RISC/os (UMIPS) System Administrator's Guide Volume II

Order Number 3206DOC

mips

The power of RISC is in the system.

RISC/os (UMIPS) System Administrator's Guide Volume II

Order Number 3206DOC

April 1989

Your comments on our products and publications are welcome. A postage–paid form is provided for this purpose on the last page of this manual.

Mfg. Part Number 84–00039–B/02–00136

MIPS Computer Systems, Inc.
930 Arques Ave.
Sunnyvale, CA 94086

Customer Service Telephone Numbers:

California:	(800)	992–MIPS
All other states:	(800)	443–MIPS
International:	(415)	330–7966

Mfg. Part Number 84–00039–B/02–00136

Introduction

This part of the UMIPS System Administrator's Guide contains procedures that can be followed to implement UMIPS functions. The procedures are concerned with system administrator input and system output but not with describing the reasons for the various actions. To understand why these procedures are performed and to understand them better in order to tailor them to your specific needs, first refer to the corresponding chapter in the first half of this document. In addition, references to particular manual pages (or "man" pages) are referenced throughout this document. They can be read in the accompanying manuals, for example, the System Administrator's Reference and the User's Reference Manual. They can also be accessed from the UMIPS system by entering:

$ **man** *command_name*

where *command_name* is the UMIPS command name.

System Administration Commands

To simplify system administration, UMIPS provides the System Administration Package or **sysadm** menu interface to help perform common administrative tasks. In addition, all administrative tasks can be performed by using the "manual" approach of the more traditional UNIX command sequences. For example, to add a new user one may simply enter:

sysadm adduser

and then follow the prompts until the user has been added to the system. Adding a user can also be performed by making a user directory, adding an entry to the **/etc/passwd** file, checking the **/etc/group** file and a few other actions also performed with UMIPS commands. The Procedures always document both approaches when they are available. Use whichever one you feel more comfortable with.

The procedures shown in this guide by-pass the higher level System Administration Menus and take you directly to the subcommands. Subcommands are the equivalent of menu selections from lower level menus. If you prefer, you may start at the main menu with the unadorned command

sysadm

You will see the list of subcommand choices:

```
  SYSTEM ADMINISTRATION

1 filemgmt  file management menu
2 machinemgmt     machine management menu
3 syssetup  system setup menu
4 ttymgmt   tty management menu
5 usermgmt  user management menu

Enter a number, a name, the initial part of a name, or
? or <number>? for HELP,  q to QUIT:
```

You can now enter a number or name to pursue the various choices. Alternatively, from the command prompt, you can get to the submenu level directly with a command like

```
$ sysadm filemgmt
Password:
```

System States

In some procedures, we state that a particular system state is required. This means that the system must be in standalone, single-user or multi-user modes. The standalone mode corresponds to the monitor and standalone shell (sash) levels. Single-user mode corresponds to run level 1, while the multi-user mode corresponds to run levels 2 or 3. Procedures for bringing the system to different system states are found in Procedure 3, System State Procedures. See the section on "Operating Levels," in Chapter 3, System States, for more information on system states.

Logins

In some procedures, we state that a particular login is required. This frequently means that you must be logged in as **root** to do the procedure. The phrase "an authorized login" is also used. The standard meaning of this term is that you must log in using a special administrative or system login name to do the procedure (see Chapter 1, System Security, for a description of these logins).

Passwords

It is strongly recommended that you set up and use passwords for administrative and system logins (see Procedure 1.4 for information on how to do this). In the procedures, we assume that such password protection has been established. When you enter a **sysadm** command as an ordinary user, therefore, you are prompted for a password. We show this with an entry like:

```
$ sysadm adduser
Password:
```

At this point, to go ahead with the procedure, you are required to enter an acceptable password. As is always the case in the UNIX system, the password is not echoed to your screen.

In procedures that require you to be logged in as **root** (that is, the super-user), you are not prompted for the **sysadm** password. Also, the pound sign (**#**) prompt is used to represent the **root** login. Here's an example:

```
# sysadm addgroup
#
```

Information in the Examples

While every effort has been made to present displays of information just as they appear on your terminal, it is possible that your system will produce slightly different output. Some displays reflect a particular machine configuration that may differ from yours. Changes between releases of the UMIPS system software may cause small differences in what appears on your terminal.

System Identification and Security Procedures

The following procedures are covered in this section:

Procedure 1.1	**Bringing up a New System** To ready the system for the following procedures.
Procedure 1.2	**Set Time and Date** To set the time and date of the internal system clock.
Procedure 1.3	**Establish or Change System Node Name** To define the formal system name, especially for the computer to be a node in a network.
Procedure 1.4	**Assign Special Administrative Passwords** To assign special passwords to administrative and system logins.
Procedure 1.5	**Forgotten Root Password Recovery** To recover from forgetting or the corruption of the root password.

Procedure 1.1: Bringing-Up a New System

This introductory procedure is designed to get the system up and running so the following procedures can be performed.

References

Release Notes
Terminal Documentation

Step 1: Many system administration functions are performed at the console terminal, which should be set to a 9600 baud rate.

Here are some things you might want to do to make sure your console terminal is configured properly. Use the Operator's Guide for your terminal to learn how to make these equipment checks.

- Set the input/output terminal speed option to 9600.
- Set the interface to 8-bit ASCII character mode, full duplex, stop bits equal to "1", with a parity of "none", "disabled" or "space" depending on the terminal you have.
- If you lose communication with the system, check to see if the terminal is still plugged in.

A printer should be part of the console equipment configuration because it provides a record of exactly what was done and how the system responded. It is essentially the system log, and it is especially advantageous when you run diagnostics. If your console terminal has this capability, the best method is to hook the printer directly to the system console. (If you have a Teletype 5425, for example, there is an auxiliary port on the rear of the terminal. Other manufacturers' brands offer the same feature.) The console printer should be independent of the LP Spooler system (discussed in Procedure 7).

Step 2: To bring up an M-Series RISComputer for the first time, power it on and then wait for the monitor prompt ">>" at the console. At this point, enter "**auto**":

```
>> auto
```

The system will come up to multiuser mode in a few minutes, performing various system checks and echoing results out to the screen. It is not necessary to answer any prompts at this time.

When the startup procedure is complete, the system will display the console login:

```
mips Console login:
```

Procedure 1.2: Set Time and Date

This procedure is used to set the system time and date.

References

Release Notes
sysadm datetime(1)
date(1)

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

Setting the clock is required when you

- first get the system
- reset nonvolatile random access memory (NVRAM)
- replace a failed clock battery

Step 1: To set the time and the date use the System Administration Menu **datetime**. For example:

```
$ sysadm datetime
Password:

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


Current time and time zone is: 04:59 EDT
Change the time zone? [y, n, q, ?] n
Current date and time:  Tue. 08/28/85 05:00
Change the date and time: [y, n, q, ?] y
Month    default 08       (1-12): <CR>
                (Using <CR> to accept the default)
Day      default 28       (1-31): <CR>
Year     default 84       (70-99): <CR>
Hour     default 05       (0-23): <CR>
Minute   default 00       (0-59): 04
Date and time will be set to:  08/28/84 05:04. OK? [y, n, q] y
The date and time are now changed.
```

Step 2: The clock also can be set using the **date** command. You must be logged in as **root** to set the date and time with **date**. The arguments to the **date** command are in the sequence of month, day, hour, minute, and year.

```
# date 0216131688
Sun Feb 16 13:16:00 EST 1988
```

Procedure 1.3: Establish or Change System Name

This procedure demonstrates how to determine the current name of the system and how to change it temporarily or "permanently".

References

uname(1)
hostname(1)

Command—uname

The name of the RISComputer can be set by using the **uname** command. Use the **uname** command to list system and node information:

```
# uname -a
mips mips 3_0 UMIPS mips
# uname -A
mips mips 3_0 UMIPS mips m120 ATT_V3_0
#
```

Changing the System Name

To change the name of your system temporarily, use the **hostname** command:

```
# hostname newname
# uname
newname
```

This name will last until the system is rebooted or the name is otherwise changed. To make a lasting name change, edit the **/etc/local_hostname** file, replacing the current name ("no_hostname_set" on a new system) with the chosen name. The contents of **/etc/local_hostname** look something like this:

```
      mips
      netmask 0xff000000 broadcast 192.255.255.255
```

Procedure 1.4: Assign Special Administrative Passwords

This procedure demonstrates how to use various commands to assign and change passwords. The M-Series systems are shipped with passwords for all logins with the exception of root, who should be assigned a password immediately.

References

sysadm syspasswd
sysadm admpasswd
passwd(1)

After you have set up your RISComputer you should assign passwords to the special administrative and system logins (see Chapter 1, System Security, for definitions of these logins). The administrative logins are: **setup**, **powerdown**, and **sysadm**,

The system logins are: **sys**, **adm**, **bin**, **uucp**, **nuucp**, **lp**, **rje**, **daemon**, and **trouble**.

Step 1: The following command will step you through all the special administrative logins to see whether you want to assign or change any of the passwords.

```
# sysadm admpasswd
```

Note: if you want to change the passwords for any of these logins, you may do it with this command. For example, the following is displayed:

```
Running subcommand 'admpasswd' from menu 'syssetup',
SYSTEM SETUP
Do you want to give passwords to administrative \
      logins? [y, n, ?, q] y
```

If you enter "y", you are prompted about each administrative login.

```
The login 'setup' already has a password.
Do you want to change the password, delete it, or skip it?
[c, d, s, q, ?] s
Password unchanged.
The login 'powerdown' already has a password.
Do you want to change the password, delete it, or skip it?
[c, d, s, q, ?] q
Password unchanged.
#
```

Step 2: To assign a password to the special system logins, enter:

```
# sysadm syspasswd
        Note: if you want to change any of these passwords,
        you must either be logged in as root or as one
        of these logins, and then execute the passwd command.
        For example, the following is displayed:
Running subcommand 'syspasswd' from menu 'syssetup',
SYSTEM SETUP
Do you want to give passwords to system \
        logins? [y, n, ?, q] y
Do you want to give the 'daemon' login a \
        password? [y, n, ?, q] n
The following system logins still do not have passwords:
        daemon
```

This command will only let you assign passwords to those logins that have never received a password in the first place.

Step 3: You can also assign passwords individually with the **passwd** command. As superuser enter:

```
# passwd username
```

and you will be prompted for the new password information. For example:

```
# passwd johnr
New password:
Re-enter new password:
#
```

User "johnr" now has a new password.

Procedure 1.5: Forgotten Root Password Recovery

If you have forgotten the root password and your system automatically boots up to multiuser mode, you are faced with the unfortunate situation of having to supply the root password to the login prompt before you can get in there and edit the **password** file. The way to get around this is to come-up only as far as single-user mode as described in this Procedure. You can then edit the password file and assign a new password.

References

shutdown(1M)
passwd(1)

If the system is in monitor mode (i.e., the prompt is ">>"), go to Step 3. If the system is in single-user state, skip to Step 4. If the system is in the multi-user state, skip to Step 2. (Refer to Chapter 3 for a discussion of system states.)

Step 1: Follow this step if the system is turned off.

If your system is off, power-up the system (i.e., press the Power button) to get to the monitor prompt:

```
>>
```

Now go to Step 3.

Step 2: From the multiuser mode, it will be necessary to shut the system down in a manner less than graceful.

First, be sure everyone is off the system. If you are unable to **write**(1) them, you will have to resort to such archaic measures as telephone or (gasp!) walking over to their terminal. Next, **sync**(1) the system a few times and then press the reset button or key switch. Your session will look something like this:

```
# who
root        tty0         May 13 09:58
johnr       ttyq0        May 16 16:31
# write johnr
Please logoff. The system is coming down for a few minutes.
# sync;sync;sync
*Intrepid* MIPS Monitor Version 4.0 MIPS OPT Wed Apr 13 \
      16:40:45 PDT 1988
Memory size: 16777216 (0x1000000) bytes
Icache size: 65536 (0x10000) bytes
Dcache size: 65536 (0x10000) bytes
>>
```

The ">>" is the monitor prompt.

User Services Procedures

The following procedures are covered in this section:

Procedure 2.1	**Add Users or Groups** To add information about new users of the system, or to name groups.
Procedure 2.2	**Modify User Information** To change information about users or groups.
Procedure 2.3	**Delete Users or Groups** To remove users or groups from the system.
Procedure 2.4	**List Users or Groups** To display information about users or groups.
Procedure 2.5	**Write to All Users** To send a message to all users logged in.

Procedure 2.1: Add Users or Groups

The following procedure identifies new users and groups to the system. The way to perform the same functions without using the **sysadm** command is also presented in this procedure.

References

sysadm adduser(1)
sysadm addgroup(1)

Step 1: Enter one of the following commands:

```
   $ sysadm adduser
Password:

   or

$ sysadm addgroup
Password:
```

Step 2: The command **sysadm adduser** is not currently implemented.

Step 3: If you enter the command **sysadm addgroup**, this sequence appears:

```
Running subcommand 'addgroup' from menu 'usermgmt',
USER MANAGEMENT

Anytime you want to quit, type "q".
If you are not sure how to answer any prompt, type "?" for help,
or see the Owner/Operator Manual.

If a default appears in the question, press <RETURN> for \
      the default.

Enter group name [?, q]: seventy7
Enter group ID number (default 45201) [?, q]: <CR>
   (Accepting default by entering <CR>)
This is the information for the new group:
   Group name:      seventy7
   group ID:        45201
Do you want to install, edit or skip this entry [i, e, s, q]? i
Group installed

Do you want to add another group?[y, n, q]   n
```

Adding Users Using Commands

Users can be added with a sequence of commands that perform the following:

create a home directory

add an entry to **/etc/passwd**

add "environment" files

In the following example, we'll set up a new user named pat. The example assumes you are logged in as superuser (root).

Step 1: Make a new directory in your user area for pat:

```
# mkdir /user/pat
```

Step 2: Using a text editor, add an entry for pat in the **/etc/passwd** file. The example entry in **/etc/passwd** shown here is explained below.

```
pat::649:40:Pat A. Jones:/user/pat:/bin/csh
```

The entry for pat shows seven colon-separated fields. The first field, "pat" is the login-name of the user. The next field is temporarily empty - in a moment we'll assign pat a password. The third field, "649", is a unique user id number , and is followed by the group number, in this case "40".a Next, is a description of the user. The last two fields are the user's home directory and login shell, **/user/pat** and **/bin/csh**, respectively.

Step 3: Save the password file and, add a password for pat with the **passwd** command:

```
# passwd pat
New Password:
Re-enter new password:
#
```

Step 4: Copy any login and environment files to the user's home directory:

```
# cp /etc/stdcshrc /user/pat/.cshrc
# chown pat /user/pat /user/pat/.cshrc
```

The "cshrc" file is copied to pat's home directory because she is a C-shell user. The directory and file permissions are changed to give pat control of the contents of **/user/pat**.

Procedure 2.2: Modify User Information

This procedure demonstrates how to later change some of the values assigned to a user initially with the previous procedure.

References
sysadm modadduser

- Enter the command:

```
$ sysadm modadduser
Password:
```

The **sysadm modadduser** command gives you the opportunity to change either or both of the default values for group ID and home (parent) directory that appear on the **adduser** form. The screen below shows an example of changing the default group number from **1** to **100**.

```
Running subcommand 'modadduser' from menu 'usermgmt'
USER MANAGEMENT

Anytime you want to quit, type "q".
If you are not sure how to answer any prompt, type "?" for help,
or see the Owner/Operator Manual.

Current defaults for adduser:
group ID                1       (other)
parent directory        /usr
Do you want to change the default group ID? [y, n, ?, q] y
Enter group ID number or group name [?, q] 100
Do you want to change the default parent \
      directory? [y, n, ?, q] n

These will be the new defaults:
group ID:               100
parent directory:       /usr
Do you want to keep these values? [y, n, q] y
Defaults installed.
$
```

Changes can also be made by directly modifying the **/etc/passwd** and **/etc/group** files. For example, a new group can be defined in **/etc/group** and then the group field in **/etc/passwd** can be edited to switch users to the new group.

Procedure 2.3: Delete Users or Groups

Use this procedure to delete users and/or groups.

References

sysadm delgroup(1)
sysadm deluser(1)

Step 1: Deleting a group ID is done with this command:

```
$ sysadm delgroup
Password:
```

Step 2: The prompt sequence is as follows:

```
Which group name do you wish to delete?[q] seventy7
Do you want to delete group name \
            'seventy7', group ID 45201?[y, n, ?, q] y
seventy7 has been deleted
Do you want to delete any other group?[y, n, q] q
```

NOTE The **sysadm delgroup** command deletes only the specified group and not the user login(s) assigned to that group. The logins belonging to the group must be deleted separately using **sysadm deluser**.

Step 3: Deleting a user's login ID requires more persistence. The user's home directory and all the files in and below that directory are deleted as well. Here is the sequence:

```
$ sysadm deluser
Password:

Running subcommand 'deluser' from menu 'usermgmt',
USER MANAGEMENT

This function COMPLETELY REMOVES THE USER, their mail file,
home directory and all files below their home directory
from the machine.  Once this is done, there is no way
guaranteed to get them all back.
        BE SURE THIS IS WHAT YOU WANT TO DO!

Enter login ID you wish to remove[q]: jqp
        'jqp' belongs to 'John Q. Public'
        whose home directory is /usr/jqp
Do you want to remove login ID 'jqp'?[y, n, ?, q] y

/usr/jqp and all files under it have been removed.

Enter login ID you wish to remove [q]: q
```

Removing Users or Groups with Command Sequences

To remove a group from your system "manually", delete the line containing that group entry from **/etc/group**. Remember, this does not remove the users. If there are users belonging to this group that you do not want removed from the system, it is necessary to move them to another group. Do this by changing the group field in their **/etc/passwd** entry to another group id number. Then use **chgrp**(1) to change the group ownership of the user's files.

To remove users from the system, remove their **/etc/passwd** entry. Also, remove the files in their home- and sub-directories and their mail files (see **mail**(1)). It is good idea to back up these files before removing them and then saving them for a while in case data in them is found to be needed later.

Procedure 2.4: List Users or Groups

Use this procedure to get a listing of users and groups.

References

sysadm lsgroup(1)

Step 1: The two **sysadm** subcommands in this procedure enable you to see what groups and what users are in the computer. The command to list groups is:

```
$ sysadm lsgroup
Password:
```

which produces a report with these column headings:

```
Groups currently in the computer
(press <RETURN> to start listing each time you hear the bell)

group  group        logins permitted to become
name   number       members using newgrp
-----  ------     ---------------------------
adm      4        root,adm,daemon
bin      2        root,bin,daemon
daemon  12        root,daemon
mail     6        root
other    1
rje      8        rje,shqer
root     0        root
sys      3        root,bin,sys,adm
$
```

Step 2: If you enter the command:

```
$ sysadm lsuser
Password:
```

the following lines appear on your terminal:

```
Users currently in the computer
(press <RETURN> to start listing each time you hear the bell)
```

<CR>

When you press <RETURN> a list in the following form is displayed:

```
Users currently in the computer:
(press <RETURN> to start printing each time you hear the bell)
login name      user name
----------     ---------
ark          Allen Kramer
fdr          Fred D. Richards
lbj          Larry John
zip          Zippy Pinhead
        .
        .
        .
$
```

Listing the Group and Password Files

Another way to view the group and user entries is to use a UMIPS to command to print the **/etc/passwd** and **/etc/group** files. For example,

```
# less /etc/passwd
```

and

```
# cat /etc/group
```

cause the two files to be output to the screen - the first command uses **less**(1) to page through the **/etc/passwd** file a screenful at a time, and the second command uses **cat**(1) to print the entire **/etc/group** file to the screen.

Procedure 2.5: Write to All Users

This procedure demonstrates how to broadcast a message to all users on the system simultaneously.

References
wall(1)

Step 1: For times when it is necessary to communicate with all users on the system at once, the UMIPS **wall** command is used.

```
# wall
```

The command reads whatever you type in at your terminal until it reads an end-of-file (indicated by typing in a **control-d**).

Step 2: The message you type in is sent immediately to the terminal of all users logged in. It is preceded by:

```
Broadcast Message from ...
```

A typical use of the **wall** command is to warn users that the system is about to be shutdown:

```
Broadcast Message from root:  System coming down in
ten minutes.  Please log off.
```

System States Procedures

The following procedures are covered in this section:

Procedure 3.1	**Powerup** To power up the system to the multi-user state.
Procedure 3.2	**Powerdown** To halt the system and turn the power off.
Procedure 3.3	**Shutdown to Single-User** To bring the system to the single-user state to do administrative tasks.
Procedure 3.4	**Return to Multi-User** To return the system to the multi-user state for standard operations.
Procedure 3.5	**Running Monitor Programs** To bring the system to the monitor mode to run standalone programs.
Procedure 3.6	**Halt and Reboot the Operating System** To halt and reboot the system from the hard disk.
Procedure 3.7	**Recovery from System Trouble** To handle system troubles due to hardware or software problems.
Procedure 3.8	**Install the Operating System** To reload the system from tape, if the system itself has been severely damaged or to change the disk partitions. Procedures include update install and a full install.

Procedure 3.1: Powerup

This procedure is used to turn on the system and bring it up to its default run-level of multi-user mode.

References
auto(1SPP)
boot(1SPP)

To power up the RISComputer perform the following procedure:

Step 1: Turn on the console and wait for the cursor to appear.

Step 2: Press the Power button on the RISComputer and you will see output similar to the following on the console:

```
MIPS Monitor Version 1.5 MIPS OPT Mon Sep 21 \
      15:41:43 PDT 1987 root
Memory size: 4194304 (0x400000) bytes
Icache size: 16384 (0x4000) bytes
Dcache size: 8192 (0x2000) bytes
```

Step 3: Enter "auto" at the boot monitor prompt: (Note that your system keyswitch must be switched on to perform this step.)

>> **auto**

```
Autoboot: waiting to load dkip(0,0,8)sash (CRTL-C to abort) . . .
81120+15232+203056 entry: 0xa0300000
MIPS Standalone Shell Version 1.5 MIPS OPT Fri Nov 13 \
      07:03:57 PST 1987 root

Loading dkip(0,0,0)/unix
557456+68640+373248 entry: 0x80021000
CPU: MIPS R2000 Processor Chip Revision: 3.0
Delay multiplier = 3, cnt = 761

UNIX System V Release 2_0 mips Version UMIPS_V
Total real memory  = 4194304
Available memory   = 2879488

if_enp1: controller not available
ips1: controller not available
root on dev 0x400 (fstyp is ffs)
Available memory   = 2752512

Checking root file system () automatically.

The system is coming up.  Please wait.

***** Normally all file systems are fscked.
***** To fsck only dirty ones, type 'yes' within 5 seconds:
***** All file systems will be fscked.
mountall: fscking /dev/usr (/usr).
** /dev/usr
** Last Mounted on
** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Cyl groups
4562 files, 103133 used, 85232 free (224 frags, \
      10626 blocks, 0.1% fragmentation)

***** FILE SYSTEM WAS MODIFIED *****
/dev/usr mounted on /usr
Internet daemons: routed portmap inetd rwhod.
NFS daemons: nfsd biod.
The system is ready.
```

Step 4: Log on to the system when the prompt `MIPS Console Login:` appears. You can log in with a system or user login. This is multiuser mode.

```
mips Console login: rootcsh
Password:
UNIX System V Release 3_0 mips
Copyright (c) 1984 AT&T
All Rights Reserved
*************************************************
*                                               *
*       MIPS - The Measure of Performance       *
*                                               *
*************************************************
#
```

Procedure 3.2: Powerdown

This procedure is used to halt the system and bring it down gracefully. The process of bringing the system down may be performed using **sysadm** commands or by using the **shutdown** script.

References

sysadm powerdown(1)
shutdown(1M)

There are differences in the procedure depending on whether you are in multi-user or single-user state.

From Multi-User

The best way to turn off the computer while the system is in the multi-user state is to enter the **sysadm powerdown** command. Entering this command causes the system to flush the system buffers, close any open files, stop all user processes and daemons currently running, unmount file systems, and bring the system down to monitor mode at which time power can be safely turned off.

Step 1: Check who is logged in before taking any action that would affect a logged-in user. Enter:

```
# sysadm whoson
```

A typical response might be:

```
These users are currently logged in:
ID           terminal number    sign-on time
-------      ---------------    ------------
root         console            18:06
jaf          tty22              22:30
```

Step 2: Notify any users that the system is shutting down via the **/etc/wall**(1M) command (see Procedure 2.5). For example:

```
# /etc/wall <CR>
Broadcast Message from root (console) on unix Wed Feb 26 07:30:27...
The system will be coming down in 5 minutes.
Please log off. <CTRL-D>
```

Step 3: Enter:

# sysadm powerdown

Observe the following output and enter appropriate prompt responses:

```
Running subcommand 'powerdown' from 'machinemgmt',
MACHINE MANAGEMENT

Once started, a powerdown CANNOT BE STOPPED.
Do you want to start an express powerdown? [y, n, ?, q] n
Enter the number of seconds to allow
between the warning messages (default 60): [?, q] 30

Shutdown started.     Thu May 16 17:10:57...

Broadcast Message from root (console)  Thu May 16 17:10:59
THE SYSTEM IS BEING SHUT DOWN NOW ! ! !
Log off now or risk your files being damaged.

INIT: New run level: 0
The system is coming down.  Please wait.
System services are now being stopped.

The system is down.
MIPS Monitor Version 1.5 MIPS OPT Mon Sep 21 \
              15:41:43 PDT 1987 root
Memory size: 4194304 (0x400000) bytes
Icache size: 16384 (0x4000) bytes
Dcache size: 8192 (0x2000) bytes
>>
```

The Power button may now be pressed to turn power off.

From Single-User

If the system is in the single-user state, use the following command to powerdown the system.

Step 1: To remove power and guarantee file system integrity, use the **shutdown** command as follows:

# shutdown −y −i0 −g0

The arguments have the following meanings:

−y	assume yes answers to all questions
−i0	go to state 0 (monitor mode)
−g0	allow grace period of 0 seconds

Observe output similar to the following on the console:

```
Shutdown started.

The system will be shutdown in 0 seconds.
Please log off now.
THE SYSTEM IS BEING SHUTDOWN NOW ! ! !
Log off now or risk your files being damaged.
The system is coming down. Please wait.

The system is down.
MIPS Monitor Version 1.5 MIPS OPT Mon Sep 21 2
            15:41:43 PDT 1987
Memory size: 4194304 (0x400000) bytes
cache size: 16384 (0x4000) bytes
Dcache size: 8192 (0x2000) bytes
>>
```

All services are stopped and power may be removed from the machine.

Procedure 3.3: Shutdown to Single-User

This procedure demonstrates how to shut the system down to single-user mode. Single-user mode is used to perform various administrative tasks that must be performed when the system is in single-user mode, for example, file backups.

References

shutdown(1M)
init(1M)

If possible, bring the system down to the single-user state during off-hours, since users at terminals do not have access to the system in the single-user state.

Step 1: Log in as **root** at the console.

Step 2: Enter:

```
# shutdown
```

By default, **shutdown** prompts you about the various broadcast messages, provides a 60-second grace period between each message, and brings the system to the single-user state. When you arrive in the single-user state, you will see the following:

```
INIT: SINGLE USER MODE
```

You may now proceed with your intended tasks.

Procedure 3.4: Return to Multi-User

Multi-user mode is the standard operating mode of the M-Series RISComputers. This procedure demonstrates how to bring the system back up to multi-user mode after it has been in single-user or monitor modes for administrative purposes.

References

init(1M)

There are three system states from which you can return the system to the multi-user state. You can bring the system back from the single-user state and the monitor mode, and you can cause the system to immediately halt and reboot (see Procedure 3.6, Halt and Reboot the Operating System).

From Single-User

After administrative tasks are finished, you can bring the system back to the multi-user state from the single-user state via the **init** command.

Step 1: At the console, enter:

```
# telinit 2
```

This causes **init** to inspect **/etc/inittab**
and execute entries that will initialize the system
to the multi-user state.
The following is displayed:

```
INIT: New run level: 2

The system is coming up.  Please wait.
```

The file systems are checked and
the current system configuration is printed out.
Finally:

```
The system is ready.

MIPS Console Login:
```

Now you can log in either as **root** or as a conventional user,
since the system is in the multi-user state.

From the Monitor

After monitor programs have been run, you can bring the system back from the monitor mode by executing the automatic boot program from the hard disk.

Step 1: When you receive the monitor prompt, enter **auto**:

```
>> auto
```

Step 2: After the the sanity of the root file system is checked (via **fsstat**(1M)), a file system check is performed if necessary (via **fsck**(1M)), the system configuration is printed out, and the system is placed in the multi-user state. Observe the prompt:

```
MIPS Console Login:
```

You may log in with an appropriate system or user login.

Procedure 3.5: Running Monitor Programs

This procedure is just designed to demonstrate how to access the monitor mode and to introduce simple use of it. For details regarding the various commands and options available with the monitor mode, refer to your M-Series Technical Reference.

References

shutdown(1M)
init(1M)

Many programs can be run while in monitor mode. In addition, by using the monitor command **boot**(1SPP), it is possible to access UNIX programs.

To bring the system to monitor mode from multi-user mode, first shutdown to single-user mode as described in Procedure 3.3. To bring the system down to monitor mode from single-user mode, enter a few **sync**(1) commands and then "init 0" at the single-user prompt:

```
# sync;sync
# telinit 0
```

You will be in monitor mode when you see the following prompt:

```
>>
```

To display a list of commands and their syntax, use the **help**(SPP) command:

```
>> help

COMMANDS:
   autoboot:       auto
   boot:           boot [-f FILE] [-n] [ARGS]
   cat:            cat FILE_LIST
   disable:        disable CONSOLE_DEVICE
   dump:           dump [-(b|h|w)] [-(o|d|u|x|c|B)] RANGE
   enable:         enable CONSOLE_DEVICE
   fill:           fill [-(b|h|w)] [-v VAL] RANGE
   get:            g [-(b|h|w)] ADDRESS
   go:             go [INITIAL_PC]
   help:           help [COMMAND]
   help:           ? [COMMAND]
   initialize:     init
   load:           load CHAR_DEVICE
   put:            p [-(b|h|w)] ADDRESS VALUE
   printenv:       printenv [ENV_VAR_LIST]
   setenv:         setenv ENV_VAR STRING
   sload:          sload CHAR_DEVICE
   spin:           spin [[-v VAL] [-c CNT] [-(r|w)(b|h|w) ADDR]]*
   test:           test [mem(c)|cpu(c)|fpu(c)|all] \
                                [test num(s)|?(help)] [l|L](oop)
   unsetenv:       unsetenv ENV_VAR
   warm:           warm

COMMAND FLAGS
   commands that reference memory take widths of:
      -b -- byte, -h -- halfword, -w -- word (default)
   RANGE's are specified as one of:
   BASE_ADDRESS#COUNT
   START_ADDRESS:END_ADDRESS
Erase single characters by CTRL-H or DEL
Rubout entire line by CTRL-U
>>
```

To display the current monitor environment, use the **printenv**(1SPP) command:

```
   >> printenv
   netaddr=97.1.0.97
   lbaud=9600
   rbaud=9600
   bootfile=dkip(0,0,8)sash
   bootmode=d
   console=1
```

To use the UMIPS **mv**(1) command, use the monitor **boot** command as follows:

>> **boot dkip()unix initfile=/bin/mv initarg=***oldname* **initarg=***newname*

where *oldname* and *newname* are filenames as supplied with standard **mv** syntax.

Procedure 3.6: Halt and Reboot the Operating System

This procedure allows you to halt the system and reboot from **/unix** on the hard disk. There are a number of reasons why you might want to do this. For example, you may suspect that the running version of the kernel has been corrupted in some way (power glitch, etc.) and wish to start it over. Or, you may wish to run a different version of the kernel.

References
sysadm reboot(1)
shutdown(1M)

One way to run a different version of the kernel is to replace **/unix** with another kernel. To do this, copy **/unix** to a backup file and replace it with the other kernel, for example:

```
# cp /unix /unix.backup
# cp unix.std /unix
```

This replaces the current kernel with the one originally shipped with the system. After doing this you could follow the steps in the procedure below to cause the system to boot the new kernel. Refer to Chapter 3, System States, for more information on how to boot different kernels.

Step 1: If you are in multi-user mode, you must first get everybody off the system. To see if anyone is on, use the **who**(1) command:

```
# who
```

If users are on the system, use the wall(1M) command as described in Procedure 2.5 to inform them that the system is coming down.

To halt and reboot the system from the hard disk, enter **sysadm reboot** (or **shutdown −i6**):

```
# sysadm reboot

Running subcommand 'reboot' from menu 'machinemgmt',
MACHINE MANAGEMENT

Once started, a reboot CANNOT BE STOPPED.
Do you want to start an express reboot? [y, n, ?, q] y

Shutdown started.     Mon Apr 25 15:45:04 PDT 1988

Broadcast Message from root (tty0) on mips Mon Apr 25 15:45:05...
THE SYSTEM IS BEING SHUT DOWN NOW ! ! !
Log off now or risk your files being damaged.


INIT: New run level: 6
```

```
The system is coming down.  Please wait.
System services are now being stopped.
cron aborted: SIGTERM

The system is down.

The system is being restarted.

Autoboot: Waiting to load dkip(0,0,8)sash (CTRL-C \
      to abort) ... loading -
81120+15232+203056 entry: 0xa0300000
MIPS Standalone Shell Version 1.5 MIPS OPT Fri Nov \
      13 07:03:57 PST 1987 root

Loading dkip(0,0,0)/unix
663072+69056+373248 entry: 0x80021000
CPU: MIPS R2000 Processor Chip Revision: 3.0
Delay multiplier = 3, cnt = 742
UNIX System V Release 2_0 mips Version UMIPS_V
Total real memory  = 4194304
Available memory   = 2772992

if_enp1: controller not available
ips1: controller not available
Buffer cache: 512 pages
              256 bufs
              16384 bytes buf headers
root on dev 0x400 (fstyp is ffs)
Available memory   = 2646016

Checking root file system () automatically.

The system is coming up.  Please wait.

***** Normally all file systems are fscked.
***** To fsck only dirty ones, type 'yes' within 5 seconds:
***** All file systems will be fscked.
mountall: fscking /dev/usr (/usr).
** /dev/usr
** Last Mounted on
** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Cyl groups
4627 files, 103421 used, 84944 free (208 frags, 10592 blocks, \
      0.1% fragmentation)
```

```
***** FILE SYSTEM WAS MODIFIED *****
/dev/usr mounted on /usr
Internet daemons: routed portmap inetd rwhod.
NFS daemons: nfsd biod.
```

[any NFS mounts]

```
The system is ready.



mips Console login:
```

Procedure 3.7: Recovery from System Trouble

This procedure demonstrates how to save important information regarding the system state in the event that something has gone wrong. This information can be used for diagnostic purposes.

References
crash(1M)
savecore(1M)

The following is an example of data collection using the **crash** command. In this example, user and process information is collected and saved in a file. Speak with your service representative regarding which data it would be most useful to save for a particular situation.

```
# script crash.script
Script started, file is crash.script
```

The **script***(1) command makes a file containing the record of an interactive session. The file records all terminal activity until a CTRL-D (^D) is entered. In this example, the session will be saved in* **crash.script**.

```
# crash
dumpfile = /dev/mem, namelist = /unix, outfile = stdout
reading symboltable..................................
```

This example uses the file **/dev/mem** *and the namelist* **/unix***, and outputs to the screen (stdout). These are the defaults so no arguments are supplied to* **crash***. If you want to use a different dumpfile, see* **savecore***(1M).*

```
> ?
b (buffer)       findslot      od            srmount
base             fs            p (proc)      stack
buf (bufhdr)     help          pcb           stat
buffer           i (inode)     pdt           stream
bufhdr           inode         pfdat         strstat

c (callout)      l (lck)       proc          t (trace)
callout          lck           q (quit)      trace
dballoc          linkblk       qrun          ts
dbfree           m (mount)     queue         u (user)
dblock           major         quit          user
defproc          map           rd (od)       v (var)
dis              mbfree        redirect      var
ds               mblock        region        vtop
f (file)         mode          s (stack)     ?
file             mount         search        !cmd
findaddr         nm            size
```

The "?" gets a list of possible commands to supply to *the* **crash** ">" *prompt.*

```
> u
PER PROCESS USER AREA FOR PROCESS 25
USER ID's:     uid: 0, gid: 1, real uid: 0, real gid: 1
PROCESS TIMES: user: 1016, sys: 85, child user: 0, child sys: 0
PROCESS MISC:
    command: crash, psargs: crash
    proc slot: 25, cntrl tty:  15,0
    start: Wed May 25 17:47:15 1988
    mem: 278f4, type: exec
    proc/text lock: none
    inode of current directory: 38
OPEN FILES AND POFILE FLAGS:
    [0]: F#53, 0          [1]: F#53, 0          [2]: F#53, 0
    [3]: F#36, 0          [4]: F#37, 0
FILE I/O:
    u_base: 100d7cd8, file offset: 2490368, bytes: 0,
    segment: data, cmask: 0022, ulimit: 4194304
    sfile mode(s): read
SIGNAL DISPOSITION:
         1:  default   2:   414600   3:  default   4:  default
         5:  default   6:  default   7:  default   8:  default
         9:  default  10:  default  11:  default  12:  default
        13:  default  14:  default  15:  default  16:  default
        17:  default  18:  default  19:  default  20:  default
        21:  default  22:  default  23:  default  24:  default
        25:  default  26:  default  27:  default  28:  default
```

"u" gets you everything you ever wanted to know about user processes.

```
> p
PROC TABLE SIZE = 300
SLOT ST PID   PPID  PGRP  UID PRI CPU  EVENT    NAME    FLAGS
   0 s     0     0     0    0   0   0 800bd1c9 sched   load sys nwak
   1 s     1     0     0    0  39   0 ffffc000 init
   2 s     2     0     0    0   0   0 80037f78 vhand   load sys nwak
   3 s     3     0     0    0  20   0 8004fb5c bdflush load sys nwak
   4 s   262     1   262  573  28   0 c011780c csh
   5 s  2313   154  2313    0  26   2 800bd28c rlogind load
   6 s  2314  2313  2313  649  30   0 800bf8a8 csh
   7 s   132     1     0    0  26   0 c0116db8 syslogd
   8 s   150     1     0    0  26   0 800bd28c routed  load
   9 s   156     1   156    0  26   0 c00036cc rwhod   load
  10 s   153     1     0    0  26   0 800bd28c portmap
  11 s   154     1     0    0  26   0 800bd28c inetd
  12 s   163     1     0    0  26   0 c000344c nfsd    poll
  13 s   165   163     0    0  26   0 c000344c nfsd
  14 s   166   163     0    0  26   0 c000344c nfsd
  15 s   167   163     0    0  26   0 c000344c nfsd
  16 s   168     1     0    0  26   0 800bd2e0 biod
  17 s   170     1     0    0  26   0 800bd2e0 biod
  18 s   172     1     0    0  26   0 800bd2e0 biod
  19 s   174     1     0    0  26   0 800bd2e0 biod
  20 s   257     1   257    0  26   0 c0116f40 cron
  21 s  2339  2314  2313    0  30   0 800c0b2c csh
  22 s  2347  2339  2313    0  28   0 c011bf64 script  load
  23 r  2348  2347  2313    0  60   1 00000000 script  load
  24 s  2349  2348  2313    0  30   0 800c0ee0 csh
  25 r  2357  2349  2313    0  62  17 00000000 crash   load
```

"p" prints the process table.

> **q**

"q" is used to exit from **crash**

# **^D**

```
Script done, file is crash.script
```

Entering a CTRL D terminates the script session. The crash.script file now contains a copy of the above session.

Procedure 3.8: Reload the Operating System

Just as file systems have backup versions, so does the UMIPS operating system - on the two tapes delivered with the system. There are two ways to reload the operating system if it is necessary.

- An update install replaces (overwrites) the core system files on the hard disk with those originally distributed.
- A complete install is a full restore and erases everything on the system disk and then loads the original version of the operating system.

Consult your latest Release Notes for details on how to perform these levels of install. The procedures here serve as examples of how these steps are performed. Read through the procedures and the corresponding section of the Release Notes for reloading the operating system before you do the procedure. If you understand all of the steps, then reload the operating system. If you do not understand the reload procedure, contact your service representative for help.

Update Install

You may want to do this procedure if you have received an update to the operating system, or simply want to replace a large number of the system files.

The following is a commented printout of an update install session. The session begins at the monitor prompt (see Procedure 3.5). Note that your system output will not be identical to this example. Also note that the tape controller (tpqic) and disk controller (dkip) used here may be different on your system (refer to your Release Notes). This procedure is performed with Tape 1 in the tape drive.

```
>> boot -f tpqic (,,2)sash
```

This loads the standalone shell (sash) from tape 1

```
137024+42912+176224 entry: 0xa0300000
MIPS Standalone Shell Version 4.0 MIPS OPT Thu Jun 16 \
        08:38:48 PDT 1988 root
sash: cp -b 16K tpqic(,,3) dkip(,,1)
```

This copies the miniroot from the tape to disk. The following series of dots indicates the copy is in progress.

```
. . . . . . . . . . . . . . . . . etc.

12288000 (0xbb8000) bytes copied
sash: boot -f dkip(,,1)unix.r2300_std root=ips0d0s1
```

This boots the miniroot. The root disk specified - ips0d0s1 - later becomes system swap space, so the temporary miniroot information is overwritten by swapping once the UMIPS system is up and running.

```
569088+70416+471296 entry: 0x80021000
CPU: MIPS R2000 Processor Chip Revision: 5.0
```

```
FPU: MIPS R2010 VLSI Floating Point Chip Revision: 2.0

UNIX System V Release 3_0 mips Version UMIPS
Total real memory  = 4194304
Available memory   = 2838528

if_enp1: controller not available
ips1: controller not available
Root on dev 0x401, swap on dev 0x401

WARNING: clock lost 168 days

WARNING: CHECK AND RESET THE DATE!
root on dev 0x401 (fstyp is ffs)
Available memory   = 2711552


UMIPS Miniroot run level 1

Making miniroot device files for m800 system...
erase=^H, kill=^U, interrupt=^C
# set -a
# Install=update
# inst
```

Set the "Install" option to "update", and start the installation process by executing **inst**.

```
MIPS software package installation


========== selecting subpackages ==========

Subpackage root will be installed.
Subpackage m800 will be installed.
Subpackage sppbin will be installed.
Subpackage usr will be installed.
Install subpackage cmplrs (y n) [n]? y
Install subpackage man (y n) [n]? y

Selected subpackages:
   root m800 sppbin usr cmplrs man
Is this what you want (y n) [y]? y
```

This selects an update installation of all parts of the package. If you are in a hurry you may want to save a few minutes here by skipping the compilers and/or man pages installations until later.

```
========== setting system clock/calendar ==========

The current value of the clock is: Thu Dec 31 16:38:17 PST 1987
Is the clock correct (y n) [y]? n
Enter the correct time and date as "mmddhhmm[yy]": 0621121688
Tue Jun 21 12:16:00 PDT 1988
The current value of the clock is: Tue Jun 21 12:16:00 PDT 1988
Is the clock correct (y n) [y]? y
```

We update the clock.

```
========== verifying single-user mode ==========

The system is in a single-user run level.

========== installing sash to volume header ==========


========== mounting filesystems ==========

/dev/root mounted on /mnt
/dev/usr mounted on /mnt/usr

========== preserving local files ==========

Running preserve -s for subpackage root... 28 files preserved.
No preserve list or findmods list for m800- preserve not executed.
No preserve list or findmods list for sppbin- preserve not executed.
Running preserve -s for subpackage usr... 18 files preserved.
No preserve list or findmods list for cmplrs- preserve not executed.
No preserve list or findmods list for man- preserve not executed.

========== verifying disk space ==========

The system will now be checked to verify that there is enough
disk space with the current configuration to successfully
install the package (and any selected optional subpackages).
For large packages (especially operating system packages),
this can be time consuming...

There is enough space.

========== stripping old links ==========

Stripping links for subpackage root...
Stripping links for subpackage m800...
Stripping links for subpackage sppbin...
Stripping links for subpackage usr...
Stripping links for subpackage cmplrs...
Stripping links for subpackage man...

========== extracting files from subpackage archives ==========

Rewinding the tape...
Verifying tape id... ok
Forward spacing the tape...

Loading subpackage: root...

Forward spacing the tape...
Loading subpackage: m800...
Forward spacing the tape...
Forward spacing the tape...
```

```
Loading subpackage: sppbin...
Forward spacing the tape...
Rewinding the tape...

Please mount umips tape number 2 and press return: <CR>
```

Replace tape 1 with tape 2 and press return.

```
Rewinding the tape...
Verifying tape id... ok
Forward spacing the tape...

Loading subpackage: usr...
Forward spacing the tape...
Loading subpackage: cmplrs...
Forward spacing the tape...
Loading subpackage: man...
Forward spacing the tape...
Rewinding the tape...

========== making device special files ==========


========== running comply ==========

running first comply pass...
running second comply pass...

========== cleaning up old versions ==========

An attempt will now be made to clean up any files left over
from previous versions of the software which has just been
installed.

Searching for old versions to remove...

========== restoring preserved user files ==========

Running preserve -r for subpackage root...
No preserve list or findmods list for m800- no files restored.
No preserve list or findmods list for sppbin- no files restored.
Running preserve -r for subpackage usr...
No preserve list or findmods list for cmplrs- no files restored.
No preserve list or findmods list for man- no files restored.

========== cleaning up ==========

Copying packaging information directory to \
      /mnt/usr/pkg/lib/umips3.0...
Unmounting filesystems...
/mnt/usr: Unmounted
/mnt: Unmounted

========== installation complete ==========
# sync
# sync
# init 0
#
```

The system is brought down to monitor mode.

```
INIT: New run level: 0

Miniroot shutdown


M/800 MIPS Monitor Version 4.0 MIPS OPT Thu May 19 \
         00:37:43 PDT 1988 root
Memory size: 4194304 (0x400000) bytes
Icache size: 65536 (0x10000) bytes
Dcache size: 65536 (0x10000) bytes
>> auto
```

The updated UMIPS system is booted.

Disk Management Procedures

Study these procedures to become familiar with the **format** command BEFORE using it - misuse of **format** can cause you to lose all information on your disk. The **format** command performs three distinct functions: writing a volume header, scanning a disk for bad blocks, and formatting a disk to prepare it to receive data. When using **format**, perform ONLY those functions needed in your situation.

NOTE To abort the **format** command at any time, enter CTRL-C.

The portion of the **format** command you need to use depends on your configuration and your needs:

- If you have SCSI drives, it is not normally necessary to do anything to them other than writing a volume header - formatting and extensive bad blocking is performed at the factory. Writing a volume header on a SCSI disk is covered in Procedure 4.3.
- If you have SMD drives, they must be formatted before they can be used. This is covered in Procedure 4.1.
- If you suspect your disk has developed bad sectors, regardless of whether it is a SCSI or SMD drive, use the scan portion of the **format** command as documented in Procedure 4.2.

In addition, these Procedures discuss use of the UMIPS **dvhtool**(1M) and **prtvtoc**(1M) commands.

Note that this section assumes a certain configuration of disk and tape controller - refer to your *Release Notes* for information regarding your particular configuration.

The following procedures are covered in this section:

Procedure 4.1	**Disk Preparation** To prepare an SMD disk for use.
Procedure 4.2	**Scanning a Disk for Bad Sectors** To check a disk for bad blocks.
Procedure 4.3	**Writing a Volume Header** To prepare a SCSI disk for use.
Procedure 4.4	**Using the dvhtool Utility** To modify the volume header from UMIPS.
Procedure 4.5	**Using the prtvtoc Utility** To print the volume header information.

PROCEDURE 4.1 - Disk Formatting

NOTE

If your M–Series RISComputer uses SCSI disk drives, the formatting procedure described here is not necessary. Normally, you will just have to write the volume header as described in Procedure 4.3. If you want to check a disk for possible bad blocks, skip to Procedure 4.2, "Scanning a Disk".

The following procedure demonstrates the formatting and partitioning of the system hard disk. NOTE: THIS DESTROYS ALL DATA ON THE DISK. This procedure would be used, for example, to install an entirely new version of the operating system or to replace one that has become hopelessly corrupted. Formatting a disk is also necessary when preparing a new disk to be added on to your system (except for SCSI disks as described above). If you have any data on the disk that you want to save, do so before performing this procedure.

References

format(1SPP)
boot(1SPP)

Step 1: Go to Monitor Mode

Go to monitor mode as explained in Procedure 3. The operation will wipe out the existing UMIPS operating system so it must be performed from the monitor.

Step 2: Boot the Format Utility

In this step, you boot the format utility and follow its prompts all the way through to a formatted, bad-blocked and partitioned disk.

```
>> boot -f tpis(,,2)format
Obtaining format
147968+51760+319824 entry: 0x80020000
```

format is booted from the tape

```
MIPS Format Utility
Version 4.0  Fri Apr 22 19:10:48 PDT 1988

name of device? dkis
LUN number? 0
target id? 0
```

Note that the name of your tape and disk device (in this case "tpis" and "dkis", respectively) may differ. Refer to your Release Notes for details.

```
choose new drive parameters (y if yes)? n

The UNIX file system partitions may be either BSD or System V
do you desire BSD file system partitions (y if yes)? y
```

BSD file partitions are requested because this will be a fast file system (ffs)

```
dump device parameters (y if yes)? n
```

```
modify device parameters (y if yes)? n
dump partition table (y if yes)? y
        Root partition is entry #0
        Swap partition is entry #1
        Default boot file is /vmunix

entry type           #blks   #cyls  cg(mod) 1st_lbn 1st_cyl num_bytes
0-a   BSD file sys   44496   103    6(7)    2160    5       22781952
1-b   BSD file sys   39312   91     5(11)   600912  1391    20127744
2-c   BSD file sys   638064  1477   92(5)   2160    5       326688768
3-d   BSD file sys   554256  1283   80(3)   46656   108     283779072
4-e   BSD file sys   339984  787    49(3)   46656   108     174071808
5-f   BSD file sys   170208  394    24(10)  386640  895     87146496
6-g   BSD file sys   510192  1181   73(13)  46656   108     261218304
7-h   BSD file sys   44064   102    6(6)    556848  1289    22560768
8     volume header 2160     5      0(5)    0       0       1105920
partition 9 size == 0
10    entire volume 640224   1482   92(10)  0       0       327794688
11-i  BSD file sys   384480  890    55(10)  2160    5       196853760
12-j  BSD file sys   253584  587    36(11)  386640  895     129835008
13-k  BSD file sys   83376   193    12(1)   556848  1289    42688512
14-l  BSD file sys   192240  445    27(13)  2160    5       98426880
15-m  BSD file sys   192240  445    27(13)  194400  450     98426880
```

We requested the partition table be displayed. Note that both the "old" lettering scheme and the "new" numbering scheme are displayed.

```
modify partition table (y if yes)? n
```

The default partitioning scheme was used.

```
formatting destroys ALL disk data, perform format (y if yes)? y

formatting

........................<etc>

scanning destroys disk data, perform scan (y if yes)? y

number of scans for bad blocks (3 are suggested)? 3
```

The "scan" phase performs bad-blocking.

```
........................<etc>

starting cylinder is 0, ending cylinder is 1482
```

The formatting is done.

PROCEDURE 4.2 - Scanning a Disk

The following procedure is simply a commented typescript of the output supplied by the **format** command when performing a scan for bad sectors. In this example, a single partition of an SMD disk is scanned. Scans performed on SCSI disks are for the entire disk - a single partition cannot be scanned independently. Note that a scan reads and writes the area scanned, consequently destroying data, so you must back-up devices before scanning them if they contain information you want saved. Refer to the *Technical Reference Manual* and your *Release Notes* for additional information on the **format** command.

```
M/800 MIPS Monitor Version 4.0 MIPS OPT Thu May 19 \
            00:37:43 PDT 1988
Memory size: 4194304 (0x400000) bytes
Icache size: 65536 (0x10000) bytes
Dcache size: 65536 (0x10000) bytes
>> boot dkip(0,0,8)format
```

format *is booted from the SMD disk. It may also be booted from tape (and from a network if you are so configured).*

```
81120+15232+203056 entry: 0xa0300000
MIPS Standalone Shell Version 1.5 MIPS OPT Fri Nov \
      13 07:03:57 PST 1987

Loading dkip(0,0,8)format
125200+28288+249376 entry: 0x80020000

MIPS Format Utility
Version 1.5  Tue Dec 1 16:05:31 PST 1987

name of device? dkip
controller number? 0
unit number? 0
read in 29 defects from 'on disk' bad sector table

choose new device parameters (y if yes)? n

The UNIX file system partitions may be either BSD or System V
do you desire BSD file system partitions (y if yes)? y
```

BSD is chosen because this is the fast file system

```
dump device parameters (y if yes)? n
modify device parameters (y if yes)? n

dump partition table (y if yes)? n
modify partition table (y if yes)? n

If the drive is directly from the factory defects can be
read from it ONLY ONCE before it is formatted.
read factory defects from the drive (y if yes)? n

formatting destroys disk data, perform format (y if yes)? n
formatting wasn't done, perform scan anyway (y if yes)? y
scan entire disk (y if yes)? n
entry number of partition to scan? 7
```

Note that if your disk is a SCSI device, the whole disk must be scanned - there is no option to scan only a single partition.

```
scanning destroys disk data, perform scan (y if yes)? y

number of scans for bad blocks (3 are suggested)? 3

....................etc.
```

A series of dots appears on the screen until the scanning is complete. If any bad blocks (i.e., sectors) are found, the bad block table is automatically updated.

```
media defect list manipulation, when prompted
choose one of (list, add, delete, quit)
command? list
        lbn      (cyl, head, sec, bytes from index)
        27525    (43, 6, 57, 36480)
        52704    (83, 6, 36, 23040)
        58431    (92, 7, 30, 19200)

    74989    (119, 0, 19, 12160)
                .
                .
                .
               etc
command? quit

mapping destroys disk data, perform map (y if yes)? y

dump bad sector table (y if yes)? y

bad sector table:
        bad block 27525 slipped
        bad block 52704 slipped
        bad block 58431 slipped
        bad block 74989 slipped
        bad block 91252 slipped
                .
                .
                .
               etc

writing bad sector table at lbn 630...
```

The new bad sector table is written to logical block number 630. The final command below writes the new volume header, containing the new block sector table to disk

```
write new volume header? (y if yes)? y
```

```
writing volume header...
exit(0) called

M/800 MIPS Monitor Version 4.0 MIPS OPT Thu May 19 \
      00:37:43 PDT 1988
Memory size: 4194304 (0x400000) bytes
Icache size: 65536 (0x10000) bytes
Dcache size: 65536 (0x10000) bytes
>>
```

PROCEDURE 4.3 - Writing a Volume Header

This procedure uses the **format** utility to write a volume header on a SCSI disk. This is normally the only procedure necessary to prepare a SCSI disk for use.

>> **boot -f bfs()**<*hostname*>**:/stand/format**
```
Obtaining <hostname>:/stand/format from server <hostname>
147968+51760+319824 entry: 0x80020000
```

In this example, the **format** *utility is booted off of a network server machine (<hostname>). The* **format** *utility could also be booted off the disk or tape.*

```
MIPS Format Utility
Version 4.0  Fri Jun 3 11:01:11 PDT 1988 root

name of device? dkis
LUN number? 0
target id? 0

choose new drive parameters (y if yes)? y
device parameters are known for:
        (9) fuji 2246sa (140Meg SCSI)
        (10) cdc  94161 (160Meg SCSI)
        (11) cdc  94171 (328Meg SCSI)
        (12) fuji 2249sa (325Meg SCSI)
enter number for one of the above? 11
The Unix file system partitions may be either BSD or System V
do you desire BSD file system partitions (y if yes)? y

dump device parameters (y if yes)? y
        number cylinders = 1482
        number heads = 9
        number sectors per track = 48
        number bytes per sector = 512
        sector interleave = 1
modify device parameters (y if yes)? n

dump partition table (y if yes)? y
        Root partition is entry #0

     Swap partition is entry #1
        Default boot file is /vmunix

entry type          #blks   #cyls cg(mod) 1st_lbn 1st_cyl num_bytes
0-a   BSD file sys  44496   103   6(7)    2160    5       22781952
1-b   BSD file sys  39312   91    5(11)   600912  1391    20127744
2-c   BSD file sys  638064  1477  92(5)   2160    5       326688768
3-d   BSD file sys  554256  1283  80(3)   46656   108     283779072
4-e   BSD file sys  339984  787   49(3)   46656   108     174071808
5-f   BSD file sys  170208  394   24(10)  386640  895     87146496
6-g   BSD file sys  510192  1181  73(13)  46656   108     261218304
7-h   BSD file sys  44064   102   6(6)    556848  1289    22560768
8     volume header 2160    5     0(5)    0       0       1105920
partition 9 size == 0
```

```
10     entire volume 640224  1482  92(10)  0       0       327794688
11-i   BSD file sys  384480  890   55(10)  2160    5       196853760
12-j   BSD file sys  253584  587   36(11)  386640  895     129835008
13-k   BSD file sys  83376   193   12(1)   556848  1289    42688512
14-l   BSD file sys  192240  445   27(13)  2160    5       98426880
15-m   BSD file sys  192240  445   27(13)  194400  450     98426880

modify partition table (y if yes)? n

formatting destroys ALL SCSI disk data, perform \
      format (y if yes)? n

scanning destroys disk data, perform scan (y if yes)? n

SCSI defect list manipulation, when prompted
choose one of (list, add, delete, quit)
command? list
        no defects in list
command? quit

write new volume header? (y if yes)? y

writing volume header...
exit(0) called
*Intrepid* MIPS Monitor Version 4.0 MIPS OPT Wed Apr 13 \
      16:40:45 PDT 1988
Memory size: 16777216 (0x1000000) bytes
Icache size: 65536 (0x10000) bytes
Dcache size: 65536 (0x10000) bytes
>>
```

PROCEDURE 4.4 - Using the dvhtool Utility

This procedure examines the disk volume header and then makes a modification to it.

Step 1: Examine the Disk Volume Header

The first step of this procedure uses the **dvhtool** command to look at the disk volume header. For information regarding the volume header, refer to Chapter 4 and **dvh**(4).

```
# dvhtool

Command? (read, vd, pt, dp, write, bootfile, or quit): read
Volume? /dev/rdsk/ips0d0vh
```

*We read the disk volume header (**ips0d0vh**) in order to examine the information it contains. The other options to the **dvhtool*** `Command?:` *prompt are "vd" for the volume directory, "pt" for the partition table, "dp" for the device parameters, "write" to write new data, "bootfile" to replace the default boot file (**unix**), and "quit" to exit **dvhtool**.*

```
Command? (read, vd, pt, dp, write, bootfile, or quit): dp

current contents:
        dp_skew:        4
        dp_gap1:        12
        dp_gap2:        12
        dp_spare0:      1
        dp_cyls:        823
        dp_shd0:        0
        dp_trks0:       10
        dp_shd1:        0
        dp_trks1:       0
        dp_secs:        64
        dp_secbytes:    512
        dp_interleave:  1
        dp_flags:       37
        dp_datarate:    0
Command? (name value, or l) <CR>
```

*"dp" is entered to get a list of the device parameters. Note that this information could be matched with the different disk types defined in **/etc/disktab** as one way to determine the type of disk you have. In this example, the 823-cylinder and 64-sector disk match the 2333-64 defined in **/etc/disktab**.*

```
Command? (read, vd, pt, dp, write, bootfile, or quit): vd

current contents:
        file    len     lbn
        bsttab: 348     640
        sash:   265064  641
Command? (d FILE, a UNIX_FILE FILE, c UNIX_FILE FILE, \
      or l) <CR>
```

"vd" is entered to list the volume directory. The example shows the bad sector table (bsttab) and the sash, their size and starting logical block number location (lbn).

```
Command? (read, vd, pt, dp, write, bootfile, or quit): pt
```

"pt" displays the partition table.

```
current contents:
        part     n_blks   1st_blk  type
        0:       39680    5120     bsd42
        1:       39680    487040   bsd42
        2:       521600   5120     bsd42
        3:       442240   44800    bsd42
        4:       264960   44800    bsd42
        5:       133120   309760   bsd42
        6:       398080   44800    bsd42
        7:       44160    442880   bsd42
        8:       2560     0        volhdr
        9:       2560     2560     trkrepl
        10:      526720   0        volume
        11:      304640   5120     bsd42
        12:      216960   309760   bsd42
        13:      83840    442880   bsd42
        14:      152320   5120     bsd42
        15:      152320   157440   bsd42

Command? (part nblks 1stblk type, or l) <CR>
```

entering a carriage return to the prompt returns the higher level prompt

```
Command? (read, vd, pt, dp, write, bootfile, or quit): bootfile
Current Boot file name is /unix
Enter name up to 16 characters: <CR>

Command? (read, vd, pt, dp, write, bootfile, or quit): quit
#
```

Step 2: Modifying the Volume Header

This is an example of using the **dvhtool** command to modify the disk volume header. UNLESS YOU KNOW WHAT YOU ARE DOING AND HAVE REASON TO DO IT, DON'T MODIFY THE VOLUME HEADER. This step of the procedure gives an example of how to replace a volume header file if one has somehow been lost. In the example, the unix file **/stand/format** is written to the disk volume header.

# dvhtool

```
Command? (read, vd, pt, dp, write, bootfile, or quit): read
Volume? /dev/rdsk/ips0d0vh
```

It is necessary to read in the volume to work with. In this case the volume is ips0d0vh, the volume header

```
Command? (read, vd, pt, dp, write, bootfile, or quit): vd
```

Enter "vd" to access the volume directory.

```
current contents:
        file     len      lbn
        bsttab:  348      640
        sash:    265064   641
```

This is the volume directory. Note that there is no **format** *command here, so we'll add one with the copy in* **/stand**

```
Command? (d FILE, a UNIX_FILE FILE, c UNIX_FILE FILE, \
      or l) a /stand/format format
```

The options are "d" for delete, "a" to add a new file, and "c" to overwrite an existing file. In this example, we add the **/stand/format** *command to the volume directory as* **format**

```
Command? (d FILE, a UNIX_FILE FILE, c UNIX_FILE FILE, or l) l

current contents:
        file     len      lbn
        bsttab:  348      640
        sash:    265064   641
        format:  277504   -1
```

"l" lists the contents of the directory, and format is now there. The "lbn" is the starting logical block number and does not read correctly until the directory is written to the volume header.

```
Command? (d FILE, a UNIX_FILE FILE, c UNIX_FILE FILE, \
      or l) <CR>
```

entering a carriage return to the prompt returns the higher level prompt

```
Command? (read, vd, pt, dp, write, bootfile, or quit): write
Volume? (/dev/rdsk/ips0d0vh)
```

write the new value to the volume header

```
Command? (read, vd, pt, dp, write, bootfile, or quit): vd

current contents:
        file     len      lbn
        bsttab: 348      640
        sash:    265064   641
        format: 277504   1159
```

Now, when we list the volume directory, the format command shows up with the correct address

```
Command? (d FILE, a UNIX_FILE FILE, c UNIX_FILE FILE, or \
      l) <CR>
Command? (read, vd, pt, dp, write, bootfile, or quit): quit
#
```

PROCEDURE 4.5 - Using the prtvtoc Utility

This procedure uses the UMIPS **prtvtoc**(1M) utility to list volume header information to the screen. The example consists of the single **prtvtoc** command supplied with the system disk device. Use this command to get information about various volumes quickly.

```
root% prtvtoc /dev/rdsk/ips0d0s0
* /dev/rdsk/ips0d0s0 (bootfile "/unix") partition map
*
* Dimensions:
*     512 bytes/sector
*      64 sectors/track
*      10 tracks/cylinder
*     823 cylinders
*     815 accessible cylinders
* Unallocated space:
*       Start         Size
*      309760      -304640
*      526720      -521600
*       44800       -39680
*      157440      -112640
*      442880      -398080
*      309760      -264960
*      487040      -329600
*      442880      -133120
*      526720       -83840
*      487040       -44160
*      526720       -39680
*
* Partition Tag Flags First Sector Sector Count  Mount Directory
       0     4    0        5120        39680      /
       1     4    0      487040        39680
       2     4    0        5120       521600
       3     4    0       44800       442240
       4     4    0       44800       264960
       5     4    0      309760       133120
       6     4    0       44800       398080
       7     4    0      442880        44160
       8     0    0           0         2560
       9     1    0        2560         2560
      10     6    0           0       526720
      11     4    0        5120       304640
      12     4    0      309760       216960
      13     4    0      442880        83840
      14     4    0        5120       152320
      15     4    0      157440       152320
root%
```

File System Administration Procedures

The following procedures are covered in this section:

Procedure 5.1	**Adding Extra Swap Space on the System Disk** Using a free partition on the disk for added swap space.
Procedure 5.2	**Create File Systems on an Extra Disk** To define additional file systems when more than one disk device is available.
Procedure 5.3	**Maintaining File Systems** To check and possibly repair file systems. To monitor disk space usage. To reorganize disk space.
Procedure 5.4	**File System Backup and Restore** To provide a storage copy of active files. To archive unneeded files. To bring files and file systems back from storage.

NOTE Some variation in these procedures may occur depending on the configuration of your system.

Procedure 5.1: Adding Extra Swap Space on the System Disk

This procedure illustrates the creation and mounting of a new file system on an already existing disk partition. The disk partition used is partition "7", a free partition on the UMIPS system disk as it is shipped. The partition is added as additional swap space.

Step 1: Partition "7" on the system disk, when it is shipped, is an already formatted but unused partition. To create a swap file system on the partition, use the **swap**(1M) command:

```
# swap -a /dev/dsk/ips0d0s7 44160
```

The syntax here is:

/etc/swap -a *special_file swaplo swaplen*

Where *special_file* is the UMIPS file name for the partition, *swaplo* is the starting 512-byte block (sector) of the swap space and *swaplen* is the size of the partition in 512-byte sectors.

Step 2: Now, check to see that the new swap file system has been recognized by the system. Use the **swap -l** command:

```
# swap -l
```

You should see something like this:

```
path               dev   swaplo blocks   free
/dev/dsk/ips0d0s1  4,1        0  39680  39680
/dev/dsk/ips0d0s7  4,23       0  44160  44160
```

Step 3: The new swap space is now recognized by the system, but has to be explicitly added each time the system is brought up. To cause the swap space to be automatically added each time the system is brought up multiuser, add this entry to **/etc/fstab**:

```
/dev/dsk/ips0d0s7  none  swap  rw,noauto  0 0
```

Procedure 5.2: Create File Systems on an Extra Disk

This procedure illustrates the creation and mounting of a new file system on an additional disk added to your system. The disk must first be formatted using the **format**(SPP) command. The formatting scheme used in this example is the default formatting scheme. The disk partition used is partition "2", the partition which consists of all available space on the disk. For information regarding how to format a disk to create new partitions, refer to the discussion on disk partitions in Chapter 4 as well as Procedure 4.2, Custom Partitioning.

Step 1: Partition "2", created with the default formatting scheme, is created as a new file system by the **newfs.ffs** command:

```
# newfs.ffs -s 521600 /dev/dsk/ips5d0s2 2333-64
```

The syntax here is:

/etc/newfs.ffs -s *sectors special_file disk_drive*

Where *sectors* is the size of the partition in 512 byte sectors, *special_file* is the UMIPS file name for the partition, and *disk_drive* specifies the hard disk device that the file system is being created on.

The system output will look something like this:

```
# newfs.ffs -v -N -s 521600 /dev/dsk/ips5d0s2 2333-64
/etc/mkfs.ffs -N /dev/dsk/ips5d0s2 521600 64 10 8192 1024 16 \
                  10 60 2048 t
/dev/dsk/ips5d0s2:  521600 sectors in 815 cylinders of 10 \
                  tracks, 64 sectors
        267.1Mb in 51 cyl groups (16 c/g, 5.24Mb/g, 2048 i/g)
super-block backups (for fsck -b#) at:
 32, 10336, 20640, 30944, 41248, 51552, 61856, 72160, 82464, 92768,
 103072, 113376, 123680, 133984, 144288, 154592, 163872, 174176,
 184480, 194784, 205088, 215392, 225696, 236000, 246304, 256608,
 266912, 277216, 287520, 297824, 308128, 318432, 327712, 338016,
 348320, 358624, 368928, 379232, 389536, 399840, 410144, 420448,
 430752, 441056, 451360, 461664, 471968, 482272, 491552, 501856,
 512160,
#
```

Step 2: Now that the file system has been created, it should be checked to make sure that all is well before mounting it and entrusting data to it. Run the **fsck.ffs** program on it:

```
# fsck.ffs /dev/dsk/ips5d0s2
```

The output looks something like:

```
# fsck.ffs /dev/dsk/ips5d0s2
** /dev/dsk/ips5d0s2
** Last Mounted on
** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Cyl groups
2 files, 9 used, xxxxx free (xx frags, xxxxx blocks, \
      0.1% fragmentation)

***** FILE SYSTEM WAS MODIFIED *****
#
```

If **fsck** reports problems, remake the file system with **newfs.ffs** as above.

Step 3: Now, it's time to mount the new file system. Make a directory on which to mount the file system and then use the **mount** command:

```
# mkdir /newfilesys
# mount /dev/dsk/ips5d0s2 /newfilesys
```

Step 4: Finally, test and use the new file system:

```
# ls /newfilesys
lost+found
# mkdir /newfilesys/newuser
# cp /etc/stdcshrc /newfilesys/newuser/.cshrc
# cp /etc/stdprofile /newfilesys/newuser/.profile
# ls -a /newfilesys/*
/newfilesys/lost+found:
./    ../

/newfilesys/newuser:
./          ../         .cshrc*     .profile*
#
```

As a separate file system, **newfilesys** can be unmounted at any time:

```
# umount /newfilesys
# ls /newfilesys
./   ../
```

To cause the new file system to be automatically mounted at boot time, add the following entry in **/etc/fstab**:

```
/dev/dsk/ips5d0s2 /newfilesys ffs rw 0 0
```

Procedure 5.3: Maintaining File Systems

This procedure demonstrates a few techniques used to ensure that files are not becoming too large, and are not just sitting on the system for a long time without being used.

References

fsck(1M)
sysadm fileage(1)
sysadm filesize(1)
du(1M)
df(1M)

File System Checking

In this procedure, a simple file system check is performed. The example shows file system check output in which no problems are encountered - the usual situation. For details regarding how to respond to various problems that may be encountered during file system checks, refer to Chapter 5 and Appendix B.

File system checking is performed with the **fsck**(1) command. The check should be performed on an unmounted file system as shown in the procedure below. Note that the file system being checked is a user file system that has been created on the system disk partition 7.

Step 1: IN SINGLE USER MODE, get a list of the currently mounted file systems using the **mount**(1M) command:

```
# mount
/dev/root on / type ffs (rw)
/dev/usr on /usr type ffs (rw)
/dev/dsk/ips0d0s7 on /user2 type ffs (rw)
#
```

The output from **mount** shows three mounted file systems: the root file system, the usr file system and a user file system created on the extra system disk partition.

Step 2: Now unmount the file system to be checked using the **umount**(1M) command (note that's Umount, not UNmount):

```
# umount /dev/dsk/ips0d0s7
#
```

A quick check with **mount** will show the file system to be unmounted:

```
# mount
/dev/root on / type ffs (rw)
/dev/usr on /usr type ffs (rw)
#
```

Step 3: Now perform the file system check with **fsck**:

```
# fsck.ffs /dev/dsk/ips0d0s7
** /dev/dsk/ips0d0s7
** Last Mounted on
** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Cyl groups
44 files, 1303 used, 19400 free (16 frags, 2423 blocks, \
     0.1% fragmentation)

***** FILE SYSTEM WAS MODIFIED *****
#
```

Step 4: Finally, re-mount the file system and, if desired, go to multiuser mode:

```
# mount /dev/dsk/ips0d0s7 /user2
# init 2
#
```

Note that this file system check could be performed automatically each time the system comes up by including it in an entry in **/etc/fstab** as described in Chapter 5.

Monitoring Disk Usage

The second part of the file system maintenance procedure involves various ways of making sure that enough space is available on hard disk to accommodate the users' needs. In the example for this procedure the **/user2** file system is monitored, but the commands could be used on any directory on the system.

Step 1: Enter the **du**(1) command to see the disk usage of the specified directory and subdirectories:

```
$ du /user2
117     /user2/johnr
2471    /user2/tmp
2588    /user2/fred
5194    /user2
$
```

NOTE: The number in the left column is in 512-byte blocks (sectors).

Step 2: Enter the **df**(1M) command to see the free disk space associated with the specified file system:

```
$ df /user2
Filesystem          Type kbytes    use  avail %use  Mounted on
/dev/dsk/ips0d0s7   ffs    20703  4667  16036  23%  /user2
$
```

NOTE: **df** reports free space in Kbytes.

Step 3: If you judge that a more detailed look at files is indicated, there are two **sysadm** commands you can use:

sysadm fileage

and

sysadm filesize

The **fileage** command causes you to be prompted for two pieces of information:

1. the full pathname of the directory to search

   It is important to be specific in your response. If you select a high-level directory, such as **/usr**, you may get a great deal more information than you want.

2. the number of days to go back. The default is 90 days.

   For example:

```
# sysadm fileage

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

Enter full path name of the directory to search
(default /usr/admin):  /user/johnr
Enter the number of days to go back (default 90):  12

FILES NOT MODIFIED IN THE LAST 12 DAYS IN /user/johnr

           file size    date of
owner    (characters)  last access   filename
-----    ------------  ------------  --------
johnr              37  Apr 12 19:19  .logout
johnr             598  Apr 12 23:15  .profile
johnr            1998  Apr 13 17:22  init.notes
johnr           20058  Apr 14 17:04  main.c
johnr             381  Apr 13 21:43  ps-ef.init1
johnr            1509  Apr 13 17:22  ps-ef.init2
johnr             254  Apr 13 18:53  rcfiles
johnr           19456  Apr 14 00:43  sysadm.menus
```

Step 5: filesize displays information on the *n* largest files (default is 10) in a directory named by you.

For example:

```
# sysadm filesize

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

Enter full path name of the directory to search
(default /usr/admin) [?, q]:  /user2
Enter the number of large files to be included in list
(default 10) [q]:<CR>
```

```
The 10 largest files in /user2:

        file size    date of
owner (characters)  last access   filename
----- ------------  ------------  --------
root        110592  Apr 27 21:00  tmp/ftp
root        110592  Apr 27 21:00  fred/ftp
root         98304  Apr 27 21:00  tmp/ci
root         98304  Apr 27 21:00  fred/ci
root         94208  Apr 27 21:00  tmp/rcs
root         94208  Apr 27 21:00  tmp/co
root         94208  Apr 27 21:00  fred/rcs
root         94208  Apr 27 21:00  fred/co
root         81920  Apr 27 21:00  tmp/uptime
root         81920  Apr 27 21:00  fred/uptime
#
```

Step 4: Use the **find**(1) command to perform similar actions. For example, you might include the following command line in a crontab file (see **crontab** (1)):

```
find /usr -type f -mtime +60 -print | mail root
```

Every time this command is run, root will receive mail which indicates files not modified in 60 days or more.

Procedure 5.4: File System Backup and Restore

The following section demonstrates how to make complete as well as partial backup and restores, and how to deal with backups spanning multiple tapes.

Complete Backup

Take the system to the single-user mode; (see Procedure 3.3, Shutdown to Single-User). To perform a file system backup, use the **dump.ffs**(1) command. To perform a dump of an entire file system use the **0** and **u** keys as shown in the example below. The "0" causes the entire file system to be dumped, and "u" updates the **/etc/dumpdates** file.

```
# dump.ffs 0u /dev/dsk/ips0d0s7
  DUMP: Date of this level 0 dump: Wed Apr 27 15:10:06 1988
  DUMP: Date of last level 0 dump: Wed Apr 27 15:07:02 1988
  DUMP: Dumping /dev/dsk/ips0d0s7 to /dev/mt/ctape0
  DUMP: mapping (Pass I) [regular files]
  DUMP: mapping (Pass II) [directories]
  DUMP: estimated 2674 tape blocks on 0.07 tape(s).
  DUMP: dumping (Pass III) [directories]
  DUMP: dumping (Pass IV) [regular files]
  DUMP: DUMP: 2676 tape blocks on 1 tape(s)
  DUMP: DUMP IS DONE
  DUMP: level 0 dump on Wed Apr 27 15:10:06 1988
  DUMP: Tape rewinding
#
```

This example shows the output when the entire **/dev/dsk/ips0d0s7** has been dumped to tape. In addition, the file **/etc/dumpdates** has been updated with the date and level of this dump:

```
# cat /etc/dumpdates
/dev/dsk/ips0d0s7             0 Wed Apr 27 15:10:06 1988
#
```

Incremental Backup

Incremental backups are performed in a similar manner:

```
# dump.ffs 1u /dev/dsk/ips0d0s7
  DUMP: Date of this level 1 dump: Thu Apr 28 14:40:42 1988
  DUMP: Date of last level 0 dump: Wed Apr 27 15:10:06 1988
  DUMP: Dumping /dev/dsk/ips0d0s7 to /dev/mt/ctape0
  DUMP: mapping (Pass I) [regular files]
  DUMP: mapping (Pass II) [directories]
  DUMP: estimated 2667 tape blocks on 0.07 tape(s).
  DUMP: dumping (Pass III) [directories]
  DUMP: dumping (Pass IV) [regular files]
  DUMP: DUMP: 2667 tape blocks on 1 tape(s)
  DUMP: DUMP IS DONE
  DUMP: level 1 dump on Thu Apr 28 14:40:42 1988
  DUMP: Tape rewinding
#
```

In this case, the same file system is dumped a day later. The "1" causes this backup to include only the files that have changed since a lower-level dump (in this case "0") has been recorded in **/etc/dumpdates**.

Restore

Login as **root**. Take the system to the single-user mode (run-level S or 1). Use the **restore.ffs**(1) command to restore files from tapes made with the **dump** command.

This example shows a complete file system restore being made from a dump tape to the "/user2" directory.

```
# ls /user2
# cd /user2
# restore.ffs r
# ls
fred/               lost+found/        tmp/
johnr/              restoresymtable
#
```

The files on the dump tape have been restored to the directory that the **restore** was run from (in this case, **/user2**).

Return the system to the normal operating configuration (see Procedure 3.4, Return to Multi-User). Store the backup tapes in a safe place.

Interactive Restore

A commented printout follows in which a file system is dumped and then restored using the -i, or interactive option.

```
# dump.ffs 0u /dev/dsk/ips0d0s7
  DUMP: Date of this level 0 dump: Thu Dec 31 17:11:40 1987
  DUMP: Date of last level 0 dump: Tue May  3 17:28:24 1988
  DUMP: Dumping /dev/dsk/ips0d0s7 to /dev/mt/ctape0
  DUMP: mapping (Pass I) [regular files]
  DUMP: mapping (Pass II) [directories]
  DUMP: estimated 1130 tape blocks on 0.03 tape(s).
  DUMP: dumping (Pass III) [directories]
  DUMP: dumping (Pass IV) [regular files]
  DUMP: DUMP: 1132 tape blocks on 1 tape(s)
  DUMP: DUMP IS DONE
  DUMP: level 0 dump on Thu Dec 31 17:11:40 1987
  DUMP: Tape rewinding
```

For starters, we dump a filesystem and then mount it.

```
# mount /dev/dsk/ips0d0s7 /user2
# rm -r /user2
rmdir: /user2: Directory is a mount point or in use
```

```
# ls /user2
# cd /user2
```

For the purposes of this demonstration, everything is removed from the file system (except the mount directory, user2, itself).

```
# restore -i
```

This starts the restore in interactive mode. The "restore>" prompt indicates restore is ready to accept commands, and the first command we give it is "?" for a list of possible commands.

```
restore > ?
Available commands are:
      ls [arg] - list directory
      cd arg - change directory
      pwd - print current directory
      add [arg] - add 'arg' to list of files to be extracted
      delete [arg] - delete 'arg' from list of files to be extracted
      extract - extract requested files
      setmodes - set modes of requested directories
      quit - immediately exit program
      verbose - toggle verbose flag (useful with ''ls'')
      help or '?' - print this list
If no 'arg' is supplied, the current directory is used
restore > ls

 Devconfig.entry     dce                 inittab
 Image/              disk.script         log
 Permissions.entry   disktab             lost+found/
 Poll.entry          dump.script         lpbug
 Scripts/            dvhtool.all         lsuser.temp
 Sysfiles.entry      dvhtool.example     lt.c
 Systems.entry       dvhtool.script      manipulate
 backup.bug          fake_master.d/      newbugs
 backup.script       files_to_save       pass
 bin/                firsttime.install   passwd
 bugmail             fred/               ps-ef.init1
 comments            fs                  ps-ef.init2
 comments1           fsck.script         rbackup
 cpio.bug            fstab               rbackup.script
 cpio.script         hosts               rcfiles
 crash.script        incdump.script      restore.script
```

An "ls" shows the various files and directories on the dump tape. We'll restore two directories and two files by adding them to a list of things to be restored and then extracting them.

```
restore > add bin
restore > add fred
restore > add rcfiles tty.script
restore > extract
You have not read any tapes yet.
Unless you know which volume your file(s) are on you should
start with the last volume and work towards the first.
```

```
Specify next volume #: 1
set owner/mode for '.'? [yn] y
restore > quit
# ls
bin/          fred/         rcfiles       tty.script
# pwd
/user2
#
```

tar and cpio

The **tar**(1) and **cpio**(1) commands are alternate ways of performing backups of files and directories.

The tar Command

tar is a useful command for dumping directories and files. It does not support dump levels like the **dump** and **restore** commands, so its use is more limited to making specific backup copies. In the following example, a directory and its subdirectories are tarred to the tape device, and then a specific file is extracted from that tar tape:

```
# tar -cdv /user2
/user2/johnr/bugmail 1 blocks
/user2/johnr/comments 1 blocks
/user2/johnr/comments1 1 blocks
/user2/johnr/disktab 9 blocks
/user2/johnr/files_to_save 1 blocks
       .
       .
       .
```

A backed-up file is then removed and restored to demonstrate the **tar** utility:

```
# rm /user2/johnr/comments
# ls /user2/johnr/comments
/user2/johnr/comments: No such file or directory
# tar -xv /user2/johnr/comments
x /user2/johnr/comments, 316 bytes, 1 tape blocks
# ls /user2/johnr/comments
/user2/johnr/comments
#
```

The cpio Command

cpio is used in conjunction with other UMIPS commands to create backups as well as perform many other functions. The simple example below is enough to hint at **cpio**'s involved syntactical requirements. The example backs up a directory and any subdirectories, and then restores any files requiring restoration (files in which the tape version is newer than the disk version or the disk version is missing altogether.)

```
# find /user2/fred/* -print | cpio -ovdc > /dev/rmt/ctape0
/user2/fred/-ef.init2
/user2/fred/bugmail
/user2/fred/ci
```

```
/user2/fred/co
/user2/fred/comments
       .
       .
       .
/user2/fred/uptime
2550 blocks
```

cpio is now used to get a list of the files on the tape:

```
# cpio -itvd < /dev/rmt/ctape0

100644 root        0  Apr 28 14:31:57 1988  /user2/fred/-ef.init2
100666 root      276  Apr 27 14:00:27 1988  /user2/fred/bugmail
100555 root    98304  Apr 27 14:00:36 1988  /user2/fred/ci
100555 root    94208  Apr 27 14:00:36 1988  /user2/fred/co
100644 root      316  Apr 27 14:00:27 1988  /user2/fred/comments
       .
       .
       .
2550 blocks
```

Finally, **cpio** is used to replace those disk files that are older or missing in relation to the tape backup:

```
# cpio -ivdc < /dev/rmt/ctape0
current </user2/fred/-ef.init2> newer or same age
/user2/fred/-ef.init2
current </user2/fred/bugmail> newer or same age
/user2/fred/bugmail
current </user2/fred/ci> newer or same age
/user2/fred/ci
current </user2/fred/co> newer or same age
       .
       .
       .
2550 blocks
#
```

System Reconfiguration Procedures

The following procedures are covered in this section:

Procedure 6.1	**Reconfigure the Kernel** To reconfigure the operating system after making a change to a kernel parameter value.

Procedure 6.1 - Reconfigure the Operating System Kernel

The following procedure is an example of a simple kernel reconfiguration. There would be no need to perform this particular reconfiguration unless you were having problems with file table overflows. In general, don't modify kernel parameters unless you know what you are doing. Chapter 6 goes into detail about how to monitor system performance and provides explanations about the function of the various kernel parameters.

Step 1: Modify the appropriate kernel parameter value

NOTE: In the following discussion, the CPU designation "rXXXX" must be replaced by the correct CPU number for your system, for example, r2300.

Change directory to the **master.d** directory and copy the standard kernel and sysgen files to local versions:

```
# cd /usr/reconfig/master.d
# cp kernel.rXXXX_std kernel.local
# cp sysgen.rXXXX_std sysgen.local
```

Now edit the appropriate `#define` statement in **kernel.local**. This example assumes there have been file table overflows occurring which indicates the file table size must be enlarged. Therefore, we increase the NFILE parameter:

```
#define NFILE 1200      /* was 1000  */
```

Step 2: Build the new kernel

(You need to be running **/bin/sh** when you execute the following commands, and the SHELL environment variable needs to be **/bin/sh**.)

```
# cd /usr/reconfig
# BUILD_TYPE=reconfig make unix.local
```

This will take a few minutes.

Note: If you get the message:

```
Make:  Don't know how to make unix.local.  Stop.
```

Try using this command line instead:

```
# BUILD_TYPE=reconfig make UNICES=unix.local
```

Step 3: Link the new kernel to **unix**

You now have a new UMIPS kernel in the current directory called **unix.local**. Move it to the root and link it to **/unix**:

```
# mv unix.local /
# ln /unix.local /unix
```

Note that the original unix kernel (**unix.rXXXX_std**) still exists, it just is no longer linked to **/unix**. If you have any problems with your new kernel, you can always go back to the original kernel.

Step 4: Boot the new kernel

```
# sync
#powerdown
.
.
.
>> auto
```

The system will now boot-up on the new kernel. Again, if there are any problems, remember that you can still boot the old kernel:

```
>> boot dkis()unix.rXXXX_std
```

where *dkis* is the controller designator and *rXXXX* is the CPU name for your system.

LP Spooler Administration Procedures

This section contains the following procedures:

Procedure 7-1	Installing the Printer Spooler To initially set-up a printer system.
Procedure 7-2	Manipulating the Printer Spooler To make changes to print requests.
Procedure 7-3	Printer Defaults To establish a default printer.

Procedure 7.1: Installing the Printer Spooler

In this procedure, the lp spooler is set-up to print to a serial printer. Substitute the name of the tty port to which your printer is attached for the one used in the example (**/dev/ttyq1**). (If you are attaching a parallel printer, use port **/dev/lp0** and the printer model **pprx**.)

Step 1: Check Files and Permissions

Check the owner and permission bits for the **pstatus** and **qstatus** files in the **/usr/spool/lp** directory. You should see something like this:

```
-rw-r--r--   1 lp      bin           0 May 11 09:00 pstatus
-rw-r--r--   1 lp      bin           0 May 11 09:00 qstatus
```

If **pstatus** and **qstatus** are not there, be sure to create them.

Also, check the permission bits and owners of **/usr/spool/lp/model**.
It should look something like this:

```
total 37
drwxrwxr-x   2 lp      bin         512 May 11 09:00 ./
drwxrwxr-x   7 lp      bin         512 May 11 09:00 ../
-rwxrwxr-x   1 bin     bin        1687 May 26 06:21 1640*
-rwxrwxr-x   1 bin     bin        3347 May 26 06:21 5310*
-rwxrwxr-x   1 bin     bin        1103 May 26 06:22 dqp10*
-rwxrwxr-x   1 bin     bin        1013 May 26 06:22 dumb*
-rwxrwxr-x   1 bin     bin        2046 May 26 06:22 f450*
-rwxrwxr-x   1 bin     bin        1897 May 26 06:22 hp*
-rwxrwxr-x   1 bin     bin        1083 May 26 06:22 lqp40*
-rwxrwxr-x   1 bin     bin         318 May 26 06:22 ph.daps*
-rwxrwxr-x   1 bin     bin        1411 May 26 06:22 pprx*
-rwxrwxr-x   1 bin     bin        1749 May 26 06:22 prx*
```

Also, make "lp" the owner of the special file where your printer will be, for example:

```
# chown lp /dev/ttyq1
```

Step 2: Shut Down the LP Scheduler

As root, enter:

```
# /usr/lib/lpshut
```

to shut down the lp scheduler (it may already be shut down which is fine). To verify that it has been shut down, enter:

```
# lpstat -r
```

You should get this response:

```
/usr/lib/lpshut: scheduler not running
```

Step 3: Run lpadmin

Now its time to inform the system of the new printer. The name of the printer installed in the example is "Proc7". Choose any name you like. The printer model used is "dumb" - replace this with the model in **/usr/spool/lp/model** that corresponds to your printer or refer to the Chapter 7 section, *Writing Interface Programs* to create a new one. The **lpadmin** command for this set-up looks like this:

```
# /usr/lib/lpadmin -pProc7 -mdumb -v/dev/ttyq1a
```

Step 4: Restart the Scheduler

Now, get the scheduler going by starting **lpsched**:

```
# /usr/lib/lpsched
```

Step 5: Enable the Printer

Enable the new printer with the **enable** command:

```
# enable Proc7
```

The system responds with:

```
enable: printer "Proc7" enabled
```

Step 6: Accept Printer Requests

To get the printer spooler to accept requests for the new printer, enter:

```
# accept Proc7
```

The system responds with:

```
accept: destination "Proc7" accepting requests
```

Step 7: Print a File

Print a file with **lp**, specifying the new printer:

```
# lp -dProc7 testfile
```

The output should go to the printer.

Procedure 7.2: Manipulating the Printer Spooler

The following procedure demonstrates various techniques used to selectively disable and enable printers, and cancel and redirect print jobs.

Step 1: Determine Printer Status

To see a summary of the spooler status, enter:

```
    $ /usr/lib/lpstat -s
    no system default destination
    device for Test: /dev/null
    device for New: /dev/null
    device for me: /dev/ttyq0
    device for Proc7: /dev/ttyq1
```

To see a summary of the printer status, enter:

```
$ /usr/lib/lpstat -p
    printer New is idle.  enabled since May 26 12:42
    printer me disabled since May 26 16:11 - reason unknown
    printer Proc7 is idle.  enabled since May 26 14:29
```

To see a summary of printer acceptance status, enter:

```
    $ /usr/lib/lpstat -a
    New accepting requests since May 26 12:43
    me accepting requests since May 26 13:31
    Proc7 accepting requests since May 26 14:26
```

To see the status of all output requests pending, enter:

```
    $ /usr/lib/lpstat -o
    New20 root     785   May 26 16:11
    New22 johnr     785   May 26 17:05
```

Step 2: Disabling Printers and Rejecting Jobs

To cancel a single job, get its id and then use the **cancel** command:

```
    $ lpstat
    me-22 johnr     785   May 26 17:05
    $ cancel me-22
    request "me-22" cancelled
    $ lpstat
    $
```

Disable a printer to stop all of its jobs from printing immediately. This example stops the printer Proc7 from printing any more requests, and will cause any currently printing job to start over once the printer is again enabled:

```
    # disable Proc7
```

At this time, the printer Proc7 will still accept jobs. To keep the printer from accepting any more jobs, enter:

```
# reject -r"temporary situation, sorry" Proc7
```

Now, if you submit any jobs to Proc7, you get:

```
# lp -dProc7 temp.c
lp: can't accept requests for destination "Proc7" -
            temporary situation, sorry
```

Step 3: Enabling Printers and Accepting Jobs

To restore the previous situation, use **enable** to enable the Proc7 printer once again:

```
# enable Proc7
```

Proc7 will now continue printing any jobs in its queue.

To allow Proc7 to once again accept jobs, use **accept**:

```
# /usr/lib/accept Proc7
```

and jobs will again be sent to printer Proc7.

Step 4: Moving Print Jobs

To move the queued jobs from one printer to another, shut down the lp scheduler and then use the **lpmove** command:

```
# /usr/lib/lpshut
# /usr/lib/lpmove New Proc7
```

This example moves the jobs queued at printer New to the queue at printer Proc7. In addition, print requests to New will be rejected until that printer is explicitly enabled.

To move specific jobs from one printer to another, use **lpmove** with the job id(s) and the name of the destination printer:

```
# /usr/lib/lpmove New20 New21 Proc7a
```

This moves the two jobs New20 and New21 to the printer Proc7. Note that this also causes the printer New to reject jobs until enabled.

Procedure 7.3 - Printer Defaults

This procedure demonstrates how to set up a particular printer as the system default.

Specifying a Default Printer

To establish one of the printers on the system (or the only printer) as the default printer, use the **lpadmin** command with the **-d** option:

```
# lpadmin -dProc7
```

This causes the default printer to become Proc7, which now receives the jobs that do not specify a particular printer, such as:

```
$ lp test.c
```

NOTE If your lp spooler system does not start automatically when you come up multiuser, refer first to your Release Notes for any lp spooler setup instructions before contacting your Field Representative.

TTY Management Procedures

Procedure 8.1	**Check TTY Line Settings** To tell what line settings are defined.
Procedure 8.2	**Make TTY Line Settings** To create new TTY line settings and hunt sequences.
Procedure 8.3	**Modify TTY Line Characteristics** To change the characteristics of TTY lines. To turn lines on or off.
Procedure 8.4	**Observe UMIPS File Changes** To demonstrate the file changes made by the preceding exercise.

Procedure 8.1: Check TTY Line Settings

This procedure uses the **sysadm lineset** command to determine the currently defined terminal line settings.

References
sysadm lineset(1)

Step 1: Enter this command to go directly to the lineset display:

```
$ sysadm lineset
Password:
```

This display appears on your terminal:

```
INSERT

Running subcommand 'lineset' from menu 'ttymgmt',
TTY MANAGEMENT


Tty Line Settings and Sequences

co_1200    co_300    co_9600   co_4800   co_2400
console   (does not sequence)
du_1200   du_300    du_9600   du_4800   du_2400
dx_1200   (does not sequence)
dx_19200  (does not sequence)
dx_2400   (does not sequence)
dx_4800   (does not sequence)
dx_9600   (does not sequence)
```

Each of the line settings is just a name, used to identify a set of tty line characteristics. During the **login** process, the line settings on one line "hunt" from left to right, moving from one to the next on receiving a **BREAK** signal. The rightmost setting on each line hunts to the first one again, forming a circular hunt sequence.

Note that some settings do not sequence. Sending a **BREAK** will not make it change in any way.

Step 2: To look at a line setting in detail:

```
Select one line setting to see it in detail [?, q]: console

Line Setting:           console
        Initial Flags:  B9600 CLOCAL
        Final Flags:    B9600 CLOCAL SANE TAB3
        Login Prompt:   HOSTNAME Console login:
        Next Setting:   console

B9600   9600 Baud
CLOCAL  Local line
SANE    Set All Modes To "Traditionally Reasonable" Values
TAB3    Expand Horizontal-tab To Spaces

Select another line setting or
<RETURN> to see the original list [?, q]: du_9600

Line Setting:           du_9600
        Initial Flags:  B9600
        Final Flags:    B9600 SANE TAB3 HUPCL
        Login Prompt:   HOSTNAME login:
        Next Setting:   du_4800

B9600   9600 Baud
HUPCL   Hang Up on Last Close
SANE    Set All Modes To "Traditionally Reasonable" Values
TAB3    Expand Horizontal-tab To Spaces

Select another line setting or
<RETURN> to see the original list [?, q]: q

#
```

Step 3: Notice that we didn't have to start with the leftmost entry of a row. Any entry can be specified.

Procedure 8.2: Make TTY Line Settings

In this procedure we create a test setting that will go from 9600 to 19200 baud when the BREAK key is pressed.

References

sysadm mklineset

Step 1: Enter this command to go directly to the **mklineset** display:

```
$ sysadm mklineset
Password:
```

Step 2: This sequence of prompts appears on your terminal:

```
Running subcommand 'mklineset' from menu 'ttymgmt',
TTY MANAGEMENT

Enter the name of the new tty line setting [?, q]: x9600
Select a baud rate [?, q]: 9600
Enter the login prompt you want (default = "login: ") [?, q]: test:
Do you want to add another tty line setting to the \
            sequence? [y, n, q] y

Enter the name of the new tty line setting [?, q]: x19200
Select a baud rate [?, q]: 19200
Enter the login prompt you want (default = "test:") [?, q]: test2:
Do you want to add another tty line setting to the \
            sequence? [y, n, q] n

Here is the tty line setting sequence you created:

x9600    x19200
Line Setting:           x9600
        Initial Flags:  B9600 HUPCL
        Final Flags:    B9600 SANE IXANY HUPCL TAB3
        Login Prompt:   test:
        Next Setting:   x19200
Line Setting:           x19200
        Initial Flags:  B19200 HUPCL
        Final Flags:    B19200 SANE IXANY HUPCL TAB3
        Login Prompt:   test2:
        Next Setting:   x9600
B19200  19200 Baud
B9600   9600 Baud
HUPCL   Hang Up on Last Close
IXANY   Enable Any Character to Restart Output
SANE    Set All Modes To "Traditionally Reasonable" Values
TAB3    Expand Horizontal-tab To Spaces
Do you want to install this sequence? [y, n, q] y
Installed.
```

Procedure 8.3: Modify TTY Line Characteristics

This procedure modifies an already existing tty line setting. The objective here is to tell the computer which port to use with the line settings defined above in Procedure 8.2.

References

sysadm modtty

Step 1: Enter this command to go directly to the **modtty** display:

```
$ sysadm modtty
Password:
```

Step 2: This sequence of prompts appears on your terminal:

```
Running subcommand 'modtty' from menu 'ttymgmt',
TTY MANAGEMENT


Changeable tty lines:
  tty1   ttyh0   ttyh11   ttyh14   ttyh2   ttyh4   ttyh6   ttyh8
  tty2   ttyh1   ttyh12   ttyh15   ttyh3   ttyh5   ttyh7   ttyh9
  tty3   ttyh10  ttyh13
Select the tty you wish to modify,
or enter ALL to see a report of all ttys [?, q]: ttyh13

ttyh13:  current characteristics:
        State           off
        Hangup Delay    off
        Line Setting    dx_19200
        Description     none LDISC0

Available states:
        off     on
Select a state (default: off) [?, q]:  on

Enter a hangup delay, in seconds, or 'off' (default: \
            off) [?, q]: <CR>

Available line settings:
co_1200  co_4800  du_1200  du_300   du_9600  dx_19200 dx_4800
co_2400  co_9600  du_2400  du_4800  dx_1200  dx_2400  dx_9600
co_300   console  x9600    x19200
Select a line setting (default: dx_19200) [?, q]:  x9600

Current description:
        none LDISC0
Enter a new description (default: current \
            description) [?, q]: <CR>


ttyh13: new characteristics:
        State           on
        Hangup Delay    off
        Line Setting    x9600
        Description     none LDISC0
Do you want to install these new characteristics? [y, n, q] y
ttyh13 now has new characteristics.

Changeable tty lines:
tty1   ttyh0   ttyh11  ttyh14  ttyh2   ttyh4   ttyh6  ttyh8
tty2   ttyh1   ttyh12  ttyh15  ttyh3   ttyh5   ttyh7  ttyh9
tty3   ttyh10  ttyh13
Select the tty you wish to modify,
or enter ALL to see a report of all ttys [?, q]: q
#
```

Procedure 8.4: Observe UMIPS File Modifications

This procedure simply points out the changes that the preceding procedures have made to UMIPS files. The same changes could have been made directly to the files by the system administrator using a standard editor.

References

grep(1)

Procedure 8.2 caused the **/etc/gettydefs** file to be modified. The changes are demonstrated below by using the **grep** command to list the occurrences of our "test" prompt in the **gettydefs** file:

```
# grep test /etc/gettydefs

x9600# B9600 HUPCL # B9600 SANE IXANY HUPCL TAB3 #test:#x19200
x19200# B19200 HUPCL # B19200 SANE IXANY HUPCL TAB3 #test2:#x9600
#
```

Procedure 8.3 modified the **/etc/inittab** file. In particular, turning the line setting "on" caused the word "off" to be replaced by "respawn", and the former **gettydefs** reference of dx_19200 has been replaced by "x9600". Again, **grep** is used to locate the new information:

```
# grep ttyh13    /etc/inittab

hd:234:respawn:/etc/getty ttyh13 x9600 none LDISC0 # none LDISC0
```

UUCP Networking Procedures

The following procedures are covered in this section:

Procedure 9.1: Set Up UUCP Networking Files

This procedure provides instructions for setting up UUCP and putting it into operation. The following topics are covered:

Set Up Devices File

Set Up **/etc/inittab**

Set Up **Systems** File

Set Up **Poll** File

Set Up **Permissions** File

Set Up **Devconfig** File

Set Up **Sysfiles** File

Add **uucp** logins

Set Up Devices File

The **Devices** file (**/usr/lib/uucp/Devices**) contains information about the devices used to call other machines.

```
#ident

# Some sample entries:
# NOTE - all lines must have at least 5 fields
#    use '-' for unused fields
# The Devices file is used in conjunction with the Dialers file.
# Types that appear in the 5th field must be either built-in
#  functions (801, Sytek, TCP, Unetserver, DK)
#  or standard functions whose name appears in the first
#  field in the Dialers file.
# Two escape characters may appear in this file:
# - \D which means don't translate the phone #/token
# - \T translate the phone #/token using the Dialcodes file
# Both refer to the phone number field in the Systems
#  file (field 5)
# \D should always be used with entries in the Dialers file,
#  since the Dialers file can contain a \T to expand the
#  number if necessary.
# \T should only be used with builtin functions that require
#  expansion
# NOTE: - if a phone number is expected and a \D or \T is
#  not present a \T is used for a builtin, and \D is used
#  for an entry referencing the Dialers file.
#  (see examples below)
#
#
# ---Standard modem line
# ACU cul02 cua02 1200 801
# ACU contty - 1200 penril
# or
# ACU contty - 1200 penril \D
#
# ---A direct line so 'cu -lculd0' will work
# Direct culd0 - 4800 direct
#
# ---A ventel modem on a develcon switch (vent is the token
#     given to the develcon to reach the ventel modem)
# ACU culd0 - 1200 develcon vent ventel
# ACU culd0 - 1200 develcon vent ventel \D
#
# ---To reach a system on the local develcon switch
# Develcon culd0 - Any develcon \D
#
# ---Access a direct connection to a system
# systemx tty00 - Any direct
#
# where the Systems file looks like
# systemx Any systemx 1200 unused  "" in:-\r\d-in: nuucp
#     word: nuucp
#     (The third field in Systems matches the first
#     field in Devices)
#
# ---To connect to any system on the DATAKIT VCS network
# DK DK 0 Any DK \D
```

```
#
# ---To connect to a system on a Datakit in nj/ho
# DKho - uucp Any DK nj/ho/\D
#
# ---To use an ACU that is connected to Datakit that DK does
#     not understand how to talk to directly
# ACU - 0 Any DK vent ventel \D
#
# ---To use a dialer that the Datakit understands how to
#      chat with
#     This is a special case where the translation must be
#     done by the Devices file processing.
# ACU DKacu 0 Any DK py/garage/door.\T
#
# TCP - 0 Any TCP ""

######## AT&T Transport Interface
#
# ---To use a STREAMS network that conforms to the AT&T
#     Transport Interface with a direct connection to login
#     service (i.e., without explicitly using the Network
#     Listener Service dial script):
#
# networkx,eg devicex - - TLIS \D
#
#     The Systems file entry looks like:
#
# systemx Any networkx - addressx in:--in: nuucp word: nuucp
#
#     You must replace systemx, networkx, addressx, and devicex
#     with system name, network name, network address and
#     network device, respectively. For example, entries for
#     machine "sffoo" on a STARLAN NETWORK might look like:
#        sffoo Any STARLAN - sffoo in:--in: nuucp word: nuucp
#     and:
#        STARLAN,eg starlan - - TLIS \D
```

```
#
# ---To use a STREAMS network that conforms to the AT&T
#     Transport Interface and that uses the Network Listener
#     Service dial script to negotiate for a server:
#
# networkx,eg devicex - - TLIS \D nls
#
#
# ---To use a non-STREAMS network that conforms to the AT&T
#     Transport Interface and that uses the Network Listener
#     Service dial script to negotiate for a server:
#
# networkx,eg devicex - - TLI \D nls
#
########
#
#
# NOTE: blank lines and lines that begin with a <space>,
#        <tab>, or # are ignored.
#     protocols can be specified as a comma-subfield of the
#        device type either in the Devices file (where
#        device type is field 1) or in the Systems file
#        (where it is field 3).
```

Set Up /etc/inittab

The **inittab** file (**/etc/inittab**) contains information on the ports to which the devices are connected. For example:

```
    t2:234:respawn:/etc/getty tty2:du_1200
```

This entry specifies port **/dev/tty2** as a dial-in port (it issues a login message due to the respawn entry), at 1200 baud. Refer to **uugetty**(1M) for information on setting-up a bidirectional line for UUCP. For further information on the **inittab** refer to Chapter 3, Processor Operations.

Set Up Systems File

The **Systems** file (**/usr/lib/uucp/Systems**) contains the information needed by **uucp** to call and log on to a remote machine. Each entry represents one remote machine that can be called by your UUCP Networking programs.

```
#ident
#
# Entries have this format:
#
#  Machine-Name Time Type Class Phone Login
#
# Machine-Name    node name of the remote machine
# Time   day-of-week and time-of-day when you may call
#  (e.g., MoTuTh0800-1700). Use "Any" for any day.
#  Use "Never" for machines that poll you, but that
#  you never call directly.
# Type   device type
# Class  transfer speed
# Phone  phone number (for autodialers) or token (for
#  data switches)
# Login  login sequence is composed of fields and subfields
#  in the format "[expect send] ...".  The expect field
#  may have subfields in the format "expect[-send-expect]".
#
# Example:
#  cuuxb Any ACU 1200 chicago8101242 in:--in: nuucp word: panzer
```

Set Up Poll File

The **Poll** file (**/usr/lib/uucp/Poll**) contains a list of the machines that are to be called (polled) by your M-Series RISComputer to see if they have anything to transmit to you. It also contains the times they are to be polled.

```
#ident
# This file (Poll) contains a list of
#  "system <tab> hour1 hour2 hour3 ..." lines for \
   polling remote systems.
# See examples below
#
# Lines starting with # are ignored.
# NOTE a tab must follow the machine name

# Examples:
#raven    2  6  10
#quail    0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 \
   19 20 21 22 23
```

Set Up Permissions File

The default **/usr/lib/uucp/Permissions** file provides the maximum amount of security for your M-Series RISComputer. The file, as delivered, contains the following entry:

```
        LOGNAME=nuucp
```

You can set additional parameters for each machine to define:

the ways it can receive files from your machine

the directories it can read and write in

the commands it can use for remote execution

See Chapter 9 for information on how to set up this file.

Set Up Devconfig File

```
   #ident
#
#  Devconfig provides a means of configuring STREAMS devices
#  by service.
#
#  FORMAT:
#
#  service=<service name>device=<device type> \
#           push=<STREAMS module list>
#
#  where service name is "uucico" or "cu", device type is the
#  device or caller type (3rd field in Systems file, 1st field
#  in Devices file), and STREAMS module list is a
#  colon-separated list of STREAMS modules to be pushed on
#  this stream.
#
#  The examples below are for the STARLAN NETWORK and assume
#  that both cu & uucico are going through login and so need
#  full tty capabilities. If uucico connects to a uucico
#  server on the remote machine without going through login,
#  you would need to push only tirdwr.
#
#  EXAMPLE:
#    service=cu    device=STARLANpush=ntty:tirdwr:ld0
#    service=uucicodevice=STARLANpush=ntty:tirdwr:ld0
#
#  Note: The tirdwr module is part of the STREAMS package.
#  Other modules needed to provide tty capabilities must be
#  supplied by the network provider.
#
```

Set Up Sysfiles File

/usr/lib/uucp/Sysfiles lets you assign different files to be used by **uucp** and **cu** as **Systems**, **Devices**, and **Dialers** files. Here are some cases where this optional file may be useful.

- You may want different **Systems** files so requests for **cu** login services can be made to addresses other than **uucp** services.
- You may want different **Dialers** files to use different chat scripts for **cu** and **uucp**.
- You may want to have multiple **Systems**, **Dialers**, and **Devices** files. The **Systems** file in particular may become large, making it convenient to split it into several smaller files.

The format of the **Sysfiles** file is described in Chapter 9. The following is an example of the file.

```
service=uucico systems=Systems.cico:Systems
                     dialers=Dialers.cico:Dialers
                     devices=Devices.cico:Devices
service=cu     systems=Systems.cu:Systems
                     dialers=Dialers.cu:Dialers
                     devices=Devices.cu:Devices
```

Add uucp logins

You must add one or more administrative logins to your system so incoming **uucp** (**uucico**) requests from remote machines can be handled properly. Each remote machine should have an entry in its **Systems** file for your machine that contains the login ID and password that you add to your **/etc/passwd** file.

An example of a common entry in the **/etc/passwd** file is shown below.

```
nuucp:????:6:1:UUCP.Admin:/usr/spool/uucppublic:\
   /usr/lib/uucp/uucico
```

This entry shows that a login request by **nuucp** is answered by **/usr/lib/uucp/uucico.** The home directory is **/usr/spool/uucppublic**. The *????* will be replaced by an encrypted password that would be added using **passwd nuucp**.

Procedure 9.2: UUCP Networking Maintenance

The purpose of the following procedures are to keep UUCP files from consuming too much disk space. They can be run automatically with **cron** or done manually as needed.

UUCP Networking Utilities comes with four shell scripts that will poll remote machines, reschedule transmissions, and clean up old log files and unsuccessful transmissions. These shell scripts should be executed regularly to keep your basic networking running smoothly. Normally, they are run automatically with **cron**(1M), though they can also be run manually. The few areas needing clean up that are not handled by these shell scripts should be maintained manually.

Automated Networking Maintenance (cron)

The UUCP Networking Utilities are delivered with entries for **uudemon** shell scripts in the **/usr/spool/cron/crontabs/uucp** file. These entries will automatically handle some UUCP administrative tasks for you. Each of these shell scripts is in **/usr/lib/uucp**.

When the M-Series RISComputer is in run state 2 (multi-user), **cron** scans the **/usr/spool/cron/crontabs/root** file every minute for entries scheduled to execute at that time. As the UUCP administrator, you should become familiar with **cron** and the four **uudemon** shell scripts.

uudemon.poll

The **uudemon.poll** shell script, as delivered, does the following:

- Reads the **Poll** file (**/usr/lib/uucp/Poll**) once an hour.
- If any of the machines in the **Poll** file are scheduled to be polled, a work file (**C.sysnxxxx**) is placed in the **/usr/spool/uucp/***nodename* directory, where *nodename* is replaced by the name of the machine.

The shell script is scheduled to run twice an hour just before **uudemon.hour** so that the work files will be there when **uudemon.hour** is called. The default root crontab entry for **uudemon.poll** is as follows:

```
1,30 * * * * "/usr/lib/uucp/uudemon.poll > /dev/null"
```

uudemon.hour

The **uudemon.hour** shell script you receive with your machine does the following:

- Calls the **uusched** program to search the spool directories for work files (C.) that have not been processed and schedules these files for transfer to a remote machine.
- Calls the **uuxqt** daemon to search the spool directories for execute files (X.) that have been transferred to your M-Series RISComputer and were not processed at the time they were transferred.

The default root crontab entry for **uudemon.hour** is as follows:

```
    39,9 * * * * /usr/lib/uucp/uudemon.hour
```

As delivered, this is run twice an hour. You may want it to run more often if you expect high failure rates.

uudemon.admin

The **uudemon.admin** shell script, as delivered, does the following:

- Runs the **uustat** command with **−p** and **−q** options. The **−q** reports on the status of work files (C.), data files (D.), and execute files (X.) that are queued. The **−p** prints process information for networking processes listed in the lock files (**/usr/spool/locks**).
- Sends resulting status information to the **uucp** administrative login via mail.

The default entry in **/usr/spool/cron/crontabs/uucp** for **uudemon.admin** is:

```
48 10,14 * * 1-5 /usr/lib/uucp/uudemon.admin
```

uudemon.cleanup

The delivered **uudemon.cleanu** shell script does the following:

- Takes log files for individual machines from the **/usr/spool/uucp/.Log** directory, merges them, and places them in the **/usr/spool/uucp/.Old** directory with other old log information. If log files get large, the ulimit may need to be increased.
- Removes work files (C.) 7-days old or older, data files (D.) 7 days old or older, and execute files (X.) two days old or older from the spool files.
- Returns to the sender mail that cannot be delivered.
- Mails a summary of the status information gathered during the current day to the UUCP administrative login (**uucp**).

The default uucp crontab entry for **uudemon.cleanup** is:

```
45 23 * * * /usr/lib/uucp/uudemon.cleanup
```

Other Maintenance

Some files may grow indirectly from **uucp** activities. The **/etc/init.d/uucp** script automatically removes **uucp** lock files when the system goes multi-user. If you want to perform other **uucp** maintenance automatically at this time, add the appropriate commands to this script.

Procedure 9.3: UUCP Networking Debugging

These procedures describe how to go about solving common problems that may be encountered with UUCP Networking Utilities.

Check for Faulty ACU/Modem

You can check if the automatic call units or modems are not working properly in several ways.

- Run **uustat −q**. This will give counts and reasons for contact failure.
- Run **cu −d −l***line*. This will let you call over a particular line and print debugging information on the attempt. The line must be defined as Direct in the devices file. (You must add a telephone number to the end of the command line if the line is connected to an autodialer or the device must be set up as **direct.**)

Check Systems File

Check that you have up-to-date information in your systems file if you are having trouble contacting a particular machine. Some things that may be out of date for a machine are its:

- Phone number
- Login
- Password

Debug Transmissions

If you are unable to contact a particular machine, you can check out communications to that machine with **Uutry** and **uucp**.

Step 1: To simply try to make contact, run:

```
$ /usr/lib/uucp/Uutry −r machine
```

where *machine* is replaced with the node name of the machine you are having problems contacting. This command will:

1. Start the transfer daemon (**uucico**) with debugging. You will get more debugging information if you are **root**.
2. Direct the debugging output to /tmp/*machine*,
3. Print the debugging output to your terminal (**tail −f**). Hit **BREAK** to end output.

You can copy the output from **/tmp/***machine* if you want to save it.

Step 2: If **Uutry** doesn't isolate the problem, try to queue a job by running:

```
$ uucp −r file machine!/dir/file
```

where *file* is replaced by the file you want to transfer, *machine* is replaced by the machine you want to copy to, and *dir/file* is where the file will be placed on the other machine. The **–r** option will queue a job but not start the transfer.

Now use **Uutry** again. If you still cannot solve the problem, you may need to call support personnel. Save the debugging output; it will help diagnose the problem.

Check Error Messages

There are two types of error messages for UUCP Networking Utilities: ASSERT and STATUS. See Appendix C for a listing of these messages.

ASSERT Error Messages

When a process is aborted, ASSERT error messages are recorded in **/usr/spool/uucp/.Admin/errors**. These messages include the file name, **sccsid**, line number, and text. These messages usually result from system problems.

STATUS Error Messages

Status error messages are stored in the **/usr/spool/uucp/.Status** directory. The directory contains a separate file for each remote machine your M-Series RISComputer attempts to communicate with. These files contain status information on the attempted communication and whether it was successful.

Check UUCP Information

There are several commands you can use to check for basic networking information.

uuname	Use this command to list those machines your machine can contact.
uulog	Use this command to display the contents of the log directories for particular hosts.
uucheck –v	Run this command to check for the presence of files and directories needed by **uucp**. This command also checks the **Permissions** file and outputs information on the permissions you have set up.

TCP/IP Networking Procedures

The following procedures are covered in this section:

Procedure 10.1: Setting-Up the Network

This procedure sets-up the necessary software to access an existing TCP/IP network. If you are setting up the initial network, configure the other machine(s) in the same way as this one, but use a different host name and address. It is also recommended that you begin to establish the network by setting up two machines, and then adding the other machines after the initial network has been verified.

If you are connecting to an existing network, use the netmask and broadcast address being used there, and consult the network administrator regarding host number and name.

References
Chapter 10, TCP-IP

Step 1: Set-up the hardware

This procedure assumes you have attached your machine to the network via ethernet cable as described in the M-Series Technical Manual and your Ethernet supplier's documentation.

Step 2: Edit the **/etc/local_hostname** File

The **/etc/local_hostname** should contain the name of your machine and the broadcast address. In this case a hostname of "oldguy" and a Class C broadcast address of 192.255.255.255 is used:

```
oldguy
netmask 0xffff0000 broadcast 192.255.255.255
```

Step 3: Edit the /etc/hosts File

The **/etc/hosts** file should be edited to include your hostname, address, and the hostname and address of some other host on the network that you will have access to. In this way it will be possible to get the complete **/etc/hosts** file from that machine once you have the network working.

For example, if the name of your machine is "oldguy", and the host you have access to is "snail", the **hosts** file for a Class C address might look like this:

```
127.1          localhost
192.0.0.1      oldguy
192.0.0.3      snail
```

Your **/etc/hosts** file will probably contain different names and numbers for your local machine and for the remote machine, but it should have the same entry for "localhost" as shown above.

Step 4: Reboot the Machine

To initialize the new information bring the system down and reboot the machine. The easiest way to do this (once any other users are off the system) is:

```
# sync;sync
# init 6
```

Step 5: Test the Local Set-up

Once the system is up in multi-user mode, the **ping**(1M) command can be used to make a quick test of the network connection. First, make sure that the local set-up is functioning by "pinging" the localhost:

```
# ping localhost
PING localhost: 56 data bytes
64 bytes from 127.0.0.1: icmp_seq=0. time=3. ms
64 bytes from 127.0.0.1: icmp_seq=1. time=3. ms
INTERRUPT
----localhost PING Statistics----
2 packets transmitted, 2 packets received, 0% packet loss
round-trip (ms)  min/avg/max = 3/3/3
```

This output can be terminated by entering the Interrupt character (usually the Delete key or CTRL-C). Without going into details (see ping(1M)), the message "0% packet loss" means that the local setup is working.

Step 6: Test the Network Connection

Now, ping the remote host that was included in the **/etc/hosts** file.

```
# ping snail
64 bytes from 192.0.0.3: icmp_seq=0. time=16. ms
64 bytes from 192.0.0.3: icmp_seq=1. time=14. ms
INTERRUPT
----snail PING Statistics----
2 packets transmitted, 2 packets received, 0% packet loss
round-trip (ms)  min/avg/max = 14/15/16
```

This output indicates all is well with the network connection. The next procedures use the TCP/IP connection to perform various tasks.

Procedure 10.2: Using Remote Login and Remote Shell

This procedure exercises the **rlogin**(1C) and **rsh**(1C) utilities. It assumes you have verified the basic network setup as described in the previous procedure. This procedure does not require root privileges.

Step 1: Remote Login

Use rlogin to remotely login to another machine you have access to (for example, the other host name supplied in the previous exercise). You must have an account on the remote machine. For example:

```
$ rlogin snail
Password:
% uname
snail
%
```

You are now on the remote host snail and can change directories, edit files, etc., as if you had logged in on a directly connected terminal.

Step 2: Adding a .rhosts entry

At this point, add a file called **.rhosts** to your home directory on the local host. This will allow you to rlogin without entering a password and will also allow you to use the remote shell utility described shortly. The **.rhosts** file simply contains a list of host-names of machines on which you also have accounts and from which you may want the ability to login remotely or use a remote shell. For example, this example **.rhosts** file in your home directory on the host named "snail":

```
oldguy
fredsbox
```

would allow you to remotely login to snail from "oldguy" and "fredsbox".

Step 3: Terminating a Remote Login

To terminate the remote login, hit Carriage Return and use "~" and a dot ".":

```
% <CR>
% ~.
Closed connection.
$
```

Step 4: Using the Remote Shell Utility

The remote shell (**rsh**(1C)) lets you access a remote machine for the duration of a single command. (**rsh** requires that you have **$HOME/.rhosts** permission on the remote host or that your local machine name is included in the remote hosts' **/etc/hosts.equiv** file.) Note that this is **/usr/ucb/rsh** and not the restricted shell **/bin/rsh**. If your $PATH variable has **/bin** before **/usr/ucb**, use the full path name **/usr/ucb/rsh**.

For example, the following sequence executes the **date** command on the remote machine "snail" but the next prompt is once again from the local machine:

```
$ rsh snail date
Wed May  4 17:06:17 PDT 1988
$
```

Procedure 10.3: Using rwho and ruptime

This procedure uses the **rwho**(1C) and **ruptime**(1C) commands to show who is on the network and what machines are up.

References
rwho(1C)
ruptime(1C)

Step 1: Verify the **rwhod** Daemon

Be sure the **rwhod** daemon is running. This should be automatically started when you go to multiuser mode. To see if it is running, enter:

```
# ps -e | grep rwhod
```

You should see something like this:

```
265 ?        6:43 rwhod
```

If the **rwhod** daemon is not running, be sure you are in multiuser mode.

Step 2: Using **rwho**

To use **rwho**, enter **rwho** at the command line and a list of who is currently on the system shows up:

```
$ rwho
adamb    escargot:ttyq5   May 17 16:11 :14
allen    electron:ttyh2   May 17 13:35 :13
anneli   quacky:ttyqf     May 17 10:30 :22
annette  escargot:ttyq4   May 17 16:47 :24
ara      admin:ttyr3      May 17 08:47 :04
babu     giant:ttyp2      May 17 17:18
.
.
.
```

Step 3: Using **ruptime**

To use **ruptime**, enter **ruptime** at the command line. A list of currently up (and down) machines is shown:

```
$ ruptime
brie        up    6+01:44,    1 user,  load 0.00, 0.00, 0.00
charlie     down  3+10:41
dude        up    1+22:03,    2 users, load 0.00, 0.00, 0.00
dunkshot    up    2+22:17,    3 users, load 0.96, 1.05, 0.91
electron    up    2+02:23,    22 users,load 0.26, 0.82, 0.91
escargot    up    12+00:48,   15 users,load 0.12, 0.23, 0.52
```

Procedure 10.4: Using ftp and telnet

This procedure demonstrates a few simple **ftp** and **telnet** functions as a means to verify that these utilities are working.

Using ftp

To initiate an **ftp** session with another host on the network simply enter **ftp** and the host name:

```
        $ ftp quacky
```

The following output appears on your terminal:

```
        Connected to quacky.
        220 quacky FTP server (4.3BSD version 4.105) ready.
        Name (quacky:johnr): 331 Password required for johnr.
        230 User johnr logged in.
        ftp>
```

Note that with **ftp** you have to supply a password even if you have an account on the system - **ftp** does not use the **.rhosts** file. To see a list of available commands, enter "help" at the ftp> prompt:

```
ftp> help
Commands may be abbreviated.  Commands are:

!         cr          macdef    proxy        send
$         delete      mdelete   sendport     status
account   debug       mdir      put          struct
append    dir         mget      pwd          sunique
ascii     disconnect  mkdir     quit         tenex
bell      form        mls       quote        trace
binary    get         mode      recv         type
bye       glob        mput      remotehelp   user
case      hash        nmap      rename       verbose
cd        help        ntrans    reset        ?
cdup      lcd         open      rmdir
close     ls          prompt    runique
ftp>


ftp> ls
200 PORT command successful.
150 Opening data connection for /bin/ls (97.4.0.30,1118) (0 bytes).
Alpha
Mail
News
Work
aardvark
afile
(etc.)

226 Transfer complete.
563 bytes received in 0.67 seconds (0.82 Kbytes/s)
```

```
        ftp> cd ..
250 CWD command successful.
ftp> ls
200 PORT command successful.
150 Opening data connection for /bin/ls (97.4.0.30,1119) (0 bytes).
allen
alt
anneli
annette
(etc.)

226 Transfer complete.
770 bytes received in 0.3 seconds (2.5 Kbytes/s)
ftp> pwd
257 "/usr/user" is current directory.
ftp> cd ~
250 CWD command successful.
ftp> get temp
200 PORT command successful.
150 Opening data connection for temp (97.4.0.30,1120) (844 bytes).
w226 Transfer complete.
local: temp remote: temp
873 bytes received in 0.017 seconds (51 Kbytes/s)
ftp> quit
221 Goodbye.
```

Using telnet

Below is a commented printout of a simple **telnet** session.

```
johnr@dunkshot_95% telnet painless
Trying...
Connected to painless.
Escape character is '^]'.


MIPS (painless)



login: johnr
Password:
UNIX System V Release 3_0 painless
Copyright (c) 1984 AT&T
All Rights Reserved

*************************************************
*                                               *
*       MIPS - The Measure of Performance       *
*                                               *
*************************************************

no mail
johnr@painless_51% date
```

```
Wed Jun 22 11:26:45 PDT 1988
johnr@painless_52% telnet
telnet> ?
Commands may be abbreviated.  Commands are:

open            connect to a site
close           close current connection
quit            exit telnet
escape          set escape character
status          print status information
options         toggle viewing of options processing
crmod           toggle mapping of received carriage returns
debug           toggle debugging
ayt             send Are You There
interrupt       send Interrupt Process
passthru        send escape character
?               print help information
telnet> quit
johnr@painless_53% logout
logout
Bye...
Connection closed by foreign host.

johnr@dunkshot_95%
```

NFS Procedures

The following procedures are covered in this section:

Procedure 11.1	**Setting-Up an NFS Server** To export a file system to the network.
Procedure 11.2	**Setting-Up an NFS Client** To access a remote file system.
Procedure 11.3	**Automatic Remote Mounts** To mount remote file systems automatically at boot time.

Procedure 11.1: Setting-Up an NFS Server

This procedure sets-up your M-Series system as an NFS server. It assumes you are already connected to the network and able to use the TCP/IP procedures described in the previous section. This procedure is performed in multiuser mode.

If you do not want to be an NFS server and only want to mount a remote file system from another server (i.e., be an NFS client), skip this procedure and go to the next procedure.

References

mount(1M)
nfsd(1M)
biod(1M)

Step 1: Check NFS Daemons

Check the process status of the NFS and block I/O daemons by entering the following:

```
# ps -d | grep nfs
```

You should see something like the following:

```
307 ?        11:32 nfsd
308 ?        12:33 nfsd
306 ?        12:33 nfsd
309 ?        12:25 nfsd
```

This means the NFS daemons are running. Now enter:

```
# ps -d | grep biod
```

and you should see something like this:

```
311 ?         0:00 biod
313 ?         0:00 biod
315 ?         0:00 biod
317 ?         0:00 biod
```

If these daemons do not show up, you are probably not in multiuser mode. Either go to multiuser mode or fire them up manually by executing the corresponding shell script:

```
# /etc/rc2.d/S40nfs
```

Step 2: Export a File System

To export a file system (make it available to the network) you must edit the **/etc/exports** file to include the name of the file system. For example, if you want to export the **/usr** file system to everyone on the network, put this entry in **/etc/exports**:

```
/usr
```

If you want to export the file system to just one machine, enter:

```
/usr    hostname
```

where *hostname* is the name of the remote host to which you want to export **/usr**.

Procedure 11.2: Setting-Up An NFS Client

This procedure sets up your M-Series system as an NFS client. It assumes you are already connected to the network and able to use the TCP/IP protocol procedures described in the previous section.

References

mount(1M)
biod(1M)

Mount a Remote File System

If you are connected to a network that has a server which is exporting a file system to you, you can mount a remote file system. Make a directory to mount it on, for example:

```
# mkdir /usr2
```

and then use the **mount** command to mount the remote file system:

```
# mount oldguy:/usr /usr2
```

This command mounts the **/usr** directory exported by the remote host "oldguy" onto the local **/usr2** directory. (If you are becoming a client on an already existing network, talk to a network administrator to find out which remote file systems can be accessed by your machine.) The **/usr** directory of "oldguy" can now be accessed just like any other local directory.

Procedure 11.3: Automatic Remote Mounts

In this procedure, we modify the **/etc/fstab** file to automatically mount the **/usr** directory from *oldguy*. In this case the remote file system is being mounted to use its man pages - so we can remove the man pages on the local machine to get more disk space. Since we are only interested in the man pages, we mount the file system "read-only". (Note that with NFS, the server cannot export just a part of a file system, for example **/usr/man**. The entire file system must be exported.)

Step 1: Modify /etc/fstab

Use an editor to add a line to the **/etc/fstab** file. The file systems specified in **/etc/fstab** are mounted when the system goes multiuser. The following entry mounts the remote **/usr** file system from the host *oldguy*:

```
oldguy:/usr /usr2 nfs ro,bg,soft,timeo=20 0 0
```

For details on what's being done here, refer to the **mount**(1M) man page. The main things that concern us here are the "nfs" and "ro" fields. "nfs" states that the file system being mounted is an NFS file system and "ro" states that the file system is to be mounted "read-only".

Step 2: Mount the File System

If you reboot your system the system will attempt to mount the remote file system when it goes mulituser. If you don't want to reboot, you can enter:

```
# mount -a
```

to cause the **mount** command to attempt to mount all the file systems specified in **/etc/fstab**.

Step 3: Verify the Remote Mount

To verify that the remote file system is mounted, enter:

```
# mount
```

The **mount** command with no arguments list all the currently mounted file systems. You can now "ls" the **/usr2** directory, for example, and see the files and directories in *oldguy*'s **/usr** directory. To use *oldguy*'s man pages, enter the following:

```
# ln -s /usr2/usr/man /usr/man
```

NOTE: This only works if you've already removed **/usr/man** from your file system.

To verify that the file system is mounted "read-only", attempt to make a new file some place in the **/usr** directory structure. Permission to save the file should be denied.

A Fast File System for UNIX*

Marshall Kirk McKusick, William N. Joy,
Samuel J. Leffler, Robert S. Fabry

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

(* UNIX is a trademark of Bell Laboratories.)

ABSTRACT

A reimplementation of the UNIX file system is described. The reimplementation provides substantially higher throughput rates by using more flexible allocation policies that allow better locality of reference and can be adapted to a wide range of peripheral and processor characteristics. The new file system clusters data that is sequentially accessed and provides two block sizes to allow fast access to large files while not wasting large amounts of space for small files. File access rates of up to ten times faster than the traditional UNIX file system are experienced. Long needed enhancements to the programmers' interface are discussed. These include a mechanism to place advisory locks on files, extensions of the name space across file systems, the ability to use long file names, and provisions for administrative control of resource usage.

Revised February 18, 1984

CR Categories and Subject Descriptors:

D.4.3 [Operating Systems]

File Systems Management –
file organization, directory structures, access methods

D.4.2 [Operating Systems]

Storage Management –
allocation/deallocation strategies, secondary storage devices

D.4.8 [Operating Systems]

Performance –
measurements, operational analysis

H.3.2 [Information Systems]

Information Storage –
file organization

Additional Keywords and Phrases: UNIX, file system organization, file system performance, file system design, application program interface.

General Terms: file system, measurement, performance.

Old File System

In the file system developed at Bell Laboratories (the "traditional" file system), each disk drive is divided into one or more partitions. Each of these disk partitions may contain one file system. A file system never spans multiple partitions. By "partition" here we refer to the subdivision of physical space on a disk drive. In the traditional file system, as in the new file system, file systems are really located in logical disk partitions that may overlap. This overlapping is made available, for example, to allow programs to copy entire disk drives containing multiple file systems. A file system is described by its super-block, which contains the basic parameters of the file system. These include the number of data blocks in the file system, a count of the maximum number of files, and a pointer to the *free list*, a linked list of all the free blocks in the file system.

Within the file system are files. Certain files are distinguished as directories and contain pointers to files that may themselves be directories. Every file has a descriptor associated with it called an *inode*. An inode contains information describing ownership of the file, time stamps marking last modification and access times for the file, and an array of indices that point to the data blocks for the file. For the purposes of this section, we assume that the first 8 blocks of the file are directly referenced by values stored in an inode itself*. * The actual number may vary from system to system, but is usually in the range 5-13. An inode may also contain references to indirect blocks containing further data block indices. In a file system with a 512 byte block size, a singly indirect block contains 128 further block addresses, a doubly indirect block contains 128 addresses of further singly indirect blocks, and a triply indirect block contains 128 addresses of further doubly indirect blocks.

A 150 megabyte traditional UNIX file system consists of 4 megabytes of inodes followed by 146 megabytes of data. This organization segregates the inode information from the data; thus accessing a file normally incurs a long seek from the file's inode to its data. Files in a single directory are not typically allocated consecutive slots in the 4 megabytes of inodes, causing many non-consecutive blocks of inodes to be accessed when executing operations on the inodes of several files in a directory.

The allocation of data blocks to files is also suboptimum. The traditional file system never transfers more than 512 bytes per disk transaction and often finds that the next sequential data block is not on the same cylinder, forcing seeks between 512 byte transfers. The combination of the small block size, limited read-ahead in the system, and many seeks severely limits file system throughput.

The first work at Berkeley on the UNIX file system attempted to improve both reliability and throughput. The reliability was improved by staging modifications to critical file system information so that they could either be completed or repaired cleanly by a program after a crash [Kowalski78]. The file system performance was improved by a factor of more than two by changing the basic block size from 512 to 1024 bytes. The increase was because of two factors: each disk transfer accessed twice as much data, and most files could be described without need to access indirect blocks since the direct blocks contained twice as much data. The file system with these changes will henceforth be referred to as the *old file system*.

This performance improvement gave a strong indication that increasing the block size was a good method for improving throughput. Although the throughput had doubled, the old file system was still using only about four percent of the disk bandwidth. The main problem was that although the free list was initially ordered for optimal access, it quickly became scrambled as files were created and removed. Eventually the free list became entirely random, causing files to have their blocks allocated randomly over the disk. This forced a seek before every block access. Although old file systems provided transfer rates of up to 175 kilobytes per second when they were first created, this rate deteriorated to 30 kilobytes per second after a few weeks of moderate use because of this randomization of data block placement. There was no way of restoring the performance of an old file system except to dump, rebuild, and restore the file system. Another possibility, as suggested by [Maruyama76], would be to have a process that periodically reorganized the data on the disk to restore locality.

New File System Organization

In the new file system organization (as in the old file system organization), each disk drive contains one or more file systems. A file system is described by its super-block, located at the beginning of the file system's disk partition. Because the super-block contains critical data, it is replicated to protect against catastrophic loss. This is done when the file system is created; since the super-block data does not change, the copies need not be referenced unless a head crash or other hard disk error causes the default super-block to be unusable.

To insure that it is possible to create files as large as $2 sup 32$ bytes with 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 a file system is recorded in the file system's super-block so it is possible for file systems with different block sizes to be simultaneously accessible on the same system. The block size must be decided at the time that the file system is created; it cannot be subsequently changed without rebuilding the file system.

The new file system organization divides a disk partition into one or more areas called *cylinder groups*. A cylinder group is comprised of one or more consecutive cylinders on a disk. Associated with each cylinder group is some bookkeeping information that includes a redundant copy of the super-block, space for inodes, a bit map describing available blocks in the cylinder group, and summary information describing the usage of data blocks within the cylinder group. The bit map of available blocks in the cylinder group replaces the traditional file system's free list. For each cylinder group a static number of inodes is allocated at file system creation time. The default policy is to allocate one inode for each 2048 bytes of space in the cylinder group, expecting this to be far more 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 redundant copies of the super-block. Thus the cylinder group bookkeeping information begins at a varying offset from the beginning of the cylinder group. The offset for each successive cylinder group is calculated to be about one track further from the beginning of the cylinder group than the preceding cylinder group. In this way the redundant information spirals down into the pack so that any single track, cylinder, or platter can be lost without losing all copies of the super-block. Except for the first cylinder group, the space between the beginning of the cylinder group and the beginning of the cylinder group information is used for data blocks. While it appears that the first cylinder group could be laid out with its super-block at the "known" location, this would not work for file systems with blocks sizes of 16 kilobytes or greater. This is because of a requirement that the first 8 kilobytes of the disk be reserved for a bootstrap program and a separate requirement that the cylinder group information begin on a file system block boundary. To start the cylinder group on a file system block boundary, file systems with block sizes larger than 8 kilobytes would have to leave an empty space between the end of the boot block and the beginning of the cylinder group. Without knowing the size of the file system blocks, the system would not know what roundup function to use to find the beginning of the first cylinder group.

Optimizing Storage Utilization

Data is laid out so that larger blocks can be transferred in a single disk transaction, greatly increasing file system throughput. As an example, consider a file in the new file system composed of 4096 byte data blocks. In the old file system this file would be composed of 1024 byte blocks. By increasing the block size, disk accesses in the new file system may transfer up to four times as much information per disk transaction. In large files, several 4096 byte blocks may be allocated from the same cylinder so that even larger data transfers are possible before requiring a seek.

The main problem with larger blocks is that most UNIX file systems are composed of many small files. A uniformly large block size wastes space. Table 1 shows the effect of file system block size on the amount of wasted space in the file system. The files measured to obtain these figures reside on one of our time sharing systems that has roughly 1.2 gigabytes of on-line storage. The measurements are based on the active user file systems containing about 920 megabytes of formatted space.
Table 1 – Amount of wasted space as a function of block size.

Space used	% waste	Organization
775.2 Mb	0.0	Data only, no separation between files
807.8 Mb	4.2	Data only, each file starts on 512 byte boundary
828.7 Mb	6.9	Data + inodes, 512 byte block UNIX file system
866.5 Mb	11.8	Data + inodes, 1024 byte block UNIX file system
948.5 Mb	22.4	Data + inodes, 2048 byte block UNIX file system
1128.3 Mb	45.6	Data + inodes, 4096 byte block UNIX file system

The space wasted is calculated to be the percentage of space on the disk not containing user data. As the block size on the disk increases, the waste rises quickly, to an intolerable 45.6% waste with 4096 byte file system blocks.

To be able to use large blocks without undue waste, small files must be stored in a more efficient way. The new file system accomplishes this goal by allowing the division of a single file system block into one or more *fragments*. The file system fragment size is specified at the time that the file system is created; each file system block can optionally be broken into 2, 4, or 8 fragments, each of which is addressable. The lower bound on the size of these fragments is constrained by the disk sector size, typically 512 bytes. The block map associated with each cylinder group records the space available in a cylinder group at the fragment level; to determine if a block is available, aligned fragments are examined. Figure 1 shows a piece of a map from a 4096/1024 file system.
Figure 1 – Example layout of blocks and fragments in a 4096/1024 file system.

Bits in map	XXXX	XXOO	OOXX	OOOO
Fragment numbers	0-3	4-7	8-11	12-15
Block numbers	0	1	2	3

Each bit in the map records the status of a fragment; an "X" shows that the fragment is in use, while a "O" shows that the fragment is available for allocation. In this example, fragments 0–5, 10, and 11 are in use, while fragments 6–9, and 12–15 are free. Fragments of adjoining blocks cannot be used as a full block, even if they are large enough. In this example, fragments 6–9 cannot be allocated as a full block; only fragments 12–15 can be coalesced into a full block.

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 remaining fragments of the block are made available for allocation to other files. As an example consider an 11000 byte file stored on a 4096/1024 byte file system. This file would uses two full size blocks and one three fragment portion of another block. If no block with three aligned fragments is available at the time the file is created, a full size block is split yielding the necessary fragments and a single unused fragment. This remaining fragment can be allocated to another file as needed.

Space is allocated to a file when a program does a *write* system call. Each time data is written to a file, the system checks to see if the size of the file has increased. A program may be overwriting data in the middle of an existing file in which case space would already have been allocated. If the file needs to be expanded to hold the new data, one of three conditions exists: There is enough space left in an already allocated block or fragment to hold the new data. The new data is written into the available space. The file contains no fragmented blocks (and the last block in the file contains insufficient space to hold the new data). If space exists in a block already allocated, the space is filled with new data. If the remainder of the new data contains more than a full block of data, a full block is allocated and the first full block of new data is written there. This process is repeated until less than a full block of new data remains. If the remaining new data to be written will fit in less than a full block, a block with the necessary fragments is located, otherwise a full block is located. The remaining new data is written into the located space. The file contains one or more fragments (and the fragments contain insufficient space to hold the new data). If the size of the new data plus the size of the data already in the fragments exceeds the size of a full block, a new block is allocated. The contents of the fragments are copied to the beginning of the block and the remainder of the block is filled with new data. The process then continues as in (2) above. Otherwise, if the new data to be written will fit in less than a full block, a block with the necessary fragments is located, otherwise a full block is located. The contents of the existing fragments appended with the new data are written into the allocated space.

The problem with expanding a file one fragment at a a time is that data may be copied many times as a fragmented block expands to a full block. Fragment reallocation can be minimized if the user program writes a full block at a time, except for a partial block at the end of the file. Since file systems with different block sizes may reside on the same system, the file system interface has been extended to provide application programs the optimal size for a read or write. For files the optimal size is the block size of the file system on which the file is being accessed. For other objects, such as pipes and sockets, the optimal size is the underlying buffer size. This feature is used by the Standard Input/Output Library, a package used by most user programs. This feature is also used by certain system utilities such as archivers and loaders that do their own input and output management and need the highest possible file system bandwidth.

The amount of wasted space in the 4096/1024 byte new file system organization is empirically observed to be about the same as in the 1024 byte old file system organization. A file system with 4096 byte blocks and 512 byte fragments has about the same amount of wasted space as the 512 byte block UNIX file system. The new file system uses less space than the 512 byte or 1024 byte file systems for indexing information for large files and the same amount of space for small files. These savings are offset by the need to use more space for keeping track of available free blocks. The net result is about the same disk utilization when a new file system's fragment size equals an old file system's block size.

In order for the layout policies to be effective, a file system cannot be kept completely full. For each file system there is a parameter, termed the free space reserve, that gives the minimum acceptable percentage of file system blocks that should be free. If the number of free blocks drops below this level only the system administrator can continue to allocate blocks. The value of this parameter may be changed at any time, even when the file system is mounted and active. The transfer rates that appear in section 4 were measured on file systems kept less than 90% full (a reserve of 10%). If the number of free blocks falls to zero, the file system throughput tends to be cut in half, because of the inability of the file system to localize blocks in a file. If a file system's performance degrades because of overfilling, it may be restored by removing files until the amount of free space once again reaches the minimum acceptable level. Access rates for files created during periods of little free space may be restored by moving their data once enough space is available. The free space reserve must be added to the percentage of waste when comparing the organizations given in Table 1. Thus, the percentage of waste in an old 1024 byte UNIX file system is roughly comparable to a new 4096/512 byte file system with the free space reserve set at 5%. (Compare 11.8% wasted with the old file system to 6.9% waste + 5% reserved space in the new file system.)

File System Parameterization

Except for the initial creation of the free list, the old file system ignores the parameters of the underlying hardware. It has no information about either the physical characteristics of the mass storage device, or the hardware that interacts with it. A goal of the new file system is to parameterize the processor capabilities and mass storage characteristics so that blocks can be allocated in an optimum configuration-dependent way. Parameters used include the speed of the processor, the hardware support for mass storage transfers, and the characteristics of the mass storage devices. Disk technology is constantly improving and a given installation can have several different disk technologies running on a single processor. Each file system is parameterized so that it can be adapted to the characteristics of the disk on which it is placed.

For mass storage devices such as disks, the new file system tries to allocate new blocks on the same cylinder as the previous block in the same file. Optimally, these new blocks will also be rotationally well positioned. The distance between "rotationally optimal" blocks varies greatly; it can be a consecutive block or a rotationally delayed block depending on system characteristics. On a processor with an input/output channel that does not require any processor intervention between mass storage transfer requests, two consecutive disk blocks can often be accessed without suffering lost time because of an intervening disk revolution. For processors without input/output channels, the main processor must field an interrupt and prepare for a new disk transfer. The expected time to service this interrupt and schedule a new disk transfer depends on the speed of the main processor.

The physical characteristics of each disk include the number of blocks per track and the rate at which the disk spins. The allocation routines use this information to calculate the number of milliseconds required to skip over a block. The characteristics of the processor include the expected time to service an interrupt and schedule a new disk transfer. Given a block allocated to a file, the allocation routines calculate the number of blocks to skip over so that the next block in the file will come into position under the disk head in the expected amount of time that it takes to start a new disk transfer operation. For programs that sequentially access large amounts of data, this strategy minimizes the amount of time spent waiting for the disk to position itself.

To ease the calculation of finding rotationally optimal blocks, the cylinder group summary information includes a count of the available blocks in a cylinder group at different rotational positions. Eight rotational positions are distinguished, so the resolution of the summary information is 2 milliseconds for a typical 3600 revolution per minute drive. The super-block contains a vector of lists called *rotational layout tables*. The vector is indexed by rotational position. Each component of the vector lists the index into the block map for every data block contained in its rotational position. When looking for an allocatable block, the system first looks through the summary counts for a rotational position with a non-zero block count. It then uses the index of the rotational position to find the appropriate list to use to index through only the relevant parts of the block map to find a free block.

The parameter that defines the minimum number of milliseconds between the completion of a data transfer and the initiation of another data transfer on the same cylinder can be changed at any time, even when the file system is mounted and active. If a file system is parameterized to lay out blocks with a rotational separation of 2 milliseconds, and the disk pack is then moved to a system that has a processor requiring 4 milliseconds to schedule a disk operation, the throughput will drop precipitously because of lost disk revolutions on nearly every block. If the eventual target machine is known, the file system can be parameterized for it even though it is initially created on a different processor. Even if the move is not known in advance, the rotational layout delay can be reconfigured after the disk is moved so that all further allocation is done based on the characteristics of the new host.

Layout Policies

The file system layout policies are divided into two distinct parts. At the top level are global policies that use file system wide summary information to make decisions regarding the placement of new inodes and data blocks. These routines are responsible for deciding the placement of new directories and files. They also calculate rotationally optimal block layouts, and decide when to force a long seek to a new cylinder group because there are insufficient blocks left in the current cylinder group to do reasonable layouts. Below the global policy routines are the local allocation routines that use a locally optimal scheme to lay out data blocks.

Two methods for improving file system performance are to increase the locality of reference to minimize seek latency as described by [Trivedi80], and to improve the layout of data to make larger transfers possible as described by [Nevalainen77]. The global layout policies try to improve performance by clustering related information. They cannot attempt to localize all data references, but must also try to spread unrelated data among different cylinder groups. If too much localization is attempted, the local cylinder group may run out of space forcing the data to be scattered to non-local cylinder groups. Taken to an extreme, total localization can result in a single huge cluster of data resembling the old file system. The global policies try to balance the two conflicting goals of localizing data that is concurrently accessed while spreading out unrelated data.

One allocatable resource is inodes. Inodes are used to describe both files and directories. Inodes of files in the same directory are frequently accessed together. For example, the "list directory" command often accesses the inode for each file in a directory. The layout policy tries to place all the inodes of files in a directory in the same cylinder group. To ensure that files are distributed throughout the disk, a different policy is used for directory allocation. A new directory is placed in a cylinder group that has a greater than average number of free inodes, and the smallest number of directories already in it. The intent of this policy is to allow the inode clustering policy to succeed most of the time. The allocation of inodes within a cylinder group is done using a next free strategy. Although this allocates the inodes randomly within a cylinder group, all the inodes for a particular cylinder group can be read with 8 to 16 disk transfers. (At most 16 disk transfers are required because a cylinder group may have no more than 2048 inodes.) This puts a small and constant upper bound on the number of disk transfers required to access the inodes for all the files in a directory. In contrast, the old file system typically requires one disk transfer to fetch the inode for each file in a directory.

The other major resource is data blocks. Since data blocks for a file are typically accessed together, the policy routines try to place all data blocks for a file in the same cylinder group, preferably at rotationally optimal positions in the same cylinder. The problem with allocating all the data blocks in the same cylinder group is that large files will quickly use up available space in the cylinder group, forcing a spill over to other areas. Further, using all the space in a cylinder group causes future allocations for any file in the cylinder group to also spill to other areas. Ideally none of the cylinder groups should ever become completely full. The heuristic solution chosen is to redirect block allocation to a different cylinder group when a file exceeds 48 kilobytes, and at every megabyte thereafter. The first spill over point at 48 kilobytes is the point at which a file on a 4096 byte block file system first requires a single indirect block. This appears to be a natural first point at which to redirect block allocation. The other spillover points are chosen with the intent of forcing block allocation to be redirected when a file has used about 25% of the data blocks in a cylinder group. In observing the new file system in day to day use, the heuristics appear to work well in minimizing the number of completely filled cylinder groups. The newly chosen cylinder group is selected from those cylinder groups that have a greater than average number of free blocks left. Although big files tend to be spread out over the disk, a megabyte of data is typically accessible before a long seek must be performed, and the cost of one long seek per megabyte is small.

The global policy routines call local allocation routines with requests for specific blocks. The local allocation routines will always allocate the requested block if it is free, otherwise it allocates a free block of the requested size that is rotationally closest to the requested block. If the global layout policies had complete information, they could always request unused blocks and the allocation routines would be reduced to simple bookkeeping. However, maintaining complete information is costly; thus the implementation of the global layout policy uses heuristics that employ only partial information.

If a requested block is not available, the local allocator uses a four level allocation strategy: Use the next available block rotationally closest to the requested block on the same cylinder. It is assumed here that head switching time is zero. On disk controllers where this is not the case, it may be possible to incorporate the time required to switch between disk platters when constructing the rotational layout tables. This, however, has not yet been tried. If there are no blocks available on the same cylinder, use a block within the same cylinder group. If that cylinder group is entirely full, quadratically hash the cylinder group number to choose another cylinder group to look for a free block. Finally if the hash fails, apply an exhaustive search to all cylinder groups.

Quadratic hash is used because of its speed in finding unused slots in nearly full hash tables [Knuth75]. File systems that are parameterized to maintain at least 10% free space rarely use this strategy. File systems that are run without maintaining any free space typically have so few free blocks that almost any allocation is random; the most important characteristic of the strategy used under such conditions is that the strategy be fast.

Performance

Ultimately, the proof of the effectiveness of the algorithms described in the previous section is the long term performance of the new file system.

Our empirical studies have shown that the inode layout policy has been effective. When running the "list directory" command on a large directory that itself contains many directories (to force the system to access inodes in multiple cylinder groups), the number of disk accesses for inodes is cut by a factor of two. The improvements are even more dramatic for large directories containing only files, disk accesses for inodes being cut by a factor of eight. This is most encouraging for programs such as spooling daemons that access many small files, since these programs tend to flood the disk request queue on the old file system.

Table 2 summarizes the measured throughput of the new file system. Several comments need to be made about the conditions under which these tests were run. The test programs measure the rate at which user programs can transfer data to or from a file without performing any processing on it. These programs must read and write enough data to insure that buffering in the operating system does not affect the results. They are also run at least three times in succession; the first to get the system into a known state and the second two to insure that the experiment has stabilized and is repeatable. The tests used and their results are discussed in detail in [Kridle83].
A UNIX command that is similar to the reading test that we used is "cp file /dev/null", where "file" is eight megabytes long. The systems were running multi-user but were otherwise quiescent. There was no contention for either the CPU or the disk arm. The only difference between the UNIBUS and MASSBUS tests was the controller. All tests used an AMPEX Capricorn 330 megabyte Winchester disk. As Table 2 shows, all file system test runs were on a VAX 11/750. All file systems had been in production use for at least a month before being measured. The same number of system calls were performed in all tests; the basic system call overhead was a negligible portion of the total running time of the tests.

Type of File System	Processor and Bus Measured	Speed	Write Bandwidth	% CPU
old 1024	750/UNIBUS	48 Kbytes/sec	48/983 5%	29%
new 4096/1024	750/UNIBUS	142 Kbytes/sec	142/983 14%	43%
new 8192/1024	750/UNIBUS	215 Kbytes/sec	215/983 22%	46%
new 4096/1024	750/MASSBUS	323 Kbytes/sec	323/983 33%	94%
new 8192/1024	750/MASSBUS	466 Kbytes/sec	466/983 47%	95%

Unlike the old file system, the transfer rates for the new file system do not appear to change over time. The throughput rate is tied much more strongly to the amount of free space that is maintained. The measurements in Table 2 were based on a file system with a 10% free space reserve. Synthetic work loads suggest that throughput deteriorates to about half the rates given in Table 2 when the file systems are full.

The percentage of bandwidth given in Table 2 is a measure of the effective utilization of the disk by the file system. An upper bound on the transfer rate from the disk is calculated by multiplying the number of bytes on a track by the number of revolutions of the disk per second. The bandwidth is calculated by comparing the data rates the file system is able to achieve as a percentage of this rate. Using this metric, the old file system is only able to use about 3–5% of the disk bandwidth, while the new file system uses up to 47% of the bandwidth.

Both reads and writes are faster in the new system than in the old system. The biggest factor in this speedup is because of the larger block size used by the new file system. The overhead of allocating blocks in the new system is greater than the overhead of allocating blocks in the old system, however fewer blocks need to be allocated in the new system because they are bigger. The net effect is that the cost per byte allocated is about the same for both systems.

In the new file system, the reading rate is always at least as fast as the writing rate. This is to be expected since the kernel must do more work when allocating blocks than when simply reading them. Note that the write rates are about the same as the read rates in the 8192 byte block file system; the write rates are slower than the read rates in the 4096 byte block file system. The slower write rates occur because the kernel has to do twice as many disk allocations per second, making the processor unable to keep up with the disk transfer rate.

In contrast the old file system is about 50% faster at writing files than reading them. This is because the write system call is asynchronous and the kernel can generate disk transfer requests much faster than they can be serviced, hence disk transfers queue up in the disk buffer cache. Because the disk buffer cache is sorted by minimum seek distance, the average seek between the scheduled disk writes is much less than it would be if the data blocks were written out in the random disk order in which they are generated. However when the file is read, the read system call is processed synchronously so the disk blocks must be retrieved from the disk in the non-optimal seek order in which they are requested. This forces the disk scheduler to do long seeks resulting in a lower throughput rate.

In the new system the blocks of a file are more optimally ordered on the disk. Even though reads are still synchronous, the requests are presented to the disk in a much better order. Even though the writes are still asynchronous, they are already presented to the disk in minimum seek order so there is no gain to be had by reordering them. Hence the disk seek latencies that limited the old file system have little effect in the new file system. The cost of allocation is the factor in the new system that causes writes to be slower than reads.

The performance of the new file system is currently limited by memory to memory copy operations required to move data from disk buffers in the system's address space to data buffers in the user's address space. These copy operations account for about 40% of the time spent performing an input/output operation. If the buffers in both address spaces were properly aligned, this transfer could be performed without copying by using the VAX virtual memory management hardware. This would be especially desirable when transferring large amounts of data. We did not implement this because it would change the user interface to the file system in two major ways: user programs would be required to allocate buffers on page boundaries, and data would disappear from buffers after being written.

Greater disk throughput could be achieved by rewriting the disk drivers to chain together kernel buffers. This would allow contiguous disk blocks to be read in a single disk transaction. Many disks used with UNIX systems contain either 32 or 48 512 byte sectors per track. Each track holds exactly two or three 8192 byte file system blocks, or four or six 4096 byte file system blocks. The inability to use contiguous disk blocks effectively limits the performance on these disks to less than 50% of the available bandwidth. If the next block for a file cannot be laid out contiguously, then the minimum spacing to the next allocatable block on any platter is between a sixth and a half a revolution. The implication of this is that the best possible layout without contiguous blocks uses only half of the bandwidth of any given track. If each track contains an odd number of sectors, then it is possible to resolve the rotational delay to any number of sectors by finding a block that begins at the desired rotational position on another track. The reason that block chaining has not been implemented is because it would require rewriting all the disk drivers in the system, and the current throughput rates are already limited by the speed of the available processors.

Currently only one block is allocated to a file at a time. A technique used by the DEMOS file system when it finds that a file is growing rapidly, is to preallocate several blocks at once, releasing them when the file is closed if they remain unused. By batching up allocations, the system can reduce the overhead of allocating at each write, and it can cut down on the number of disk writes needed to keep the block pointers on the disk synchronized with the block allocation [Powell79]. This technique was not included because block allocation currently accounts for less than 10% of the time spent in a write system call and, once again, the current throughput rates are already limited by the speed of the available processors.

File System Functional Enhancements

The performance enhancements to the UNIX file system did not require any changes to the semantics or data structures visible to application programs. However, several changes had been generally desired for some time but had not been introduced because they would require users to dump and restore all their file systems. Since the new file system already required all existing file systems to be dumped and restored, these functional enhancements were introduced at this time.

Long File Names

File names can now be of nearly arbitrary length. Only programs that read directories are affected by this change. To promote portability to UNIX systems that are not running the new file system, a set of directory access routines have been introduced to provide a consistent interface to directories on both old and new systems.

Directories are allocated in 512 byte units called chunks. This size is chosen so that each allocation can be transferred to disk in a single operation. Chunks are broken up into variable length records termed directory entries. A directory entry contains the information necessary to map the name of a file to its associated inode. No directory entry is allowed to span multiple chunks. The first three fields of a directory entry are fixed length and contain: an inode number, the size of the entry, and the length of the file name contained in the entry. The remainder of an entry is variable length and contains a null terminated file name, padded to a 4 byte boundary. The maximum length of a file name in a directory is currently 255 characters.

Available space in a directory is recorded by having one or more entries accumulate the free space in their entry size fields. This results in directory entries that are larger than required to hold the entry name plus fixed length fields. Space allocated to a directory should always be completely accounted for by totaling up the sizes of its entries. When an entry is deleted from a directory, its space is returned to a previous entry in the same directory chunk by increasing the size of the previous entry by the size of the deleted entry. If the first entry of a directory chunk is free, then the entry's inode number is set to zero to indicate that it is unallocated.

File Locking

The old file system had no provision for locking files. Processes that needed to synchronize the updates of a file had to use a separate "lock" file. A process would try to create a "lock" file. If the creation succeeded, then the process could proceed with its update; if the creation failed, then the process would wait and try again. This mechanism had three drawbacks. Processes consumed CPU time by looping over attempts to create locks. Locks left lying around because of system crashes had to be manually removed (normally in a system startup command script). Finally, processes running as system administrator are always permitted to create files, so were forced to use a different mechanism. While it is possible to get around all these problems, the solutions are not straight forward, so a mechanism for locking files has been added.

The most general schemes allow multiple processes to concurrently update a file. Several of these techniques are discussed in [Peterson83]. A simpler technique is to serialize access to a file with locks. To attain reasonable efficiency, certain applications require the ability to lock pieces of a file. Locking down to the byte level has been implemented in the Onyx file system by [Bass81]. However, for the standard system applications, a mechanism that locks at the granularity of a file is sufficient.

Locking schemes fall into two classes, those using hard locks and those using advisory locks. The primary difference between advisory locks and hard locks is the extent of enforcement. A hard lock is always enforced when a program tries to access a file; an advisory lock is only applied when it is requested by a program. Thus advisory locks are only effective when all programs accessing a file use the locking scheme. With hard locks there must be some override policy implemented in the kernel. With advisory locks the policy is left to the user programs. In the UNIX system, programs with system administrator privilege are allowed override any protection scheme. Because many of the programs that need to use locks must also run as the system administrator, we chose to implement advisory locks rather than create an additional protection scheme that was inconsistent with the UNIX philosophy or could not be used by system administration programs.

The file locking facilities allow cooperating programs to apply advisory *shared* or *exclusive* locks on files. Only one process may have an exclusive lock on a file while multiple shared locks may be present. Both shared and exclusive locks cannot be present on a file at the same time. If any lock is requested when another process holds an exclusive lock, or an exclusive lock is requested when another process holds any lock, the lock request will block until the lock can be obtained. Because shared and exclusive locks are advisory only, even if a process has obtained a lock on a file, another process may access the file.

Locks are applied or removed only on open files. This means that locks can be manipulated without needing to close and reopen a file. This is useful, for example, when a process wishes to apply a shared lock, read some information and determine whether an update is required, then apply an exclusive lock and update the file.

A request for a lock will cause a process to block if the lock can not be immediately obtained. In certain instances this is unsatisfactory. For example, a process that wants only to check if a lock is present would require a separate mechanism to find out this information. Consequently, a process may specify that its locking request should return with an error if a lock can not be immediately obtained. Being able to conditionally request a lock is useful to "daemon" processes that wish to service a spooling area. If the first instance of the daemon locks the directory where spooling takes place, later daemon processes can easily check to see if an active daemon exists. Since locks exist only while the locking processes exist, lock files can never be left active after the processes exit or if the system crashes.

Almost no deadlock detection is attempted. The only deadlock detection done by the system is that the file to which a lock is applied must not already have a lock of the same type (i.e. the second of two successive calls to apply a lock of the same type will fail).

Symbolic Links

The traditional UNIX file system allows multiple directory entries in the same file system to reference a single file. Each directory entry "links" a file's name to an inode and its contents. The link concept is fundamental; inodes do not reside in directories, but exist separately and are referenced by links. When all the links to an inode are removed, the inode is deallocated. This style of referencing an inode does not allow references across physical file systems, nor does it support inter-machine linkage. To avoid these limitations *symbolic links* similar to the scheme used by Multics [Feiertag71] have been added.

A symbolic link is implemented as a file that contains a pathname. When the system encounters a symbolic link while interpreting a component of a pathname, the contents of the symbolic link is prepended to the rest of the pathname, and this name is interpreted to yield the resulting pathname. In UNIX, pathnames are specified relative to the root of the file system hierarchy, or relative to a process's current working directory. Pathnames specified relative to the root are called absolute pathnames. Pathnames specified relative to the current working directory are termed relative pathnames. If a symbolic link contains an absolute pathname, the absolute pathname is used, otherwise the contents of the symbolic link is evaluated relative to the location of the link in the file hierarchy.

Normally programs do not want to be aware that there is a symbolic link in a pathname that they are using. However certain system utilities must be able to detect and manipulate symbolic links. Three new system calls provide the ability to detect, read, and write symbolic links; seven system utilities required changes to use these calls.

In future Berkeley software distributions it may be possible to reference file systems located on remote machines using pathnames. When this occurs, it will be possible to create symbolic links that span machines.

Rename

Programs that create a new version of an existing file typically create the new version as a temporary file and then rename the temporary file with the name of the target file. In the old UNIX file system renaming required three calls to the system. If a program were interrupted or the system crashed between these calls, the target file could be left with only its temporary name. To eliminate this possibility the *rename* system call has been added. The rename call does the rename operation in a fashion that guarantees the existence of the target name.

Rename works both on data files and directories. When renaming directories, the system must do special validation checks to insure that the directory tree structure is not corrupted by the creation of loops or inaccessible directories. Such corruption would occur if a parent directory were moved into one of its descendants. The validation check requires tracing the descendents of the target directory to insure that it does not include the directory being moved.

Quotas

The UNIX system has traditionally attempted to share all available resources to the greatest extent possible. Thus any single user can allocate all the available space in the file system. In certain environments this is unacceptable. Consequently, a quota mechanism has been added for restricting the amount of file system resources that a user can obtain. The quota mechanism sets limits on both the number of inodes and the number of disk blocks that a user may allocate. A separate quota can be set for each user on each file system. Resources are given both a hard and a soft limit. When a program exceeds a soft limit, a warning is printed on the users terminal; the offending program is not terminated unless it exceeds its hard limit. The idea is that users should stay below their soft limit between login sessions, but they may use more resources while they are actively working. To encourage this behavior, users are warned when logging in if they are over any of their soft limits. If users fails to correct the problem for too many login sessions, they are eventually reprimanded by having their soft limit enforced as their hard limit.

Acknowledgements

We thank Robert Elz for his ongoing interest in the new file system, and for adding disk quotas in a rational and efficient manner. We also acknowledge Dennis Ritchie for his suggestions on the appropriate modifications to the user interface. We appreciate Michael Powell's explanations on how the DEMOS file system worked; many of his ideas were used in this implementation. Special commendation goes to Peter Kessler and Robert Henry for acting like real users during the early debugging stage when file systems were less stable than they should have been. The criticisms and suggestions by the reviews contributed significantly to the coherence of the paper. 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.

References

Almes78

Almes, G., and Robertson, G.
"An Extensible File System for Hydra"
Proceedings of the Third International Conference on Software Engineering,
IEEE, May 1978.

Bass81

Bass, J.
"Implementation Description for File Locking",
Onyx Systems Inc, 73 E. Trimble Rd, San Jose, CA 95131
Jan 1981.

Feiertag71

Feiertag, R. J. and Organick, E. I.,
"The Multics Input-Output System",
Proceedings of the Third Symposium on Operating Systems Principles,
ACM, Oct 1971. pp 35-41

Ferrin82a

Ferrin, T.E.,
"Performance and Robustness Improvements in Version 7 UNIX",
Computer Graphics Laboratory Technical Report 2,
School of Pharmacy, University of California,
San Francisco, January 1982.
Presented at the 1982 Winter Usenix Conference, Santa Monica, California.

Ferrin82b

Ferrin, T.E.,
"Performance Issuses of VMUNIX Revisited",
;login: (The Usenix Association Newsletter), Vol 7, #5,
November 1982. pp 3-6

Kridle83

Kridle, R., and McKusick, M.,
"Performance Effects of Disk Subsystem Choices for VAX Systems Running 4.2BSD UNIX",
Computer Systems Research Group, Dept of EECS,
Berkeley, CA 94720,
Technical Report #8.

Kowalski78

Kowalski, T.
"FSCK - The UNIX System Check Program",
Bell Laboratory, Murray Hill, NJ 07974. March 1978

Knuth75

Kunth, D.
"The Art of Computer Programming",
Volume 3 - Sorting and Searching,
Addison-Wesley Publishing Company Inc, Reading, Mass, 1975.
pp 506-549

Maruyama76

Maruyama, K., and Smith, S.
"Optimal reorganization of Distributed Space Disk Files",
CACM, 19, 11. Nov 1976. pp 634-642

Nevalainen77

Nevalainen, O., Vesterinen, M.
"Determining Blocking Factors for Sequential Files by Heuristic Methods",
The Computer Journal, 20, 3. Aug 1977. pp 245-247

Pechura83

Pechura, M., and Schoeffler, J.
"Estimating File Access Time of Floppy Disks",
CACM, 26, 10. Oct 1983. pp 754-763

Peterson83

Peterson, G.
"Concurrent Reading While Writing",
ACM Transactions on Programming Languages and Systems,
ACM, 5, 1. Jan 1983. pp 46-55

Powell79

Powell, M.
"The DEMOS File System",
Proceedings of the Sixth Symposium on Operating Systems Principles,
ACM, Nov 1977. pp 33-42

Ritchie74

Ritchie, D. M. and Thompson, K.,
"The UNIX Time-Sharing System",
CACM 17, 7. July 1974. pp 365-375

Smith81a

Smith, A.
"Input/Output Optimization and Disk Architectures: A Survey",
Performance and Evaluation 1. Jan 1981. pp 104-117

Smith81b

Smith, A.
"Bibliography on File and I/O System Optimization and Related Topics",
Operating Systems Review, 15, 4. Oct 1981. pp 39-54

Symbolics81

"Symbolics File System",
Symbolics Inc, 9600 DeSoto Ave, Chatsworth, CA 91311
Aug 1981.

Thompson78

Thompson, K.
"UNIX Implementation",
Bell System Technical Journal, 57, 6, part 2. pp 1931-1946
July-August 1978.

Thompson80

Thompson, M.
"Spice File System",
Carnegie-Mellon University,
Department of Computer Science, Pittsburg, PA 15213
#CMU-CS-80, Sept 1980.

Trivedi80

Trivedi, K.
"Optimal Selection of CPU Speed, Device Capabilities,
and File Assignments",
Journal of the ACM, 27, 3. July 1980. pp 457-473

White80

White, R. M.
"Disk Storage Technology",
Scientific American, 243(2), August 1980.

Fsck – The UNIX† File System Check Program

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 and 4.3BSD 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.

Introduction

This document reflects the use of **fsck** with the 4.2BSD and 4.3BSD 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.

Overview of the File System

Superblock

A file system is described by its *super-block*. The super-block is built when the file system is created (*newfs*(1FFS)) 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 actual number may vary from system to system, but is usually in the range 5-13. 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 (the triple indirect block is never needed in practice).

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.

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.

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 *i*th cylinder group. In this way, the redundant information spirals down into the pack; any single track, cylinder, or platter can be lost without losing all copies of the super-blocks. Except for the first cylinder group, the space between the beginning of the cylinder group and the beginning of the cylinder group information stores data.

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.

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,* or by manual operator intervention with the *sync*(1M) 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.

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.

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

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.

Free Block Checking

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

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

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.

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.

Inode Links

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

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

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

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

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

Fsck checks the range of each block number claimed by an inode. If the block number is lower than the first data block in the file system, or greater than the last data block, then the block number is a *bad block number*. Many bad blocks in an inode are usually caused by an indirect block that was not written to the file system, a condition which can only occur if there has been a hardware failure. If an inode contains bad block numbers, **fsck** prompts the operator to clear it.

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.

Checking the Data Associated With an Inode

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

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

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

The directory inode number entry for "**.**" must be the first entry in the directory data block. The inode number for "**.**" must reference itself; e.g., it must equal the inode number for the directory data block. The directory inode number entry for "**..**" must be the second entry in the directory data block. Its value must equal the inode number for the parent of the directory entry (or the inode number of the directory data block if the directory is the root directory). If the directory inode numbers are incorrect, **fsck** will replace them with the correct values. If there are multiple hard links to a directory, the first one encountered is considered the real parent to which "**..**" should point; *fsck* recommends deletion for the subsequently discovered names.

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.

McKusick84

McKusick, M., Joy, W., Leffler, S., and Fabry, R.
A Fast File System for UNIX,
ACM Transactions on Computer Systems 2, 3.
pp. 181-197, August 1984.

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.

Fsck Error Conditions

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.

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 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**(1M) 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: ignore this error condition. ignore this file system and continues checking the next file system given.

UNDEFINED OPTIMIZATION IN SUPERBLOCK (SET TO DEFAULT)
The superblock optimization parameter is neither OPT_TIME nor OPT_SPACE. Possible responses to the SET TO DEFAULT prompt are: The superblock is set to request optimization to minimize running time of the system. (If optimization to minimize disk space utilization is desired, it can be set using **tunefs**(1M).) ignore this error condition.

IMPOSSIBLE MINFREE=*D* IN SUPERBLOCK (SET TO DEFAULT)
The superblock minimum space percentage is greater than 99% or less then 0%. Possible responses to the SET TO DEFAULT prompt are: The minfree parameter is set to 10%. (If some other percentage is desired, it can be set using **tunefs**(1M).) ignore this error condition.

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 S.spLY NEEDED INFORMATION; SEE fsck(1M).
The super block has been corrupted. An alternative super block must be selected from among those listed by *newfs* (1FFS) 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: 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". 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: attempt to continue to run the file system check. It will retry the read and print out the message:

THE FOLLOWING SECTORS COULD NOT BE READ: *N*
where *N* indicates the sectors that could not be read. If **fsck** ever tries to write back one of the blocks on which the read failed it will print the message:
WRITING ZERO'ED BLOCK *N* **TO DISK**
where *N* indicates the sector that was written with zero's. If the disk is experiencing hardware problems, 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". 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; check the write protect lock on the drive. If that is not the problem, see a guru. Possible responses to the CONTINUE prompt are: attempt to continue to run the file system check. The write operation will be retried with the failed blocks indicated by the message:
THE FOLLOWING SECTORS COULD NOT BE WRITTEN: *N*
where *N* indicates the sectors that could not be written. If the disk is experiencing hardware problems, 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". terminate the program.

bad inode number DDD to ginode
An internal error has attempted to read non-existent inode *DDD*. This error causes **fsck** to exit. See a guru.

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** and **PARTIALLY TRUNCATED INODE** are fatal if the file system is being preen'ed.

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: 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. ignore this error condition.

PARTIALLY TRUNCATED INODE I=*I* (SALVAGE)
Fsck has found inode *I* whose size is shorter than the number of blocks allocated to it. This condition should only occur if the system crashes while in the midst of truncating a file. When preen'ing the file system, **fsck** completes the truncation to the specified size. Possible responses to SALVAGE are: complete the truncation to the size specified in the inode. ignore this error condition.

LINK COUNT TABLE OVERFLOW (CONTINUE)
An internal table for **fsck** containing allocated inodes with a link count of zero cannot allocate more memory. Increase the virtual memory for **fsck**. Possible responses to the CONTINUE prompt are: 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. 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 (see next paragraph) 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: 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. terminate the program.

BAD STATE DDD TO BLKERR
An internal error has scrambled **fsck**'s state map to have the impossible value *DDD*. **Fsck** exits immediately. See a guru.

B* DUP I=*I
Inode *I* contains block number *B* that 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: 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. terminate the program.

DUP TABLE OVERFLOW (CONTINUE)
An internal table in **fsck** containing duplicate block numbers cannot allocate any more space. Increase the amount of virtual memory available to **fsck**. Possible responses to the CONTINUE prompt are: 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. terminate the program.

PARTIALLY ALLOCATED INODE I=*I* (CLEAR)
Inode *I* is neither allocated nor unallocated. Possible responses to the CLEAR prompt are: de-allocate inode *I* by zeroing its contents. 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: replace the block count of inode *I* with *Y*. ignore this error condition.

Phase 1B: Rescan for More Dups

When a duplicate block is found in the file system, the file system is rescanned to find the inode that 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.

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, and directory integrity checks. All errors in this phase are fatal if the file system is being preen'ed, except for directories not being a multiple of the blocks size and extraneous hard links.

ROOT INODE UNALLOCATED (ALLOCATE)
The root inode (usually inode number 2) has no allocate mode bits. This should never happen. Possible responses to the ALLOCATE prompt are: allocate inode 2 as the root inode. The files and directories usually found in the root will be recovered in Phase 3 and put into *lost+found*. If the attempt to allocate the root fails, **fsck** will exit with the message:
CANNOT ALLOCATE ROOT INODE. fsck will exit.

ROOT INODE NOT DIRECTORY (REALLOCATE)
The root inode (usually inode number 2) is not directory inode type. Possible responses to the REALLOCATE prompt are: clear the existing contents of the root inode and reallocate it. The files and directories usually found in the root will be recovered in Phase 3 and put into *lost+found*. If the attempt to allocate the root fails, **fsck** will exit with the message:
CANNOT ALLOCATE ROOT INODE. fsck will then prompt with **FIX** Possible responses to the FIX prompt are: replace the root inode's type to be a directory. If the root inode's data blocks are not directory blocks, many error conditions will be produced. terminate the program.

DUPS/BAD IN ROOT INODE (REALLOCATE)
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 REALLOCATE prompt are: clear the existing contents of the root inode and reallocate it. The files and directories usually found in the root will be recovered in Phase 3 and put into *lost+found*. If the attempt to allocate the root fails, **fsck** will exit with the message: **CANNOT ALLOCATE ROOT INODE. fsck** will then prompt with **CONTINUE**. Possible responses to the CONTINUE prompt are: 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 many other error conditions. terminate the program.

NAME TOO LONG *F*
An excessively long path name has been found. This usually indicates 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).

I OUT OF RANGE I=*I* NAME=*F* (REMOVE)
A directory entry *F* has an inode number *I* that is greater than the end of the inode list. Possible responses to the REMOVE prompt are: the directory entry *F* is removed. ignore this error condition.

UNALLOCATED I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* *type*=*F* (REMOVE)
A directory or file entry *F* points to an unallocated inode *I*. The owner *O*, mode *M*, size *S*, modify time *T*, and name *F* are printed. Possible responses to the REMOVE prompt are: the directory entry *F* is removed. ignore this error condition.

DUP/BAD I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* *type*=*F* (REMOVE)
Phase 1 or Phase 1b have found duplicate blocks or bad blocks associated with directory or file entry *F*, 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: the directory entry *F* is removed. 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: the directory entry *F* is removed; this will always invoke the BAD/DUP error condition in Phase 4. 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: increase the size of the directory to the minimum directory size. ignore this directory.

DIRECTORY *F* LENGTH *S* NOT MULTIPLE OF *B* (ADJUST)
A directory *F* has been found with size *S* that is not a multiple of the directory blocksize *B*. Possible responses to the ADJUST prompt are: the length is rounded up to the appropriate block size. This error can occur on 4.2BSD file systems. Thus when preen'ing the file system only a warning is printed and the directory is adjusted. ignore the error condition.

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: throw away all entries up to the next directory boundary (usually 512-byte) boundary. This drastic action can throw away up to 42 entries, and should be taken only after other recovery efforts have failed. skip up to the next directory 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: change the inode number for '.' to be equal to *I*. 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: build an entry for '.' with inode number equal to *I*. 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: remove the extra entry for '.'. 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: change the inode number for '..' to be equal to the parent of *I* ("**..**" in the root inode points to itself). 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: build an entry for '..' with inode number equal to the parent of *I* ("**..**" in the root inode points to itself). 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. The file system should be mounted and the second entry in the directory moved elsewhere. The file system should then be unmounted and **fsck** should be run again.

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: remove the extra entry for '..'. leave the directory unchanged.

***N* IS AN EXTRANEOUS HARD LINK TO A DIRECTORY *D* (REMOVE)**
Fsck has found a hard link, *N*, to a directory, *D*. When preen'ing the extraneous links are ignored. Possible responses to the REMOVE prompt are: delete the extraneous entry, *N*. ignore the error condition.

BAD INODE *S* TO DESCEND
An internal error has caused an impossible state *S* to be passed to the routine that descends the file system directory structure. **Fsck** exits. See a guru.

BAD RETURN STATE *S* FROM DESCEND
An internal error has caused an impossible state *S* to be returned from the routine that descends the file system directory structure. **Fsck** exits. See a guru.

BAD STATE *S* FOR ROOT INODE
An internal error has caused an impossible state *S* to be assigned to the root inode. **Fsck** exits. See a guru.

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: 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. ignore this error condition. This will always invoke the UNREF error condition in Phase 4.

NO lost+found DIRECTORY (CREATE)
There is no *lost+found* directory in the root directory of the file system; When preen'ing **fsck** tries to create a *lost+found* directory. Possible responses to the CREATE prompt are: create a *lost+found* directory in the root of the file system. This may raise the message:
NO SPACE LEFT IN / (EXPAND)
See below for the possible responses. Inability to create a *lost+found* directory generates the message:
SORRY. CANNOT CREATE lost+found DIRECTORY
and aborts the attempt to linkup the lost inode. This will always invoke the UNREF

error condition in Phase 4. abort the attempt to linkup the lost inode. This will always invoke the UNREF error condition in Phase 4.

lost+found IS NOT A DIRECTORY (REALLOCATE)
The entry for *lost+found* is not a directory. Possible responses to the REALLOCATE prompt are: allocate a directory inode, and change *lost+found* to reference it. The previous inode reference by the *lost+found* name is not cleared. Thus it will either be reclaimed as an UNREF'ed inode or have its link count ADJUST'ed later in this Phase. Inability to create a *lost+found* directory generates the message:
SORRY. CANNOT CREATE lost+found DIRECTORY
and aborts the attempt to linkup the lost inode. This will always invoke the UNREF error condition in Phase 4. abort the attempt to linkup the lost inode. This will always invoke the UNREF error condition in Phase 4.

NO SPACE LEFT IN /lost+found (EXPAND)
There is no space to add another entry to the *lost+found* directory in the root directory of the file system. When preen'ing the *lost+found* directory is expanded. Possible responses to the EXPAND prompt are: the *lost+found* directory is expanded to make room for the new entry. If the attempted expansion fails **fsck** prints the message:
SORRY. NO SPACE IN lost+found DIRECTORY
and aborts the attempt to linkup the lost inode. This will always invoke the UNREF error condition in Phase 4. Clean out unnecessary entries in *lost+found*. This error is fatal if the file system is being preen'ed. abort the attempt to linkup the lost inode. This will always invoke the UNREF error condition in Phase 4.

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.

DIRECTORY *F* LENGTH *S* NOT MULTIPLE OF *B* (ADJUST)
A directory *F* has been found with size *S* that is not a multiple of the directory blocksize *B* (this can reoccur in Phase 3 if it is not adjusted in Phase 2). Possible responses to the ADJUST prompt are: the length is rounded up to the appropriate block size. This error can occur on 4.2BSD file systems. Thus when preen'ing the file system only a warning is printed and the directory is adjusted. ignore the error condition.

BAD INODE *S* TO DESCEND
An internal error has caused an impossible state *S* to be passed to the routine that descends the file system directory structure. **Fsck** exits. See a guru.

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, and bad or duplicate blocks in files, symbolic links, and directories. 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: 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*. 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: deallocate the inode mentioned in the immediately previous error condition by zeroing its contents. ignore this error condition.

NO lost+found DIRECTORY (CREATE)
There is no *lost+found* directory in the root directory of the file system; When preen'ing **fsck** tries to create a *lost+found* directory. Possible responses to the CREATE prompt are: create a *lost+found* directory in the root of the file system. This may raise the message:
NO SPACE LEFT IN / (EXPAND)
See below for the possible responses. Inability to create a *lost+found* directory generates the message:
SORRY. CANNOT CREATE lost+found DIRECTORY
and aborts the attempt to linkup the lost inode. This will always invoke the UNREF error condition in Phase 4. abort the attempt to linkup the lost inode. This will always invoke the UNREF error condition in Phase 4.

lost+found IS NOT A DIRECTORY (REALLOCATE)
The entry for *lost+found* is not a directory. Possible responses to the REALLOCATE prompt are: allocate a directory inode, and change *lost+found* to reference it. The previous inode reference by the *lost+found* name is not cleared. Thus it will either be reclaimed as an UNREF'ed inode or have its link count ADJUST'ed later in this Phase. Inability to create a *lost+found* directory generates the message:
SORRY. CANNOT CREATE lost+found DIRECTORY
and aborts the attempt to linkup the lost inode. This will always invoke the UNREF error condition in Phase 4. abort the attempt to linkup the lost inode. This will always invoke the UNREF error condition in Phase 4.

NO SPACE LEFT IN /lost+found (EXPAND)
There is no space to add another entry to the *lost+found* directory in the root directory of the file system. When preen'ing the *lost+found* directory is expanded. Possible responses to the EXPAND prompt are: the *lost+found* directory is expanded to make room for the new entry. If the attempted expansion fails **fsck** prints the message:
SORRY. NO SPACE IN lost+found DIRECTORY
and aborts the attempt to linkup the lost inode. This will always invoke the UNREF error condition in Phase 4. Clean out unnecessary entries in *lost+found*. This error is fatal if the file system is being preen'ed. abort the attempt to linkup the lost inode. This will always invoke the UNREF error condition in Phase 4.

LINK COUNT *type* I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* COUNT=*X* SHOULD BE *Y* (ADJUST)
The link count for inode *I*, 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 unless the number of references is increasing, a condition that should never occur unless precipitated by a hardware failure. When the number of references is increasing under preen mode, **fsck** exits with the message:
LINK COUNT INCREASING Possible responses to the ADJUST prompt are: replace the link count of file inode *I* with *Y*. ignore this error condition.

UNREF *type* **I=***I* **OWNER=***O* **MODE=***M* **SIZE=***S* **MTIME=***T* **(CLEAR)**
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, 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: de-allocate inode *I* by zeroing its contents. ignore this error condition.

BAD/DUP *type* **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 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: de-allocate inode *I* by zeroing its contents. ignore this error condition.

Phase 5 - Check Cyl groups

This phase concerns itself with the free-block and used-inode 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. It also lists error conditions resulting from free inodes in the used-inode maps, allocated inodes missing from used-inode maps, and the total used-inode 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.

BLK(S) MISSING IN BIT MAPS (SALVAGE)
A cylinder group block map is missing some free blocks. During preen'ing the maps are reconstructed. Possible responses to the SALVAGE prompt are: reconstruct the free block map. ignore this error condition.

SUMMARY INFORMATION BAD (SALVAGE)
The summary information was found to be incorrect. When preen'ing, the summary information is recomputed. Possible responses to the SALVAGE prompt are: reconstruct the summary information. ignore this error condition.

FREE BLK COUNT(S) WRONG IN SUPERBLOCK (SALVAGE)
The superblock free block information was found to be incorrect. When preen'ing, the superblock free block information is recomputed. Possible responses to the SALVAGE prompt are: reconstruct the superblock free block information. ignore this error condition.

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 standard auto-reboot script distributed with 4.3BSD 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.

Introduction

Appendix C describes directories and files of interest to a system administrator.

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

bin	Directory containing public commands.
boot	Directory containing configurable object files created by the **/etc/mkboot**(1M) program.
dev	Directory containing special files that define all of the devices on the system.
etc	Directory containing administrative programs and tables.
lib	Directory containing public libraries.
lost+found	Directory used by **fsck**(1M) to save disconnected files.
tmp	Directory used for temporary files.
usr	Directory used to mount the **/usr** file system.

The following configurator files and directories are important in the administration of the RISComputer:

- **/etc/checklist**
- **/etc/cshrc**
- **/etc/fstab**
- **/etc/gettydefs**
- **/etc/group**
- **/etc/hosts**
- **/etc/init.d Directory**
- **/etc/inittab**
- **/etc/local_hostname**
- **/etc/reconfig/master.d**
- **/etc/motd**
- **/etc/passwd**
- **/etc/profile**
- **/etc/rc0**
- **/etc/rc0.d Directory**
- **/etc/rc2**
- **/etc/rc2.d Directory**
- **/etc/rc.d Directory**
- **/etc/rc3**
- **/etc/rc3.d Directory**

- **/etc/shutdown**
- **/etc/stdcshrc**
- **/etc/stdprofile**
- **/etc/TZ**
- **/etc/utmp**
- **/etc/wtmp**
- **/usr/lib/cron/log**
- **/usr/lib/spell/spellhist**
- **/usr/news Directory**
- **/usr/spool/cron/crontabs Directory**

Each of these files is briefly described in this appendix.

Files

/etc/checklist

The **/etc/checklist** file is used to define a default list of file system devices to be checked for consistency by **/etc/fsck** and **/etc/ncheck**. The character (raw) device partition for the file system should be identified. The devices listed normally correspond to those mounted when the system is in the multi-user mode (run level 2). Remember that with the exception of **root**, a file system must be unmounted to be checked. Therefore, the **checklist** file is a convenience for use when in the single-user mode of operation with only the **root** file system mounted. When the system is delivered, this file contains only the root.

/etc/cshrc

This is the initial environment settings for users of the C-shell. It is the basic set of values assigned to all C-shell users on the system. A typical **/etc/cshrc** looks like this:

```
# default settings for all users

#

umask 022
set path = (~/bin /usr/net /bin /usr/bin /usr/ucb .)

cat -s /etc/motd
if ($?LOGNAME == 0) then
      echo "$0": LOGNAME: parameter not set
      exit 1
else
      set mail=/usr/mail/$LOGNAME
endif
if ( { /bin/mail -e } ) then
      echo 'You have mail.'
endif
if ( $LOGNAME != root ) then
      news -n
endif
```

Figure C-1: Typical **/etc/cshrc** File

/etc/fstab

The **/etc/fstab** file is used as an argument to the **/etc/mountall** command. The **fstab** file specifies the file system(s) to be mounted by **/etc/mountall** and remote file system(s) to be mounted by **/etc/rmountall**. A typical **/etc/fstab** file is shown in Figure C-2. The format of the file is the block device name followed by the mount point name. (See the **mountall**(1M) manual page in the *System Administrator's Reference*

Manual for additional information.)

```
/dev/root    /      ffs rw 0 0
/dev/usr     /usr   ffs rw 0 0
/dev/dsk/ips0d0s7 none        swap rw,noauto 0 0
yoda:/cmos   /cmos            nfs rw,bg,soft,timeo=20 0 0
yoda:/usr    /yoda/usr        nfs rw,bg,soft,timeo=20 0 0
```

Figure C-2: Typical **/etc/fstab** File

/etc/gettydefs

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

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

Each line entry in the **gettydefs** file is followed by a blank line. (Refer to the **gettydefs**(4) manual page in the *Programmer's Reference Manual* for complete information.) Figure C-3 shows a typical RISComputer **/etc/gettydefs** file.

```
console# B9600 CLOCAL # B9600 CLOCAL SANE TAB3 #\r\fn\n$HOSTNAME Console \
                    login: #console

co_9600# B9600 CLOCAL # B9600 CLOCAL SANE TAB3 #\r\n\n$HOSTNAME login: #co_4800

co_4800# B4800 CLOCAL # B4800 CLOCAL SANE TAB3 #\r\n\n$HOSTNAME login: #co_2400

co_2400# B2400 CLOCAL # B2400 CLOCAL SANE TAB3 #\r\n\n$HOSTNAME login: #co_1200

co_1200# B1200 CLOCAL # B1200 CLOCAL SANE TAB3 #\r\n\n$HOSTNAME login: #co_300

co_300# B300 # B300 SANE TAB3 #\r\n\n$HOSTNAME login: #co_9600

dx_19200# B19200 CLOCAL # B19200 CLOCAL SANE TAB3 #\r\n\n$HOSTNAME \
                    login: #dx_19200

dx_9600# B9600 CLOCAL # B9600 CLOCAL SANE TAB3 #\r\n\n$HOSTNAME login: #dx_9600

dx_4800# B4800 CLOCAL # B4800 CLOCAL SANE TAB3 #\r\n\n$HOSTNAME login: #dx_4800

dx_2400# B2400 CLOCAL # B2400 CLOCAL SANE TAB3 #\r\n\n$HOSTNAME login: #dx_2400

dx_1200# B1200 CLOCAL # B1200 CLOCAL SANE TAB3 #\r\n\n$HOSTNAME login: #dx_1200

du_9600# B9600 # B9600 SANE TAB3 HUPCL #\r\n\n$HOSTNAME login: #du_4800

du_4800# B4800 # B4800 SANE TAB3 HUPCL #\r\n\n$HOSTNAME login: #du_2400

du_2400# B2400 # B2400 SANE TAB3 HUPCL #\r\n\n$HOSTNAME login: #du_1200

du_1200# B1200 # B1200 SANE TAB3 HUPCL #\r\n\n$HOSTNAME login: #du_300

du_300# B300 # B300 SANE TAB3 HUPCL #\r\n\n$HOSTNAME login: #du_9600
```

Figure C-3: Typical **gettydefs** File

/etc/group

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

group name:password:group id:login names

Explanations for these fields are as follows:

group name	The first field defines the group name. The group name is from three to six characters long. The first character is alphabetic. The rest of the characters are alphanumeric. No uppercase characters appear.

password	The second field contains the encrypted group password. The encrypted group password contains 13 bytes (characters). The actual password is limited to a maximum of 8 bytes. The encrypted password can be followed by a comma and up to 4 more bytes of password aging information. The use of group passwords is discouraged.
group id	The third field contains the group identification number, which must be between 0 and 60,000. Group identification numbers 0 through 99 are reserved; 0 indicates the super-user (root). Commas are not entered in this field.
login names	The fourth field contains a list of all login names in the group. Names in the list are separated by commas. The names listed may use the **/etc/newgrp** command to become a member of the group.

Figure C-4 shows a typical RISComputer **/etc/group** file.

```
root::0:root
other::1:root
bin::2:root,bin,daemon
sys::3:root,bin,sys,adm
adm::4:root,adm,daemon
mail::6:root
rje::8:rje
daemon::12:root,daemon
progs::120:greg,tom,dvv,ads,memo
staff::121:mar,nnr,bob
```

Figure C-4: Typical **/etc/group** File

/etc/hosts Directory

The /etc/hosts directory is used by the networking software to determine the name and addresses of accessible hosts.

```
# NET : NET-ADDR : NETNAME :
# GATEWAY : ADDR, ADDR : NAME : CPUTYPE : OPSYS : PROTOCOLS :
# HOST : ADDR, ALTERNATE-ADDR (if any): HOSTNAME,NICKNAME : CPUTYPE :

97.4.0.1        agate
97.4.0.3        diamond di
97.4.0.5        jade
```

Figure C-5: Typical **/etc/hosts** File

/etc/init.d Directory

The **/etc/init.d** directory contains executable files used in upward and downward transitions to all system run levels. These files are linked to files beginning with **S** (start) or **K** (stop) in **/etc/rc*n*.d**, where *n* is the appropriate run level. Files are not executed from this directory. They are only executed from **/etc/rc*n*.d** directories.

/etc/inittab

The **/etc/inittab** file contains instructions for the **/etc/init** command. The instructions define the processes that are to be created or terminated for each initialization state. Initialization states are called run levels or run-states.

run levels

By convention, run level 1 (or S or s) is single-user mode; run levels 2 and 3 are multi-user modes. Chapter 3, "System Levels", summarizes the various run levels and describes their uses. (See the **inittab**(4) manual page in the *Programmer's Reference Manual* for additional information.) Figure C-6 shows a typical RISComputer **/etc/inittab** file. The typical entry is a series of fields separated by a colon (:):

identification:run-state:action:process

Explanations for these fields are as follows:

identification	The identification field is a one- or two-character identifier for the line entry. The identifier is unique for a line.
run-state	The run-state defines the run level in which the entry is to be processed.
action	The action field defines how **/etc/init** treats the process field. (Refer to the **inittab**(4) manual page in the *Programmer's Reference Manual* for complete information.)
process	The process field defines the shell command that is to be executed.

```
#  Copyright (c) 1984 AT&T
#    All Rights Reserved

#  THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF AT&T
#  The copyright notice above does not evidence any
#  actual or intended publication of such source code.

#ident   "$ Header $"

#
# Field #2 indicates the system's default run level.
# (X for unspecified.)
#
is:2:initdefault:

#
# Boot time system initialization.
#
fs::sysinit:/etc/bcheckrc >/dev/syscon 2>&1
#  Check (& fsck) root fs.
mt::sysinit:/etc/brc >/dev/syscon 2>&1
#  Initialize /etc/mtab.

#
# Run level changes.
#
s1:1:wait:/etc/shutdown -y -iS -g0 >/dev/syscon <&1 2>&1
s2:23:wait:/etc/rc2 >/dev/syscon <&1 2>&1
s3:3:wait:/etc/rc3 >/dev/syscon <&1 2>&1

#
# System shut down.
#
#   telinit 0 = Shutdown and halt.
#   telinit 6 = Shutdown and then reboot.
#   telinit 5 = Like 6, but asks operator which unix
#     to boot. (DOESN'T WORK.)
#

s0:056:wait:/etc/rc0 >/dev/syscon <&1 2>&1
#  Always run rc0.
of:0:wait:/etc/uadmin 2 0 >/dev/syscon <&1 2>&1
fw:5:wait:/etc/uadmin 2 2 >/dev/syscon <&1 2>&1
RB:6:wait:echo "The system is being restarted." \
      >/dev/syscon <&1 2>&1
rb:6:wait:/etc/uadmin 2 1 >/dev/syscon <&1 2>&1
#
# Gettys
#
# Enabled if field #3 is "respawn";
# Disabled if field #3 is "off".
#
# The h* entries are for M500/800/1000/2000.  The d* entries
# are for M120 with the DIGI board.
```

```
#
co:234:respawn:/etc/getty console console none LDISC0
   # (console == tty0)
t1:234:off:/etc/getty tty1 co_9600 none LDISC0
t2:234:off:/etc/getty tty2 co_9600 none LDISC0
t3:234:off:/etc/getty tty3 co_9600 none LDISC0
h0:234:off:/etc/getty ttyh0 dx_19200 none LDISC0
h1:234:off:/etc/getty ttyh1 dx_19200 none LDISC0
h2:234:off:/etc/getty ttyh2 dx_19200 none LDISC0
h3:234:off:/etc/getty ttyh3 dx_19200 none LDISC0
h4:234:off:/etc/getty ttyh4 dx_19200 none LDISC0
    .
    .
    .
```

Figure C-6: Typical **/etc/inittab** File

/usr/reconfig/master.d Directory

The **/usr/reconfig/master.d** directory contains files that define the configuration of hardware devices, software drivers, system parameters and aliases. The files are used by **/etc/mkboot** to obtain device information for the generation of device driver and configurable module files. The first step in reconfiguring the system to run with different tunable parameters is to edit the appropriate files in the **/usr/reconfig/master.d** directory. (Refer to the **master**(4) manual page in the *Programmer's Reference Manual* for additional information.)

/etc/motd

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

/etc/passwd

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

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

Explanations for these fields are as follows:

login name	The first field defines the login name. The login name is from three to six characters long. The first character is alphabetic. The rest of the characters are alphanumeric. No uppercase characters appear.

passwd	The second field contains the encrypted login password. The encrypted login password contains 13 bytes (characters). The actual password is limited to a maximum of 8 bytes. The encrypted password can be followed by a comma and up to 4 more bytes of password aging information.
user id	The third field contains the user identification number, which must be between 0 and 60,000. Group identification numbers 0 through 99 are reserved; 0 indicates the super-user (root). Commas are not entered in this field.
group id	The fourth field contains the group identification number, which must be between 0 and 60,000. Group identification numbers 0 through 99 are reserved; 0 indicates the super-user (root). Commas are not entered in this field.
account	The fifth field is used by accounting programs. This field typically contains the user name, department number, and bin number.
login directory	The sixth field defines the full path name of the login directory.
program	The seventh field defines the program to be executed after login. If it is null, the shell (**/bin/sh**) is invoked.

Figure C-7 shows a typical **/etc/passwd** file. (See the **passwd**(4) manual page in the *Programmer's Reference Manual* for additional information.)

```
root::0:1:0000-Admin(0000):/:
daemon:*:1:1:0000-Admin(0000):/:
bin:*:2:2:0000-Admin(0000):/bin:
sys:*:3:3:0000-Admin(0000):/usr/src:
adm:*:4:4:0000-Admin(0000):/usr/adm:
uucp:*:5:5:0000-uucp(0000):/usr/lib/uucp:
nuucp:*:10:10:0000-uucp(0000):\
      /usr/spool/uucppublic:/usr/lib/uucp/uucico
nobody:*:14:14:for weak daemons:/tmp:
rje:*:18:18:0000-rje(0000):/usr/rje:
lp:*:71:2:0000-lp(0000):/usr/spool/lp:
man:*:99:1:0000-Admin(0000):/:
setup:*:0:0:general system administration:/usr/admin:/bin/rsh
sysadm:*:0:0:general system administration:/usr/admin:/bin/rsh
```

Figure C-7: Typical **/etc/passwd** File

/etc/profile

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

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

Figure C-8 shows the RISComputer default profile.

```
#ident

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

trap ""  2 3

umask 022
MAIL=/usr/mail/${LOGNAME:?}
export LOGNAME USER

# This method of setting TZ is obsoleted by the new login.
# . /etc/TIMEZONE

#  Login and -su shells get /etc/profile services.
#  -rsh is given its environment in its .profile.
case "$0" in
-su )
   # omit '.' to close security hole
   PATH=/usr/net:/bin:/usr/bin:/etc:/usr/ucb
   export PATH
   ;;

-sh )
   PATH=$HOME/bin:/usr/net:/usr/bin:/bin:/usr/ucb:.
   export PATH

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

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

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

trap  2 3
```

Figure C-8: Standard **/etc/profile** File

/etc/rc0

The **/etc/rc0** file contains a shell script that is executed by **/etc/shutdown** for transitions to single-user state, and by **/etc/init** on transitions to run levels 0, 5, and 6. Files in the **/etc/rc0.d** directories are executed when **/etc/rc0** is run. The file **K00ANNOUNCE** in **/etc/rc0.d** prints the message "`System services are now being stopped.`"

```
 .
 .
 .
# system cleanup functions ONLY (things that end fast!)
for f in /etc/rc0.d/S*
   {
      if [ -s ${f} ]
      then
         /bin/sh ${f} start
      fi
   }
fi
trap "" 15
kill -15 -1
sleep 10
/etc/killall  9
sleep 10
sync;sync;sync
/etc/umountall
stty sane 2>/dev/null
sync;  sync
echo '
The system is down.'
sync
```

Figure C-9: Typical **/etc/rc0** File

/etc/rc0.d Directory

The **/etc/rc0.d** directory contains files executed by **/etc/rc0** for transitions to system run levels 0, 5, and 6. Files in this directory are linked from the **/etc/init.d** directory, and begin with either a **K** or an **S**. **K** indicates processes that are stopped, and S indicates processes that are started when entering run levels 0, 5, or 6.

/etc/rc2

The **/etc/rc2** file contains a shell script that is executed by **/etc/init** on transitions to run level 2 (multi-user state). Executable files in the **/etc/rc.d** and any executable files beginning with **S** or **K** in **/etc/rc2.d** directories are executed when **/etc/rc2** is run. All files in **rc2.d** are linked from files in the **/etc/init.d** directory. These files are prefixed with an **S** or a **K** and a number in the **/etc/rc2.d** directory.

Other files may also be added to **/etc/rc2.d** and **/etc/rc.d** directories as a function of adding hardware or software to the system. Figure C-10 shows a typical RISComputer **/etc/rc2** file.

```
#! /bin/sh

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

#      Pickup start-up packages for mounts, daemons, services, etc.
set 'who -r'
if [ $9 = "S" ]
then
      echo 'The system is coming up.  Please wait.'
      BOOT=yes
      if [ -f /etc/rc.d/PRESERVE ]
      # historical segment for vi and ex
      then
            mv /etc/rc.d/PRESERVE /etc/init.d
            ln /etc/init.d/PRESERVE /etc/rc2.d/S02PRESERVE
      fi

elif [ $7 = "2" ]
then
      echo 'Changing to state 2.'
      if [ -d /etc/rc2.d ]
      then
            for f in /etc/rc2.d/K*

            {
                  if [ -s ${f} ]
                  then
                        /bin/sh ${f} stop
                  fi
            }
      fi
fi
```

Figure C-10: Typical **/etc/rc2** File (screen 1 of 2)

```
if [ -d /etc/rc2.d ]
then
      for f in /etc/rc2.d/S*
      {
            if [ -s ${f} ]
            then
                  /bin/sh ${f} start
            fi
      }
fi
if [ "${BOOT}" = "yes" ]
then
      stty sane tab3 2>/dev/null
fi

if [ "${BOOT}" = "yes" -a -d /etc/rc.d ]
then
      for f in 'ls /etc/rc.d'
      {
            if [ ! -s /etc/init.d/${f} ]
            then
                  /bin/sh /etc/rc.d/${f}
            fi
      }
fi

if [ "${BOOT}" = "yes" -a $7 = "2" ]
then
      echo 'The system is ready.'
elif [ $7 = "2" ]
then
      echo 'Change to state 2 has been completed.'
fi
```

Figure C-10: Typical **/etc/rc2** File (screen 2 of 2)

/etc/rc2.d Directory

The **/etc/rc2.d** directory contains files executed by **/etc/rc2** for transitions to system run level 3. Files in this directory are linked from the **/etc/init.d** directory, and begin with either a **K** or an **S**. **K** indicates processes that should be stopped, and **S** indicates processes that should be started when entering run levels 2 or 3.

/etc/rc.d Directory

The **/etc/rc.d** directory contains executable files that do the various functions needed to initialize the system to run level 2. The files are executed when **/etc/rc2** is run. (Files contained in this directory prior to UNIX System release 3.0 were moved to **/etc/rc2.d**. This directory is only maintained for compatibility reasons.)

/etc/rc3

The **/etc/rc3** file is executed by **/etc/init**. It executes the shell scripts in **/etc/rc3.d** on transitions to system run level 3 (the Remote File Sharing state).

/etc/rc3.d Directory

The **/etc/rc3.d** directory contains files executed by **/etc/rc3** for transitions to system run level 3 (multi-user mode). Files in this directory are linked from the **/etc/init.d** directory, and begin with either a **K** or an **S**. **K** indicates processes that should be stopped, and **S** indicates processes that should be started when entering run level 3.

/etc/shutdown

The **/etc/shutdown** file contains a shell script to shut down the system gracefully in preparation for system backup or scheduled downtime. After stopping all nonessential processes, the **shutdown** script executes **/etc/rc0** for transition to run level s or S. For transitions to other run levels, the **shutdown** script calls **/etc/init**. Figure C-11 shows a typical RISComputer **/etc/shutdown** file.

```
#! /bin/sh

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

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

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

#!     chmod +x ${file}

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

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

grace=60
askconfirmation=yes
initstate=s
```

Figure C-11: Typical **/etc/shutdown** File (screen 1 of 3)

```
while [ $# -gt 0 ]
do
      case $1 in
      -g[0-9]* )
            grace='expr "$1" : '-g)''
            ;;
      -i[Ss0156] )
            initstate='expr "$1" : '-i)''
            ;;
      -i[234] )
            initstate='expr "$1" : '-i)''
            echo "$0:  Initstate $i is not for system shutdown"
            exit 1
            ;;
      -y )
            askconfirmation=
            ;;
      -* )
            echo "Illegal flag argument '$1'"
            exit 1
            ;;
      * )
            echo "Usage:  $0 [ -y ] [ -g<grace> ] [ -i<initstate> ]"
            exit 1
      esac
      shift
done

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

Figure C-11: Typical **/etc/shutdown** File (screen 2 of 3)

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

echo '\nShutdown started.     \c'
date
echo

sync
cd /
trap "exit 1"  1 2 15

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

      !
      sleep ${grace}
fi

/etc/wall <<-!
      ^GTHE SYSTEM IS BEING SHUT DOWN NOW ! ! !^G
      ^GLog off now or risk your files being damaged.^G

!
sleep ${grace}

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

Figure C-11: Typical **/etc/shutdown** File (screen 3 of 3)

/etc/stdcshrc

This file is the standard cshrc that a system administrator can place in new C shell user's home directory as a starter **$HOME/.cshrc**. A typical **/etc/stdcshrc** is shown below.

```
#ident
#      This is the default standard profile provided to a user.
#      They are expected to edit it to meet their own needs.

umask 022

stty line 1 erase '^H' kill '^U' intr '^C' echoe
eval 'tset -S -Q'

# list directories in columns
alias ls 'ls -C'
```

Figure C-12: Typical **/etc/stdcshrc** File

/etc/stdprofile

This file is the standard profile that a system administrator can place in new Bourne shell user's home directory as a starter **$HOME/.profile**. A typical **/etc/stdprofile** is shown below.

```
#ident      "@(#)sadmin:etc/stdprofile     1.2"
#      This is the default standard profile provided to a user.
#      They are expected to edit it to meet their own needs.

umask 022

stty line 1 erase '^H' kill '^U' intr '^C' echoe
eval 'tset -S -Q'

# list directories in columns
ls()  { /bin/ls -C $*; }
```

Figure C-13: Typical **/etc/stdprofile** File

/etc/TZ

The **/etc/TZ** file sets the time zone shell variable TZ. A typical **/etc/TZ** file is shown below. The first line of this file should correspond to your time zone.

```
PST8PDT
      # Timezone value used by login and /etc/TIMEZONE
      # The value must start on the beginning of the first line
      # and a newline must follow the value.
#ident
```

Figure C-14: Typical **/etc/TZ** File

/etc/utmp

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

/etc/wtmp

The **/etc/wtmp** file contains a history of system logins. The owner and group of this file must be **adm**, and the access permissions must be 664. Each time **login** is run this file is updated. As the system is accessed, this file increases in size. Periodically, this file should be cleared or truncated. The command line **>/etc/wtmp** when executed by **root** creates the file with nothing in it. The following command line limits the size of the **/etc/wtmp** file to the last 3600 characters in the file:

```
tail -3600c /etc/wtmp > /tmp/wtmp; mv /tmp/wtmp /etc/wtmp
```

Note that **/etc/cron**, **/etc/rc0**, or **/etc/rc2** can be used to clean up the **wtmp** file. To use one of these functions, add the appropriate command line to the **/usr/spool/cron/crontab/root**, **/etc/shutdown.d/...**, or **/etc/rc.d/, rc2.d, rc3.d ...** file.

/usr/lib/cron/log

A history of all actions taken by **/etc/cron** is recorded in the **/usr/lib/cron/log** file. The **/usr/lib/cron/log** file should be periodically truncated to keep the size of the file within a reasonable limit. Note that **/etc/cron**, **/etc/rc0**, or **/etc/rc2** can be used to clean up the **/usr/lib/cron/log** file. To use one of these functions to limit the size of a log file, add the appropriate command line to the **/usr/spool/cron/crontab/root**, **/etc/shutdown.d/...**, or **/etc/rc.d/ rc2.d, rc3.d ...** file, as applicable. The following command line limits the size of the log file to the last 100 lines in the file:

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

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

```
! *** cron started ***  pid = 81 Tue Sep 22 18:32:26 1987
>  CMD: /usr/lib/sa/sa1
>  sys 85 c Tue Sep 22 19:00:00 1987
<  sys 85 c Tue Sep 22 19:00:00 1987
>  CMD: /usr/lib/sa/sa1
>  sys 88 c Tue Sep 22 20:00:00 1987
<  sys 88 c Tue Sep 22 20:00:00 1987
>  CMD: /usr/lib/sa/sa1
>  sys 91 c Tue Sep 22 21:00:00 1987
<  sys 91 c Tue Sep 22 21:00:00 1987
>  CMD: /usr/lib/sa/sa1
>  sys 94 c Tue Sep 22 22:00:00 1987
<  sys 94 c Tue Sep 22 22:00:00 1987
>  CMD: /usr/lib/sa/sa1
>  sys 97 c Tue Sep 22 23:00:00 1987
```

Figure C-15: Typical **/usr/lib/cron/log** File

/usr/lib/spell/spellhist

If the Spell Utilities is installed, a history of all words that **spell**(1) fails to match is kept in the **/usr/lib/spell/spellhist** file. Periodically, this file should be reviewed for words that should be added to the dictionary. After the **spellhist** file is reviewed, it can be cleared.

/usr/news

The **/usr/news** directory contains news files. The file names are descriptive of the contents of the files; they are analogous to headlines. When a user reads the news, using the **news** command, an empty file named **.news_time** is created in his or her login directory. The date (time) of this file is used by the **news** command to determine if a user has read the latest news file(s).

/usr/spool/cron/crontabs

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

As **root**, you can either use the **crontab**(1) command or edit the appropriate file under **/usr/spool/cron/crontabs** to make the desired entries. Revisions to the file take effect at the next reboot. The line entry format of a **/usr/spool/cron/crontabs/***logname* file is as follows:

minute hour day month day-of-week command

The various fields of a **crontabs/***logname* line entry are the following:

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

The following syntax applies to the first five fields:

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

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

Figure C-16 shows a typical **/usr/spool/cron/crontabs/***logname* file. The data shown are the **sys** file. The file entries support system activity report activities (see Chapter 6). Remember, you can use the **cron** function to decrease the number of data terminal driven system administration tasks: include recurring and habitual tasks in your crontab file.

```
#ident      @(#)adm:sys 1.2
#ident      %W%
#
# The sys crontab should be used to do performance
# collection. See cron and performance manual pages
# for details on startup.
#
0 * * * 0-6 /usr/lib/sa/sa1
20,40 8-17 * * 1-5 /usr/lib/sa/sa1
5 18 * * 1-5 /usr/lib/sa/sa2 -s 8:00 -e 18:01 -i 1200 -A
```

Figure C-16: Typical **/usr/spool/cron/crontabs/sys** File

(Refer to the **crontab**(1) manual page in the *User's Reference Manual* for additional information.)

Glossary

address	a number, label, or name that indicates the location of information in the computer's *memory*.
a.out	the default name of a freshly compiled *object file*, pronounced 'A-dot-out'; historically a.out signified assembler output.
archive	1. a collection of data gathered from several *files* into one file. 2. especially, such a collection gathered by **ar**(1) for use as a library.
automatic calling unit	a hardware *device* used to dial stored telephone numbers; allows the system to contact another system over phone lines without manual intervention.
bad block	a section of a storage medium which cannot store data reliably. This is a 512-byte disk block, otherwise known as a *sector*.
block	the basic unit of *buffering* in the *kernel*, 8192 bytes; see *indirect*, *logical*, and *physical blocks* and *sectors*.
block device	a *device* upon which a *file system* [1] can be *mounted*, typically a permanent storage device such as a tape or disk drive, so called because data transfers to the device occur by *blocks*; cf. *character device*.
boot	to start the operating system, so called because the *kernel* must bootstrap itself from secondary storage into an empty machine. No *login* [3] or *process* persists across a boot.
boot program	loads the *operating system* into ram.
buffer	1. a staging area for input-output where arbitrary-length transactions are collected into convenient units for system operations; the *file system* [3] uses buffers, as does *stdio*. 2. to use buffers.
buffer pool	a region of store available to the *file system* [3] for holding *blocks;* all but *raw* [2] input-output for *block devices* goes through the buffer pool so read and write operations may be independent of device blocks.
cartridge tape	a storage medium that consists of a magnetic tape wound on spools housed in a plastic container.
character device	a *device* upon which a *file system* [1] cannot be *mounted* such as a terminal or the *null device*. (See *raw device*.)
child process	see *fork*.
client	a *host* that has *mount*ed an exported file system from an *NFS server*.
command	1. an instruction to the *shell,* usually to run a *program* [1] as a *child process*. 2. by extension, any *executable file,* especially a *utility program*.

command file	same as *shell script*.
configuration	the arrangement of the software or hardware of a system, peripheral, or network as defined by the nature, number, and chief characteristics of its functional units.
controller	a *device* that directs the transmission of data over the data links of a *network*.
core file	a *core image* of a terminated *process* saved for debugging; a core file is created under the name 'core' in the *current directory* of the process.
core image	a copy of all the *segments* of a running or terminated program; the copy may exist in main storage, in the *swap area,* or in a *core file*.
crash	If a hardware or software *error* condition develops that the system can't handle, it takes itself out of service, or crashes. Such conditions occur when the system can't allocate resources, manage *processes,* respond to requests for system functions, or when the electrical power is unstable.
cron	a command which creates a daemon that invokes commands at specified dates and times.
cylinder	the set of all *tracks* on a *disk* which are the same distance from the axis about which the disk rotates.
daemon	a *background* process, often perpetual, that performs a system-wide public function, e.g. **calendar**(1) and **cron**(1M); the affected spelling is an ancient legacy.
destination	the remote system that will ultimately receive a *file* transferred over a *network*.
device	1. a *file* [2] that is not a *plain file* or a *directory,* such as a tape drive, or the *null device*; a *special file*. 2. a physical input-output unit.
diagnostic	a message printed at your terminal that identifies and isolates *program* errors.
directory	a *file* that comprises a catalog of *filenames* [2]; the organizing principle of the *file system* [2], a directory consists of *entries* which specify further *files* (sense 2, including directories), and constitutes a node of the *directory tree*.
directory entry, entry	1. an association of a name with an *inode number* appearing as an element of a *directory*. 2. the name part of such an association.
directory hierarchy	the tree of all *directories,* in which each is reachable from the *root* via a chain of *subdirectories*.
directory tree	same as *directory hierarchy*.
disk	a platter coated with magnetic material on which data can be stored.

drive	the hardware device that holds magnetic disks and tapes while they are in use.
dump	a copy of the *core image* of the operating system.
dvh	disk volume header. A volume header (or "volume table of contents") at the beginning of each disk contains information regarding the physical device and the logical partitions. It is manipulated by the standalone **format** and the UMIPS **dvhtool**(1M) commands, and can be viewed by using **prtvtoc**(1M). It is designated in a device name as the "vh" partition, for example: **ips0d0svh**.
environment	1. a set of strings, distinct from the *arguments*, made available to a *process* when it *executes* [2] a *file*; the environment is usually inherited across *exec*(2) operations. 2. a specific environment [2] maintained by the *shell*. 3. a nebulously identified way of doing things, as in 'interactive environment': a deprecated usage, not always expunged from these manuals.
error	occurs when a hardware or software condition prevents the successful *execution* of a system or a user *process*.
error message	a message sent from the system to the *system console* when an *error* occurs.
exec	a system call which allows the user to request the execution of another program.
executable file	1. an *object file* that is ready to be copied into the *address* space of a *process* to run as the code of that process. 2. a file that has execute *permission*, either an *executable file* [1] or a *shell script*.
execute	1. informally, to run a *program*. 2. to replace the *text segment* and *data segments* of a *process* with a given *program* [1].
FIFO	a named permanent *pipe* which allows two unrelated *processes* to exchange information using a pipe connection.
file	1. in general, a potential source of input or destination for output. 2. most specifically, an *inode* and/or associated contents, i.e. a *plain file*, a *special file*, or a *directory*. 3. a *directory entry*; several directory entries may name the same file [2]. 4. most loosely, a *plain file*.
file descriptor	a conventional integer quantity that designates an *open file*.
filename	1. a *pathname*. 2. the last component name in a pathname.
file system	1. a collection of *files* that can be *mounted* on a block *special file*; each file of a file system appears exactly once in the *i-list* of the file system and is accessible via some *path* from the *root* directory of the file system. 2. the collection of all *files* on a computer. 3. the part of the kernel that deals with file systems [1].

filter	a *program* [1] that reads from the *standard input* and writes on the *standard output*, so called because it can be used as a data-transformer in a *pipeline*.
flush	to empty a *buffer*, for example to throw away unwanted input-output upon *interrupt* or to release output from the clutches of *stdio*.
fork	to split one *process* into two, the **parent process** and **child process**, with separate, but initially identical, *text*, *data*, and *stack segments*.
formatting	the process of imposing an addressing scheme on a *disk*. This includes the establishment of a *dvh*, and the mapping of the disk into *tracks* and *sectors*.
free list	in a *file system* [1], the list of *blocks* that are not occupied by data.
getty	one of a series of *processes* which connect the user to the UNIX system. *getty* is invoked by *init*, and in turn invokes *login*.
group	1. a set of *permissions* alternative to *owner* permissions for access to a *file*. 2. a set of *userids* that may assume the privileges of a group [1]. 3. the *groupid* of a file.
groupid	an integer value, usually associated with one or more *login names*; as the *userid* of a process becomes the *owner* of files *created* by the process, so the groupid of a process becomes the *group* [3] of such files.
host	a computer that is configured to share *resources* in a *networked* environment.
i-list	the index to a *file system* [1] listing all the *inodes* of the file system; cf. *inode number*.
indirect blocks	data blocks that are not directly referenced by a *inode* (because the file has more blocks than can be specified in the inode itself. The inode has 3 *addresses* that indirectly reference (by a cascade of pointers) some 2,114,114 data blocks (an extremely large potential *file* size). the inode has 1 address that points to 128 more data blocks; a second address that points to 128 blocks that each point to 128 data blocks; and finally a third address that points to 128 blocks each of which point to another 128 blocks, each of which point to 128 data blocks!
init	a general *process* spawner which is invoked as the last step in the *boot* procedure; it regularly checks a table that defines what processes should run at what *run level*.
inode	an element of a *file system* [1]; an inode specifies all properties of a particular *file* [2] and locates the file's contents, if any.

inode number, i-number	the position of an *inode* in the *i-list* of a *file system* [1].
instruction	see *address*.
integrity	in a *file system*, the quality of being without errors due to *bad blocks*.
interface programs	*shell scripts* and *programs* furnished with the LP *spooling* software which interface between the user and the printer.
interrupt	1. a *signal* that normally terminates a *process*, caused by a break or an interrupt character. 2. a signal generated by a hardware condition or a peripheral *device*. 3. loosely, any *signal*.
IPC	an acronym for interprocess communication.
kernel	the UNIX system proper; resident code that implements the *system calls*.
kernel address space	a portion of memory used for data and code addressable only by the *kernel*.
line discipline	a module to handle protocol or data conversion for a *stream* [2]. A line discipline, unlike a *filter*, is part of the *kernel*.
link	1. to add an entry for an existing *file* to a directory; converse of *unlink*. 2. by extension, a *directory entry*. 3. loosely, any but one putatively primary directory entry for a given *inode*; either linked [1] or a *symbolic link*.
link count	the number of *directory entries* that pertain to an *inode*; a *file* ceases to exist when its link count becomes zero and it is not *open*.
load device	designates the physical *device* from which a program will be loaded into main *memory*.
log files	contain records of transactions that occur on the system; software that *spools*, for example, generates various log files.
logical block	a unit of data as it is handled by the software; the UMIPS system handles data in 8192-byte logical blocks.
login	1. the *program* that controls logging in. 2. the act of *logging in*. 3. by extension, the computing session that follows a login [2].
memory	1. same as *memory image*. 2. physical memory represents the available space in main memory; *programs* are either *swapped* or *paged* into physical memory for *execution*. 3. virtual memory management techniques permit *programs* to treat *disk* storage as an extension of main memory.
memory image	same as *core image*.

mode, file mode	the *permissions* of a *file*; colloquially referred to by a 3-digit octal number, e.g. 'a 755 file'; see *chmod*(1).
mount	to extend the *directory hierarchy* by associating the *root* of a *file system* [1] with a *directory entry* in an already mounted file system; converse is unmount, spelled 'umount'.
namelist	same as *symbol table*.
network	the hardware and software that constitute the interconnections between computer systems, permitting electronic communication between the systems and associated peripherals.
networking	for computer systems, means sending data from one system to another over some communications medium (coaxial cable, phone lines, etc.). Common networking services include *file* transfer, remote *login*, remote *execution*.
node name	an up-to-six character name for the system; used as the official name of the machine in a *network*.
null device	a *device* [1] that always yields *end of file* on reading and discards all data on writing (e.g., /dev/null).
nvram	the Non-Volatile Random Access Memory (NVRAM), which permanently holds a few, special programs.
object file	a *file* of machine language code and data; object files are produced from source programs by compilers and from other object files and libraries by the link editor; an object file that is ready to run is an *executable file* [1].
operating system	the *program* for managing the resources of the computer. It takes care of such things as input/output procedures, process scheduling, the file system, removing this burden from user programs.
open file	1. the destination for input or output obtained by *opening* a *file* or creating a *pipe*; a *file descriptor*; open files are shared across *forks* and persist across *executes* [2]. 2. loosely, a file that has been opened, however an open file [1] need not exist in a *file system* [1], and a file [2] may be the destination of several *open files* simultaneously.
other	1. a set of *permissions* regulating access to a *file* by processes with *userid* different from the *owner* and *groupid* different from the *group* of the file. 2. the customary name of the default *group* [2] assigned upon *login*.
owner	the *userid* of the *process* that created a *file*; the owner has distinctive *permissions* for a file.
page	a fixed length, 1024-byte block that has a virtual *address*, and that can be transferred between main and secondary storage.

paging	the process by which *programs* are truncated into *pages* and transferred between main and secondary storage by the virtual handler (or paging *daemon*).
parent process	see *fork*.
partitions	units of storage space on disk, corresponding to the logical *file systems*.
path, pathname	a chain of names designating a *file*; a **relative pathname** leads from the current directory, for example, a path to *directory* A, thence to directory B, thence to *file* C is denoted A/B/C; a **full pathname** begins at the *root*, indicated by an initial '/', as in /A/B/C.
permission	a right to access a *file* in a particular way; read, write, execute (or look up in, if a directory); permissions are granted separately to *owner, group,* and *others*. **permission bit** a permission, so called because each permission is encoded into one bit in an *inode*.
physical block	a unit of data as it is actually stored and manipulated; the MIPS systems handle data in 512-byte physical blocks, or "sectors".
physical	**see** *memory*.
pipe	a direct stream connection between *processes*, whereby data written on an *open file* in one process becomes available for reading in another.
pipeline	a sequence of *programs* [1] connected by *pipes*.
polling	the interrogation of *devices* by the *operating system* to avoid contention, determine operation status, or ascertain readiness to send or receive data.
ports	the point of physical connection between a peripheral *device* (such as a terminal or a printer) and the device *controller* (ports board), which is part of the computer hardware.
process	a connected sequence of computation; a process is characterized by a *core image* with instruction location counter, *current directory*, a set of *open files, control terminal, userid,* and *groupid*.
process id	an integer that identifies a *process*.
process number	same as *process id*.
profile	1. an optional *shell script*, '.profile', or '.cshrc' and/or '.login', conventionally used by the *shell* upon *logging in* to establish the *environment* [3] and other working conditions customary to a particular user. 2. to collect a histogram of values of the instruction location counter of a *process*.
program	1. an *executable file*. 2. a *process*. 3. all the usual meanings.

queue	a line or list formed by items in a system waiting for service.
raw device	a *character device* where read and write operations are not *buffered,* that is, characters are written directly to, and read directly from, the device.
reboot	same as *boot.*
region	a group of machine *addresses* that refer to a base address.
release	a distribution of fixes or new functions for an existing software product.
resource	a directory that is *advertised* in a *Remote File Sharing* environment. When a *resource* is *mount*ed on a *client,* the contents of the directory (files, devices, and named pipes) and any of its subdirectories are potentially available to users on the *client.*
re-tension	the process of re-winding the tape in a *cartridge tape device* to make sure it is at the correct tautness for accurate recording of data (see mt(7)).
root	1. a distinguished directory that constitutes the origin of the *directory hierarchy* in a *file system* [1]. 2. specifically, the origin for the *file system* [2], with the conventional *pathname* '/'. 3. the origin of the directory hierarchy in a *file system* [1].
rotational gap	the gap between the actual *disk* locations of blocks of data belonging to the same *file*; the rotational gap compensates for the continuous, high-speed rotation of the disk so that when the controller is ready to reference the next physical block the read-write head is positioned correctly at the beginning of that block.
run level	a software *configuration* of the system which allows a particular group of *processes* to exist.
schedule	to assign resources— main store and CPU time—to *processes.*
scheduler	a permanent *process,* with *process number* 1, and associated *kernel* facilities that does scheduling.
search path	in the *shell,* a list of *pathnames* of *directories* that determines the meaning of a *command*; the command name is prefixed with members of the search path in turn until a pathname of an *executable file* [2] results; the search path is given by the shell variable PATH.
sector	A 512-byte portion of a disk *track* which is usually the smallest addressable section of the disk.
segment	a contiguous range of the address space of a *process* with consistent store access capabilities; the four segments are (i) the **text segment**, occupied by executable code, (ii) the **data segment**, occupied by *static* data that is specifically initialized, (iii) the **bss segment**, occupied by static data that is initialed by default to zero values, and (iv) the

	stack segment, occupied by *automatic* data, see *stack*; sometimes (ii), (iii), and (iv) are collectively called data segments.
semaphore	an IPC facility which allows two or more processes to be synchronized.
server	a *host* that is actively sharing one of its *advertise*d *resources* with another *host* in a *Remote File Sharing* environment.
set userid	a special *permission* for an *executable file* [1] that causes a *process* executing it to have the access rights of the *owner* of the file; the owner's *userid* becomes the **effective userid** of the process, distinguished from the **real userid** under which the process began.
set userid bit	the associated *permission bit*.
shared memory	an IPC facility which allows two or more processes to share the same data space.
shell	1. the program *sh*(1), which causes other programs to be executed on *command*; the shell is usually started on a user's behalf when the user *logs in*. 2. by analogy, any program started upon logging in.
shell script	an executable *file* of *commands* taken as input to the *shell*.
signal	an exceptional occurrence that causes a *process* to terminate or divert from the normal flow of control; see *interrupt*, *trap*.
single-user	a state of the operating system in which only one user is supported.
source file	1. the uncompiled version of a *program*. 2. generally, the unprocessed version of a *file*.
special file	an *inode* that designates a *device*, further categorized as either (i) a **block special file** describing a *block device*, or (ii) a **character special file** describing a *character device*.
spool	(*s*imultaneous *p*eripheral *o*perations *o*n *l*ine) to collect and serialize output from multiple *processes* competing for a single output service.
spool area	a *directory* in which a spooler collects work.
spooler	a *daemon* that spools.
stack	a *segment* of the *address* space into which *automatic* data and subroutine linkage information is allocated in last-in-first-out fashion; the stack occupies the largest data addresses and grows downward towards *static* data.
standard error	one of three files described below under *standard output*.
standard input	the second of three files described below under *standard output*.

standard output	*open files*, customarily available when a *process* begins, with *file descriptors* 0, 1, 2 and *stdio* names 'stdin', 'stdout', 'stderr'; where possible, utilities by default read from the standard input, write on the standard output, and place error comments on the standard error file. Initially, all three of these files default to your terminal.
startup	same as *boot*
sticky bit	a *permission* flag that identifies a file as a *sticky file*.
sticky file	a special *permission* for a *shared text* file that causes a copy of the *text segment* to be retained in the *swap area* to improve system response.
super block	the second *block* in a *file system* [1], which describes the allocation of space in the file system; cf. *boot block*.
	userid 0, which can access any *file* regardless of *permissions* and can perform certain privileged *system calls*, e.g. setting the clock.
swap	to move the *core image* of an executing program between main and secondary storage to make room for other *processes*.
swap area	the part of secondary store to which *core images* are *swapped*; the swap area is disjointed from the *file system*.
symbolic link	an *inode* that contains the *pathname* of another. References to the symbolic link become references to the named inode.
symbol table	information in an *object file* about the names of data and functions in that file; the symbol table and *address* relocation information are used by the link editor to compile *object files* and by debuggers.
System Administration	when capitalized, refers to the package of screens and interactive prompts, invoked through the **sysadm**(1) command, that help you accomplish most system administration tasks.
system calls	1. the set of system primitive functions through which all system operations are allocated, initiated, monitored, manipulated, and terminated. 2. the system primitives invoked by user *processes* for system-dependent functions, such as I/O, process creation, etc.
system console	the directly connected terminal used for communication between the operator and the computer.
system name	an up-to-six character name for the system; resides in the SYS parameter.
TCP/IP	the Transmission Control Protocol/Internet Protocol communications software on an M-Series system. An M-Series system can communicate across an Ethernet local area network with other hosts and terminals using TCP/IP communications software. this permits operations between machines such as file transfer and remote

	login services.
table	an array of data each item of which may be uniquely identified by means of one or more arguments.
text file, ASCII	a *file*, the bytes of which are understood to be in ASCII code.
track	an addressable ring of *sections* on a *disk*; each disk has a predefined number of concentric tracks, which allows the disk head to properly access *sections* of data.
trap	a method of detecting and interpreting certain hardware and software conditions via software; a trap is set to catch a *signal* (or *interrupt*), and determine what course of action to take.
tunable parameters	variables used to set the sizes and thresholds of the various control structures of the *operating system*.
tuning	1. modifying the *tunable parameters* so as to improve system performance. 2. the reconfiguration of the *operating system* to incorporate the modifications into *executable* version of the system.
userid	an integer value, usually associated with a *login name*; the userid of a *process* becomes the *owner* of files *created* by the process and descendent (*forked*) processes.
utility, utility program	a standard, generally useful, permanently available *program*.
version	a separate *program* product, based on an existing one, but containing significant new code or new functions.
virtual memory	see *memory*.

Index