed.

Possible responses to the FIX prompt are:

YES replace the count in the super-block by the actual count.

NO ignore this error condition.

4.9. Phase 6 - Salvage Cylinder Groups

This phase concerns itself with the free-block maps reconstruction. No error messages are produced.

4.10. Cleanup

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

***V* files, *W* used, *X* free (*Y* frags, *Z* blocks)**

This is an advisory message indicating that the file system checked contained *V* files using *W* fragment sized blocks leaving *X* fragment sized blocks free in the file system. The numbers in parenthesis breaks the free count down into *Y* free fragments and *Z* free full sized blocks.

******* REBOOT UNIX *******

This is an advisory message indicating that the root file system has been modified by *fsck*. If UNIX is not rebooted immediately, the work done by *fsck* may be undone by the in-core copies of tables UNIX keeps. When preen'ing, *fsck* will exit with a code of 4. The auto-reboot script interprets an exit code of 4 by issuing a reboot system call.

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

This is an advisory message indicating that the current file system was modified by *fsck*. If this file system is mounted or is the current root file system, *fsck* should be halted and UNIX rebooted. If UNIX is not rebooted immediately, the work done by *fsck* may be undone by the in-core copies of tables UNIX keeps.