Part Number 800-1085-01
Revision: F of 1 November 1983
For: Sun System Release 1.0

System Manager's Manual

for the

Sun Workstation — Models 100U/150U

Sun Microsystems, Inc.,
2550 Garcia Avenue,
Mountain View,
California 94043
(415) 960-1300

This manual supersedes the *User's Guide for the Sun Workstation* dated 15th October 1982. The prior User's Guide was specific to Sun Workstations running the Version 7 UNIX* system.

Acknowledgements

The tutorials in the final section of this manual were originally derived from the work of various people associated with various institutions. Their names and the titles of the original works are:

Building UNIX Systems with Config
was originally written by Samuel J. Leffler, formerly of the Computer Systems Research Group, University of California at Berkeley.

Fsck — The UNIX File System Check Program
was originally written by T. J. Kowalski, Bell Laboratories, Murray Hill, New Jersey; and was revised by Marshall Kirk McKusick, Computer Systems Research Group, University of California at Berkeley.

SENDMAIL — An Internetwork Mail Router
was originally written by Eric Allman, formerly of Project Ingres, University of California at Berkeley.

SENDMAIL — Installation and Operation Guide
was originally written by Eric Allman, formerly of Project Ingres, University of California at Berkeley.

Uucp Implementation Description
was originally written by D. A. Nowitz, Bell Laboratories, Murray Hill, New Jersey.

Line Printer Spooler Manual
was originally written by Ralph Campbell, Computer Systems Research Group, University of California at Berkeley.

* UNIX is a trademark of Bell Laboratories.

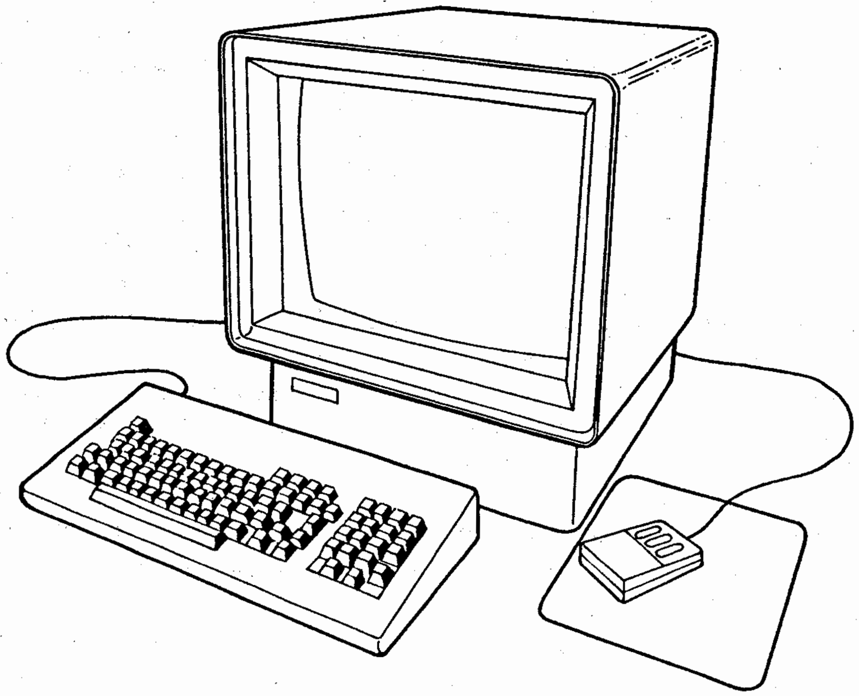
Revision History

Revision	Date	Comments
A	19th February 1983	First β release of this System Manager's Guide.
B	11th March 1983	Corrected errors and misinformation in original release.
C	15th April 1983	Corrected errors in setup information. Added material on setting up UUCP, Mail System, and Printers.
D	1st August 1983	Rewrote for 100U/150U. Added information on subsystem set-up, system set-up and operation, and upgrading. Revised hardware documentation, and included Xylogics 450 disk controller. Revised and updated installation procedures.
E	15th September 1983	Updated for new Sun CPU and Memory Expansion Boards. Added several tutorials on new aspects of the system, and revised operation and maintenance procedures to reflect new software.
F	1st November 1983	Revised for 1.0 Release.

System Manager's Manual

Table of Contents

INTRODUCTION



Welcome to the Sun Workstation.

This manual is meant to help you get the Sun Workstation Models 100U (desktop) and 150U (rackmount) up and running. The manual has three basic divisions. The first, *System Installation and Maintenance Guide*, gives unpacking and and set-up directions for new desktop and rackmount workstations and their subsystems; walks you through UNIX installation procedures for Sun systems; and covers the basics on hardware and software operation, maintenance, and upgrading. The second section is a reference manual for UNIX commands and utilities relevant to system management. The final section includes several installation tutorials.

Summary of Contents

This manual consists of three major sections. The first section, *System Installation and Maintenance Guide*, covers hardware assembly for new 100U and 150U systems, walks you through UNIX installation, and supplies basic information on system configuration, maintenance, and software upgrading. You can use this guide to take you safely from unpacking your shipping cartons to doing backups. Contents are:

Chapter 1 — *Unpacking and Setting Up the 100U* — is a guide to getting a new Model 100U out of its shipping cartons and setting up the desktop workstation ready to run. If you have disk and/or tape subsystems, work through Chapter 3 next to complete your hardware set up.

Chapter 2 — *Unpacking and Setting Up the 150U* — is a guide to getting a new Model 150U out of its shipping cartons and setting up the rackmount workstation ready to run. If you have disk and/or tape subsystems, work through Chapter 3 next to complete your hardware set up.

Chapter 3 — *Subsystem Set-up* — covers unpacking, mounting, and cabling up the disk and tape subsystems which may be supplied with either the 100U or the 150U.

Chapter 4 — *Installing UNIX for the First Time* — guides you through the procedure for installing the UNIX System on a new Sun Workstation from a half-inch or quarter-inch distribution tape. Network installation procedures are also covered.

Chapter 5 — *Installing UNIX on Systems Without Tape Support* — describes the procedure for installing the UNIX System onto a Sun Workstation equipped with only a disk drive (no tape), over the Ethernet from a system equipped with a tape drive.

Chapter 6 — *System Set-up and Operation* — covers basic system set-up and maintenance procedures. Topics from *uucp* and *sendmail* installation to doing dumps are covered.

Chapter 7 — *Upgrading System Software* — gives a brief review of how to upgrade your system to a new software release level.

Chapter 8 — *Hardware Configuration and Expansion* — describes the 7-slot Multibus cardcage in the Model 100U and the 15-slot cage in the 150U, and provides configuration details for each board in the cardcage. You must use this chapter to determine placement and switch-settings for new boards when you add boards or peripherals. Also useful for troubleshooting, if you need to get inside the cardcage and identify the boards.

Appendix A — *The Sun Workstation Monitor* — discusses the startup and boot functions of the Sun Workstation monitor, lists diagnostic messages which it currently displays, and covers monitor commands and routines which enable terminal emulation. It also describes the operational features of SUN-1 keyboards.

The second section of this manual is a reference guide to UNIX commands and utilities of special interest to system managers. Section 8 covers the procedures and commands needed for day-to-day administration of the UNIX system on the Sun Workstation. The pages are arranged in alphabetical order.

The final section of the manual contains several installation tutorials.

This manual is still under development and will undergo several revisions before it reaches its final form. It will evolve over time to cover the kinds of things you do on a day to day basis; and to address the kinds of problems that you are likely to encounter, and what to do about those problems. To help direct our development — and to help us maintain the currency and accuracy of this material — we have supplied a reader comment sheet at the end of this guide. Please use the comment sheet to list errors, omissions and outright lies. Your responses will help a great deal in our efforts to keep our documentation cogent.

Applicable Documents

We emphasize that this manual outlines rather than exhausts many of its topics. References to applicable documents supplied with your system are given throughout, however, and we urge you to read these documents when you can.

Hardware and System Documentation

Part Number	*Description*
800-0274	Interphase SMD 2180 Storage Module Controller/Formatter User's Guide.
800-0277	Fujitsu M2311K/M2312K Microdisk Drives CE Manual.
800-0339	Sun 1024 Video Board User Manual.
800-0393	3COM 3C400 Multibus Ethernet Controller Reference Manual.
800-0397	Sun 1M Memory Board User's Manual.
800-0398	Sun Color Video Board User's Manual.
800-0402	User's Guide for the Sun Workstation Mouse Subsystem.
800-0415	Sun 1/4" Tape Interface User Manual.
800-0485	Power Drain Requirements for Sun Workstation Boards
800-0620	CPC TAPEMASTER Product Specification
800-0622	CPC TAPEMASTER Application Note
800-0623	CDC Streaming Tape Unit 9218X Reference Manual (1/2")
800-0628	Archive Product Manual (Sidewinder 1/4" Streaming Cartridge Tape Drive)
800-1000	Philips Preliminary Service/Operator Manual High Resolution 17" CRT Display
800-1002	BARCO GD 33 Color Display 120VAC Operation Instruction (13" Color Monitor Manual)
800-1003	BARCO Color Display CD 251 120/220 VAC Operation Guide (19" Color Monitor Manual)
800-1011	Fujitsu M2351A/AF Mini-Disk Drive CE Manual.
800-1025	Xylogics Model 450 Peripheral Processor SMD Disk Subsystems Maintenance and Reference Manual
800-1052	Zilog Serial Communications Controller Technical Manual

Software Documentation

Part Number	*Description*
800-1083	User's Manual for the Sun UNIX System.
800-1084	System Interface Manual for the Sun UNIX System.
800-1091	SunCore Programmer's Manual
800-1092	Programmer's Reference Manual for the Sun Window System
800-1093	System Internals Manual

Notations Used In This Guide

References to commands and utilities from the *User's Manual for the Sun UNIX System* and the *System Interface Manual for the Sun UNIX System* use the notation:

passwd(1)

to indicate the *passwd* page in section 1 of the User's Manual. (The User's Manual contains sections 1, 6, and 7.)** Similarly, *passwd*(5) means refer to the *passwd* manual page in section 5 of the System Interface Manual. (The System Interface Manual contains sections 2, 3, 4, and 5.) The notation *spline*(1G) means that this is a Graphics command in section 1 of the User's Manual. Finally, */etc/reboot*(8) implies that this is a command from section 8, which is in this manual. (The System Manager's Manual contains section 8.)

Throughout the examples, there are numbers in the notation:

0xnnnn.

A number with the notation *0x* in front of it is a hexadecimal (base 16) number. Those wishing more information should consult any of the literature on the C programming language.

Finally, this manual provides a number of tutorial "walk through" examples which use certain formatting conventions. In all these examples, what the user types is shown in **bold faced text, like this.** The system's response is shown in roman type, like this. Variable items which you must substitute — which depend on your system's specific configuration — are shown in *italic type, like this.* Be especially careful with all such substitutions. Lastly, text which appears [within brackets] is commentary; usually, we use brackets to abbreviate a long series of system response messages, or to offer explanation of a process.

** For an explanation of this weird numbering scheme, see the *Introduction to the Sun UNIX System Manuals* in the User's Manual.

System Installation and Maintenance Guide

for the

Sun Workstation — Models 100U/150U

CAUTION: This document does not cover hardware upgrades; it is intended for use with new systems. If you are upgrading your 100/150, follow the hardware upgrading procedures given by Sun Microsystems BEFORE making use of this manual.

Sun Microsystems, Inc.,
2550 Garcia Avenue,
Mountain View,
California 94043
(415) 960-1300

Table of Contents

1. UNPACKING AND SETTING UP THE 100U

This chapter outlines how to unpack and set up your new Sun Model 100U and some of its peripherals (subsystems are covered in the chapter *Subsystem Set-up*).

If you are *upgrading* your Sun Model 100 rather than starting with a system 'fresh from the box,' please heed the warning on the front page of this manual. SUN-1 systems which are upgrading must not attempt to install this release of the system until all Multibus memory has been removed or disabled, and — for an upgrade from a Xylogics 440 to a Xylogics 450 — cabling has been re-routed. Make sure you have followed Sun's hardware upgrade procedures before proceeding.

If you are adding boards and/or peripherals to an existing system, the chapter on *Hardware Configuration and Expansion* will give you the information you need.

1.1. Safety Precautions

DO NOT attempt procedures which are not described in this manual; if you do so, you may void your warranty. Please refer all servicing not described in this document to qualified service personnel. If in doubt, contact your authorized Sun service representative.

Observe common-sense safety precautions as you would for any electrical or electronic equipment. Always disconnect power before opening any system enclosure.

Even after power is disconnected from the workstation, very high voltages remain in the capacitors behind the picture tube. It takes about ten minutes for the capacitors to lose their charge.

The following sections contain set-up and configuration specifications for the Sun Workstation. Consult the additional hardware manuals supplied with your workstation (and with user-supplied Multibus accessory boards, if any) for further information and precautions.

1.2. Environmental Considerations

This section describes the environmental requirements for the Sun Model 100U.

Attention: This equipment generates, uses, and can radiate radio frequency energy and if not installed and used in accordance with the instructions manual, may cause interference to radio communications. It has been tested and found to comply with the limits for a Class A computing device pursuant to Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference when operated in a commercial environment. Operation of this equipment in a residential area is likely to cause interference in which case the user at his own expense will be required to take whatever measures may be required to correct the interference.

1.2.1. Physical Environment

The Sun Model 100U is manufactured for the following physical environment:

	Operating	*Non-Operating*
Ambient Temperature	10 ° C - 40 ° C	-40 ° C - 65 ° C
Humidity (Non condensing)	5% - 90%	5% - 90%
Altitude	10,000 feet	40,000 feet

1.2.2. Power Drain Requirements

Model 100U desktop workstations are equipped with a 200 watt four-output power supply, which delivers + 5, –5, + 12, and –12 volts. The power supply is rated as follows*:

Voltage	*Maximum Current in Amps*
+ 5	35.0
–5	1.5
+ 12	4.0
–12	1.5

It is possible that a full cardcage will exceed the ratings of this supply. The table which follows shows the power consumption of each Sun-supplied board. Boards documented are those current for this release.

*NOTE that this rating is for the Model 100U current with this release; older systems (pre-0.4 release of August 1983) have 175 watt, 30 amp power supplies. The peak amperage for these systems is: + 5: 30 amps; -5: 2.0 amps; + 12: 4.0 amps; -12: 2.0 amps.

PART NUMBER	*BOARD*	*AMPS @ + 5V*	*TOTAL WATTS*
370-1012	Xylogics SMD Disk Controller Board	8.0	45
501-0289	Color Display Controller	6.0	36
590-0059	Black and White Display Controller	5.0	25
590-0217	Interphase SMD Disk Controller Board	4.0	23
590-0288	10Mbit Ethernet Board	5.0	31
590-0502	1/2-inch Tape Controller Board	4.0	20
590-0526	1/4-inch Tape Controller Board	3.0	15
590-1007	SUN-2 CPU Board, Keyboard, and Mouse	6.0	30
590-1013	1M Memory Board	3.0	15
590-1013	SUN-2 (low power) Memory Board	2.5	13

Note that the keyboard and mouse draw their power from the CPU board.

Review the above list carefully to determine exactly what configuration of boards you can have in the cardcage. If you get boards from other suppliers, make sure you obtain the current consumption information from them, so that you can factor that information in with the list above. Overloading the rated capacity of the power supply can cause system malfunction (which often shows up as "glitches" on the video screen), and may damage the power supply.

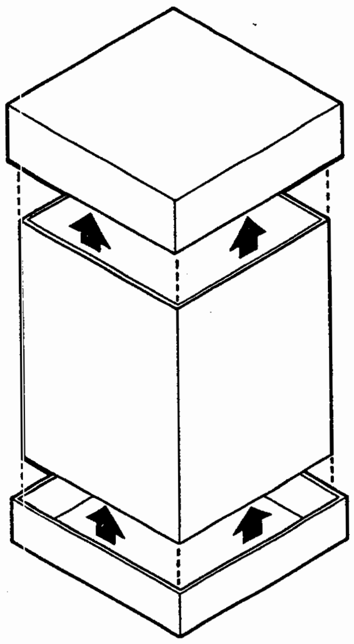
1.3. Receipt and Unpacking Instructions

Note that the shipping weight of the Sun Workstation and its shipping carton is approximately 105 pounds. The weight of the workstation itself is about 90 pounds.

1. When you receive your shipment, inspect all shipping cartons *immediately* for evidence of damage. If any shipping carton is severely damaged, request that the carrier's agent be present when the carton is opened. If the carrier's agent is not present when a carton is opened and the contents are found to be damaged, keep all contents and packing materials for the agent's inspection.
2. To open the shipping carton, cut the binding straps on it with a knife or scissors. The shipping carton is in the form of a square 'tube' with two lids on it, one on the bottom, and one on the top. The workstation is encased in styrofoam packing materials inside the lids.
3. Remove the top lid from the 'tube', and you will find the keyboard, a set of diagnostic PROM's*, the distribution tape, the power cord, and an envelope containing manuals in the recesses in the top of the packing material. Remove these items.

* The Sun Workstation is shipped with an auxiliary set of PROM's which contain diagnostic programs to aid in the checkout of the system in the event of failure.

4. Lift out the rubber and styrofoam packing material (the fit is fairly tight; try rocking it out). At this point you may find your mouse and pad tucked along the side of the monitor; if you ordered an 84 MByte disk subsystem, these will be packed with it instead.



5. Remove the square 'tube' from its bottom lid. Don't try to lift the Sun Workstation out of the shipping carton: it's too heavy. When you remove the 'tube' from the bottom lid, the workstation is left standing in the bottom piece of packing material. You can then lift the workstation out of the bottom shipping material, remove the plastic wrapping, and place the workstation on its working surface.

We recommend that you save the salvageable shipping cartons and packing material for future use in case the product must be reshipped.

1.4. Set-up

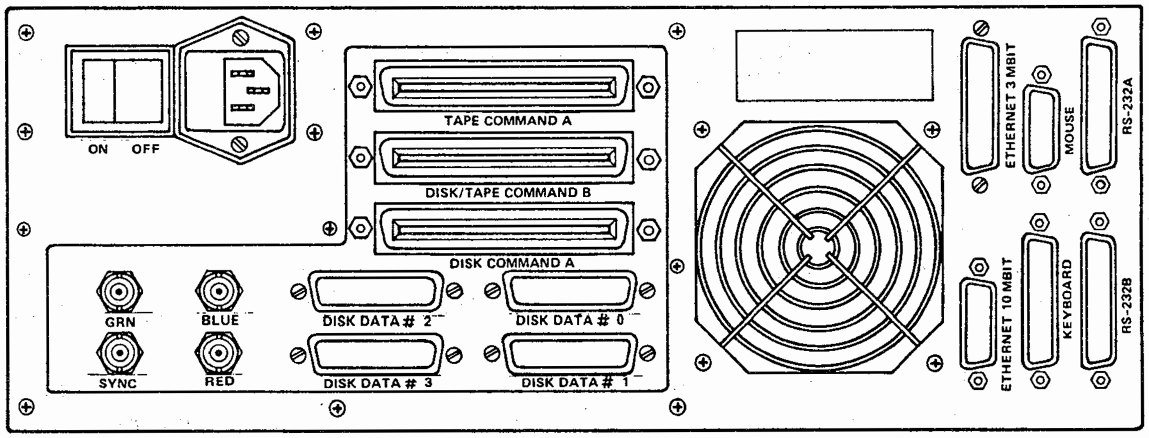
The following subsections describe how to cable up your Sun Model 100U and its basic peripherals.

CAUTION: Before plugging in the power cord of any component of your Sun system, be sure that the line power supply voltage and frequency are as specified on the back panel of your workstation. Use only three-prong (grounded) outlets. Always disconnect power before opening any system enclosure or servicing any system component. All servicing should be performed by qualified personnel.

1.4.1. Keyboard and Mouse

The Sun keyboard should be plugged into the connector labelled "KEYBOARD" on the workstation back panel. If you wish to use the keyboard as your console input device (as is normally done) you must power-on the workstation *AFTER* plugging in the keyboard. The Sun resident PROM monitor will try to use serial port A for input if it does not find an attached keyboard.

Plug the mouse into the "MOUSE" connector on the backpanel.



1.4.2. Ethernet

If you purchased a system with an Ethernet, your system is shipped with the Ethernet Controller Board installed in your workstation's Multibus cardcage. You will receive a package containing the transceiver and transceiver cable necessary to complete implementation of the Ethernet for a single workstation, and an *Ethernet Controller Reference Manual*. The coaxial cable necessary to implement a network with multiple machines may be purchased separately from Sun.

Setting up an Ethernet with all Sun-supplied components is fairly straightforward:

1. Terminate the 50 ohm coaxial cable by placing a transceiver at both ends. Screw the cable into the transceiver N connectors; cover the open N connector on each transceiver with a terminator. In lieu of a transceiver, you may terminate the cable with two barrel connectors and terminators. Handle the coaxial cable with some care, as it is fragile; don't run it in an area where it may be run over.
2. For each workstation, plug one end of the transceiver cable into the 15-pin D connector on the transceiver, and the other end into the "ETHERNET 10 MBIT" connector on the workstation backplane.

Please note that there are certain cabling limitations which must be observed for proper Ethernet implementation:

Ethernet Cabling Limitations	
MAXIMUM contiguous length of coaxial cable segments	500.0 meters
MINIMUM distance between transceivers	2.5 meters
MAXIMUM length of transceiver cable	50.0 meters

For more information, see the Ethernet manual shipped with your system. If you have questions about repeaters or non-stardard Ethernet configurations, please contact Sun Microsystems.

1.4.3. Asynchronous Serial Ports

You may attach a terminal, modem, printer, plotter, or other device which uses the RS-232 interfaces, to one of the serial port connectors labelled "RS232 A" and "RS232 B" on the back panel.

The serial ports on the Sun Workstation were designed primarily for connecting output peripherals, such as printers and plotters, and can drive these output lines at speeds up to 19.2 Kbaud. Both ports provide the CTS, RTS, DTR, DSR, and DCD control lines required by some devices (such as modems) in addition to the transmit and receive lines. Both ports are also wired as DTE (Data Terminal Equipment) ports, and thus permit direct connection of modems. Computers and other DTE devices can be connected using the null modem cable supplied by Sun. See the *Asynchronous Serial Ports* section of the *Hardware Configuration and Expansion* chapter for wiring specifications.

NOTE that older workstations (shipped prior to the 1.0 Release) have serial port A configured as a DCE (Data Communications Equipment) and serial port B configured as a DTE. In addition, only serial port A on the older workstations has full modem control. This means that external cables built for the SUN-1 or SUN-1.5 CPU may have to be modified to work correctly with the SUN-2 CPU, or a null modem cable (such as that supplied by Sun) may be required. See the *Asynchronous Serial Ports* section of the *Hardware Configuration and Expansion* chapter for wiring specifications.

2. UNPACKING AND SETTING UP THE 150U

This chapter outlines how to unpack and set up your new Sun Model 150U and some of its peripherals (subsystems are covered in the chapter *Subsystem Set-up*).

> If you are *upgrading* your Sun Model 150 rather than starting with a system 'fresh from the box,' please heed the warning on the front page of this manual. SUN-1 systems which are upgrading must not attempt to install this release of the system until all Multibus memory has been removed or disabled, and — for an upgrade from a Xylogics 440 to a Xylogics 450 — cabling has been re-routed. Make sure you have followed Sun's hardware upgrade procedures before proceeding.

If you are adding boards and/or peripherals to an existing system, the chapter on *Hardware Configuration and Expansion* will give you the information you need.

2.1. Safety Precautions

DO NOT attempt procedures which are not described in this manual; if you do so, you may void your warranty. Please refer all servicing not described in this document to qualified service personnel. If in doubt, contact your authorized Sun service representative.

Observe common-sense safety precautions as you would for any electrical or electronic equipment. Always disconnect power before opening any system enclosure.

> *Even after power is disconnected from the workstation, very high voltages remain in the capacitors behind the picture tube. It takes about ten minutes for the capacitors to lose their charge.*

The following sections contain set-up and configuration specifications for the Sun Rackmount Workstation. Consult the additional hardware manuals supplied with your workstation (and with user-supplied Multibus accessory boards, if any) for further information and precautions.

2.2. Environmental Considerations

This section describes the environmental requirements for the Sun Model 150U.

Attention: This equipment generates, uses, and can radiate radio frequency energy and if not installed and used in accordance with the instructions manual, may cause interference to radio communications. It has been tested and found to comply with the limits for a Class A computing device pursuant to Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference when operated in a commercial environment. Operation of this equipment in a residential area is likely to cause interference in which case the user at his own expense will be required to take whatever measures may be required to correct the interference.

2.2.1. Physical Environment

The Sun Model 150U is manufactured for the following physical environment:

	Operating	*Non-Operating*
Ambient Temperature	10 ° C - 40 ° C	-40 ° C - 65 ° C
Humidity (Non condensing)	5% - 90%	5% - 90%
Altitude	10,000 feet	40,000 feet

2.2.2. Power Drain Requirements

Model 150U rackmount systems are supplied with a 1000 watt, four-output power supply, which delivers + 5, –5, + 12, and –12 volts. The power supply is rated as follows:

Voltage	*Maximum Current in Amps*
+ 5	150.0
–5	5.0
+ 12	10.0
–12	10.0

It is unlikely that a full cardcage will exceed the ratings of this supply. The table which follows shows the power consumption for Sun-supplied boards. Boards documented are current with this release.

PART NUMBER	*BOARD*	*AMPS @ + 5V*	*TOTAL WATTS*
370-1012	Xylogics SMD Disk Controller Board	8.0	45
501-0289	Color Display Controller	6.0	36
590-0059	Black and White Display Controller	5.0	25
590-0217	Interphase SMD Disk Controller Board	4.0	23
590-0288	10Mbit Ethernet Board	5.0	31
590-0502	1/2-inch Tape Controller Board	4.0	20
590-0526	1/4-inch Tape Controller Board	3.0	15
590-1007	SUN-2 CPU Board, Keyboard, and Mouse	6.0	30
590-1013	1M Memory Board	3.0	15
590-1013	SUN-2 (low power) Memory Board	2.5	13

Note that the keyboard and mouse draw their power from the CPU board.

Review the above list carefully to determine exactly what configuration of boards you can have in the cardcage. If you get boards from other suppliers, make sure you obtain the current consumption information from them, so that you can factor that information in with the list above. Overloading the rated capacity of the power supply can cause system malfunction (which often shows up as "glitches" on the video screen), and may damage the power supply.

2.3. Receipt and Unpacking Instructions

The basic Model 150U is shipped in two separate cartons: one contains the Sun Workstation video monitor on its pedestal, and the second contains a rack-mountable metal enclosure with the workstation cardcage and power supply. Each subsystem ordered comes in its own carton — see the chapter *Subsystem Set-up* for unpacking instructions. You may also find yourself with a few small boxes: the Sun System documentation is shipped in its own package, as are the brackets and power supply for the 169 MByte disk.

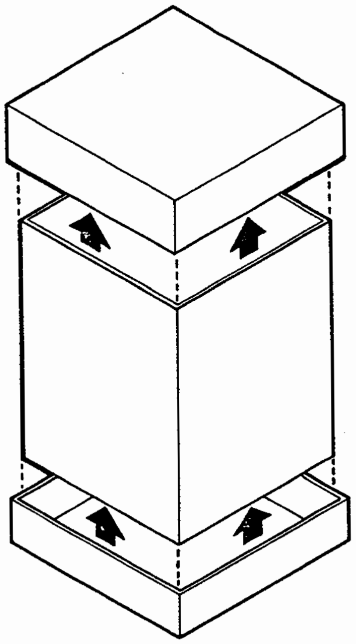
When you receive your shipment, inspect all shipping cartons *immediately* for evidence of damage. If any shipping carton is severely damaged, request that the carrier's agent be present when the carton is opened. If the carrier's agent is not present when a carton is opened and the contents are found to be damaged, keep all contents and packing materials for the agent's inspection.

The following two subsections describe how to unpack the Model 150U monitor and cardcage enclosure. Mounting instructions for the cardcage enclosure are also given.

2.3.1. 150U Monitor — Unpacking

Note that the shipping weight of the Model 150U monitor and its shipping carton is approximately 80 pounds. The weight of the monitor itself is about 70 pounds.

1. To open the monitor's shipping carton, cut the binding straps on it with a knife or scissors. The shipping carton is in the form of a square 'tube' with two lids on it, one on the bottom, and one on the top. The monitor is encased in styrofoam packing materials inside the lids.



2. Remove the top lid from the 'tube', and you will find the keyboard, the power cord, a set of diagnostic PROM's*, the distribution tape, and an envelope containing manuals in the recesses in the top of the packing material. Remove these items.
3. Lift out the rubber and styrofoam packing material (the fit is fairly tight; try rocking it out). At this point you may find your mouse and pad boxed and tucked along the side of the monitor; if you ordered an 84 MByte disk subsystem, these will be packed with it instead.
4. Remove the square 'tube' from its bottom lid. Don't try to lift the monitor out of the shipping carton: it's too heavy. When you remove the 'tube' from the bottom lid, the monitor is left standing in the bottom piece of packing material. Two people can then lift the monitor out of the bottom shipping material, remove the plastic wrapping, and place the monitor on its working surface.

We recommend that you save the salvageable shipping carton and packing material for future use in case the product must be reshipped.

* The Sun Workstation is shipped with an auxiliary set of PROM's. These PROM's contain diagnostic programs to aid in the checkout of the system in the event of failure.

2.3.2. 150U Cardcage Enclosure — Unpacking and Mounting

Note that the shipping weight of the Model 150U enclosure and its shipping carton is approximately 60 pounds. The weight of the enclosure itself is about 50 pounds.

1. Open the enclosure's shipping carton as you did the monitor's.
2. Remove the top lid from the shipping 'tube', and you will find a cardboard piece with the door for the enclosure; power cord; and extension keyboard, mouse, and video cables on it. Take these items off and set them aside.
3. Pull up the cardboard and styrofoam packing material, and you will expose the unit, covered in plastic.
4. Remove the square 'tube' from its bottom lid. Don't try to lift the enclosure out of the shipping carton: it's too heavy. When you remove the 'tube' from the bottom lid, the unit is left standing in the bottom piece of packing material. Two people can then lift the enclosure out of the bottom shipping material, using the handles on its sides.

   We recommend that you save the salvageable shipping carton and packing material for future use in case the product must be reshipped.
5. The M150U enclosure comes supplied with a set of clips containing captive nuts, and a set of bolts for bolting the unit into a standard 19-inch rack. To mount the enclosure, clip the clips into the holes in the vertical metal rail in the rack. Then commandeer two cohorts to position the unit in the rack while you secure it by the bolts provided.
6. Lastly, slip the enclosure front door on. If you mount the unit with the door on, you run the risk of shearing off the front light and switch.

2.4. Set-up

The following subsections describe how to cable up your Sun Model 150U and its basic peripherals.

CAUTION: Before plugging in the power cord of any component of your Sun system, be sure that the line power supply voltage and frequency are as specified on the back panel of your workstation. Use only three-prong (grounded) outlets. Always disconnect power before opening any system enclosure or servicing any system component. All servicing should be performed by qualified personnel.

2.4.1. Keyboard and Mouse

The Sun keyboard should be plugged into the connector labelled "KEYBOARD" on the cardcage back panel. If you wish to use the keyboard as your console input device (as is normally done) you must power-on the workstation *AFTER* plugging in the keyboard. The Sun resident PROM monitor will try to use serial port A for input if it does not find an attached keyboard.

Plug the mouse into the "MOUSE" connector on the back panel.

2.4.2. Ethernet

If you purchased a system with an Ethernet, your system is shipped with the Ethernet Controller Board installed in your workstation's Multibus cardcage. You will receive a package containing the transceiver and transceiver cable necessary to complete implementation of the Ethernet for a single workstation, and an *Ethernet Controller Reference Manual*. The coaxial cable necessary to implement a network with multiple machines may be purchased separately from Sun.

Setting up an Ethernet with all Sun-supplied components is fairly straightforward:

1. Terminate the 50 ohm coaxial cable by placing a transceiver at both ends. Screw the cable into the transceiver N connectors; cover the open N connector on each transceiver with a terminator. In lieu of a transceiver, you may terminate the cable with two barrel connectors and terminators. Handle the coaxial cable with some care, as it is fragile; don't run it in an area where it may be run over.
2. For each workstation, plug one end of the transceiver cable into the 15-pin D connector on the transceiver, and the other end into the "ETHERNET 10 MBIT" connector on the cardcage enclosure backplane.

Please note that there are certain cabling limitations which must be observed for proper Ethernet implementation:

Ethernet Cabling Limitations	
MAXIMUM contiguous length of coaxial cable segments	500.0 meters
MINIMUM distance between transceivers	2.5 meters
MAXIMUM length of transceiver cable	50.0 meters

For more information, see the Ethernet manual shipped with your system. If you have questions about repeaters or non-stardard Ethernet configurations, please contact Sun Microsystems.

2.4.3. Asynchronous Serial Ports

You may attach a terminal, modem, printer, plotter, or other device which uses the RS-232 interfaces, to one of the serial port connectors labelled "RS232 A" and "RS232 B" on the back panel.

NOTE that older workstations (shipped prior to the 1.0 Release) have serial port A configured as a DCE (Data Communications Equipment) and serial port B configured as a DTE. In addition, only serial port A on the older workstations has full modem control. This means that external cables built for the SUN-1 or SUN-1.5 CPU may have to be modified to work correctly with the SUN-2 CPU, or a null modem cable (such as that supplied by Sun) may be required. See the *Asynchronous Serial Ports* section of the *Hardware Configuration and Expansion* chapter for wiring specifications.

The serial ports on the Sun Workstation were designed primarily for connecting output peripherals, such as printers and plotters, and can drive these output lines at speeds up to 19.2 Kbaud. Both ports provide the CTS, RTS, DTR, DSR, and DCD control lines required by some devices (such as modems) in addition to the transmit and receive lines. Both ports are also wired as DTE (Data Terminal Equipment) ports, and thus permit direct connection of modems. Computers and other DTE devices can be connected using the null modem cable supplied by Sun. See the ***Asynchronous Serial Ports*** section of the *Hardware Configuration and Expansion* chapter for wiring specifications.

3. SUBSYSTEM SET-UP

This chapter gives step by step instructions on how to unpack, mount, and cable up the various disk and tape subsystems which may be supplied with either the Model 100U or Model 150U. Mounting instructions are of course relevant only for the Model 150U; please skip these steps if you are dealing with a Model 100U.

3.1. M2312K (Fujitsu 84 MByte) Disk Subsystem

Your Sun Workstation may be supplied with a disk subsystem which is the Fujitsu Model M2312K disk drive. The Fujitsu disk drive uses the industry standard SMD interface which is supported by either the Interphase SMD 2180 disk controller or the Xylogics 450 SMD disk controller supplied with your disk subsystem.

Note that if you are upgrading from an Interphase controller to a Xylogics controller, the DIP switches on the top surface of the disk drive must be re-set. For an Interphase controller, the proper settings are: on DIP Switch 2, switches 2, 4, 5, and 7 *ON*; switches 1, 3, and 6 *OFF*; on DIP Switch 3, switch 3 *ON*; all others *OFF*. For a Xylogics 440 or 450 controller, the proper settings are: on DIP Switch 2, all switches *ON*; on DIP Switch 3, switch 3 *ON*; all others *OFF*.

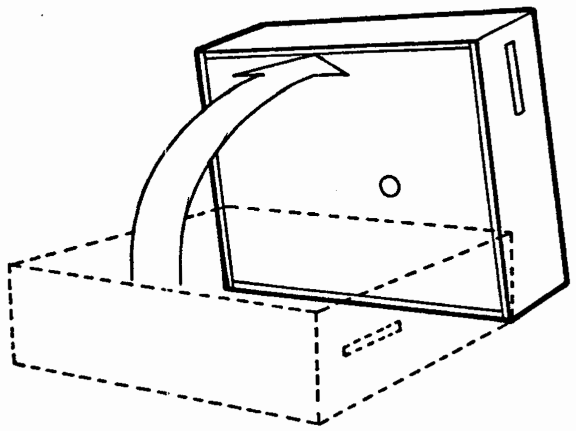
Two disk drives may be attached to a single SMD controller. Fujitsu disk drives are shipped in in a box nested inside another box. The package containing the disk drive is wrapped in a plastic bag. To unpack, mount, and cable up the subsystem, follow the procedures below.

Note that the shipping weight of the 84 MByte Disk Subsystem and its shipping carton is approximately 85 pounds. The weight of the unit itself is about 75 pounds.

1. To open the 84 MByte disk's shipping carton, cut the binding straps on it with a knife or scissors.
2. Open the outer shipping carton and remove the top styrofoam piece. This exposes packing foam containing several components (these vary depending on your system configuration): manuals, a power cord, Ethernet cables and transceiver, mouse and pad, and — for a 150U only — the mounting hardware for the disk subsystem. Remove these items and set them aside.
3. Remove the foam piece, and take out the cardboard below it, to expose four styrofoam corner pieces and another box. Remove the inner box from the shipping carton.
4. Peel off the tape which seals this box. DON'T CUT OPEN the box, or you may damage the finish on the unit. Remove the metal unit — this holds the 84 MByte disk subsystem, and may also contain a 1/4-inch tape drive.

   We recommend that you save the salvageable shipping carton and packing material for future use in case the product must be reshipped.
5. If you are mounting the subsystem in a 19-inch rack, proceed with the next three steps; otherwise, unlock the heads on the disk drive by following the procedure in step 7, and then proceed with cabling in step 9.

6. To mount the subsystem, first mount the slides at the chosen position in the rack.
7. Before *carefully* sliding the subsystem into the rack, you must unlock the heads on the disk drive. The head lock for the 84 MByte drive is on the bottom of the subsystem enclosure, so you must either tilt or elevate the enclosure to get at it. To unlock the heads, use a large screwdriver to rotate the white plastic knob on the bottom of the unit to the *OFF* position. ALWAYS lock the heads by rotating this knob to the *ON* position before moving the drive — even from table to table. More detailed instructions are in the Fujitsu Microdisk Drives CE Manual.



8. Now, slide the subsystem into the rack.
9. The disk subsystem is attached to either the Model 100U workstation backpanel or the Model 150U cardcage enclosure backpanel via two flat ribbon cables. The (wider) control cable comes out of the back of the Fujitsu disk drive enclosure, and is labelled "DISK". This cable plugs into the connector labelled "DISK COMMAND A" on the Sun backpanel. The (narrower) data or "B" cable for each of up to four drives plugs into one of the four SMD-B connectors labelled "DISK DATA #x" on the back panel, where *x* is 0 through 3. The first disk drive must be cabled to connector "DISK DATA #0"; the second should be cabled to "DISK DATA #1".

   The disk controller installed in the cardcage of both the Model 100U and the 150U is a Xylogics 450 controller which *can* control up to four drives; however, the system is configured for a maximum of two drives per controller. As shipped, each drive is assumed to be the first drive, that is, unit number 0.

The Fujitsu subsystem is very simple to operate; it has only a power cord and an on-off switch. However, we suggest that you take some time to look through the installation and operation sections of the Fujitsu manual before using the drive, to understand general operating requirements and precautions.

3.2. M2284 (Fujitsu 169 MByte) Disk Subsystem

The 169 MByte disk subsystem is currently shipped in several boxes: the disk drive itself, the power supply for the subsystem, and the mounting hardware for the Model 150U each come packaged separately.

When you follow the unpacking and mounting procedures below, we recommend that you have the Fujitsu *M228X Fixed Disk Unit, Customer Engineering Manual* close at hand. This manual contains information essential to the process.

Note that the shipping weight of the 169 MByte Disk Subsystem and its shipping carton is approximately 90 pounds. The weight of the unit itself is about 80 pounds.

1. Open the 169 MByte disk's outer shipping carton by removing the four white plastic pins on the lid (squeeze together wings and pull), and then lifting off the lid.
2. When the lid is removed, the internal carton containing the disk subsytem is exposed. Lift out the internal carton, and take off the corner pads.
3. Cut open the internal carton, and remove the styrofoam packing material. You expose the power and control cables, on the sides of the box; the front panel for the unit; and the subsystem itself, wrapped in a polyethylene bag.
4. With another person, lift the unit out of its bottom pad. Grasp the unit at the bottom to avoid damage to the sub-assemblies.

   We recommend that you save the salvageable shipping carton and packing material for future use in case the product must be reshipped.
5. If you are mounting the subsystem in a standard 19-inch rack, continue with the next step; otherwise, continue with step 7.
6. To mount the subsystem, first mount the slides at the chosen position in the rack. Then — if the unit is sufficiently elevated to allow you to get underneath it easily — carefully slide the unit into the rack. If your drive is positioned in the rack so that you can't get to its underside — which is necessary to unlock the spindle (procedure below) — don't mount the subsystem until you have completed the next step.

   When you are *completely through* with sliding the disk around, you can screw the pair of retaining bars onto the front of the subsystem, and snap on the front panel.
7. Before applying power to the subsystem, you must unlock the spindle, the actuator, and the motor on the disk drive. These are locked during shipment to protect the subsystem, and must be locked any time the unit is moved. For procedures, see the Fujitsu *M228X Fixed Disk Unit, Customer Engineering Manual*, sections 3.4.3, 3.4.4, and 3.4.5, respectively.
8. The M2284 disk drive is connected to its power supply unit via three cables. There are two power cables known as the 'A' (AC power cable) and 'B' (DC power cable) power cables, and there is a power control cable for power-up sequencing when multiple disk drives are installed.
9. There are two switches of interest on the power supply unit. The POWER ON/OFF switch must be left ON all the time after installation. The LOCAL/REMOTE switch must be set to REMOTE at all times.
10. After connecting the M2284 to its power supply, hook up its control and data cables as follows. The M2284 is connected to the cardcage backpanel via two flat ribbon cables. The wider of the cables is a 60-pin control cable. Plug the control cable into the socket entitled 'DISK COMMAND A' on the backpanel. The narrower cable is the disk data cable, and

must be plugged into the socket labelled 'DISK DATA #x' on the backpanel, where *x* is 0 through 3. The first disk drive must be cabled to connector "DISK DATA #0"; the second should be cabled to "DISK DATA #1".

The disk controller installed in the cardcage of both the Model 100U and the 150U is a Xylogics 450 controller which *can* control up to four drives; however, the system is configured for a maximum of two drives per controller. As shipped, each drive is assumed to be the first drive, that is, unit number 0.

3.3. M2351 (Fujitsu 474 MByte) Disk Subsystem

The 474 MByte disk subsystem is shipped in a single carton, along with its kits and cables. The drive itself is shipped in nested boxes, and wrapped in a polyethelyne bag. To unpack, mount, and cable up the M2351, follow the procedures below.

We recommend that you have the Fujitsu *M2351A/AF Mini-Disk Drive CE Manual* close at hand as you follow the instructions; it contains essential information.

1. Open the 474 MByte disk's outer shipping carton by cutting the binding straps with a knife or scissors, and opening the flaps.
2. When the box flaps are opened, the shock absorbers surrounding the internal carton are exposed. Remove the shock absorbers, and have AT LEAST two people lift out the internal carton.
3. Cut open the internal carton, and you expose the subsystem itself, wrapped in a polyethylene bag.
4. With another person, lift the unit out of its bottom pad. Grasp the unit at the bottom to avoid damage to the sub-assemblies.

   We recommend that you save the salvageable shipping carton and packing material for future use in case the product must be reshipped.
5. Remove the cover on the drive. Loosen the two phillips screws that hold the cardcage upright, and gently pivot the cardcage to a horizontal position to expose the 'Vibration Preventative Block' under the disk enclosure housing. Remove the 'VPB'. Return the cardcage to its upright position, and tighten the screws.
6. Locate the rotary actuator ('VCM') at the rear of the drive, near the cables — there is a small label which says 'LOCK ←→ FREE'; to the right of this label as you look at the back of the drive is a phillips screw. Loosen the screw and rotate the locking lever to the unlocked position; tighten the screw in this second hole. ALWAYS lock the rotary actuator before moving the drive. For an illustration, see the Fujitsu manual, section 2.3.
7. For mounting instructions, see the Fujitsu manual, section 2.4.
8. The M2351 is connected to the cardcage backpanel via two flat ribbon cables. The wider of the cables is a 60-pin control cable. Plug the control cable into the socket entitled 'DISK COMMAND A' on the backpanel. The narrower cable is the disk data cable, and must be plugged into the socket labelled 'DISK DATA #x' on the backpanel, where *x* is 0 through 3. The first disk drive must be cabled to connector "DISK DATA #0"; the second should be cabled to "DISK DATA #1".

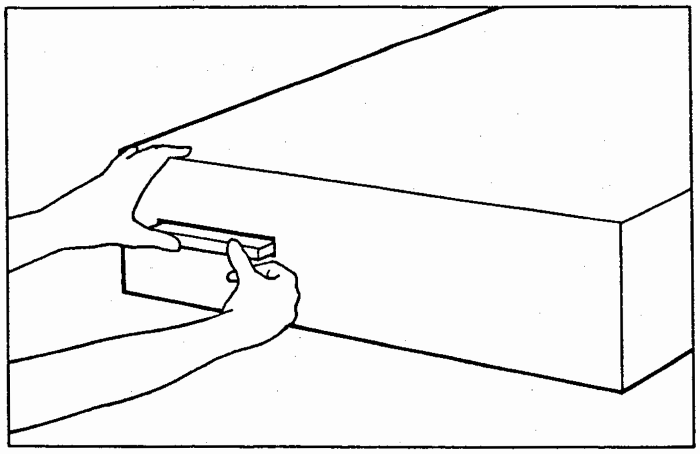
The disk controller installed in the cardcage of both the Model 100U and the 150U is a Xylogics 450 controller which *can* control up to four drives; however, the system is configured for a maximum of two drives per controller. As shipped, each drive is assumed to be the first drive, that is, unit number 0.

3.4. Quarter-Inch Magnetic Tape Subsystem

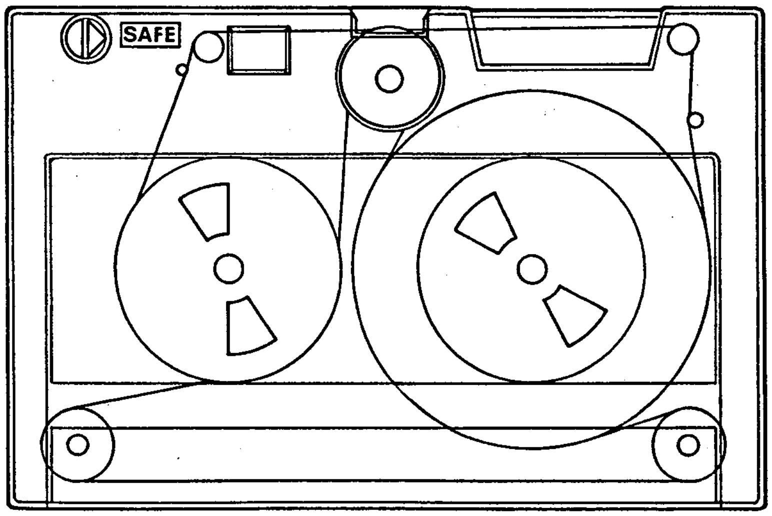
The Sun Workstation may be shipped with a quarter-inch magnetic tape cartridge subsystem as part of the Fujitsu disk drive. A tape cartridge can store up to 20 Megabytes of data.

The magnetic tape unit is controlled via a controller board manufactured by Sun Microsystems. The tape controller contains the hardware for managing the magnetic tape, and also contains 256 Kbytes of memory which is accessible on the Multibus (**which must be disabled if you are using the board with a SUN-2 CPU board**). The tape controller connects to the tape drive via a 50-pin flat ribbon cable. The tape controller cable comes out of the back of the Fujitsu disk enclosure, and is labelled "TAPE". This cable plugs into the back panel connector labelled "TAPE COMMAND A".

The tape cartridge is in the correct orientation when the little window with the word "SAFE" in it is at the top left hand corner of the cartridge. Firmly push the cartridge into the slot in the front of the tape drive. It will suddenly go all the way in with a definite "click". The tape is now ready for use.



To protect the tape from being written on, turn the arrow that is on the left of the window marked "SAFE" until the arrow points at the word "SAFE" — the tape cannot be written when the arrow is in this position. Turning the arrow 180 degrees so that it points away from the word "SAFE" allows writing on the tape.



3.5. Nine-Track Magnetic Tape Subsystem

The Sun Workstation may be shipped with a Control Data Streaming Tape Unit (STU) in the 92180 family of tape drives. This tape unit is an industry standard tape unit which can operate at either 25 inches per second start/stop mode, or at 100 inches per second streaming mode.

The magnetic tape unit is controlled via a Computer Products Corporation TAPEMASTER tape controller. Up to eight magnetic tape drives can be controlled from the one controller.

To unpack, mount, and cable up the nine-track magnetic tape subsystem, follow the procedures below.

Note that the shipping weight of the 1/2-inch Streaming Tape Unit and its shipping carton is approximately 117 pounds. The weight of the unit itself is about 100 pounds.

1. Cut open the top of the Streaming Tape Unit's shipping carton with a short-bladed knife (so as not to damage the finish on the unit).
2. Fold back the flaps of the shipping carton to expose the inner tray, with manual and installation kit taped on top. Remove these items.
3. Remove the tray, and you will see the top of the unit with attached shipping frame.
4. Have a cohort grasp one side of the frame while you grasp the other; remove the unit from the carton and bottom inner tray, and place it on a table top.

   We recommend that you save the salvageable shipping carton and all packing material for future use in case the product must be reshipped.

5. Carefully cut and remove the non-metallic band securing the Streaming Tape Unit's door.
6. Remove the filler blocks which are between the upper and lower PC board rear-mounted hinges.
7. Remove the door support blocks from under the door assembly. Leave the one-inch frame support block in place until the unit is ready for rack mounting. If you're not mounting the unit, simply leave the shipping frame on.
8. Remove the filler block which is between the shipping frame and underside of the PC boards by carefully pressing downward on the block, and sliding it backward and out from under the PC boards.
9. Remove the door stud from the installation kit. Screw the threaded end into the receptacle block inside the dust cover door. The tape unit will not run if the stud is not in place to engage the interlock switch.
10. To mount the unit in a 19-inch rack, first bolt the two hinge assemblies and stiffener bar to the rack. If you're not mounting the unit, skip to step 18 for cabling instructions.
11. Support the Streaming Tape Unit in a vertical position on the shipping frame, so that you can remove the four screws and detach the unit from the frame. After unmounting the unit, make sure your cohort is around to help you lift and manuver it — definitely a two-person operation.
12. When the frame is removed, install the two hinges on the top of the unit with the provided screws and lockwashers.
13. Position the Streaming Tape Unit onto the mounting hinges. The unit must be perpendicular to the rack for the hinges to be mated.
14. Place the unit in a closed position. Mark the area at which the adjustable pawl fastener of the unit contacts the mounting rail.
15. With the unit in an open position, install the bumper assembly into the mounting rail approximately one to two inches above the point at which the pawl fastener contacts the mounting rail.
16. Adjust the bumper assembly so that the tape deck is parallel to the mounting rack when the Streaming Tape Unit is in a closed position.
17. Place the unit in a closed position and secure it with the adjustable pawl fastener. Continue turning the pawl fastener clockwise until the unit is secure against the bumper assembly.
18. The TAPEMASTER tape controller inside the Sun Workstation cardcage connects to the tape drive via two 50-pin flat ribbon cables. Look at the back of the tape drive: there is a large square circuit board with two edge connectors on it. Connect the top edge connector ("J1") via its 50-pin ribbon cable to the socket labelled "TAPE COMMAND A" on the backpanel of the cardcage enclosure. Connect the bottom ("J2") edge connector via its 50-pin ribbon cable to the socket labelled "DISK/TAPE COMMAND B" on the backpanel of the enclosure.

Please read the Control Data STU Reference Manual for more information on the nine-track drive.

4. INSTALLING UNIX FOR THE FIRST TIME

This chapter describes how to install the UNIX system for the very first time on the Sun Workstation. We assume that you are starting with a Sun Workstation fresh from the box, bare disk(s), and the distribution software either on a nine-track half-inch tape or on two quarter-inch cassette tapes. This chapter guides you through the installation procedure.

> **CAUTION**: If you are *upgrading* your Sun Model 100 or 150 rather than starting with a system 'fresh from the box,' please heed the warning on the front page of this manual. SUN-1 systems which are upgrading must not attempt to install this release of the system until all Multibus memory has been removed or disabled, and — for an upgrade from a Xylogics 440 to a Xylogics 450 — cabling has been re-routed. Make sure you have followed Sun's hardware upgrade procedures before proceeding.
>
> **CAUTION**: If you are *upgrading* your system software, rather than beginning for the first time, read the *Upgrading System Software* chapter BEFORE proceeding here. The chapter gives procedures for saving and rebuilding your existing root and */usr* file systems

If you are installing UNIX on a machine without a tape drive, see the procedures in the next chapter, *Installing UNIX on Systems Without Tape Support*.

We begin with a description of what is on the distribution tape. This is followed by an overview of the installation procedure, including some non-trivial remarks on device abbreviations, what to do when you make mistakes while performing the procedure, and what facilities are and are not available to you during installation. Then we do a walkthrough of an actual installation.

4.1. What is on the Distribution Tape?

The software needed to load the UNIX system is contained either on a 9-track magnetic tape, or on two quarter-inch tape cartridges, distributed with the system. The tape(s) contains eleven files*, as follows:

File 1	A general purpose boot program which knows how to boot from the various devices that can be attached to the Sun Workstation. The PROM monitor boots this general purpose boot program.
File 2	A copy of a program called *diag*. *Diag* is used to format and label disks.
File 3	A stand alone *copy* program which can copy from specified sources to specified destinations.
File 4	An image of a miniature version of the root file system, which contains enough machinery to get something up and running.
File 5	Copyright file.
File 6	The complete root file system for the UNIX operating system. This is on the tape in *dump* format and is loaded in by the *xtr* utility.

* Note that the boot command follows the convention of numbering the first file on the tape as file #0. If you go off the installation path, and need to load files, please remember to decrement the numbers here by 1.

File 7	Copyright file.
File 8	Copyright file. Note that this is the eighth file on the nine-track distribution tape; if the distribution is on a quarter-inch cartridge tape, this file is actually the first file on the second cartridge.
File 9	An image of the */usr* file system that was placed on the tape by the *tar*(1) command. The *setup* program transfers this file system to the disk.
File 10	A *tar* format dump of the */usr/man* (online manual sources), */usr/games* (games), and */usr/demo* (demonstration programs) directories. These directories are optional and need only be loaded onto your system if they are specifically required. Note that in a system which is being formatted for diskless users, these directories may be too big to fit into the public partition. You can exclude parts of the dump, such as demonstration programs, if you wish.
File 11	Copyright file.

4.2. Overview of the Installation Procedure

The object of the exercise here is to load the UNIX system from the magnetic tape onto the disk subsystem of your Sun Workstation. The major steps in the procedure for loading the UNIX system are:

1. Use the resident PROM monitor to determine the Ethernet addresses for this workstation or cluster of workstations. This information is required for later use.
2. Use the resident PROM monitor to boot the general purpose bootstrap program from the magnetic tape.
3. Use the bootstrap program to load the *diag* utility from the magnetic tape.
4. Format and label the disk with the *diag* utility.
5. Use the bootstrap program to load the stand alone copy utility from the magnetic tape.
6. Use the stand alone copy utility to copy the mini UNIX root file system from the tape onto the swap area on the disk.
7. Boot the mini UNIX operating system from the swap area.
8. Restore the full root file system using the *xtr* utility.
9. Boot the UNIX operating system in single-user state.
10. Run the *setup* program to configure your system.
11. Boot the full UNIX system.
12. Reconfigure the system kernel. This step is mandatory for systems with only 1 MByte of memory, and highly recommended for systems with more.
13. Configure the local network. If you have a network, the */etc/rc*, */etc/rc.local*, */etc/hosts.equiv*, and */.rhosts* files must be edited to set it up properly. Again, this step is mandatory for systems with only 1 MByte of memory, as it results in significant system performance improvements.
14. Load the manuals, demos, and games directories if desired.
15. Continue with installation of utilities such as the *mail* system and *uucp*, configuration of a line printer, etc., by following procedures in the *System Set-up and Operation* chapter.

The procedures described below walk you through what you actually have to do. In all the examples, what *you* type is shown in **bold face text like this**. Everything shown in bold face should be typed exactly as it appears. Where parts of a command are shown in *italic text like this,* they refer to a variable which you have to substitute from a selection; it is up to you to make the proper substitution.

Two very important variables in the examples are *disk,* which should be replaced with the UNIX device name for your disk:

	Device Abbreviations
Abbreviation	Device
xy	Xylogics 450/440 disk controller
ip	Interphase SMD-2180 disk controller

and *tape,* which should be replaced with the abbreviation for your tape drive:

	Device Abbreviations
Abbreviation	Device
mt	Nine-track magnetic tape
ar	Archive quarter-inch tape

For example, a common configuration would load from the Archive quarter-inch magnetic tape cartridge, onto a Fujitsu disk driven from a Xylogics 450 disk controller. In this case, *tape* would be replaced by **ar** (**Ar**chive tape) and *disk* would be replaced by **xy** (**Xy**logics disk) everywhere.

If (when) you make a typing mistake during the installation procedure, you can use the DEL key to erase and back up over incorrect characters you just typed. You can use the control-U key to kill the entire line you just typed, and start typing a new line.

If you need to get back to the monitor for any reason, you can abort by typing 'SET-UP A' (hold down the SET-UP key while typing 'A') or 'ERASE-EOF A' at any time.

During a large part of the installation procedure, you are using the single-user UNIX system. Please note that the shell you are talking to in this state is the Bourne Shell, not the C Shell — this means that there is no history facility available.

4.3. Installation Walkthrough

4.3.1. Determining Ethernet Addresses

> *This step is only necessary if you have an Ethernet board in your Sun system. You can skip to the next subsection if you don't have Ethernet.*

Power up the Sun Workstation (if you haven't already done so). You should see the PROM monitor's sign on messages. Stop the auto boot immediately by aborting — type 'SET-UP A' (hold down the SET-UP key while typing 'A') or 'ERASE-EOF A' — and then you should see the monitor's prompt, which is a > sign:

```
Self Test completed successfully.

Sun Workstation, Model Sun-1/100U or Sun-1/150U, type of keyboard
Serial #number, ROM Rev L, some_number_MBytes memory installed

Auto-boot in progress . . .

[ abort by typing 'SET-UP A' or 'ERASE-EOF A' here ]

Abort at some address
>
```

You use the workstation monitor program to examine the PROM locations where the Ethernet address is stored. The Ethernet addresses start in location fe0400 (base 16). The Ethernet address occupies six bytes. You find out what is stored there with the monitor's **e** (examine) command. The notation *return* in the example below stands for carriage return:

```
> e fe0400  return
FE0400: 0260?  return
FE0402: 8C00?  return
FE0404: 1142? q  return
>
```

The numbers you are interested in at this time are the lower three bytes of the six bytes (001142 in this example). Make a note of these numbers. You must determine the Ethernet addresses for all machines in a network if you are installing a cluster of workstations. The addresses you get from the monitor are in hexadecimal (base 16).

4.3.2. Loading the Tape

In the case of the nine-track tape, load the tape onto the tape drive and put the drive on-line. In the case of a quarter-inch tape, insert the tape cartridge into the drive as described in the section, *Quarter-Inch Magnetic Tape Subsystem* in the *Subsystem Set-up* chapter.

4.3.3. Loading the Bootstrap Program

Power up the Sun Workstation (if you haven't already done so). You should see the PROM monitor's sign on messages. Stop the auto boot immediately by aborting — type 'SET-UP A' (hold down the SET-UP key while typing 'A') or 'ERASE-EOF A' — and then you should see the monitor's prompt, which is a > sign:

> Self Test completed successfully.
>
> Sun Workstation, Model Sun-1/100U or Sun-1/150U, *type of keyboard*
> Serial #*number*, ROM Rev L, *some_number_MBytes* memory installed
>
> Auto-boot in progress . . .
>
> [**abort** by typing 'SET-UP A' or 'ERASE-EOF A' here]
>
> Abort at *some address*
> \>

You boot the general purpose bootstrap program from the tape by typing a **b** (for boot) command to the monitor. The monitor echoes the boot command back to you, with the parameters filled in. A default boot command looks like this:

> \> **b** ***tape*()**
> Boot: *tape*(0,0,0)

Remember: *tape* here should be replaced by **mt** for the nine-track tape, or **ar** for the Archive quarter-inch tape.

The bootstrap program displays a sign on message, and then displays a prompt sign of **Boot:** to indicate that it is ready to accept commands. Whenever the bootstrap program is ready for a command, it displays the **Boot:** sign at the start of the line. So, the initial dialogue with the monitor and the bootstrap program would look like this:

> \> **b** ***tape*()**
> Boot: *tape*(0,0,0)
>
> Boot:

Now the standalone boot program is in control. All programs that it loads in eventually return control to it. When the bootstrap program loads another program, it displays some numbers which are the sizes of the different portions of the program it loaded.

4.3.4. Using the Diag Utility

You boot the *diag* (disk format utility) from tape by typing a bootstrap command like this:

> Boot: *tape***(0,0,1)**

where *tape* is replaced with **mt** for the nine-track tape, or **ar** for the Archive quarter-inch tape. The bootstrap program should boot *diag* from the magnetic tape. When *diag* starts up, it displays a sign on message:

> Disk Initialization and Diagnosis
>
> When asked if you are sure, respond with 'y' or 'Y'

Diag first asks you a series of questions about the disk you will be using. The answers to these questions 'configure' *diag* to work with that specific hardware. The first thing *diag* wants to know is the type of disk controller to use:

```
specify controller:
        0 - Interphase SMD-2180
        1 - Xylogics 440 (prom set 926)
        2 - Xylogics 450
        3 - Adaptec ACB 4000 - SCSI
which one? 2
```

In this example, we specified that the controller is type 2 for a Xylogics 450 SMD controller.

Diag's next question is what address this controller occupies on the Multibus. When you give *diag* the address (see table below)*, it echoes the address back to you:

Specify controller address on the Multibus (in hex): ***select the address from the table below***

Device address: ***whatever address you selected***

The table below shows the default addresses for disk controllers*.

Disk Controller Addresses

Controller	Address (hex)
Interphase SMD-2180	40
Xylogics 450/440	ee40

Next, *diag* wants to know the unit number of the disk:

```
Which unit? 0
```

"0" is the correct response if you have one disk drive attached to the controller specified above.

Now *diag* wants to know the type of disk drive you are working with. *Diag* displays a menu of the different disks. When you have specified which disk drive you wish to operate on, *diag* then displays a table of the physical data about that disk, which includes the number of cylinders, number of alternate cylinders, number of heads, and number of sectors per track. In this example, we are formatting the Fujitsu M2312K (Sun "D84") 84 MByte disk:

* Note that if you have one Xylogics SMD Controller and one Interphase Controller in your system, you cannot use the default addresses (as the Interphase Controller sees only four bytes). The Interphase must be configured to be at address 48, and the Xylogics at ee40. See the *Hardware Configuration and Expansion* chapter for board configuration procedures.

```
Specify drive:
0 - Fujitsu-M2312K (Sun D84)
1 - Fujitsu-M2284 (Sun D169)
2 - Fujitsu-M2351 Eagle (Sun D474)
3 - Fujitsu-M2294
4 - CDC-Lark-cartridge
5 - CDC-Lark-fixed
6 - CDC-LarkII-cartridge
7 - CDC-LarkII-fixed
8 - CDC-9766
9 - CDC-9730-160
10 - CDC-9730-80
11 - CDC-9730-24
12 - CDC-9730-12
13 - Ampex-Scorpio
14 - Ampex-Capricorn
15 - Other

which one? 0

ncyl 586 acyl 3 nhead 7 nsect 34 interleave 1
status:
drive status ready
```

At this point, *diag* knows all it needs to know about the disk, and it displays its prompt:

```
diag>
```

One of the responses to this prompt can be the **h** (for help) command. If you use the **h** command, *diag* displays a list of its commands.

4.3.4.1. Formatting the Disk

Next, you use *diag*'s disk formatting function. Begin by clearing any outstanding errors (especially important if you are using an Interphase disk controller) with the **clear** command:

```
diag> clear
clear
diag>
```

Now, type the **format** command. *Diag* warns you that formatting a disk destroys all information which might already be stored on that disk, and asks for confirmation before going ahead and doing the job:

```
diag> format
format/verify, DESTROYS ALL DISK DATA!
are you sure? y
# of surface analysis passes ? 5
```

When formatting a disk for the first time, five surface analysis passes is a reasonable number. The surface analysis phase of the formatting is for detecting bad spots on the disk and mapping them to alternate places. The format phase takes about 30 minutes for one surface analysis pass, using an Interphase SMD-2180 disk controller, and slightly less time using a Xylogics 450 or 440 controller. The time to format and analyze for *n* passes is about *n* times 30 minutes. It is well worth taking the time to do this when formatting a new disk. *Diag* then formats the disk, and displays the current cylinder number as it formats each cylinder. You can use just one surface analysis pass if you are reformatting a disk that has been formatted before, and is known not to have any bad spots.

At the end of the format process, *diag* displays a message telling you that it has finished, then tells you to use the label command to label the disk:

```
cyl 588 format complete - 0 bad track(s)
Use the label command to label the disk
diag>
```

This is the next step.

4.3.4.2. Labelling the Disk

A disk must be labelled after it has been formatted. The purpose of a label is to record (in a well known place on the disk) information about how the disk is divided up into partitions for different purposes such as paging space and file systems.

Diag labels a disk in response to the **label** command. Here is an example of using *label* to write a label on the disk which was formatted in the discussion above. When you give the command to *diag*, it asks if you want to use the logical partition map that is 'built in' to the program:

```
diag> label
label this disk...
OK to use logical partition map 'Fujitsu-M2312K (Sun D84)' ? y
Are you sure you want to write? y
```

Then, when *diag* has labelled the disk, it verifies the label it has just written. There is in fact a separate verify command, but *diag* automatically does a verify as part of the labelling process:

```
verify label
id: <Fujitsu-M2312K cyl 586 alt 3 hd 7 sec 34>
      Partition a: starting cyl=0, # blocks=15884    [ #s vary with disk controller ]
      Partition b: starting cyl=67, # blocks=33440
      Partition c: starting cyl=0, # blocks=131264
      Partition g: starting cyl=208, # blocks=81760
diag>
```

This is the end of the disk formatting and labelling process. Now you get back to the bootstrap program by typing the **q** (quit) command, and the bootstrap displays its prompt sign again:

```
diag> q

Boot:
```

4.3.5. Loading the Mini UNIX System

After the disk is formatted and labelled, the next job is to load in the mini UNIX root file system from the tape. This is done by loading in the standalone copy program which is the third file on the tape. The copy program prompts you for the source and destination of the copy, as shown here. (As always, the example assumes that ***tape*** stands for a tape device — such as **mt** for the nine-track tape, or **ar** for the Archive quarter-inch tape — and that ***disk*** stands for a disk device — such as **xy** for the Xylogics disk controller, or **ip** for the Interphase disk controller):

```
Boot: tape(0,0,2)
20480+ 4096+ 113384 bytes        [ #s vary with system level ]
Standalone Copy
From: tape(0,0,3)
To: disk(0,0,1)
```

Note that after you have typed in the tape information for the 'From' prompt, the copy program moves the tape for about ten seconds before displaying the 'To' question. Copying in the mini-UNIX system takes about three minutes, using a half-inch tape, and about six minutes, using a quarter-inch cartridge. At the end of the copy, the copy program returns control to the bootstrap program:

```
Copy completed

Boot:
```

4.3.6. Booting the Mini UNIX System

Next, you tell the bootstrap program to boot the mini-UNIX system from the disk. Because this mini-UNIX system boot is an unusual one, you must specify the **–a** (for ask me) option on the boot command, and also the **–s** (come up single user) option, as follows:

```
Boot: disk(0,0,1)vmunix –as
243712+ 30720+ 49768 start 0x4000        [ #s vary with system level ]

[ . . . about a dozen lines of configuration messages . . . ]
```

The mini-UNIX system displays some messages about the configuration of the system on which it is running, and finally displays a message asking for its root file system. The root file system at this stage is "*disk*0*", which has a special meaning to the small UNIX system. Since this notation looks ambiguous, let me specify: if you have a Xylogics disk controller, your root device is **xy0*** ; if you have an Interphase controller, it is **ip0*** — the asterisk is part of the device name:

```
root device? disk0*
```

At this point the system may gently remind you to reset the system date and time:

```
WARNING: clock gained 16845 days -- CHECK AND RESET THE DATE!
```

You can do this later, when you have finished installation, with the *date*(1) command.

4.3.7. Loading the Root File System

Now the mini-UNIX system starts up and displays its prompt sign: a # character. You then tell it to extract the real root file system from the tape with the following sequence:

```
# disc=disk tape=tape xtr
```

Replace *disk* with **xy** for the Xylogics disk controller, or **ip** for the Interphase disk controller. Replace *tape* with **mt** for the nine-track tape, or **ar** for the Archive quarter-inch tape. The *xtr* command is a shell script which creates a root file system and extracts it from the tape. The *xtr* shell displays a lot of messages which look something like this:

```
# disc=disk tape=tape xtr
+ cd /dev
+ ./MAKEDEV disk0 tape0

[ Possible messages from MAKEDEV ]

+ echo /dev/disk0a:/a:xx:1:1
+ sync
+ /etc/newfs /dev/rdisk0a
Warning: 20 sector(s) in last cylinder unallocated
/dev/rdisk0a: 15884 sectors in 71 cylinders of 7 tracks, 32 sectors
              8.1Mb in 5 cyl groups (16c/g, 1.84Mb/g, 672i/g)
super-block backups (for fsck -b#) at:
      32, 3648, 7264, 10880, 14496,
+ sync
+ /etc/fsck /dev/rdisk0a
** /dev/rdisk0a

[ . . . Informative messages from fsck . . . ]

+ /etc/mount /dev/disk0a /a
+ mt -f /dev/rtape0 rew
+ cd /a
+ /etc/restore rsf 6 /dev/rtape0

[ . . . Restoring the file system takes about ten minutes with
  a 1/2-inch tape, and twenty with a 1/4-inch cartridge . . . ]

+ rm -f /a/restoresymtable
+ cd /a/dev
+ ./MAKEDEV std disk0 tape0
+ cd /
+ /etc/umount /dev/disk0a
+ sync
+ /etc/fsck /dev/rdisk0a
** /dev/rdisk0a

[ . . . Informative messages from fsck . . . ]

+ echo Root filesystem extracted
Root filesystem extracted
#
```

At this point, the UNIX system has a complete root file system. There is as yet no */usr* file system, and that is part of the next job.

4.3.8. Booting the UNIX System

Now you must escape back to the monitor by typing 'SET-UP A' (hold down the SET-UP key while typing 'A') or 'ERASE-EOF A'. The monitor responds by displaying a message to the effect:

> Abort at *some address*

When you see the monitor > sign, you boot the UNIX system by typing a **b** command to the monitor, and the monitor responds by filling in the full details of the boot command. We must boot the UNIX system with the **–s** option (come up single user), because there is no *usr* file system as yet:

```
> b vmunix -s
Boot: disk(0,0,0)vmunix -s
Load: disk(0,0,0)boot
```

The UNIX system startup procedure is automatic. It displays a progress report as it goes through the initialization sequence.

```
Boot: disk(0,0,0)vmunix
219136+ 28672+ 29088
Sun UNIX 4.2 (Berkeley beta release) (GENERIC) #145: Tue Aug 30 20:35:13 PDT 1983
Copyright (c) 1983 by Sun Microsystems, Inc.

[ . . . Several lines of configuration messages . . . ]

#
```

4.3.9. Using Setup to Configure Your System

At this point you invoke the *setup* program to configure your system.

Setup is essentially a system configurator in three parts: it consists of a worksheet which you complete if you need a non-standard configuration, an interactive front-end which gathers the information necessary to configure your system by conducting a dialogue with you, and a non-interactive back-end which uses this information to do the actual configuration. During the dialogue, *setup* does consistency and error checking to ensure that the configuration will work. If errors are detected, they are reported to you, and you are asked to enter corrected information.

You will use *setup* differently depending on whether you are configuring

- A standalone system,
- A network server with equal-sized client partitions, or
- A network server with different-sized client partitions.

If you are configuring a standalone system, the dialogue with *setup* is fairly straightforward. Glance through the next section, *Path 1: Standalone System Configuration*, to familiarize yourself with the program's conventions.

If you are configuring a network server with one or more diskless nodes ("clients"), you can establish equal or unequal client partitions. The choice depends in part on your clients' storage needs — do they vary or are they roughly equivalent? — and also on your available resources — you may need to allocate unequal partitioning to accommodate all clients, for example. Please read the following before making the decision, as it is extremely difficult to 'undo' an established configuration.

This release of the system runs diskless workstations off one or more server disks by keeping a read-only copy of common files in the */pub* directory (partition) of one of your drives. On this drive, the server machine uses partition 'a' as its root and partition 'b' as its paging area. The remainder of the drive (partition 'g') contains the */pub* filesystem, followed by 'subpartitions' for the diskless clients. Each diskless client machine is allocated a file system area for private disk storage and a paging area (see *nd*(4) to understand how this is done). A generic abstract disk would thus look something like this:

	User	*Minimum*	*Default*
a	Server's root	7.7	7.7
b	Server's paging area	16.3	16.3
g	*/pub*	17.0	20.0
	Client 1's file system area	x.0	x.0
	Client 1's paging area	4.0	6.0
	Client 2's file system area	x.0	x.0
	Client 2's paging area	4.0	6.0

Partitions 'a' and 'b' were sized during the disk-labelling phase of *diag*, above; they are 7.7 and 16.3 MBytes respectively. The minimum configuration provided by *setup* then allocates 17 MBytes for */pub*, 4 MBytes for each client's paging area, and then equally subdivides the remaining disk space for each client's file system. These minimum values are **bare** minimums; we suggest using them only if you have to (for example, to fit two two clients on an 84 MByte disk). The default values noted above are much more livable partition sizes.

If you want to include the files containing the online manual pages, demos, or games, the public partition must be increased by the following amounts:

Optional /pub Directories	
Manual Pages	2.0 MBytes
Demos	3.6 MBytes
Games	2.0 MBytes

If you want to allocate equal amounts of disk space to each client, read through the subsection, *Path 2: Standard Server Configuration*. This path allows you to select the size of the public partition, and a size for a standard paging partition which will be given to each client. The remaining disk space will be divided equally among the clients as their file system area.

To assign different amounts of disk space to clients, read through the subsection, *Path 3: Non-standard Server Configuration*. This path allows you to select the size of the public partition, each client's paging area, and each client's file system area. Since the path for establishing a non-standard system configuration is slightly more complex than the other two paths, it requires some pre-planning. We have therefore included several copies of a worksheet which will help you consolidate the information you need before using *setup*. Please complete the worksheet before attempting configuration. If you decide not to complete the worksheet, please complete the worksheet BEFORE calling Sun Microsystems when you get hung up during configuration. Now please continue with the worksheet in the subsection, *Path 3: Non-standard Server Configuration*.

4.3.9.1. Path 1: Standalone System Configuration

This subsection provides an example of the *setup* interactive dialogue which is typical for a standalone workstation. In the example, what you might type in is shown in **bold face type like this**; whatever is simply displayed on the workstation monitor is shown in Roman type like this.

We invoke the *setup* program by typing the *setup* command. *Setup* begins by requesting global information:

```
# setup

Sun Microsystems Configuration System

Global Information

    1) network disk server
    2) standalone
    3) standalone with remote tape

Enter the number for your environment: 2

You have a standalone system; is this correct? (y/n): y
```

Note that the *setup* program asks you to verify configuration information after each 'phase' of configuration is complete. It is extremely difficult to undo system configuration, so be careful with your responses: 'y' casts things in concrete, and 'n' allows you to start the phase over again. You can also type 'q' to any prompt to quit the *setup* program and return to the shell — this allows you to annihilate what you have done and start over, if you have to.

Next, *setup* establishes your workstation's identity: name and — if you are tied into a network — address:

```
Host Information

Enter your hostname: **lucifer**

Is this workstation on a network? (y/n): **y**

Enter the hexadecimal ethernet address for lucifer: **1030**

Your hostname and ethernet address are:
    lucifer:    1030

Is this correct? (y/n): **y**
```

Note that you use the ethernet address established at the outset of the installation procedure here if your machine is on a network. Make sure you have your address at hand before starting the dialogue.

The last phase of *setup* configuration requests information about your tape subsystem:

```
Tape Information

    1) 1/4" SCSI tape (st)
    2) 1/2" magnetic tape (mt)
    3) 1/4" archive tape (ar)

Enter the number for the type of tape: (1-3): **3**

You have specified a 1/4" archive tape; is this correct? (y/n): **y**
```

When this last phase has been completed, *setup* asks you whether you want to institute the configuration you have designed and, if you confirm, proceeds to edit several of the database files:

```
You have completed the configuration questions.
Continuing will destroy any existing files under /usr
and any client partitions if you are a server.

Do you want to begin configuration? (y/n): **y**

Updating /etc/hosts
[ . . . A few lines of configuration messages . . . ]
```

If your distribution is on two 1/4-inch tape cartridges, *setup* will prompt you to change to the second cartridge about two minutes into its back-end routine. Insert the second tape and type 'RETURN' to continue the routine; it takes approximately 45 minutes to complete. When *setup* completes its back-end work, your shell prompt returns. You can then continue your installation by booting the full UNIX system, as described near the end of this chapter.

4.3.9.2. Path 2: Standard Server Configuration

This subsection provides an example of the *setup* interactive dialogue which is typical for a standard network server configuration. This path allows you to select the size of your public partition, and establish the size of a standard paging partition which is allotted to each client. The remaining disk space is equally divided to make each client's file system subpartition.

In the example, what you might type in is shown in **bold face type like this**; whatever is simply displayed on the monitor is shown in Roman type like this.

We invoke the *setup* program by typing the *setup* command. *Setup* begins by requesting global information:

```
# setup

Sun Microsystems Configuration System

Global Information

    1) network disk server
    2) standalone
    3) standalone with remote tape

Enter the number for your environment: 1

You have a network disk server system; is this correct? (y/n): y
```

Note that the *setup* program asks you to verify configuration information after each 'phase' of configuration is complete. It is extremely difficult to undo system configuration, so be careful with your responses: 'y' casts things in concrete, and 'n' allows you to start the phase over again. You can also type 'q' to any prompt to quit the *setup* program and return to the shell — this allows you to annihilate what you have done and start over, if you have to.

After determining your environment, *setup* gathers information about your disk configuration. It determines the type of disk controller being used for installation, and asks you for device information:

```
You have booted off of a Xylogics disk controller

Enter the number of disks attached to the Xylogics controller: (1-2): 2

The Xylogics disk controller has 2 disk(s):
    xy0
    xy1

Is this configuration correct? (y/n): y
```

For the remainder of the dialogue, *setup* will refer to your disk(s) by their UNIX device abbreviations. Next, *setup* begins the partitioning process for the first-named disk. If your configuration includes more than one disk on a single controller, as in our example, *setup* will completely finish partitioning the first disk and then turn to the second. If you have multiple controllers, you will be asked to repeat this entire phase for each controller. Your configuration will be validated after all disks have been partitioned. If errors are found at that time — for example, if all clients have not been allocated both file system and paging areas — you will be asked to edit your specified configuration until it is correct.

The partitioning phase of the dialogue with *setup* looks like this:

```
Disk Partition Information

Would you like to partition device xy0 into "n" equal sized clients? (y/n): y

Enter the pub partition size (nnnM or nnnK)
            - minimum is 17408K
            - default is 20480K: 17M
    Partition size rounded to 17472K.

Enter the swap partition size (nnnM or nnnK)
            - minimum is 4096K
            - default is 6144K: 8M
    Partition size rounded to 8288K.

Enter the number of clients on device xy0: (1-3): 2

Enter a client's name: adam

Enter a client's name: eve
```

You are asked to declare partition sizes in K Bytes or M Bytes, and ***setup*** takes care of rounding your partitions to cylinder boundaries for performance reasons. You can always type 'RETURN' as a response to a sizing prompt to get the default partitioning.

This partitioning phase of the dialogue sets up a ***/pub*** partition of whatever size you select (if this is the first device being partitioned), allocates a standard paging area for each of your specified clients (again, you determine the size), and carves the remaining free disk space into equally sized user subpartitions for each of them.

This completes partitioning for the first disk. If you have another controller, you are asked for its type, and the number of devices attached to it at this time, and are then asked to repeat the partitioning process for each disk. When all disks have been partitioned, ***setup*** asks for network information for each of your specified clients and your network server:

```
Network Information

Enter the hexadecimal ethernet address for each of the clients

Client adam: 1305

Client eve: 1A4B

The clients and their addresses are:

    1) adam:   1305
    2) eve:    1A4B

Are the ethernet addresses correct? (y/n): y

Server Information

Enter the name of the server:  sun

Enter the hexadecimal ethernet address for sun: 1030

The server and its ethernet address are:
    sun:   1030

Is this correct? (y/n): y
```

Note that you use the ethernet addresses established at the outset of the installation procedure here. Make sure you have each node's address at hand before starting the dialogue.

The last phase of *setup* configuration requests information about your tape subsystem:

```
Tape Information

    1) 1/4" SCSI tape (st)
    2) 1/2" magnetic tape (mt)
    3) 1/4" archive tape (ar)

Enter the number for the type of tape: (1-3): 3

You have specified a 1/4" archive tape; is this correct? (y/n): y
```

When this last phase has been completed, *setup* asks you whether you want to institute the configuration you have designed and, if you confirm, proceeds to edit several of the database files:

```
You have completed the configuration questions.
Continuing will destroy any existing files under /usr
and any client partitions if you are a server.

Do you want to begin configuration? (y/n): y

Updating /etc/hosts
[ . . . A few lines of configuration messages . . . ]
```

If your distribution is on two 1/4-inch tape cartridges, *setup* will prompt you to change to the second cartridge about two minutes into its back-end routine. Insert the second tape and type 'RETURN' to continue the routine; it takes approximately 45 minutes to complete. When *setup* completes its back-end work, your shell prompt returns. You can then continue your installation by booting the full UNIX system, as described below.

4.3.9.3. Path 3: Non-Standard Server Configuration

4.3.9.3.1. Configuration Worksheet for Path 3

The purpose of this worksheet is to allow you to gather and stabilize your system configuration before you cast it in concrete with the *setup* program. Basically the worksheet helps you determine your real storage resources, the number of users you want to support, and how you want to allocate your free disk space. We have provided several copies of the worksheet so that you can draft and redraft if necessary; it will be to your advantage to write everything out.

There are several things to keep in mind during this process. One important point is that once the disk has been divided up in a certain way, it is extremely difficult to alter that structure. It can be changed only by going through the entire first time installation procedure again. Therefore, you might give some thought to what your future requirements are before you complete the worksheet. For example, you may want to "reserve" a client or two for future diskless stations that will be added in the coming months. The storage that you allocate for those future clients does not have to remain unused. You can make such partitions available as extra mounted filesystems for the original set of clients.

Now simply fill in the blanks.

Sample Worksheet #1

1. Number of disks and size (MBytes) of each disk:

First Controller	
Disk Unit	Size
#0	
#1	

Second Controller	
Disk Unit	Size
#0	
#1	

The following table shows approximate free space for allocating client file system, client paging, and */pub* subpartitions for Sun-supplied disk subsystems. The numbers are approximate because formatted capacity depends on the type of controller being used with the drive. Use the table to determine your real total storage resources.

Available Disk Space			
Disk	Raw	Formatted	With Server
Fujitsu SMD 8-inch	84 MByte	65 MByte	40 MByte
Fujitsu SMD 14-inch	169 MByte	133 MByte	106 MByte
Fujitsu Eagle	474 MByte	384 MByte	359 MByte

2. Number of clients and size of file system and paging subpartitions for each client. Include the */pub* partition with the clients for disk units with */pub* partitions.

   Remember that *setup*'s minimum values for these subpartitions are 4 MBytes file system area + 4 MBytes paging area = 8 MBytes per client. Default values — to give you more livable figures — are 6 MBytes file system area + 6 MBytes paging area = 12 MBytes per client. The */pub* partition must be at least 17 MBytes (for a */pub* without manual pages, demos, or games); 20 MBytes is the default. If you wish to include the manual pages, demos, and games, allow 2 MBytes, 3.6 MBytes, and 2.0 MBytes respectively.

User Name	User Space	Paging Space
1 */pub*		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		
17		

3. Now, use the next page or pages to lay out your disks by blocking off your 'real resource' disk size and # MBytes for each */pub*, client file system, and client paging subpartition. Please remember that since *setup* rounds partitions to cylinder boundaries and these vary depending on controller type and disk size, these numbers will be approximate — variance will be about ± 5%. We have included the server's root and server's paging partitions ('a' and 'b') in the diagrams; these partitions are allocated automatically by *diag* (and are 7.7 MBytes and 16.3 MBytes respectively).

	Partitioned Disk #0
MBytes	*User*
5	Server's root
10	Server's root
15	Server's paging
20	Server's paging
25	Server's paging
30	
35	
40	
45	
50	
55	
60	
65	
70	
75	
80	
85	
90	
95	
100	
105	
110	
115	
120	
125	
130	
135	
140	
145	
150	
155	
160	
165	
170	
175	
180	
185	
190	
195	
200	

	Partitioned Disk #1
MBytes	*User*
5	
10	
15	
20	
25	
30	
35	
40	
45	
50	
55	
60	
65	
70	
75	
80	
85	
90	
95	
100	
105	
110	
115	
120	
125	
130	
135	
140	
145	
150	
155	
160	
165	
170	
175	
180	
185	
190	
195	
200	

	Partitioned Disk #0
MBytes	*User*
5	Server's root
10	Server's root
15	Server's paging
20	Server's paging
25	Server's paging
30	
35	
40	
45	
50	
55	
60	
65	
70	
75	
80	
85	
90	
95	
100	
105	
110	
115	
120	
125	
130	
135	
140	
145	
150	
155	
160	
165	
170	
175	
180	
185	
190	
195	
200	

	Partitioned Disk #1
MBytes	*User*
5	
10	
15	
20	
25	
30	
35	
40	
45	
50	
55	
60	
65	
70	
75	
80	
85	
90	
95	
100	
105	
110	
115	
120	
125	
130	
135	
140	
145	
150	
155	
160	
165	
170	
175	
180	
185	
190	
195	
200	

Sample Worksheet #2

1. **Number of disks and size (MBytes) of each disk:**

First Controller	
Disk Unit	Size
#0	
#1	

Second Controller	
Disk Unit	Size
#0	
#1	

The following table shows approximate free space for allocating client file system, client paging, and *Ipub* subpartitions for Sun-supplied disk subsystems. The numbers are approximate because formatted capacity depends on the type of controller being used with the drive. Use the table to determine your real total storage resources.

Available Disk Space			
Disk	Raw	Formatted	With Server
Fujitsu SMD 8-inch	84 MByte	65 MByte	40 MByte
Fujitsu SMD 14-inch	169 MByte	133 MByte	106 MByte
Fujitsu Eagle	474 MByte	384 MByte	359 MByte

2. Number of clients and size of file system and paging subpartitions for each client. Include the */pub* partition with the clients for disk units with */pub* partitions.

   Remember that *setup*'s minimum values for these subpartitions are 4 MBytes file system space + 4 MBytes paging area = 8 MBytes per client. Default values — to give you more livable figures — are 6 MBytes file system space + 6 MBytes paging area = 12 MBytes per client. The */pub* partition must be at least 17 MBytes (for a */pub* without manual pages, demos, or games); 20 MBytes is the default. If you wish to include the manual pages, demos, and games, allow 2 MBytes, 3.6 MBytes, and 2.0 MBytes respectively.

User Name	User Space	Paging Space
1 */pub*		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		
17		

3. Now, use the next page or pages to lay out your disks by blocking off your 'real resource' disk size and # MBytes for each */pub*, client file system, and client paging subpartition. Please remember that since *setup* rounds partitions to cylinder boundaries and these vary depending on controller type and disk size, these numbers will be approximate — variance will be about ± 5%. We have included the server's root and server's paging partitions ('a' and 'b') in the diagrams; these partitions are allocated automatically by *diag* (and are 7.7 MBytes and 16.3 MBytes respectively).

MBytes	Partitioned Disk #0
	User
5	Server's root
10	Server's root
15	Server's paging
20	Server's paging
25	Server's paging
30	
35	
40	
45	
50	
55	
60	
65	
70	
75	
80	
85	
90	
95	
100	
105	
110	
115	
120	
125	
130	
135	
140	
145	
150	
155	
160	
165	
170	
175	
180	
185	
190	
195	
200	

MBytes	Partitioned Disk #1
	User
5	
10	
15	
20	
25	
30	
35	
40	
45	
50	
55	
60	
65	
70	
75	
80	
85	
90	
95	
100	
105	
110	
115	
120	
125	
130	
135	
140	
145	
150	
155	
160	
165	
170	
175	
180	
185	
190	
195	
200	

	Partitioned Disk #0
MBytes	*User*
5	Server's root
10	Server's root
15	Server's paging
20	Server's paging
25	Server's paging
30	
35	
40	
45	
50	
55	
60	
65	
70	
75	
80	
85	
90	
95	
100	
105	
110	
115	
120	
125	
130	
135	
140	
145	
150	
155	
160	
165	
170	
175	
180	
185	
190	
195	
200	

	Partitioned Disk #1
MBytes	*User*
5	
10	
15	
20	
25	
30	
35	
40	
45	
50	
55	
60	
65	
70	
75	
80	
85	
90	
95	
100	
105	
110	
115	
120	
125	
130	
135	
140	
145	
150	
155	
160	
165	
170	
175	
180	
185	
190	
195	
200	

Sample Worksheet #3

1. Number of disks and size (MBytes) of each disk:

First Controller	
Disk Unit	Size
#0	
#1	

Second Controller	
Disk Unit	Size
#0	
#1	

The following table shows approximate free space for allocating client file system, client paging, and */pub* subpartitions for Sun-supplied disk subsystems. The numbers are approximate because formatted capacity depends on the type of controller being used with the drive. Use the table to determine your real total storage resources.

Available Disk Space			
Disk	Raw	Formatted	With Server
Fujitsu SMD 8-inch	84 MByte	65 MByte	40 MByte
Fujitsu SMD 14-inch	169 MByte	133 MByte	106 MByte
Fujitsu Eagle	474 MByte	384 MByte	359 MByte

2. Number of clients and size of file system and paging subpartitions for each client. Include the */pub* partition with the clients for disk units with */pub* partitions.

   Remember that ***setup***'s minimum values for these subpartitions are 4 MBytes file system space + 4 MBytes paging area = 8 MBytes per client. Default values — to give you more livable figures — are 6 MBytes file system space + 6 MBytes paging area = 12 MBytes per client. The */pub* partition must be at least 17 MBytes (for a */pub* without manual pages, demos, or games); 20 MBytes is the default. If you wish to include the manual pages, demos, and games, allow 2 MBytes, 3.6 MBytes, and 2.0 MBytes respectively.

User Name	User Space	Paging Space
1 */pub*		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		
17		

3. Now, use the next page or pages to lay out your disks by blocking off your 'real resource' disk size and # MBytes for each */pub*, client file system and client paging subpartition. Please remember that since ***setup*** rounds partitions to cylinder boundaries and these vary depending on controller type and disk size, these numbers will be approximate — variance will be about ± 5%. We have included the server's root and server's paging partitions ('a' and 'b') in the diagrams; these partitions are allocated automatically by ***diag*** (and are 7.7 MBytes and 16.3 MBytes respectively).

	Partitioned Disk #0
MBytes	*User*
5	Server's root
10	Server's root
15	Server's paging
20	Server's paging
25	Server's paging
30	
35	
40	
45	
50	
55	
60	
65	
70	
75	
80	
85	
90	
95	
100	
105	
110	
115	
120	
125	
130	
135	
140	
145	
150	
155	
160	
165	
170	
175	
180	
185	
190	
195	
200	

	Partitioned Disk #1
MBytes	*User*
5	
10	
15	
20	
25	
30	
35	
40	
45	
50	
55	
60	
65	
70	
75	
80	
85	
90	
95	
100	
105	
110	
115	
120	
125	
130	
135	
140	
145	
150	
155	
160	
165	
170	
175	
180	
185	
190	
195	
200	

	Partitioned Disk #0
MBytes	User
5	Server's root
10	Server's root
15	Server's paging
20	Server's paging
25	Server's paging
30	
35	
40	
45	
50	
55	
60	
65	
70	
75	
80	
85	
90	
95	
100	
105	
110	
115	
120	
125	
130	
135	
140	
145	
150	
155	
160	
165	
170	
175	
180	
185	
190	
195	
200	

	Partitioned Disk #1
MBytes	User
5	
10	
15	
20	
25	
30	
35	
40	
45	
50	
55	
60	
65	
70	
75	
80	
85	
90	
95	
100	
105	
110	
115	
120	
125	
130	
135	
140	
145	
150	
155	
160	
165	
170	
175	
180	
185	
190	
195	
200	

4.3.9.3.2. Setup Walkthrough for Path 3

This subsection provides an example of the *setup* interactive dialogue which is typical for the non-standard network server configuration. This path allows you to select the size of the public partition, the size of each user's paging partition, and the size of each user's file system area. In the example, what you might type in is shown in **bold face type like this**; whatever is simply displayed on the monitor is shown in Roman type like this.

We invoke the *setup* program by typing the *setup* command. *Setup* begins by requesting global information:

```
# setup

Sun Microsystems Configuration System

Global Information

    1) network disk server
    2) standalone
    3) standalone with remote tape

Enter the number for your environment: 1

You have a network disk server system; is this correct? (y/n): y
```

Note that the program asks you to verify configuration information after each 'phase' of configuration is complete. It is extremely difficult to undo system configuration, so be careful with your responses: 'y' casts things in concrete, and 'n' allows you to start the phase over again. You can also type 'q' to any prompt to quit the *setup* program and return to the shell — this allows you to annihilate what you have done and start over, if you have to.

After determining your environment, *setup* gathers information about your disk configuration. It determines the type of disk controller being used for installation, and asks you for device information:

```
You have booted off of a Xylogics disk controller

Enter the number of disks attached to the Xylogics controller: (1-2): 2

The Xylogics disk controller has 2 disk(s):
    xy0
    xy1

Is this configuration correct? (y/n): y
```

For the remainder of the dialogue, *setup* will refer to your disk(s) by their UNIX device abbreviations. Next, *setup* begins the partitioning process for the first-named disk. If your configuration includes more than one disk on a single controller, as in our example, *setup* will completely finish partitioning the first disk and then turn to the second. If you have multiple controllers, you will be asked to repeat this entire phase for each controller. Your configuration will be validated after all disks have been partitioned. If errors are found at that time — for example, if all clients have not been allocated both */usr* and paging areas — you will be asked to edit your specified configuration until it is correct.

The partitioning phase of the dialogue begins with an opportunity to allocate equal partitioning (Path 2); this is followed by optional allocation of the */pub* subpartition:

```
Disk Partition Information

Would you like to partition device xy0 into "n" equal sized clients? (y/n): n

Partitions for disk xy0 - 40880K bytes free

Do you want the public partition on this disk unit? (y/n): y

Enter the pub partition size (nnnM or nnnK)
            - minimum is 17408K
            - default is 20480K: 17M
    Partition size rounded to 17472K.
```

Throughout this phase, *setup* declares the amount of free space on your disk and lists any allocated partitions before asking you to alter the partitioning in any way. You are asked to declare partition sizes in K Bytes or M Bytes, and *setup* takes care of rounding your partitions to cylinder boundaries for performance reasons. You can always type 'RETURN' as a response to a sizing prompt to get the default partitioning.

The next step in partitioning is to allocate client */usr* and paging areas (which need not be on the same disk, although each client must have both). If you want to 'reserve' space for future users, allocate "other" partitions of appropriate size:

```
Partitions for disk xy0 - 23408K bytes free
1)    public:        17472K

Do you want to add or edit a partition? (y/n/1): y

Enter the partition type (user/swap/other): user

Enter the user partition size (nnnM or nnnK)
            - minimum is 4096K
            - default is 6144K: 8M
    Partition size rounded to 8288K.

Enter the name of the client: blaise
```

Continue this process until you have finished partitioning your first disk. *Setup* then allocates any remaining disk space to "other", and asks you to verify your work. If something has gone wrong, you can edit a partition by responding with its number when you are prompted for adding or editing. Both size and type fields can be changed. Our final screen might look something like this:

```
Partitions for disk xy0 - 0K bytes free
1)    public:                 17472K
2)    blaise's    user:        6272K
3)    blaise's    swap:         4256K
4)    albert's    user:        6272K
5)    albert's    swap:         4256K
6)    other:                   2352K

Is this partitioning correct? (y/n): y
```

This completes partitioning for the first disk. If you have another controller, you are asked for its type, and the number of devices attached to it at this time, and are then asked to repeat the partitioning process for each disk. When all disks have been partitioned, and the configuration has been validated, *setup* asks for network information for each of your specified clients and your network server:

```
Network Information

Enter the hexadecimal ethernet address for each of the clients

Client blaise: 1030

Client albert: 1305

Client isaac: 1A4B

Client johannes: C27F

Client immanuel: 1142

Client tycho: FAC3

The clients and their addresses are:

    1) blaise:      1030
    2) albert:      1305
    3) isaac:       1A4B
    4) johannes:    C27F
    5) immanuel:    1142
    6) tycho:       FAC3

Are the ethernet addresses correct? (y/n): y

Server Information

Enter the name of the server:  archimedes

Enter the hexadecimal ethernet address for archimedes: 4F4F

The server and its ethernet address are:
    archimedes:  4F4F

Is this correct? (y/n): y
```

Note that you use the ethernet addresses established at the outset of the installation procedure here. Make sure you have each node's address at hand before starting the dialogue.

The last phase of *setup* configuration requests information about your tape subsystem:

```
Tape Information

    1) 1/4" SCSI tape (st)
    2) 1/2" magnetic tape (mt)
    3) 1/4" archive tape (ar)

Enter the number for the type of tape: (1-3): 2

You have specified a 1/2" magnetic tape; is this correct? (y/n): y
```

When this last phase has been completed, *setup* asks you whether you want to institute the configuration you have designed and, if you confirm, proceeds to edit several of the database files:

```
You have completed the configuration questions.
Continuing will destroy any existing files under /usr
and any client partitions if you are a server.

Do you want to begin configuration? (y/n): y

Updating /etc/hosts
[ . . . A few lines of configuration messages . . . ]
```

If your distribution is on two 1/4-inch tape cartridges, *setup* will prompt you to change to the second cartridge about two minutes into its back-end routine. Insert the second tape and type 'RETURN' to continue the routine; it takes approximately 45 minutes to complete. When *setup* completes its back-end work, your shell prompt returns. You can then continue your installation by booting the full UNIX system, as described below.

4.3.10. Booting the Full UNIX System

Finally, you boot the full UNIX system. You must first halt the system, using the */etc/halt* command. This shuts the system down in an orderly manner, and returns control to the monitor:

```
# /etc/halt
Syncing disks . . . . done
>
```

and now you can simply boot the UNIX system:

```
> b
Boot: disk(0,0,0)vmunix
Load: disk(0,0,0)boot
Boot: disk(0,0,0)vmunix
Size: 266240+ 32768+ 35316 bytes
Sun UNIX 4.2 (Berkeley beta release) (GENERIC) #145: Tue Aug 30 20:35:13 PDT 1983
Copyright (c) 1983 by Sun Microsystems, Inc.

[ . . . Many lines of configuration messages . . . ]

gaia login: root
#
```

You can now use the UNIX system on this machine, and now boot any clients that will be served by this machine. Continue now with kernel configuration and, if you have an Ethernet, with network configuration. Both of these steps are crucial for system performance reasons.

4.3.11. Kernel Configuration

Sun Microsystems' implementation of UNIX provides for a *configurable kernel* — that is, certain system parameters that were hardwired in previous implementations can now be changed. As a final step in system installation, all systems with only 1 MByte of memory **must** — and all other systems should — configure the UNIX system kernel to suit your particular system. This reconfiguration reduces the kernel size, thus giving a larger effective memory size to programs. This is especially important if you intend to run the Sun Window System.

This section begins with a lockstep walkthrough of configuration procedures, which we hope provides enough information to take you safely through reconfiguration. If there is anything you don't understand or feel comfortable with, please read the sections just following the walkthrough; they give some explanation of what's going on (describe the layout of the kernel code, and the format of the configuration file used by the utility */usr/etc/config* to build your system configuration).

For a full discussion of configuring and building system images, see the document *Building UNIX Systems with Config* in the *Tutorials* section of this manual.

This configurable system also provides for adding new device drivers to the system, since all the kernel object files required to build a new system are present. For procedures, see the *Device Driver Tutorial* in the Sun *System Internals Manual*.

4.3.11.1. Making a New Configuration

This subsection walks you through the procedure for making a new system configuration.

1. Choose a name for your configuration of the system; here, GAIA. Note that — by convention — the name should be in all uppercase letters.
2. Change directory to the */sys/conf* directory, and create the configuration file and the directory in which the new system will be built. In this example, we assume that you will make a copy of the GENERIC file provided with this release, and edit the copy to create your new configuration file:

```
# cd /sys/conf
# cp GENERIC GAIA
# mkdir ../GAIA
```

3. Edit *GAIA* to reflect your system. This is the part of the procedure which takes some thought. On the next page, we provide a copy of the GENERIC file. We have indicated both classes of lines as follows:

   - Mandatory general system description lines are tagged "**mandatory**". These lines **must** be included in your system configuration file, either exactly as they stand or, if commentary indicates variables, with the variables adjusted to fit your system.
   - System-specific device description lines are interpreted; often, we also refer you to the manual entry which covers the device in question. When you edit the *GENERIC* file, you should include **only** the lines which describe your system's devices.

   If you need more information, see the following subsections and/or the *Building UNIX Systems with Config* paper in the *Tutorials* section of this manual.

```
#
# GENERIC SUN
#
machine         sun             #**Mandatory**#
cpu             "SUN2"          #**Mandatory**#
ident           GENERIC         #**Mandatory** Use "GENERIC" only if
                                    config line, below, has only "swap generic" clause#
timezone        8 dst           #**Mandatory** Number and "dst" are variable#
maxusers        2               #**Mandatory** Number may vary#
options         INET            #**Mandatory** INET means include Internet code#
options         SYSACCT         #Optional; include only with pseudo-device sysacct.
                                 Controls inclusion of code to do process accounting — see acct(2) and acct(5).#

config          vmunix  swap generic            #**Mandatory** Specify root and swap devices#

pseudo-device   pty             #Pseudo-tty's. Needed for network or window system.#
pseudo-device   bk              #Berknet line discipline for high speed tty input - see bk(4).#
pseudo-device   sysacct         #Include only with SYSACCT options clause, above.#
pseudo-device   inet            #**Mandatory** Internet code - see inet(4).#
pseudo-device   ether           #ARP code. Must include if using Ethernet — see arp(4).#
pseudo-device   loop            #**Mandatory** Software loop back network
                                    device driver; must include if inet — see lo(4).#
pseudo-device   nd              #Network disk. Needed if server or diskless - see nd(4).#
pseudo-device   win             #Window system.#
pseudo-device   ms              #Mouse; required for window system - see ms(4).#
pseudo-device   kb              #Optional; required if using any Sun keyboard;
                                  omit if using serial terminal for console.#
controller      mb0 at nexus ?  #**Mandatory** Multibus code.#
controller      ipc0 at mb0 csr 0x40 priority 2         #1st 2180 controller - see ip(4).#
controller      ipc1 at mb0 csr 0x44 priority 2         #2nd 2180 controller.#
disk            ip0 at ipc0 drive 0                     #1st disk on 1st 2180 controller#
disk            ip1 at ipc0 drive 1                     #2nd disk on 1st 2180 controller#
disk            ip2 at ipc1 drive 0                     #1st disk on 2nd 2180 controller#
disk            ip3 at ipc1 drive 1                     #2nd disk on 2nd 2180 controller#
controller      xyc0 at mb0 csr 0xee40 priority 2       #1st Xylogics controller - see xy(4).#
controller      xyc1 at mb0 csr 0xee48 priority 2       #2nd Xylogics controller#
disk            xy0 at xyc0 drive 0                     #1st disk on 1st Xylogics controller#
disk            xy1 at xyc0 drive 1                     #2nd disk on 1st Xylogics controller#
disk            xy2 at xyc1 drive 0                     #1st disk on 2nd Xylogics controller#
disk            xy3 at xyc1 drive 1                     #2nd disk on 2nd Xylogics controller#
controller      sc0 at mb0 csr 0x80000 priority 2       #1st SCSI controller#
disk            sd0 at sc0 drive 0 flags 0              #1st disk on 1st SCSI controller#
disk            sd1 at sc0 drive 1 flags 0              #2nd disk on 1st SCSI controller#
tape            st0 at sc0 drive 32 flags 1             #SCSI tape#
controller      sc1 at mb0 csr 0x84000 priority 2       #2nd SCSI controller#
disk            sd2 at sc1 drive 0 flags 0              #1st disk on 2nd SCSI controller#
disk            sd3 at sc1 drive 1 flags 0              #2nd disk on 2nd SCSI controller#
tape            st1 at sc1 drive 32 flags 1             #SCSI tape#
device          ropc0 at mb0 csr 0xee0800 priority 1    #**Mandatory** Raster Op chip - see ropc(4).#
device          sky0 at mb0 csr 0x2000 priority 2       #Sky Floating Point board.#
device          zs0 at mb0 csr 0xeec800 flags 3 priority 6 #Optional. UART (tty) driver - see zs(4).#
device          zs1 at mb0 csr 0xeec000 flags 3 priority 6 #Optional; required if using Sun-2 keybd and mouse
                                                         on Model 120. UARTs on Sun 2 Video board.#
device          zs2 at mb0 csr 0x80800 flags 3 priority 6  #Optional. UARTs on first SCSI board.#
device       zs3 at mb0 csr 0x81000 flags 3 priority 6  #Optional. UARTs on first SCSI board.#
device       zs4 at mb0 csr 0x84800 flags 3 priority 6  #Optional. UARTs on second SCSI board.#
device       zs5 at mb0 csr 0x85000 flags 3 priority 6  #Optional. UARTs on second SCSI board.#
device       oct0 at mb0 csr 0x520 flags 0xff priority 4 #Central Data Octal Card - see oct(4).#
device       ec0 at mb0 csr 0xe0000 priority 3          #1st 3COM Ethernet Controller - see ec(4)#
device       ec1 at mb0 csr 0xe2000 priority 3          #2nd 3COM Ethernet Controller#
device       en0 at mb0 csr 0x100 flags 126 priority 3  #Sun 3Mbit Ethernet Controller — see en(4).#
controller   tm0 at mb0 csr 0xa0 priority 3             #1st TAPEMASTER tape controller - see tm(4).#
controller   tm1 at mb0 csr 0xa2 priority 3             #2nd TAPEMASTER tape controller#
tape         mt0 at tm0 drive 0 flags 1                 #1st 1/2" tape drive on 1st controller.#
tape         mt1 at tm1 drive 0 flags 1                 #1st 1/2" tape drive on 2nd controller.#
device       ar0 at mb0 csr 0x200 priority 3            #1st 1/4" tape drive - see ar(4).#
device       ar1 at mb0 csr 0x208 priority 3            #2nd 1/4" tape drive.#
device       cg0 at mb0 csr 0xe8000 priority 3          #Sun Color Board - see cg(4).#
device       vp0 at mb0 csr 0x400 priority 2            #Ikon Versatec Board - see vp(4).#
device       pi0 at mb0 csr 0xee2000 priority 1         #Parallel input.  Only used on Sun Models 100U and 150U,
                                                             for keyboard and mouse.  Not needed if using terminal
                                                             for console.#
```

The following configuration SMALLXY is a stripped down version of the GENERIC system for a system with only a single local peripheral: a Xylogics controller supporting one disk drive. It's a good example of a standard configuration file:

```
#
# SMALL SUN WITH ONE XY DISK
#
machine         sun
cpu             "SUN2"
ident           SMALLXY
timezone        8 dst
maxusers        2
options         INET

config          vmunix  root on xy

pseudo-device   pty
pseudo-device   inet
pseudo-device   ether
pseudo-device   loop
pseudo-device   win
pseudo-device   ms
pseudo-device   kb
controller      mb0 at nexus ?
controller      xyc0 at mb0 csr 0xee40 priority 2
disk            xy0 at xyc0 drive 0
device          zs0 at mb0 csr 0xeec800 flags 3 priority 6
device          ec0 at mb0 csr 0xe0000 priority 3
device          ropc0 at mb0 csr 0xee0800 priority 1
device          pi0 at mb0 csr 0xee2000 priority 1
```

4. When you have finished editing GAIA, run config:

```
# /usr/etc/config GAIA
```

/usr/etc/config uses *GAIA*, *files*, and *files.sun* as input, and generates a number of files in the ../*GAIA* directory. These generated files are:

ioconf.c
contains tables for configuring the UNIX system at startup time.

param.c
which contains the details of number of users, timezone, and so on.

makefile
containing the commands necessary to build the new kernel.

A number of *.h* files
which describe the maximum number of devices of any given type that the system can handle.

5. Now change directory to the new configuration directory, ../*GAIA* in this case, and make the new system:

```
# cd ../GAIA
# make depend
[ lots of output ]
# make
[ lots of output ]
```

The **make depend** command creates the dependency tree for all system source files.

6. Now you can install the new system and try it out.

```
# mv /vmunix /vmunix.org
# cp vmunix /vmunix
# /etc/halt
     The system goes through the halt sequence, then
     the monitor displays its prompt, at which point you
     can boot the system in single-user state
> b vmunix -s
     The system boots up in single user state, and
     then you can try things out
#
```

7. If the system appears to work, all is well. If the new kernel doesn't seem to be functioning properly, boot */vmunix.org*, move it back to */vmunix*, and go about fixing your new kernel:

```
# /etc/halt
> b vmunix.org -s
# mv /vmunix /vmunix.oops
# mv /vmunix.org /vmunix
# ^D    [ Brings the system up multi-user ]
#
```

You can now continue with network configuration, if your system includes an Ethernet, by turning to the *Network Configuration* section. The subsections immediately below give background information for configuring the kernel.

4.3.11.2. Configuration Directory Layout

The system as shipped contains the essential source and object files in the */sys* directory. The */sys* directory contains several subdirectories that in turn contain the object code modules required to build different parts of the system. The subdirectories in */sys* are:

OBJ	Contains the kernel object code plus the header files describing the number of devices of each type.
conf	Contains the files describing the configuration of the system and what files are required to build the new system. Also contains a *README* file describing how to make a new kernel. Finally, contains the *GENERIC* system configuration file supplied with the distribution. This file may be copied and edited to produce your specific system configuration file; procedures are given above.

h	Header files containing definitions and data structures.
machine	Machine-dependent routines for the Sun system — this is a symbolic link to the *sun* directory.
net	Networking files.
netimp	Arpanet IMP support code.
netinet	Internet protocols.
netpup	3 Mbit Xerox PUP protocol support.
sun	Sun-specific mainline code.
sundev	Sun-specific device drivers.
sunif	Sun network interface code.
sys	General system routines.

4.3.11.3. Building a Configuration

Building a new system is a semi-automatic process; most of the fine detail is handled by a configuration build utility called */usr/etc/config*. */usr/etc/config* uses three files as input:

Specific Configuration File
: This file contains a description of the characteristics of a specific system, plus descriptions of the devices that that system can support. Every new version of the system has another file of this type.

files
: Contains a list of the files required to build the basic kernel.

files.sun
: Contains a description of the files required to build the specific Sun system. You add the names of new driver modules in here when adding drivers to the kernel.

/usr/etc/config should be run from the *conf* subdirectory of the system source or object files. */usr/etc/config* assumes that there is already a directory *../System_Name* created, and it places all its output files in there. The output of */usr/etc/config* consists of a number of files:

ioconf.c	Contains a description of I/O devices attached to the system.
makefile	For building the system.
Header Files	A set of header files which contain the number of various devices that will be compiled into the system.

After running */usr/etc/config*, you must then change directory to the directory in which the new makefile was created, and use *make* to create the dependency tree for the new system. */usr/etc/config* reminds you of this when it completes:

```
# cd ../System_Name
# make depend
#
```

If you get any other error messages from */usr/etc/config*, fix the problems in your configuration file and try again. If you try to compile a system that had configuration errors, you will meet with failure.

4.3.11.4. Format of the Configuration File

There are two main classes of information in the configuration file:

- Lines which describe general things about your system.
- Lines which describe the devices on the system, and what those devices are connected to.

The two subsections below contain descriptions of the two classes of information.

In theses descriptions, a number can be a decimal integer, a whole octal number or a whole hexadecimal number. Hexadecimal and octal are specified to */usr/etc/config* in the same way they are specified to the C compiler, namely, a number starting with '0x' is a hexadecimal number and a number starting with just a '0' is an octal number. When specifying the timezone, you may also use a floating point number.

Comments are specified in a configuration file with the character '#'. All characters from a '#' to the end of a line are ignored.

Lines beginning with tabs are considered continuations of the previous line.

4.3.11.4.1. General System Description Lines

The first seven general description lines in the configuration file are mandatory. They are:

machine *type*
This system is to run on the machine type specified. Only **one** machine type can appear in the configuration file. The legal *type* for a Sun system is **sun.**

cpu *type*
This system is to run on the cpu type specified. For a Sun system, legal *type* is "**SUN2**" (enclose in double quotes).

ident *name*
Gives the system identifier — a name for the machine or machines that run this kernel.

timezone *number* [**dst**]
Specifies the timezone you are in. This is measured in the number of hours west of GMT you are. 5 is EST, 8 is PST. Negative numbers indicate hours east of GMT. If you specify **dst**, the system will operate under daylight savings time.

maxusers *number*
The maximum expected number of simultaneously active users on this system is *number*. This number is used to size several system data structures. Typical values are 2 for 1 MByte memory single-user Sun Workstation, 4 for Sun Workstations with 2 MBytes of memory, and 8 for Sun Fileservers.

options *optlist*
Compile the listed options into the system. Options in this list are seperated by commas. There is a list of options that you may specify in the generic makefile. A line of the form 'options FUNNY,HAHA' yields -DFUNNY -DHAHA to the C compiler. An option may be given a value, by following its name with '=' then the value enclosed in (double) quotes. None of the standard options use such a value.

config *kernelname config_clauses*
Specifies a bootable system image; multiple bootable images may be specified in a single configuration file. Here *kernelname* is the name of the loaded kernel image (normally, **vmunix**). The *config_clauses* specify the root and swap devices. To create a generic configuration, only the clause **swap generic** should be specified; any extra clauses cause an error.

4.3.11.4.2. Device Description Lines

The second class of line in the configuration file describes what devices your system has and what they are connected to (for instance, I have a Xylogics xy450 on the Multibus). The device description lines for all supported devices are given in the section 4 pages of the Sun *System Interface Manual*; see the 'synopsis' section of the entry for each device.

Each device description line has the following format:

dev_type *dev_name* **at** *con_dev* *more_info*

The meaning of the fields is:

Dev_type
is one of **controller**, **tape**, **disk**, **device**, or **pseudo-device**. These types have the following meanings:

controller
is a Multibus disk controller or a Multibus tape controller.

disk or **tape**
are devices that are connected to a contoller.

device
is something which plugs into the Multibus, like a cartridge tape interface.

pseudo-device
is something which should be conditionally loaded, but is not really a device. Current examples are the the pseudo-tty driver and various network subsystems. For pseudo-devices, *more_info* may be specified as an integer, that gives the value of the symbol defined in the header file created for that device, and is generally used to indicate the number of instances of the pseudo-device to create. If you load a subsystem you may find it necessary to enable conditional code using an **options** specification.

dev_name
is the name of the device you are specifying. If it is not a pseudo-device, you must give a number afterwards — for instance, **xyc0** for a Xylogics controller or **ar0** for an Archive cartridge tape controller.

con_dev
is what the thing you are specifying is connected to. For example, disk **xy1** is connected to controller **xyc0**.

more_info
is a sequence of the following:

csr *addr*
Specifies the address of the csr (command and status registers) for a device. Must be specified for all Multibus controllers and for all devices connected to the Multibus.

drive *number*
For a disk or tape, specifies which drive this is.

flags *number*
These flags are passed to the device driver at system initialization time.

priority *level*
For devices which interrupt on the Multibus, specifies the interrupt level at which the device operates.

A ? may be substituted for a number in two places and the system will figure out what to fill in for the ? when it boots. You can put question marks on a *con_dev* (for example, at xyc?), or on a drive number (for example, drive ?). This allows redundancy as a single system can be built which will boot on different hardware configurations.

4.3.12. Network Configuration

4.3.12.1. Reducing Network Overhead

It is possible to have most of the advantages of living in a network environment without having all the network daemons running. It is particularly important for systems with limited memory to omit extraneous daemons, or run them only on specific machines in a network, as they consume significant portions of memory otherwise available to users.

Specifically, we suggest you do not run *rwhod*(8C) unless you absolutely have to, and that you run *routed*(8C) only on a gateway machine.

Rwhod allows the programs *rwho*(1C) and *ruptime*(1C) to return status information about machine usage of hosts on the local network. *Routed* maintains network routing information that enables your machine to pick the best path for sending packets to external networks.

When running, *routed* and *rwhod* are actively doing something many times a minute, and so leave many of their pages in memory almost all the time. For a 1 MByte system with 600K of available user memory, the memory thus effectively consumed is about 7%. Combined, both daemons use up about 14% of available user memory.

By default, *rwhod* is not run. If you choose to run it, remove the comment mark (a '#' in column 1) from the following line in */etc/rc*:

```
#/usr/etc/in.rwhod & echo -n ' rwhod'            >/dev/console
```

To reduce the impact of *routed*, you can run the daemon only on a designated gateway machine. All machines on the local network will redirect packets going to an external network to this gateway machine, and use its routing tables. Set this all up by editing the */etc/rc.local* file on all machines **except** the gateway machine (*gateway*, in the example). Find the line that says:

```
echo -n 'local daemons:'                         >/dev/console
```

Insert the following two lines just before it:

```
/usr/etc/route add 0 gateway 1
echo '  /usr/etc/route add 0 gateway 1'          >/dev/console
```

Then, find the following three lines and comment out (insert a '#' before each line) or remove them:

```
if [ -f /usr/etc/in.routed ]; then
      /usr/etc/in.routed & echo -n ' routed'     >/dev/console
fi
```

If your gateway machine goes down, you can redirect your packets to another gateway — see *route*(8C).

4.3.12.2. Handling Network Security with /etc/hosts.equiv and .rhosts

Network security is implemented at two levels: first, at the machine or node level, and, second, at the individual user level. The */etc/hosts.equiv* and *.rhosts* files, respectively, control access at these levels.

The security-checking process goes something like this. When a user initiates a remote process on another machine (*rlogin*, *rsh* or *rcp*, for example), the system checks for his machine's hostname in the other machine's */etc/hosts.equiv* file. If no entry is found, the system next checks for his hostname in the *.rhosts* file in his home directory on the other machine. If no entry is found there, the user is prompted for a password before he is allowed to *rlogin*. If no entry appears in either file, he is **not** allowed to *rcp* from or use *rsh* on this machine; he gets "Permission denied" messages when attempting remote processes. The single exception to this security scenario is the super-user: the system skips the first level check, and goes directly to checking */.rhosts*.

So, if you want to allow access to your machine by all users on another specific machine, include the machine's hostname from your */etc/hosts.equiv* file. For example, if my machine's hostname is gaia, and I want to allow anyone on host kepler to gain access to gaia, I simply edit my */etc/hosts.equiv* file as follows. The file is just a list of hostnames, one per line:

```
core
ganymede
krypton
```

Add kepler's name to the list:

```
core
ganymede
krypton
kepler
```

Now all users who can gain access to kepler can also freely *rlogin*(1) to gaia (without being asked for a password), and can *rcp*(1) from and use *rsh*(1) on gaia.

If you want to allow access to some users on a particular machine but not all, do not put the machine's hostname in */etc/hosts.equiv*. Instead, put it in the *.rhosts* file in each user's home directory on your machine (*~USERNAME/.rhosts*). The *.rhosts* file has the same format as */etc/hosts.equiv*: one hostname per line. Thus, I can allow user irena from host kepler to gain access to gaia, and can keep clem from kepler out by making sure that an entry for kepler appears in */usr/irena/.rhosts* but not in */usr/clem/.rhosts* on gaia.

4.3.13. Loading the Manuals, Demos, and Games Directories

The last file on the distribution tape contains *tar* images of the */usr/man* directory (online manuals), */usr/demo* directory (demonstration programs), and */usr/games* directory (games). You can load all or part of this file if you want to.

There is one basic procedure for loading to a standalone system from a quarter-inch tape, and one for the nine-track tape. These follow. If your system is a file server rather than a standalone system, simply change directory to */pub/usr* rather than */usr* initially, then complete the identical sequence of commands.

To load from quarter-inch tape:

1) Become super-user.
2) Insert the second quarter-inch tape cartridge.
3) Type the following sequence of commands (choosing the directories you want, at the last):

```
# cd /usr  (or /pub/usr if you are loading on a fileserver)
# mt -f /dev/nrar0 rew
# mt -f /dev/nrar0 fsf 2
# tar xvf /dev/rar0 man demo games
```

To load from half-inch tape:

1) Become super-user.
2) Mount the half-inch tape.
3) Type the following sequence of commands (choosing the directories you want, at the last):

```
# cd /usr  (or /pub/usr if you are loading on a fileserver)
# mt rew
# mt fsf 9
# tar xv man demo games
```

5. INSTALLING UNIX ON SYSTEMS WITHOUT TAPE SUPPORT

This chapter describes the process for installing the UNIX system onto a workstation which does not have a tape drive installed. The process involves installing the UNIX system across the Ethernet from a system which does have a tape drive. Throughout this chapter we refer to the "remote host" and "target" machines. The remote host is the machine with the tape drive; the target is the machine without it.

CAUTION: If you are *upgrading* your Sun Model 100 or 150 rather than starting with a system 'fresh from the box,' please heed the warning on the front page of this manual. SUN-1 systems which are upgrading must not attempt to install this release of the system until all Multibus memory has been removed or disabled, and — for an upgrade from a Xylogics 440 to a Xylogics 450 — cabling has been re-routed. Make sure you have followed Sun's hardware upgrade procedures before proceeding.

CAUTION: If you are *upgrading* your system software, rather than beginning for the first time, read the *Upgrading System Software* chapter BEFORE proceeding here. The chapter gives procedures for saving and rebuilding your existing root and */usr* file systems

5.1. Overview of the Installation Procedure

Here are the steps required to install the system over the network:

1. Complete installation of the UNIX system on a Sun System equipped with a tape drive — your "remote host" or "network disk server" — as described in the chapter, *Installing UNIX for the First Time*.
2. Determine Ethernet addresses for both the "remote host" and the "target" machine.
3. If your remote host is configured as a standalone system rather than as a network disk server, you must turn its */usr* file system into a public network disk. If your remote host is already configured as a server, this step is unnecessary.
4. Load the mini file system onto the public disk from the boot tape.
5. Boot *diag* over the network; run *diag* to format (if necessary) and label your disk.
6. Boot the standalone *copy* program over the network. Run *copy* to copy a mini-file system over the network into the swap area on your disk.
7. Boot the mini-UNIX system.
8. Edit the */etc/hosts* and */.rhosts* files.
9. Mount the release tape on the remote host.
10. Extract the root file system from the tape.
11. Boot UNIX in single user state.
12. Run the *setup* program to extract the */usr* file system from the tape and initialize the network files.
13. Boot the full UNIX system
14. Load the manuals, demos, and games directories if you wish.

Now read the sections which follow. There should be enough detail there to walk through the remote installation procedure. In all the examples, what **you type is shown in bold-faced text like this**. Bold-faced text is typed exactly as it appears. Some places we have shown parts of command in *italic text like this,* and this means that there is something that must be substituted at that point. Italic characters are variables. For example, we show the phrase *server_address* in several examples — this means that at that point you must type the hexadecimal ethernet address of the remote host.

Two very important variables which you will see in the examples — and have already seen if you have installed UNIX on your remote host — are *tape* and *disk*. Where *tape* appears, replace it with the correct device abbreviation for your tape drive: **mt** for the nine-track tape, or **ar** for the Archive quarter-inch tape. Where *disk* appears, replace it with the correct device abbreviation for your disk: **xy** for the Xylogics disk controller, or **ip** for the Interphase disk controller.

You can correct typing errors at any time. The DEL key backspaces over and erases characters, and the control-U key (hold down the CONTROL key while typing the letter U) erases the entire line typed so far.

If you need to get back to the monitor for any reason, you can abort by typing 'SET-UP A' (hold down the SET-UP key while typing 'A') or 'ERASE-EOF A' at any time.

5.2. Determining Ethernet Addresses

Before beginning the installation procedure, you need to know the Ethernet addresses of both the remote host and the target machine. You will use these addresses later in the installation procedure.

For each machine, proceed as follows. Power up the Sun Workstation (if you haven't already done so). You should see the PROM monitor's sign on messages. Stop the auto boot immediately by aborting — type 'SET-UP A' (hold down the SET-UP key while typing 'A') or 'ERASE-EOF A' — and then you should see the monitor's prompt, which is a > sign:

```
Self Test completed successfully.

Sun Workstation, Model Sun-1/100U or Sun-1/150U, type of keyboard
Serial #number, ROM Rev L, some_number_MBytes memory installed

Auto-boot in progress . . .

abort by typing 'SET-UP A' or 'ERASE-EOF A' here

Abort at some address
>
```

You use the workstation monitor program to examine the PROM locations where the Ethernet address is stored. The Ethernet addresses start in location fe0400 (base 16). The Ethernet address occupies six bytes. You find out what is stored there with the monitor's **e** (examine) command. The notation *return* in the example below stands for carriage return:

```
> e fe0400 return
FE0400: 0260? return
FE0402: 8C00? return
FE0404: 1142? q return
>
```

The numbers you are interested in at this time are the lower three bytes of the six bytes (001142 in this example). Make a note of these numbers. You must determine the Ethernet addresses for all machines in a network if you are installing a cluster of workstations. The addresses you get from the monitor are in hexadecimal (base 16).

5.3. Setting up the Remote Host

DO THIS IF THE REMOTE HOST **IS NOT** *A NETWORK DISK SERVER.*

If your remote host is already configured as a server, skip to the next step.

If the remote host machine is not a network disk server, turn the */usr* file system into a public network disk by adding two lines to *nd.local* to reference */dev/ip0g* (for an Interphase disk controller) or */dev/xy0g* (for a Xylogics disk controller). The lines should look like this:

```
user 0 0 /dev/disk0g -1 -1 -1
son
```

Then enable the network disk server:

```
# cd /dev
# MAKEDEV nd
# /etc/nd - < /etc/nd.local
```

Next, make the */usr* disk into a public disk so that it can be accessed across the network by doing the following sequence of commands:

```
# mkdir /usr/stand
# cp /stand/* /usr/stand
# ln -s /usr /pub
# cp /boot /pub/boot
# cd /usr/mdec
# installboot bootnd /dev/disk0g
#
```

Then proceed with the next step.

5.4. Loading the Mini File System to the Server

Now load the mini file system onto the public disk from the boot tape with the following sequence of commands. Remember to replace *tape* with **mt** for the nine-track tape, or **ar** for the Archive quarter-inch tape:

```
# mt -f /dev/nrtape0 rew
# mt -f /dev/nrtape0 fsf 3
# dd if=/dev/nrtape0 of=/pub/minifs bs=20b
#
```

This takes about three minutes using a 1/2-inch tape, and about six using a 1/4-inch cartridge.

5.5. Running Diag

Now, you start to work on your target machine, and install UNIX from the remote host. The first step is to format the disk with the *diag* utility. You boot *diag* from the network server with a boot command:

> **b nd(*server_address*)stand/diag**

server_address is the Ethernet address (in hexadecimal) for the server machine on the net. The monitor should boot *diag* from the network disk server. When *diag* starts up, it displays a sign on message:

```
Disk Initialization and Diagnosis

When asked if you are sure, respond with 'y' or 'Y'
```

Diag first asks you a series of questions about the disk you will be using. The answers to these questions 'configure' *diag* to work with that specific hardware. The first thing *diag* wants to know is the type of disk controller to use. *Diag* displays a menu of the different disk controller types.

Diag's second question concerns the address of the controller on the Multibus. The table below shows the default addresses for disk controllers*.

Disk Controller Addresses

Controller	Address (hex)
Interphase SMD-2180	40
Xylogics 450/440 SMD	ee40

Next, *diag* wants to know the unit number of the disk — 0 is usually the correct answer if you only have one disk, and the type of disk drive you are working with — *diag* displays a menu of the different disks. When you have specified which disk drive you are using, *diag* then displays a table of the physical data about that disk, which includes the number of cylinders, number of alternate cylinders, number of heads, and number of sectors per track.

At this point, *diag* knows all it needs to know about the disk, and it displays its prompt:

```
diag>
```

One of the responses to this prompt can be the **h** (for help) command. If you use the **h** command, *diag* displays a list of its commands.

* Note that if you have one Xylogics SMD Controller and one Interphase Controller in your system, you cannot use the default addresses (as the Interphase Controller sees only four bytes). The Interphase must be configured to be at address 48, and the Xylogics at ee40. See the *Hardware Configuration and Expansion* chapter for board configuration procedures.

5.5.1. Formatting the Disk

If you are starting with a completely new disk, you will need to format the disk first. If you are not starting with a completely new disk, you can skip to the next topic, *Labelling the Disk.*

Before formatting, clear any outstanding errors by typing the **clear** command:

```
diag> clear
clear
diag>
```

Next, type the **format** command. *Diag* then warns you that formatting a disk destroys all information which might already be stored on that disk, and asks for confirmation before going ahead and doing the job:

```
diag> format
format/verify, DESTROYS ALL DISK DATA!
are you sure? y
# of surface analysis passes? 5
```

Use five surface analysis passes for a completely new disk; use one pass for a disk which has been formatted in the past and is known not to have any bad spots. Formatting a disk takes anywhere from a half-hour to four hours depending on how many passes you select, the size of the disk, and the type of controller you happen to have. For example, the format phase takes about thirty minutes for one surface analysis pass, using an Interphase SMD-2180 disk controller, and a slightly shorter time using a Xylogics 450 or 440 controller. *Diag* displays the current cylinder number as it formats each cylinder.

5.5.2. Labelling the Disk

A disk must be labelled after it has been formatted. The purpose of a label is to record (in a well known place on the disk) information about how the disk is divided up into partitions for different purposes such as swap space and file systems. *Diag* labels a disk in response to the *label* command. Here is an example of using the label command to write a label on the disk which was formatted in the discussion above. When you give the *label* command to *diag,* it asks if you want to use the logical partition map that is "built in" to the program:

```
diag> label
label this disk...
OK to use logical partition map 'Fujitsu-M2312K (Sun D84)' ? y
Are you sure you want to write? y
```

Then, when *diag* has labelled the disk, it verifies the label it has just written. There is in fact a separate verify command, but *diag* automatically does a verify as part of the labelling process.

When the 'diag>' prompt returns, get back to the monitor by responding "q" (for 'quit').

5.6. Loading the Mini UNIX System

Now boot the standalone copy program from the disk server (remember to replace *disk* with **xy** for the Xylogics disk controller, or **ip** for the Interphase disk controller):

```
> b nd (server_address)stand/copy
Boot: nd(server_address)stand/copy
Load: nd(server_address,0,0)boot

Boot: nd(server_address,0,0)stand/copy
[ . . . messages displaying sizes of copy program . . . ]
Standalone Copy
From: nd(server_address,0,0)minifs
To: disk(0,0,1)
[ Numbers flash while the copy proceeds ]
```

The load process takes about four minutes. This process loads the mini file system into the swap area on the disk. When it completes, the copy program returns control to the bootstrap program:

```
Copy completed
Boot:
```

5.7. Booting the Mini UNIX System

The next step is to boot UNIX in single user state and ask for its root file system. When UNIX comes up and asks for its root file system, tell it "***disk*0***". Since this notation looks a bit ambiguous, let me clarify: if you are using a Xylogics controller, your root device is **xy0***; if you are using an Interphase controller, it is **ip0*** — the asterisk is part of the device name:

```
Boot: disk(0,0,1)vmunix -as

[ . . .lots of messages . . . ]

root device: disk0*
#
```

At this point, the system may gently remind you to reset the date and time:

```
WARNING: clock gained 16845 days -- CHECK AND RESET THE DATE!
```

You can do this later, when you have finished installation, with the *date*(1) command.

5.8. Editing the /etc/hosts and /.rhosts Files

Now you must edit the */etc/hosts* file on both the remote host and target machines, to make them aware of each other's existence. On the target machine, find the line in the */etc/hosts* file which contains the string 'noname' — it should look like this:

```
125.9999        noname
```

change it to read

```
125.0xnnnn        server_name
```

where *nnnn* is your remote host's hexadecimal Ethernet address (remember to precede *nnnn* with 0x to indicate that it is a hexadecimal number), and *server_name* is the name of your remote host.

Add a similar entry below it with your hexadecimal Ethernet address and your target machine's name.

Then, run the following command on the target machine:

```
# /etc/ifconfig ec0 your_target_name
```

On the remote host machine, simply add an entry in */etc/hosts* for the target machine (address and name). You don't need to run *ifconfig*.

Finally, edit the */.rhosts* file on the remote host only, adding an entry for the target machine. If the file does not exist, create it.

5.9. Loading the Root File System

Now you must mount the distribution tape on the remote host's tape drive. Then run the *rxtr* (remote extract) shell script on the target machine to copy the root file system across the Ethernet. Again, use **xy** for the Xylogics disk controller, or **ip** for the Interphase disk controller to replace *disk*, and **mt** for the nine-track tape, or **ar** for the Archive quarter-inch tape to replace *tape*:

```
# disc=disk tape=tape host=server_name rxtr

[. . .incredible amounts of messages . . .]

Root filesystem extracted
#
```

The extraction process takes ten to twenty minutes. The next job is to configure your system and load the */usr* file system.

5.10. Booting the UNIX System in Single-User State

Now type a couple of **sync** commands to flush all I/O activity to the disks, then get back to the monitor by typing 'SET-UP-A' or 'ERASE-EOF-A'. The monitor responds by displaying a message like:

```
Abort at some address
```

When you see the monitor > sign, boot the UNIX system in single-user state:

```
> b vmunix -s
[. . .incredible amounts of messages . . .]
#
```

5.11. Using Setup to Configure Your System

At this point you invoke the *setup* program to configure your system.

Setup is essentially a system configurator in two parts: it consists of an interactive front-end which gathers the information necessary to configure your system by conducting a dialogue with you, and a non-interactive back-end which uses this information to do the actual configuration. During the dialogue, *setup* does consistency and error checking to ensure that the configuration will work. If errors are detected, they are reported to you, and you are asked to enter corrected information.

For standalone systems booting from a remote tape drive, the *setup* dialogue runs as follows. In the example, what you might type in is shown in **bold face type like this**; whatever is simply displayed on the workstation monitor is shown in Roman type like this. *Italic items* are variables which you must substitute.

We invoke the *setup* program by typing the *setup* command. *Setup* begins by requesting global information:

```
# setup

Sun Microsystems Configuration System

Global Information

    1) network disk server
    2) standalone
    3) standalone with remote tape

Enter the number for your environment: 3

You have a standalone system with remote tape; is this correct? (y/n): y
```

Note that the program asks you to verify configuration information after each 'phase' of configuration is complete. It is extremely difficult to undo system configuration, so be careful with your responses: 'y' casts things in concrete, and 'n' allows you to start the phase over again. You can also type 'q' to any prompt to quit the *setup* program and return to the shell — this allows you to annihilate what you have done and start over, if you have to.

Next, *setup* establishes your workstation's identity: name and address:

```
Host Information

Enter your hostname: your_hostname

Enter the hexadecimal ethernet address for your_hostname: your_address

Your hostname and ethernet address are:
    your_hostname:    your_address

Is this correct? (y/n): y
```

Note that you use the Ethernet address for your target machine established at the outset of the installation procedure here. Make sure you have your address at hand before starting the dialogue.

The last phase of *setup* configuration requests information about your remote host's tape subsystem:

```
Tape Information

    1) 1/4" SCSI tape (st)
    2) 1/2" magnetic tape (mt)
    3) 1/4" archive tape (ar)

Enter the number for the type of tape: (1-3): 3

You have specified a 1/4" archive tape; is this correct? (y/n): y

Enter the name of the remote host that the 1/4" archive tape is attached to: server_name

Enter the hexadecimal ethernet address for server_name: server_address

The 1/4" archive tape is attached to server_name at ethernet address server_address

Is this correct? (y/n): y
```

When this last phase has been completed, *setup* asks you whether you want to institute the configuration you have designed and, if you confirm, proceeds to edit several of the database files:

```
You have completed the configuration questions.
Continuing will destroy any existing files under /usr
and any client partitions if you are a server.

Do you want to begin configuration? (y/n): y

Updating /etc/hosts
[ . . . A few lines of configuration messages . . . ]
```

When *setup* completes its back-end work, your shell prompt returns. You can then continue your installation by booting the full UNIX system, as described below.

5.12. Booting the Full UNIX System

Finally, you boot the full UNIX system. You must first halt the system, using the */etc/halt* command. This shuts down the system in an orderly manner, and returns control to the monitor:

```
# /etc/halt
Syncing disks . . . . done
>
```

and now you can simply boot the UNIX system:

```
> b
Boot: disk(0,0,0)vmunix
Load: disk(0,0,0)boot
Boot: disk(0,0,0)vmunix
Size: 266240+ 32768+ 35316 bytes
Sun UNIX 4.2 (Berkeley beta release) (GENERIC) #145: Tue Aug 30 20:35:13
PDT 1983
Copyright (c) 1983 by Sun Microsystems, Inc.

[. . . Several lines of configuration messages . . . ]

gaia login: root
#
```

You can now use the UNIX system on this machine.

5.13. Loading the Manuals, Demos, and Games Directories

The last file on the distribution tape contains *tar* images of the */usr/man* directory (online manual pages), */usr/demo* directory (demonstration programs), and */usr/games* directory (games). If these have been loaded on your remote host, you can load all or part of these files if you want to:

1) Become super-user.
2) Mount the 1/2" distribution tape or insert the 1/4" distribution tape.
3) Type the following on your target machine:

```
# cd /usr
# rsh server_name mt -f /dev/nrtape0 rew
# rsh server_name mt -f /dev/nrtape0 fsf n
```

where *n* is **9** for a half-inch tape, or **2** for an Archive quarter-inch tape. Then type the following command, specifying only the directories you want (**man** and/or **demo** and/or **games**):

```
# rsh server_name dd if=/dev/nrtape0 bs=20b | tar xfpB - man demo games
```

6. SYSTEM SET-UP AND OPERATION

This chapter describes procedures used to set-up and operate a Sun Workstation UNIX system. Set-up procedures apply to a first-time installation; they provide orientation to various available utilities — like *uucp* and the *mail* system — you may or may not wish to use. Operations procedures described here are used periodically, to reboot the system, analyze error messages from devices, do disk backups, monitor system performance, and so on.

6.1. Setting Up the Mail System

The mail system consists of the following commands and files:

/bin/mail	old standard mail program (from V7)
/usr/ucb/mail	UCB mail program, described in *mail*(1)
/usr/lib/sendmail	mail routing program
/usr/lib/sendmail.cf	configuration file for mail routing
/usr/spool/mail	mail spooling directory
/usr/spool/mqueue	spool directory for mail going out over the network
/usr/spool/secretmail	secure mail directory
/usr/bin/xsend	secure mail sender
/usr/bin/xget	secure mail receiver
/usr/lib/aliases	mail forwarding information
/usr/ucb/newaliases	command to rebuild binary forwarding database
/usr/ucb/biff	mail notification enabler
/usr/etc/in.comsat	mail notification daemon
/etc/syslog	error message logger, used by *sendmail*

Mail is normally sent and received using the *mail*(1) command, which provides a front-end to edit the messages sent and received, and passes the messages to *sendmail*(8) for routing. The routing algorithm uses knowledge of the network name syntax, aliasing and forwarding information, and network topology, as defined in the configuration file */usr/lib/sendmail.cf*, to process each piece of mail. Local mail is delivered by giving it to the program */bin/mail* which adds it to the mailboxes in the directory */usr/spool/mail*, using a locking protocol to avoid problems with simultaneous updates. After the mail is delivered, the local mail delivery daemon */usr/etc/in.comsat* notices, and it notifies users who have issued a "biff y" command that mail has arrived.

Mail for *username* is normally accessible in the file */usr/spool/mail/username*. In the distribution system, your mail file is readable only by you; however, anyone with the super-user password can read others' files, including their mail. To send mail which is secure against any possible perusal (except by a code-breaker), use the secret mail facility, which encrypts the mail so that no one can read it. See *xsend*(1). Note that this facility does not work over the network.

The following subsections give some instruction on *sendmail* installation; for more detailed information, see the *Sendmail Installation and Operation Guide* in the final section of this manual.

6.1.1. Picking a Name for your Domain

The network mail delivery program, ***sendmail***, uses Internet standard mail addressing formats which make it possible for any Internet system in the world to send or receive mail with any other Internet system. To accomplish this, each system must have a unique name. To make it easier to assign names, the world of mail addresses is broken up into domains and subdomains.

For example, many systems funded by the U.S. Government's Advanced Research Projects Agency are in the domain "arpa". Many of the systems which send mail via the "uucp" protocols are in the domain "uucp". Within each domain, names have to be unique — there can't be two machines called "MIT-Multics.arpa" or it would be impossible to decide how to deliver mail to them.

Domains nest inside one another like directories, except that the names go from right to left. Thus "joe.sun.uucp" is host "joe" in subdomain "sun" which is in domain "uucp" just like "usr/lib/crontab" is file "crontab" in subdirectory "lib" in directory "usr". To make life easier for the people who maintain mailers, there are only a small number of "top-level" domains like "uucp" and "arpa".

If your workstation is on the Arpanet, or shares an Ethernet with a machine on the Arpanet, or with a machine on the Arpa Internet, you should probably pick a name in the "arpa" domain, and register it with the Internet naming authority at the Arpanet Network Information Center. Otherwise, pick a name in the "uucp" domain.

If your workstation and/or Ethernet have no phone lines or connections to other networks, the name you pick within the "uucp" domain doesn't matter much. You may, however, have to change the name if you get other network connections and somebody else is already using that name.

If your organization already has hosts on the uucp network or the Usenet, ask a system administrator for help in picking a name.

If you are connected to the uucp network, you must be talking with at least one other computer on the network. Check with the system administrator of that machine to see if the name you have picked is already in use in the uucp network. If they are on the Usenet, they can look in newsgroup "net.map"; if not, they just have to guess.

In a network of Suns, the name of the "main machine" becomes the name of your subdomain, and each other machine can truly have any name you want. For example, a main machine at a site might be called "fred"; various other machines on its Ethernet could be "shirl," "sal," etc. The *fully qualified* name of the "fred" machine is "fred.uucp" and must be unique in the uucp world; but the full name of "shirl" is "shirl.fred.uucp" and its uniqueness is guaranteed by the uniqueness of "fred.uucp"".

If you only have one machine, your host name and your domain name (within the "uucp" or "arpa" top-level domain) are the same.

6.1.2. Picking a "Main Machine" for Mail Forwarding

To begin configuration, you must tell *sendmail* whether your system is standalone, the main system in a network, or a subsidiary system in a network.

If your machine is not attached to an Ethernet, it is standalone system. The distribution system is already set up for this configuration; you can go on to the next subsection.

If your machine is attached to an Ethernet and is also attached to phone lines, it is a "main machine". Pick one such machine on the network and make it your mail machine; treat all the rest as "subsidiary machines" for configuration purposes. The distribution system is already set up for the one system you picked; keep reading to deal with the other machines.

If your machine is attached to an Ethernet and you don't have any phone lines, but there is an Arpanet host on your Ethernet, you can pick any workstation as your "main machine". If your Arpanet host runs *sendmail*, it can be your "main machine". All the other Suns will be subsidiary machines.

Similarly, if you have several machines on an Ethernet, and none of them have phones, pick one as the main machine and leave the rest as subsidiary machines.

6.1.3. Editing Mail Configuration Files

Special note for file server configurations:

On file servers and diskless clients the directory */usr/lib* resides on the shared, public disk. Since this disk is read-only and is shared by all the systems, files that need to be writable or need to be different on each machine cannot reside here. Instead there is a directory called */private* that contains the files that would normally reside in */usr/lib*. The *setup* program replaces the files in */usr/lib* by symbolic links to the files in */private* so the files can still be referenced by their normal names. The files (and directories) so affected are listed below:

```
/usr/lib/sendmail.cf
/usr/lib/aliases
/usr/lib/crontab
/usr/lib/uucp
/usr/lib/news
```

When the following instructions tell you to copy or edit one of the above files, instead use the corresponding file in */private*.

6.1.3.1. Configuring Subsidiaries for Mail Forwarding

If your system is a "subsidiary" machine, you must change its mailer configuration so it forwards mail to the main machine. The following commands accomplish this. **Remember** that if your machine is a file server or a diskless client, the *sendmail.cf* file is in */private/sendmail.cf.*

```
# rm /usr/lib/sendmail.cf
# cp /usr/lib/sendmail.generic.cf /usr/lib/sendmail.cf
```

This installs the "generic" configuration file as the active configuration, instead of the "domain" configuration, which is the default.

6.1.3.2. Telling Sendmail your Domain Name

To tell *sendmail* what your domain is, edit the file */usr/lib/sendmail.cf* on the "main machine" and all the subsidiary machines. **Remember** that if your machine is a file server or a diskless client, the *sendmail.cf* file is in */private/sendmail.cf.* Near the top of the file is a block of lines that looks like this:

```
# local domain names
DDsun
CDsun
```

This defines our domain name as "sun" within the "uucp" domain — in other words, "sun.uucp". Change these lines to reflect the name you picked. For example,

```
# local domain names
DDfred
CDfred
```

defines your domain name as "fred.uucp". The hostname of your main machine (as set up by the *hostname* command) is "fred," and subsidiary machines, if you have any, will be called (for example) "shirl.fred.uucp" for a machine whose hostname is "shirl".

If your top-level domain is not "uucp", find the lines that look like:

```
# domain-spec for local domain within universe (e.g., what domains are above?)
# should be integrated into mainline (e.g. Berkeley) config files
DUuucp
```

They should be the next thing in the file. Change "uucp" to your top-level domain name (for example, "DUarpa").

Now, if you are editing *sendmail.cf* for a subsidiary machine, look for a block of lines like this:

```
# major relay host
DRmailhost
CRmailhost
```

Change the name "mailhost" to the name of your domain:

```
# major relay host
DRfred
CRfred
```

If you are editing *sendmail.cf* for a subsidiary machine which has phone lines attached to it, you can have *sendmail* route uucp mail to certain hosts via the local phone lines, rather than having all uucp traffic go through the "main machine". To do this, find the lines that look like:

```
# local UUCP connections -- not forwarded to mailhost
CV
```

Put the names of the local uucp sites on the end of the "CV" line, or on additional CV lines. For example,

```
CV rome prussia georgia
```

This completes the *sendmail.cf* editing for the subsidiary machine. Note that if you have more than one subsidiary with no local uucp connections, you can edit the file just once and then copy it to all the subsidiary machines with *rcp*(1).

On your main machine, you can make one optional change. If one of the machines with which you have a uucp or Ethernet connection is on the Arpanet and will relay mail for you, look for a block of lines like this:

```
# major relay mailer
DMuucp

# major relay host
DRucbarpa
CRucbarpa
```

Note that similar lines appear twice in the file — you should change the *second* set. Edit in the mailer that you connect to this host with ("uucp" or "ether") and the name of the relay host. For example, if you share an Ethernet with a machine called "cmu-cs-vlsi," which is on the Arpanet, your entry might look like this:

```
# major relay mailer
DMether

# major relay host
DRcmu-cs-vlsi
CRcmu-cs-vlsi
```

On the other hand, your relay point might be uucp host "ucbvax":

```
# major relay mailer
DMuucp

# major relay host
DRucbvax
CRucbvax
```

This change will let you mail to addresses like "charlie@mit-ai.arpa" and even though you aren't on the Arpanet, the message will get through. If you don't have an Arpanet relay point, don't worry about this.

This completes the mailer configuration process.

6.1.3.3. Setting up the "Postmaster" Alias

Edit the file */usr/lib/aliases* (or */private/aliases* for a file server or diskless client workstation) and look for the entry for "postmaster". This is where people will send mail if they want to get in touch with someone at your site who can deal with mailer problems (you can send mail to "postmaster"s at other network sites if you have problems with mail that comes from there). The line will look like this:

```
# Following alias is required by the new mail protocol, RFC 822
# Set it to the address of a HUMAN who deals with this system's mail problems
Postmaster:root
```

If you manage the mail system for your site, change the name "root" to the user name you usually log in with, so that messages directed to the Postmaster of your machine will arrive with your usual mail.

If you manage the mail system for several workstations, change the file on all of them to forward Postmaster mail to wherever you usually read mail. For example, if you are "henry" on machine "shirl," make the line read:

Postmaster: henry@shirl

You can also create aliases for people or groups of people ("mailing lists") at this time. See the examples in the *aliases* file. You can edit this file now or at any later time.

6.1.4. Testing your Mailer Configuration

The first thing to do is to reboot all the systems whose configuration files you have changed. Then, send test messages from various machines on the network with a command like this:

```
host% /usr/lib/sendmail -v </dev/null addresses
host%
```

This will send a null message to the specified address, and display messages about what it is doing. Test that:

- You can send mail to yourself, or other people, on the local machine, by addressing the message to a plain user name ("root", for example).
- If you have an Ethernet, you can send mail to someone on another machine ("root@hobo," for example). Try this in three directions — from the main machine to a subsidiary machine, *vice versa,* and from a subsidiary machine to another subsidiary machine, if you have two. (Note that */etc/hosts.equiv* must be set up on at least the main machine before this will work. See the subsection, *Handling Network Security with /etc/hosts.equiv and /.rhosts* in the *Installing UNIX for the First Time* chapter.)
- If you have a phone line and you have set up a uucp connection to another host, you can send mail to someone there and they can send it back (or call you on the phone, if they receive it). Try having them send mail to you. For example, you could send to "ucbvax!joe" if you have a connection to ucbvax. *Sendmail* won't be able to tell you whether the message really got through — since it just hands it off to uucp for delivery so you have to ask a human at the other end. You might be able to get some idea of what's going on by looking in */usr/spool/uucp/LOGFILE*; see the *Uucp Implementation Description* in the *Tutorials* section of this manual.
- Mail something to Postmaster on various machines and make sure that it comes to your usual mailbox, so when other sites send you mail as Postmaster, you're sure you will see it.

6.1.5. Diagnosing Troubles with Mail Delivery

The best tools for diagnosis of mail problems are:

- "Received" lines in the header of the broken message. These give a trace of which systems the message was relayed through on its way. Note that in the uucp network there are many sites that do not update these lines, and that in the Arpanet the lines often get rearranged. You can straighten them out by looking at the date and time in each line. Don't forget to take time zones into account.
- Messages from "MAILER-DAEMON" on various systems. These messages typically report delivery problems. More and more systems are producing these messages, rather than simply throwing away mail that they can't deliver.

- The system log, for delivery problems in your group of workstations. Sendmail records what it is doing all the time, and this information is kept in the system log. In the distributed system, the logs are kept for a week, then discarded. Log files are kept in */usr/spool/log* on your network server machine (the system log configuration is taken care of by the *setup* program during UNIX installation — (see *syslog*(8)). Today's log is in file *syslog*; the previous day's is *syslog.0*; two days' back is *syslog.1*, etc. If you have chronic trouble with mail, look at the log once in a while. At Sun, *cron*(8) runs a shell script nightly which searches the log for SYSERR messages and mails any that it finds to "Postmaster". This way, problems are often fixed before anyone notices them, and the mail system runs more smoothly.

6.2. Setting Up a UUCP Connection

A *uucp* network link using modems or dedicated lines may be established between two machines running UNIX systems. To establish a connection between two sites that both have modems, one site must have an automatic call unit (an auto-dial modem) and the other must have a dialup port (an auto-answer modem). It is better if both sites have one of each. This section gives a brief overview of *uucp* and points out the most important steps in its installation.

Support for *uucp* is located in three major directories: */usr/bin* (which contains user commands), */usr/lib/uucp* (operational commands), and */usr/spool/uucp* (spooling area).

The commands in */usr/bin* are:

/usr/bin/uucp	file-copy command
/usr/bin/uux	remote execution command
/usr/bin/uusend	binary file transfer using mail
/usr/bin/uuencode	binary file encoder (for *uusend*)
/usr/bin/uudecode	binary file decoder (for *uusend*)
/usr/bin/uulog	scans session log files

The important files and commands in */usr/lib/uucp* are:

/usr/lib/uucp/L-devices	list of dialers and hardwired lines
/usr/lib/uucp/L-dialcodes	dialcode abbreviations
/usr/lib/uucp/L.cmds	commands remote sites may execute
/usr/lib/uucp/L.sys	systems to communicate with, how to connect, and when
/usr/lib/uucp/SEQF	sequence numbering control file
/usr/lib/uucp/USERFILE	remote site pathname access specifications
/usr/lib/uucp/uuclean	cleans up garbage files in spool area
/usr/lib/uucp/uucico	*uucp* protocol daemon
/usr/lib/uucp/uuxqt	*uucp* remote execution server

The spooling area, */usr/spool/uucp*, contains the following important files and directories:

/usr/spool/uucp/C.	directory for command, "C." files
/usr/spool/uucp/D.	directory for data, "D.", files
/usr/spool/uucp/D.*hostname*	directory for local "D." files
/usr/spool/uucp/LOGFILE	log file of *uucp* activity
/usr/spool/uucp/SYSLOG	log file of *uucp* file transfers

If both you and the site(s) you wish to connect with have autodial units and ports, install *uucp* as follows. If you have only a dialup your situation will be different; see below.

1. Select a site name (less than 8 characters): the name of the machine which will be your *uucp* connection to the outside world. We will call this machine your '*uucp* host'; its name is your '*uucp* hostname'.

   If this machine is your "main machine" (see *Setting Up the Mail System*, above), then your *uucp* hostname should be the same as your domain name.

2. Change the */usr/spool/uucp/D.noname* directory to your own site's */usr/spool/uucp/D.hostname* directory with the following:

   # mv /usr/spool/uucp/D.noname /usr/spool/uucp/D.*hostname*

   Use your *uucp* hostname for *hostname*.

3. Create a *uucp* account in the password file on your *uucp* host machine by entering a line of the following form in */etc/passwd*:

   U*hostname*:*:4:4::/usr/spool/uucp:/usr/lib/uucp/uucico

   Now use the *passwd* command to establish a password for your host:

   # passwd U*hostname*

4. The *L.sys* file contains the phone numbers and login sequences required to establish a connection with a *uucp* daemon on another machine. Edit */usr/lib/uucp/L.sys*, adding a line of the following form for each site you want to talk to:

   their_host **Any** *device baud phone#* **login:–EOT–login: uucp ssword: pass**

   The first field is the *uucp* hostname of the other site, the second indicates when their host may be called, the third field specifies how their host is connected (through an ACU, a hardwired line, etc.), then comes the baud rate of the line, phone number to use in connecting through an auto-call unit, and finally a login sequence. The phone number may contain common abbreviations which are defined in the *L-dialcodes* file. The device specification (third field) should refer to devices specified in the *L-devices* file. Note that the only modem type currently supported by Sun Microsystems is the Ventel MD212 modem. Indicating only "ACU" causes the *uucp* daemon, *uucico*, to search for any available auto-call unit in *L-devices*. For example, our *L.sys* file looks something like:

   ```
   adiron Any ACUVENTEL 1200 7620883 login:–EOT–login: uucp ssword: secret
   ucbvax Any,20 ACUVENTEL 1200 6728212% login:-EOT-login: uucp ssword: almama
   ucbarpa Any,20 ACUVENTEL 1200 6424351 login:-EOT-login: uucp ssword: netnut
   decvax Any ACUVENTEL 1200 6039941241 login:-EOT-login: Ujedi ssword: cannon
   ```

For hardwired lines, your *L.sys* lines should contain the tty device name in the third and fifth fields:

```
anyname Any ttyb 1200 ttyb login:-EOT-login: uucp ssword: sunrise
```

5. Connect your auto-dial modem to ttya (the port labelled 'RS-232 A' on the backpanel of your Sun Workstation) on your host machine. Connect your auto-answer modem to ttyb (the port labelled 'RS-232 B' on the backpanel of the Sun Workstation) on your host machine.
6. Create the appropriate devices for your modems with the following series of commands:

```
# cd /dev
# ln ttya cua0
# chmod 666 cua0
# ln ttyb ttyd0
```

7. Edit */etc/ttys* to include an entry for *ttyd0* (see *ttys*(4)). Insert the line: "13ttyd0". Then reboot your system.
8. Make sure your dialer is hooked up properly by running *tip* with the phone number of a known machine:

```
# tip 6423345
```

If you get a "dialing . . . connected" response, all is well. If you get any other response, reconnect your modem.

9. As a final test, run *uucp* with the debug option (–x), as follows:

```
# /usr/lib/uucp/uucico –r1 –ssitename –x7
```

With the -x option, the higher the number, the more debugging output you get; 1, 4, and 7 are reasonable choices. If you get immense quantities of output from this command, everything is fine; you can go on to edit the following files.

10. Edit the files *uucp.day*, *uucp.noon*, and *uucp.night*, in the */usr/lib/uucp* directory. Each of these files has a 'for' loop which arranges to call sites you want to call at designated times for mail. In each file, find the line that says:

```
for i in sys.name
```

and change "sys.name" to the site name(s) you want to poll. For example:

```
for i in ucbvax ucbarpa Shasta navajo
```

11. Add the *uucp.day*, *uucp.noon*, *uucp.night*, *uucp.hour*, and *uucp.week*, files to */usr/lib/crontab* (see *cron*(8)), which arranges for the appropriate sites to be polled at the appropriate times. For example, the entries in *crontab* might look like:

```
5 6 * * * su uucp < /usr/lib/uucp/uucp.day
15 12 * * * su uucp < /usr/lib/uucp/uucp.noon
30 23 * * * su uucp < /usr/lib/uucp/uucp.night
10,30,50 * * * * su uucp < /usr/lib/uucp/uucp.hour
0 7 * * 2 su uucp < /usr/lib/uucp/uucp.week
```

If you have only a dialup, you can be a second-class citizen on the *uucp* net. You must find another site that has a dialer, and have them poll you regularly (once a day is about the minimum that is reasonable). When you send mail to another site, you must wait for them to call you. To handle installation for a passive node, just complete steps one through four in the procedures above. When you come to the final step, editing */usr/lib/uucp/L.sys*, you don't need to specify all the information called for in the step: only the first two fields of *L.sys* are necessary, and in practice only the first field (site name) is looked at. A typical *L.sys* for a passive node might be:

```
ucbvax    None
research  None
```

where the first field on each line is a site that will poll you. Next, put a password on the *uucp* login. Then let the other site know your phone number, *uucp* login name, and password.

As *uucp* operates it creates (and removes) many small files in the directories underneath */usr/spool/uucp*. Sometimes files are left undeleted; these are most easily purged with the *uuclean* program. Instructions in the *uucp.day* file take care of doing this daily clean-up for you. The *uucp* log files can grow without bound unless trimmed back; *uulog* is used to maintain these files. *uucp.day* and *uucp.week* manage this housekeeping. If you decide to prune these directories yourself, be careful: randomly removing files from */usr/spool/uucp* may cause *uucp* to generate error messages when it tries to access a file another file claims is there. (For instance, each mail transaction creates three files.) You do, however, need to clean the */usr/spool/uucppublic* directory 'manually'; this is a place for people at other sites to send to when sending files to users at your site.

uucp occasionally sends mail about minor problems to "uucp" or "root". Your maintenance person should also read and toss these messages. You can redirect the mail to your mailbox by putting an entry for "uucp" in */usr/lib/aliases*. Look at the examples there.

Under normal conditions, *uucp* calls your designated sites at specified times and, while it's at it, checks to see if anything should come back. If you ever need to invoke *uucp* 'on command', the line:

```
# /usr/lib/uucp/uucico -r1 -ssitename
```

forces *uucp* to poll *sitename*, even if there is nothing waiting. If you do run *uucp* in this fashion, don't run it as super-user, since the suid* bit will not be honored. If you are having trouble with the connection, run *uucp* with the debugging option, as described in installation step 7 above.

For more information, read the *Uucp Implementation Description* in the final section of this manual. It describes in detail the file formats and conventions, and will give you necessary context.

6.3. Talking to a VAX via ARP

ARP (Address Resolution Protocol) should be enabled on Sun systems in three major cases:

- If you have a Vax running 4.2 UNIX with an Interlan Ethernet Controller Board, and would like to attach a Sun Workstation to the network, or

* "suid" stands for "set user i.d."; if you set this bit on an executable file, UNIX will grant or deny file access based on the permissions of the file's owner, rather than the permissions of the person who executes the file. *Uucp* uses this facility to ensure that all the files in its spool directories are readable and writeable no matter who invokes the *uucp* program.

- If you have machines with Ethernet controller boards from various manufacturers on the same network (and each of the other machines can also do ARP), or
- If you are using a Class B or Class C network (DARPA, for example). The standard Sun system uses a Class A network.

If none of these are applicable, you may skip this section. If you do need ARP, you must perform the following two steps **on each machine in your network**:

1) Edit the */etc/hosts* file, and assign each machine on the network a unique host address of less than 1024. The host address is the last component of the Internet address entry in */etc/hosts*:

```
125.83          krypton
125.85          titan
```

In the example, "125.85" is titan's Internet address; titan's host address is "85". Host addresses are arbitrary except that they **must** be less than 1024 for ARP to be performed.

Your */etc/hosts* file may not look like our example. This is because Internet addresses have different formats for Class A, B, and C networks:

Internet Address Formats

Class A	Class B	Class C
network.host	part1.part2.host	part1.part2.part3.host
network = (0-127) network: 1 byte	part1 = (128-191) part1: 1 byte part2: 1 byte	part1 = (192-255) part1: 1 byte part2: 1 byte part3: 1 byte
host: 3 bytes	host: 2 bytes	host: 1 byte

To repeat, make sure the host address of each machine is less than 1024. Otherwise, its Internet address is taken as its actual hardware address in network communications.

2) Edit the */etc/rc.local* file on each machine, and add an *ifconfig* command immediately after the */bin/hostname* command. They should look like:

```
/bin/hostname krypton
/etc/ifconfig ec0 '/bin/hostname'
```

The *ifconfig* does the mapping between hostnames, Internet addresses, and hardware addresses. It is given a device (*ec0*) which has the hardware address, and it finds the hostname in */etc/hosts* to determine the Internet address. The *ifconfig* command is used to map a hostname to an Ethernet address, and the */etc/hosts* file maps a hostname to an Internet host address. For more information, see *ifconfig*(8C).

To understand how this all fits together, assume that you have two hosts called krypton and titan whose */etc/hosts* files look like our first example:

```
125.83          krypton
125.85          titan
```

At boot time, both krypton and titan had run an *ifconfig* command to obtain their Internet addresses from their respective */etc/hosts* files.

Now assume you are on krypton and wish to talk to titan. The address resolution goes like this:

1) Krypton sends out a broadcast packet saying, "I would like to talk to the machine at Internet address 125.85."
2) Titan now receives krypton's broadcast packet, and responds with a packet saying, "I'm Internet address 125.85, and my hardware address (which I got from my Ethernet Controller board) is XXXXXX."
3) Krypton now has titan's hardware ethernet address and communication can be established.

6.4. Setting Up the Line Printer System

The line printer system consists of at least the following files and commands:

/usr/ucb/lpq	spooling queue examination program
/usr/ucb/lprm	program to delete jobs from a queue
/usr/ucb/lpr	program to enter a job in a printer queue
/etc/printcap	printer configuration and capability data base
/usr/lib/lpd	line printer daemon, scans spooling queues
/etc/lpc	line printer control program

The file */etc/printcap* is a master data base; it describes line printers directly attached to a machine and printers accessible across a network. The manual page *printcap*(5) describes the format of this data base and also indicates the default values for such things as the directory in which spooling is performed. The line printer system handles multiple printers, multiple spooling queues, local and remote printers, and printers attached via serial lines which require line initialization such as baud rate. Raster output devices such as a Varian or Versatec, and laser printers such as an Imagen, can also be handled by the line printer system.

Remote spooling via the network is handled with two spooling queues, one on the local machine and one on the remote machine. When a remote printer job is initiated with *lpr*, the job is queued locally and a daemon process is created to oversee the transfer of the job to the remote machine. If the destination machine is unreachable, the job will remain queued until it is possible to transfer the files to the spooling queue on the remote machine. The *lpq* program shows the contents of spool queues on both the local and remote machines.

To configure your line printers, consult the *printcap* manual page (in the Sun *System Interface Manual*) and the *Line Printer Spooler Manual* in the *Tutorials* section of this manual.

6.5. System Log Configuration

Various system daemons and programs record information in the system log to aid in problem analysis. They send this information as Internet datagrams directed to the 'syslog' daemon on host 'loghost'. The daemon receives these datagrams and records the information or notifies users of problems. See *syslog*(8) for more details on this process.

The default configuration runs a syslog daemon on each machine, and causes the datagrams from that machine to stay on that machine. This works well for standalone systems, but in a network it's easier to track problems if all machines log their information in a single place.

If you followed the first time UNIX installation procedures in this document, and configured your system for a network server, the *setup* program took care of installing *syslog* so that it runs on the server only and not on the clients — this is the most useful configuration for a network environment. This is done by adding the 'loghost' alias for the server to the */etc/hosts* file. Also, the 'syslog' daemon (*/etc/syslog*) is moved to */etc/syslog.server* on all the clients. This prevents *syslog* from running on them.

If you have more than one server machine, it's best to pick one of them to receive all the logs. Fix */etc/hosts* on all systems so that 'loghost' is an alias for the machine you picked, rather than for their server machine. You can also move */etc/syslog* to */etc/syslog.server* on all your servers except the one you picked.

Test your system log configuration by running:

```
% tail -f /usr/spool/log/syslog
```

on the loghost machine, then sending any kind of mail on the various other machines. Each message sent will generate four or five lines of output if things are working.

6.6. Adding Users

New users can be added to the system by adding a line to the password file */etc/passwd*. The procedure for adding a new user is described in *adduser*(8).

You should add accounts for the initial user community, giving each a directory and a password, and putting users who will wish to share software in the same groups.

6.7. Bootstrap and Shutdown Procedures

In a normal reboot, the system checks the disks and comes up multi-user without intervention at the console. Such a reboot can be stopped (after it prints the date) with a ^C (interrupt). This will leave the system in single-user mode, with only the console terminal active.

If booting from the console command level is needed, then the command:

```
> b
```

boots the system from the default device.

You can boot to single user mode with:

```
> b -s
```

To take the system down to a single user state you must be super-user. Then you kill off the */etc/init* process with:

```
# kill 1
```

or use the */etc/shutdown*(8) command (which is much more polite, if there are other users logged in) when you are up multi-user. Either command kills all processes and gives you a shell on the console, as if you had just booted single-user. Previously mounted file systems remain mounted after the system is taken single-user. If you wish to come up multi-user again, you have to change directory to root, unmount all mounted file systems, then log out of single-user state:

```
# cd /
# /etc/umount -a
# ^D    [ Brings the system up multi-user ]
```

Each system shutdown, crash, processor halt and reboot is recorded in the file */usr/adm/shutdownlog* with the cause.

6.8. Device Errors and Diagnostics

When errors occur on peripherals or in the system, the system prints a warning diagnostic on the console. These messages are collected regularly and written into a system error log file */usr/adm/messages* by *dmesg*(8), which is started by *cron*(8).

Error messages printed by the devices in the system are described with the drivers for the devices in the Section 4 pages of the *System Interface Manual*. If errors occur indicating hardware problems, you should contact your hardware support group or field service. It is a good idea to examine and truncate the error log file regularly — see the section below on *Files which Need Periodic Attention*.

6.9. File System Checks, Backups, and Disaster Recovery

Periodically (say every week or so in the absence of any problems) and always (usually automatically) after a crash, all the file systems should be checked for consistency by */etc/fsck* (see *fsck*(1)). The procedures of */etc/reboot* (see *reboot*(8)) should be used to get the system to a state where a file system check can be performed manually or automatically.

Dumping of the file systems should be done regularly, since once the system is going it is easy to become complacent. Complete and incremental dumps are easily done with */etc/dump* (see *dump*(8)). Most people do a level-0 dump on a weekly or monthly basis, and a level-9 dump daily.

Dumping files by name is best done by *tar*(1), but the amount of data that can be moved in this way is limited to a single tape.

It is desirable that full dumps of the root file system be made regularly. This is especially true when only one disk is available. Then, if the root file system is damaged by a hardware or software failure, you can rebuild a workable disk by doing a restore in the same way that the initial root file system was created.

In the normal nature of things, disk space gets used up until your */usr* file system is full. At this point, there are several places to look for files you can remove:

- Every time a crash occurs, several MBytes of system image get dumped into the */usr/crash* directory. You can clean the *vm** files out if you don't need them for debugging or trouble reporting. Don't remove the */usr/crash/bounds* file.
- Files in */usr/adm* grow and must be pruned regularly.
- Check your accumulated mail file(s) — some diligent mail-writers have been known to generate about a quarter of a MByte of mail per user per month.
- Network news accumulates at a fearsome rate.

- Finally, police your own directories regularly.

Useful mechanisms for monitoring disk usage are *du*(1) and *df*(1); *quot*(8) can be used to figure out exactly who is filling filesystem space.

6.10. Moving Filesystem Data

If you have the equipment, the best way to move a file system is to dump it to magnetic tape using */etc/dump* (see *dump*(8)), use */etc/newfs* (see *newfs*(8)) to create the new file system, and restore the tape, using */etc/restore* (see *restore*(8)). If for some reason you don't want to use tape, *dump* accepts an argument telling where to put the dump; you might use another disk. The *restore* program uses an "in-place" algorithm which allows file system dumps to be restored without concern for the original size of the file system. Further, portions of a file system may be selectively restored in a manner similar to the tape archive program. If you have to merge one file system into another, the best bet is to use *tar*(1). If you must shrink a file system, the best bet is to dump the original and restore it onto the new file system.

Note that a level zero dump must be done after a full restore, because the restore relocates the files, although it does not change their content. Thus, you must do another level zero dump to get a tape image reflecting the new file positions, so that later incremental dumps will be correct.

6.11. Monitoring System Performance

The *vmstat* program provided with the system is designed to be an aid to monitoring systemwide activity. Together with the *ps*(1) command (as in "ps av"), it can be used to investigate systemwide virtual memory activity. By running *vmstat* when the system is active you can judge the system activity in several dimensions: job distribution, virtual memory load, paging and swapping activity, disk and cpu utilization. Ideally, there should be few blocked (b) jobs, there should be little paging or swapping activity, there should be available bandwidth on the disk devices (most single arms peak out at 30-35 tps in practice), and the user cpu utilization (us) should be high (above 60%).

If the system is busy, then the count of active jobs may be large, and several of these jobs may often be blocked (b). If the virtual memory is active, then the paging daemon will be running (sr will be non-zero). It is healthy for the paging demon to free pages when the virtual memory gets active; it is triggered by the amount of free memory dropping below a threshold and increases its pace as free memory goes to zero.

If you run *vmstat* when the system is busy (a "vmstat 1" gives all the numbers computed by the system), you can find imbalances by noting abnormal job distributions. If many processes are blocked (b), then the disk subsystem is overloaded or imbalanced. If you have several non-dma devices or open teletype lines that are "ringing", or user programs that are doing high-speed non-buffered input/output, then the system time may go high (60-70% or higher). It is often possible to pin down the cause of high system time by looking to see if there is excessive context switching (cs), interrupt activity (in) or system call activity (sy).

If the system is heavily loaded, or if you have little memory for your load (1 MByte is "little" in most any case), then the system may be forced to swap. This is likely to be accompanied by a noticeable reduction in system performance and pregnant pauses when interactive jobs such as editors or window system programs swap out. If you expect to be in a memory-poor environment for an extended period you might consider administratively limiting system load.

6.12. Making Local Modifications

Locally written commands are kept in */usr/src/local* and their binaries are kept in */usr/local*. This allows */usr/bin*, */usr/ucb*, and */bin* to correspond to the distribution tape (and to the system manuals). People wishing to use */usr/local* commands should be made aware that they aren't in the base manual.

A */usr/junk* directory to throw garbage into, as well as binary directories */usr/old* and */usr/new* are useful. The *man* command supports manual directories such as */usr/man/manj* for junk and */usr/man/manl* for local manual page entries to make this or something similar practical.

6.13. Accounting

Optionally, UNIX records two kinds of accounting information: connect-time accounting and process-resource accounting. Connect-time accounting information is stored in the */usr/adm/wtmp* file, which is summarized by the program */etc/ac* (see *ac*(8)). Process-time accounting information is stored in */usr/adm/acct*, and analyzed and summarized by the program */etc/sa* (see *(8))*.

If you need to charge for computing time, you can implement procedures based on the information provided by these commands. A convenient way to do this is to give commands to the clock daemon */etc/cron* to be executed every day at a specified time. This is done by adding lines to */usr/lib/crontab*; see *cron*(8) for details.

6.14. Resource Control

Resource control in the current version of UNIX is rather primitive. The resources consumed by any single process can be voluntarily limited by the mechanisms of *setrlimit*(8). Disk space usage can be monitored by *quot*(8) or *du*(1). No system-enforced procedure for controlling a user's disk space usage is implemented under the current system, although a modicum of control can be obtained by dividing user groups between different disk partitions.

6.15. Network troubleshooting

If you have anything more than a trivial network configuration, from time to time you are bound to run into problems. Before blaming the software, first check your network connections. On networks such as the Ethernet a loose cable tap or misplaced transceiver cable can result in severely deteriorated service. The *netstat*(8) program may be of aid in tracking down hardware malfunctions. In particular, look at the **–i** and **–s** options on the manual page.

If you believe you have a faulty coaxial Ethernet cable, test the cable to make sure it is delivering 50 ohms: remove the terminator from the N connector on the transceiver terminating the cable, and use an Ohm Meter to test resistance (use the pin 'inside' N connector for signal, and the housing as ground). If you are getting something other than 50 ohms, your cable may be damaged. Contact Sun Microsystems for assistance.

If you believe the routing daemon is malfunctioning, its actions — and even all the packets sent and received — may be printed out. To create a log file of routing daemon actions, just supply a file name when you start the daemon up, for example:

```
# /etc/routed /etc/routerlog
```

Whenever a route is added, deleted, or modified, a log of the action and a history of the previous packets sent and received will be printed in the log file. To force full packet tracing, the –t option may be specified when the daemon is started up. Beware though — on a busy network this will generate almost constant output.

6.16. Files which Need Periodic Attention

We conclude the discussion of system operations by listing the files that require periodic attention or are system-specific

/etc/fstab	how disk partitions are used
/etc/printcap	printer data base
/etc/remote	names and phone numbers of remote machines for *tip*(1)
/etc/group	group memberships
/etc/motd	message of the day, printed at login
/etc/passwd	password file; each account has a line
/etc/rc.local	local system restart script; runs reboot; starts daemons
/etc/hosts	host name data base
/etc/networks	network name data base
/etc/services	network services data base
/etc/hosts.equiv	hosts under same administrative control
/etc/securetty	restricted list of ttys where root can log in
/etc/ttys	enables/disables ports
/etc/ttytype	terminal types connected to ports
/usr/lib/crontab	commands that are run periodically
/usr/lib/aliases	mail forwarding and distribution groups
/usr/adm/acct	raw process account data
/usr/adm/messages	system error log
/usr/adm/shutdownlog	log of system reboots
/usr/adm/wtmp	login session accounting

7. UPGRADING SYSTEM SOFTWARE

This chapter outlines procedures for upgrading software to a new release level.

7.1. Step 1: What To Save

No matter what version of the system you may be running, you will have to rebuild your root and */usr* file systems. The easiest way to do this is to save the important files on your existing system, perform a bootstrap as if you were installing a software release on a brand new machine, then merge the saved files into the new system. The following lists the standard set of files you will want to save (**in addition to all user files**) and indicates directories in which site-specific files should be present. This list will probably be augmented with non-standard files you have added to your system; be sure to do a tar of the directories */etc/*, */lib/*, and */usr/lib/* so that you don't miss anything the first time around.

/.profile	root sh startup script
/.login	root csh startup script
/.cshrc	root csh startup script
/dev/MAKEDEV.local	for the LOCAL case for making devices
/etc/fstab	disk configuration data
/etc/group	group data base
/etc/passwd	user data base
/etc/rc.local	for any local additions
/etc/ttys	terminal line configuration data
/etc/ttytype	terminal line to terminal type mapping data
/etc/termcap	for any local entries which may have been added
/usr/include/*	for local subdirectory and any other additions
/usr/lib/aliases	mail forwarding data base
/usr/lib/crontab	cron daemon data base
/usr/lib/tabset/*	for locally developed tab setting files
/usr/lib/tmac/*	for locally developed troff/nroff macros
/usr/lib/uucp/*	for local uucp configuration files
/usr/spool/*	for current mail, news, uucp files, etc.
/usr/*	all users' directories

This can be accomplished with the following commands. Run them as "root":

```
# cd /
# tar c .??* dev/MAKEDEV.local etc lib usr/include usr/lib
```

This makes a tape containing system files which you will need to set up your system after the upgrade. NOW CHANGE TO ANOTHER BLANK TAPE, and run the following command:

```
# tar c usr/{spool,usera,userb,userc,userd...}
```

This makes a tape containing users' files. Replace "*usera* . . ." with the names of all users on your system. (You can double check by looking in /etc/passwd or by doing "ls usr".)

You may want to use other options on the *tar* commands, such as **f** to specify which tape drive to use, **b** to specify a large blocking factor such as 400 on Archive quarter-inch tapes, or **v** to list each file as it is processed. See *tar*(1) for more information.

Once you have saved the appropriate files in a convenient format, the next step is to dump all your file systems to magnetic tape with */etc/dump* (see *dump*(8)).

When you have completed your system dump, install the new release from the distribution tape just as described in the chapter, *Installing UNIX for the First Time*; or, if you are installing the release on a machine without a tape drive, in the chapter *Installing UNIX on Systems Without Tape Support*. Then proceed with the section below.

7.2. Step 2: Merging

When your system is booting reliably and you have the root and */usr* file systems fully installed, you will be ready to proceed to the next step in the conversion process: merging your old files into the new system.

Using the first tar tape you created in Step 1, extract the appropriate files into a scratch directory, say */usr/convert*:

```
# mkdir /usr/convert
# cd /usr/convert
# tar x
```

Certain files, such as those from the */etc* directory, may simply be copied into place:

```
# cp passwd group fstab ttys ttytype /etc
# cp crontab /usr/lib
```

Other files, however, must be merged into the distributed versions by hand. In particular, be careful with */etc/termcap*.

The spooling directories and user files saved on the second *tar* tape may be restored in their eventual resting places without too much concern. Be sure to use the **p** option to *tar* so that files are recreated with the same file modes (you may want to add other options, as described in step 1):

```
# cd /
# tar xp
```

Whatever else is left is likely to be site-specific or require careful scrutiny before placing in its eventual resting place. Refer to the documentation (*hier*(7), for example) before arbitrarily overwriting a file.

8. HARDWARE CONFIGURATION AND EXPANSION

This chapter describes the seven-slot Multibus cardcage in the Model 100U and the fifteen-slot Multibus cardcage in the 150U, and gives brief configuration details of the boards which may be supplied with them. For specific details of the configuration and descriptions of the boards and their jumpers, see the board-level manuals for individual boards.

In general, the Sun Workstation is shipped with all its boards (tape and disk controllers and such) already installed. However, if you are upgrading a Sun with more peripherals, you can use this section to discover how to install the boards and how to set the switches for them. This chapter can also be used during troubleshooting, if you need to poke inside the cardcage and identify the boards.

8.1. The 100U Cardcage

The cardcage holding the Model 100U circuit boards lives in the black cabinet below the monitor housing.

Turn off the power before opening up the Sun Workstation's enclosure.

To get to the Model 100U cardcage, first unscrew the six screws on the sides of the black cabinet.

Stand at the back of the Sun Workstation where all the cable connectors are. *Gently* slide out the enclosure. There are many cables and wires connecting the enclosure to various parts of the workstation. You must pull the enclosure out gently to avoid breaking anything. Eventually, you won't be able to pull it any further because the wires are stretched as far as they can go.

As you look at the enclosure from the back of the workstation, the power supply is in the metal case on the *left* of the enclosure, and the Multibus cardcage is on the *right* of the enclosure. The *top* of the cardcage itself is uppermost, and the *top* of the boards is to the right.

The following diagrams show a few typical configurations for the desktop workstation. In the diagrams, the notation '**P2**' indicates slots spanned by the P2 bus connector. Cards are shown in the slots where they are installed at the factory. Detailed explanations of the reasons for placing certain cards in specific slots follow this section.

Basic Standalone Workstation
Minimum — with Disk and 1/4" Tape Subsystem

1	*1/4" Tape Controller*	
2		**P2**
3	**68010 Processor**	**P2**
4	**1 MByte Main Memory**	**P2**
5	*One-board SMD Disk Controller*	
6		
7	**Monochrome Display Controller**	

Expansion Option for the Standalone Workstation
1 MByte Additional Memory

1	*1/4" Tape Controller*	
2	*1 MByte Memory Expansion*	**P2**
3	**68010 Processor**	**P2**
4	**1 MByte Main Memory**	**P2**
5	*One-board SMD Disk Controller*	
6		
7	**Monochrome Display Controller**	

Network Node
with Local Disk and Tape Storage

1	*Ethernet*	
2	*1 MByte Memory Expansion*	**P2**
3	**68010 Processor**	**P2**
4	**1 MByte Main Memory**	**P2**
5	*One-board SMD Disk Controller*	
6	*1/4" Tape Controller*	
7	**Monochrome Display Controller**	

Basic Diskless Network Node

1	*Ethernet*	
2	*1 MByte Memory Expansion*	**P2**
3	**68010 Processor**	**P2**
4	**1 MByte Main Memory**	**P2**
5		
6		
7	**Monochrome Display Controller**	

Diskless Node with Color Option

1	*Ethernet*	
2	*1 MByte Memory Expansion*	**P2**
3	**68010 Processor**	**P2**
4	**1 MByte Main Memory**	**P2**
5		
6	*Color Display Controller*	
7	**Monochrome Display Controller**	

Note that the configuration for a minimum standalone workstation (with 1/4" tape and disk) normally takes five slots, leaving two for expansion options. One such option is a second MByte of main memory. Note that due to power and operating temperature considerations, a color option may not be added to a desktop workstation with local disk storage. Further, since a minimum standalone workstation requires 25 amps (and its peak current rating is 35 amps), an Ethernet, or a second MByte of memory, or a customer's own Multibus board may be added to such a system *but Sun has only tested those configurations shown above*. Three principal constraints (slots, power, and cooling) affect the viability of a proposed configuration.

Also, please note that these diagrams illustrate our current understanding of the best board configurations; by all means check with Sun Microsystems or refer to the most recent configuration guides released by Sun if you intend to change your cardcage layout in any way.

8.1.1. 100U Multibus Mastering Scheme

The Model 100U uses a Multibus backplane employing serial bus arbitration. This places a limit on the number of Multibus master cards (cards which use direct memory access — DMA). The limit today, as recommended by Intel and verified for our systems by Sun, is three Multibus masters. Multibus arbitration logic requires approximately 30 nanoseconds per card; the 10 MHz arbitration clock imposes the 3 card limit. Of the cards sold by Sun Microsystems, the CPU, the SMD disk controllers, and the 1/2-inch (but not the 1/4-inch) tape controller board are Multibus master cards. A system with all three of these cards may not operate reliably if you add any additional DMA cards to the system. Note that non-DMA cards, such as Sun display controllers, Sun Ethernet controllers, and many other Multibus cards, are not Multibus masters and do not come under this restriction.

In the serial Multibus mastering scheme, the order of the cards in the cardcage determines their priority. Slot number 1 is the highest priority bus master, and slot number 7 is the lowest. Bus master devices must be in *electrically* adjacent slots — this means that they must either be physically adjacent, or separated by a board which passes the bus arbitration signal straight through. Currently, the only Sun boards which are 'transparent' in this sense are the 1 MByte Memory board and Rev. C or 'greater' 1/4" Tape Controller boards. These boards may therefore be inserted between bus masters.

8.1.2. 100U Multibus P2 Connector

The P2 bus is a high-speed synchronous memory bus that allows memory access without wait states. The design of the connector for the P2 bus requires that the processor board and the main memory expansion boards be inserted into adjacent slots. The Model 100U cardcage has a P2 connector across slots 2, 3, and 4. The processor board is usually inserted in slot 3, with main memory cards in slots 4 and — possibly — 2.

Sun Workstation cards use the standard Multibus backplane signals, as defined in IEEE standard 796 for the P1 bus. Various Sun cards use the P2 bus in different ways, which may not be compatible with other vendors' use of this bus. In particular, Sun processor and Sun main memory cards do not support extended (24-bit) Multibus addressing on the P2 bus. They use the P2 bus for no-wait-state access to main memory. As a general rule, Sun boards and other vendors' boards which use the P2 bus should therefore not be installed on the same P2 section.

The signalling conventions used by the various Sun cards on the P2 bus are documented in the hardware reference manual for each board. For further details, please refer to the appropriate reference manual.

8.1.3. 100U Card Insertion Order

There are a few restrictions on the placement of cards in the cardcage:

- As stated above, the design of the P2 connector requires that the processor board and memory boards be inserted in adjacent slots.
- With serial arbitration, bus masters (DMA devices) must be in *electrically* adjacent slots; boards which pass the bus arbitration signal through (currently, memory boards and Rev. C or greater 1/4" tape controller boards) may be inserted between bus masters.
- In a desktop workstation which includes a disk controller, the disk controller must be below the processor for proper handling of the serial bus priority. If present, the 1/2-inch tape controller, a third bus master, must be inserted above the disk controller, because the very high throughput of the disk controller might otherwise lock out the lower-priority device for excessive periods:

  Standalone Workstation
  with Disk and 1/2" Tape Subsystem

1		
2		**P2**
3	**68010 Processor**	**P2**
4	**1 MByte Main Memory**	**P2**
5	*1/2" Tape Controller*	
6	*Xylogics 450 SMD Disk Controller*	
7	**Monochrome Display Controller**	

  For a customer's own DMA devices, the correct placement order must be determined empirically.
- The order in which cards are inserted into the Multibus card cage affects the ability of a desktop workstation to maintain proper operating temperatures. Cards that require high current also dissipate large amounts of heat and should be placed in slots in the card cage that receive the maximum air flow. The top and bottom slots in the card cage receive the least amount of cooling. Air flow is greatest across the center three slots in the system. The color display controller consumes the most power, and is best placed toward the center of the card cage. (In fact, if there are unused slots in the cage, one should be left open directly above the color controller board.)

8.2. The 150U Cardcage

Open the door of the rackmount system cardcage enclosure: the cardcage is the vertically mounted unit with fifteen slots. The following diagram shows a typical configuration of the rack-mountable workstation. In the diagram, the notations 'P2*a*', 'P2*b*', and 'P2*c*' indicate slots spanned by a single connector on the P2 bus. Cards are shown in the slots where they are installed at the factory. Detailed explanations of the reasons for placing certain cards in specific card slots follow this section.

Please note that this diagram illustrates our current understanding of the best board configuration; by all means check with Sun Microsystems or refer to the most recent configuration guides released by Sun if you intend to change your cardcage layout in any way.

Model 150U Cardcage

1	**68010 Processor**	**P2***a*
2	**1 MByte Main Memory**	**P2***a*
3	*1 MByte Memory Expansion*	**P2***a*
4		**P2***a*
5		**P2***a*
6		**P2***a*
7		**P2***a*
8	*Ethernet Controller*	**P2***b*
9	*Half-Inch Tape Controller*	**P2***b*
10		**P2***b*
11	*Xylogics 450 SMD Disk Controller*	**P2***b*
12		**P2***b*
13	*Color Display Controller*	**P2***c*
14		**P2***c*
15	**Monochrome Display Controller**	**P2***c*

Note: the Sun-1 Model 150U card cage is mounted vertically, with slot 1 at the left.

8.2.1. 150U Multibus Mastering Scheme

The Model 150U cardcage uses parallel priority bus arbitration. In these systems, there is no limitation on the number of bus master cards (DMA devices) which may be inserted in the cardcage. The relative priority of each master is determined only by its relative slot number; slot 1 is the highest priority. In a parallel cardcage, bus master cards need not be physically adjacent.

8.2.2. 150U Multibus P2 Connectors

The Model 150U cardcage has three separate P2 connectors. Slots 1 through 7 share the first P2 connector; slots 8 through 12 share the second; and slots 13 through 15 share the third. Sun suggests placing the processor card and main memory cards on the first P2 connector, and the color graphics controller on the third connector. Disk, tape, Ethernet, and any non-Sun controllers should be installed on the second P2 connector.

Sun Workstation cards use the standard Multibus backplane signals, as defined in IEEE standard 796 for the P1 bus. Sun cards use the P2 bus in different ways, which may not be compatible with other vendors' use of this bus. For example, the Sun processor and Sun main memory cards do not support extended (24-bit) Multibus addressing on the P2 bus. They use the P2 bus for no-wait state access to main memory. As a general rule, Sun boards and other vendors' boards which use the P2 bus should therefore be installed in slots with separate P2 backplane connectors.

The signalling conventions used by the various Sun cards on the P2 bus are documented in the hardware reference manual for each board. For further details, please refer to the appropriate reference manual.

8.2.3. 150U Card Insertion Order

There are a few restrictions on the placement of cards in the card cage:

1) The design of the connector for the P2 bus requires that the processor board and the main memory expansion boards be inserted into adjacent slots.

2) The 3COM Ethernet Board must not share a P2 bus with any other board which uses signals from the P2 bus.

3) Because of its high power dissipation, the Xylogics 450 SMD Disk Controller Board should be placed in the most central position available in the backplane, as air flow is greatest across the center of the cardcage. However, the Xylogics board must not share a P2 bus with any Sun Microsystems board. For maximum air circulation, leave the slot to the left of the Xylogics controller empty if possible.

4) The Sky Floating Point Board must not share a P2 backplane with any Sun Microsystems board (it uses the P2 for power and ground).

5) A single Color Display Controller Board may be placed in any slot, since it uses no P2 signals; it should, however, be placed toward the middle of the cardcage to take advantage of the greater air circulation. If there are unused slots in the cage, leave one open to the immediate left of the Color Controller Board.

6) The 1/2-inch Tape Controller Board must not share a P2 backplane with the SUN-2 CPU or Memory boards. If present in a configuration which also has a Xylogics SMD disk controller, the 1/2-inch tape controller should be inserted 'above' the disk controller, because the very high throughput of the disk controller might otherwise lock out the lower-priority device for excessive periods.

8.3. Central Processor Board

The SUN-2 CPU (Central Processor Board) of the Sun Workstation is usually close to the top of the Multibus cardcage — ***usually*** in the first slot of the 150U cardcage, and the third slot of the 100U cardcage. The CPU board has two fifty pin cable connectors coming off from the top of the board. One of the cables splits in two and goes to the connectors labelled "RS232 A" and "RS232 B". See the subsection, ***Asynchronous Serial Ports*** for wiring information. The other cable also splits in two and goes to the connectors labelled "Keyboard" and "Mouse".

The jumpers on the CPU Board should be wired as follows:

• J700	Bus Priority In (BPRN/) Should be installed in the Model 100U only. See the subsection below, *CPU Multibus Priority,* for more information.
• J701	Common Bus Request (CBRQ/) (not installed) See the subsection below, *CPU Multibus Priority,* for more information.
• J702	Bus Clock (BCLK) (installed)
• J703	Constant Clock (CCLK) (installed)
• J801	Mouse VCC (installed)
• J400	27128 EPROM's (installed)
• J401	27256 EPROM's (not installed)

Only one of J400 and J401 must be installed at a time.

8.3.1. CPU Multibus Priority

As shipped, the Sun central processor board (CPU) is always configured as the highest-priority Multibus master by placing it in a higher priority slot that all other bus masters in the cardcage.

In a Model 100U, because of its serial cardcage, configuring the CPU as the highest-priority master also involves installing a jumper at location J700 (along the bottom of the processor board) to ground Bus Priority In (BPRN). The processor board can also be run at a lower bus priority by removing jumper J700, and thus allowing the processor to receive its BPRN signal from the arbitration circuit on the master with higher priority.

In the Model 150U, the CPU is configured as the highest priority master by simply placing the CPU in the highest priority slot in the parallel cardcage.

For both models, if the CPU board is used in conjunction with a Multibus DMA board, such as a disk controller, that does *not* support Common Bus Request (CBRQ), the CPU board must be configured such that it gives up the Multibus after every Multibus cycle. This is done by jumpering location J701, which is the second jumper from the left along the bottom of the board. This also will cause three additional wait states for each Multibus access. The Interphase 2180 is an example of a Multibus DMA board requiring this configuration. No other Sun-supplied boards require this configuration **if they are properly configured**. See the subsection on each board to determine how to do this. If you have boards from other vendors, check this carefully.

On the other hand, if all Multibus DMA devices (Bus Masters) *do* support CBRQ (or if there are no Multibus DMA devices), the CBRQ jumper on the processor board is not required. Instead, the CPU board will retain bus mastership until a lower priority master requests it by asserting CBRQ. Following a CBRQ, the processor board will yield mastership for at least one cycle. For certain machine configurations (especially those with color), significant speed enhancements result from removing this jumper.

8.3.2. Asynchronous Serial Ports

The Sun Models 100U and 150U provide two asynchronous serial ports. The ports are controlled by the SCC (Serial Communications Controller) on the SUN-2 CPU board. Both ports have RS-232-C cabling conventions but use RS-423 signalling. The two serial port connectors are labelled "RS-232 A" and "RS-232 B" on the cardcage enclosure back panel.

Note that older workstations (pre-1.0 Release) have only two serial ports — serial port A is configured as a DCE (Data Communications Equipment) and serial port B configured as a DTE. In addition, only serial port A on the older workstations has full modem control. This means that external cables built for the SUN-1 or SUN-1.5 CPU either may have to be modified to work correctly with the SUN-2 CPU, or a null modem cable may be required.

The serial ports on the Sun Workstation were designed primarily for connecting output peripherals — such as terminals, modems, printers, and plotters — and can drive these output lines at speeds up to 19.2 Kbaud. All ports provide the CTS, RTS, DTR, DSR, and DCD control lines required by some devices (such as modems) in addition to the transmit and receive lines. All ports are wired as DTE (data terminal equipment) ports, and thus permit direct connection of modems and the like (25 pins straight through). Computers and other DTE devices can be connected by using the null modem cable supplied with your Sun system, or any similar null modem cable. The null modem cable provided by Sun has pins 2 and 3 crossed, 4 and 5 crossed, and 6 and 20 crossed; pins 1 (Frame Ground) and 7 (Signal Ground) are carried through.

On the CPU Board, I/O channels A and B appear on 50-pin connector J1. The pin assignments for J1 are:

J1			
Pin	Signal	Pin	Signal
3	TxD A	28	TxD B
4	DB A	29	DB B
5	RxD A	30	RxD B
7	RTS A	32	RTS B
8	DD A	33	DD B
9	CTS A	34	CTS B
11	DSR A	36	DSR B
13	GND A	38	GND B
14	DTR A	39	DTR B
15	DCD A	40	DCD B
22	DA A	47	DA B
24	BSY A	49	BSY B

Connector pins not mentioned are not connected to anything.

The flat cables that connect to J1 on your CPU board convert this pinout to standard RS-232 pin assignments. The channels are implemented with three Zilog Z8530 dual UART chips. See the Zilog Serial Communications Controller Technical Manual (available from Sun Microsystems, Part Number: 800-1052-01) for programming information.

Getting the cabling right is a problem that should be familiar to anyone who has had to connect RS-232 equipment; sometimes it is most easily solved by experimenting. A piece of equipment known as an EIA Interface Adapter and Monitor — or, colloquially, a 'breakout box' — is invaluable in these exercises. Consult a hardware technician or Sun Microsystems if you need help.

Most devices (other than modems which are answering phones) do not assert carrier properly. There is therefore a software facility to simulate asserting carrier: see *zs*(4S) and *oct*(4S).

8.4. Memory Expansion Board

The Sun Memory Expansion Board(s) live on the P2 bus with the CPU — usually in the second and/or third slots at the top of the 150U Multibus cardcage, and in the second and/or fourth slots of the 100U cardcage. In both the Model 100U and 150U, memory boards must be physically adjacent to CPU board, due to the design of the P2 bus. In the Model 100U, memory boards may be placed between the CPU and other bus masters, as bus priority is passed through the memory expansion board. This is not an issue for the 150U, as its cardcage uses parallel bus arbitration.

The Sun memory expansion board provides 1 Mbyte of additional main memory for the Sun CPU. It is connected to the processor board via the Multibus P2 connector, and permits access by the 10MHz MC68010 without wait states. It uses the Multibus P1 connector only for + 5 Volt and Ground connections. Up to two of these boards can be added to either the Model 100U or 150U.

To configure a memory expansion board, the only choice you have to make is to select its starting memory location as either 0 or 1M within the on-board memory address space. This is done by setting the eight position DIP switch located at position U506 on the memory expansion board as follows:

- The *first* memory expansion board should have switch 1 on the DIP switch *closed* and switches 2 through 8 *open*.
- The *second* memory expansion board should have switch 2 on the DIP switch *closed* and switch 1 and switches 3 through 8 *open*.

8.5. Video Board

The Sun graphics system is a high-resolution bit-mapped frame buffer and display processor on one Multibus board. This board — called the "video board" or "monochrome display controller" — lives in the bottom slot in the Multibus cardcage. The video board has a 10-pin 'D shell' connector on the right hand end as you look at the top of the board.

The video board has only one DIP switch. This switch is used to set the board's Multibus base address (in Multibus Memory Space). It may be set to any address multiple of 0x20000 between 0x0 and 0xE0000. The standard address for the video board is 0xC0000. Additional video boards are placed at successively lower addresses.

The following table indicates which switch to set for each of the possible addresses. Note that *ONLY* the indicated switch should be turned on; all other switches should be in the off position.

Video Board Address Selection	
address	*DIP switch*
0x00000	8
0x20000	7
0x40000	6
0x60000	5
0x80000	4
0xA0000	3
0xC0000	2*
0xE0000	1

* This is the standard setting.

8.6. Xylogics 450 Disk Controller

The Xylogics 450 SMD disk controller board has five cable connectors on its top: four 26-pin connectors and one 60-pin connector. In the Model 100U cardcage, the disk controller board must be below the CPU for proper handling of bus arbitration; if there is a 1/2" tape controller board in the cardcage, it should be placed in a higher priority position than the disk controller. In the Model 150U cardcage, the disk controller is placed on the second P2 bus; again, it should occupy a lower priority slot than the 1/2" tape controller.

The Xylogics 450 SMD is an intelligent storage module controller/formatter, using bipolar microprocessor technology. It plugs directly into the Multibus and is a Bus Master during data transfers, using a variable burst length DMA technique. It directly connects via industry standard A and B cables to from one to four storage module drives which are available from a number of manufacturers. Sun Microsystems supports a family of such drives: the Fujitsu M2312K and M2284 Microdisk Drives, and the M2351 Mini-Disk Drive.

If you need to install the board yourself, the following placement considerations and configuration details apply:

- The Xylogics board should not share a P2 connector with the SUN-2 CPU or Memory Boards, as it has P2 traces which are incompatible with the CPU/Memory P2 bus.
- Because the Xylogics board is a Multibus Master, its relative slot number determines its priority (slot 1 is the highest priority master). The board must be placed in a lower priority position than the SUN-2 CPU board for proper handling of bus arbitration. It should also be placed in a lower priority position than the 1/2-inch Tape Controller Board, if there is one in the system.
- Since the Xylogics board dissipates a fair amount of heat, it should be placed in the most central position *possible* in the backplane (subject to the considerations listed above). For maximum air circulation, leave the slot to the left of the board (as installed) empty if possible.

Several sets of straps are provided for configuring the disk controller board. For proper operation in the Sun environment, the following options must be selected:

- Configure the 450 for 16-bit I/O addressing at address EE40 by installing exactly the following jumpers on jumper group JA through JE: JA3-JB3, JA8-JB8, JR1-JC1, JC2-JD2, JC3-JD3, JC4-JD4, and JE4-JE5.
- Configure the 450 for 20-bit memory addressing. Jumpers are: JM1-JM2 Out; JM3-JM4 In.
- Select interrupt level 2: wire pin E2 to pin JX4 (second pin from left in top row of JX pins).
- Enable the BPRO signal: jumper JE1 to JE2.
- Verify that jumpers JH1-JH2 and JN1-JN2 are NOT installed and leave other jumpers as shipped from the factory.

The first controller is configured at address ee40 by setting the JA through JE jumper group as described above. If you are configuring a second board, it should start at address ee48: the jumper at JC4-JD4 should be out and one at JR4-JC4 should be in. The system can support a maximum of two Xylogics 450 controller boards.

Please note that if you have an Interphase 2180 disk controller board in your system, only one Xylogics board can be supported. The Xylogics board should be configured for address ee40; the Interphase should be configured for address 48 as described below.

For the full specification and detailed description of the disk controller, see the supplied document: *Xylogics Model 450 Peripheral Processor SMD Disk Subsystems Maintenance and Reference Manual.*

8.7. Interphase 2180 Disk Controller

The Interphase Disk Controller Board has five cable connectors on its top: four 26-pin connectors and one 60-pin connector. In the Model 100U cardcage, the Interphase board must be below the CPU for proper handling of bus arbitration; if there is a 1/2" tape controller board in the cardcage, it is placed above the disk controller. In the Model 150U cardcage, the Interphase disk controller is placed on the second P2 bus; again, it should occupy a lower priority slot than the 1/2" tape controller.

Note that the Interphase controller does not generate CBRQ/; the CPU board must be configured to take this into account. On the CPU board, the jumper at location J701 must be installed. See the section, *CPU Multibus Priority* for more information.

The Interphase SMD 2180 is an intelligent storage module controller/formatter, using bipolar microprocessor technology. It plugs directly into the Multibus and is a Bus Master during data transfers, using a variable burst length DMA technique. It directly connects via industry standard A and B cables to from one to four storage module drives which are available from a number of manufacturers. Sun Microsystems supports two such drives: the Fujitsu M2312K ('D84') and M2284 ('D169') Microdisk Drives. Note that the Interphase 2180 is not suitable for use with the Fujitsu M2351 ('D474') Mini-Disk Drive, also supported by Sun.

Four sets of straps and two 8-bit DIP switches are provided for configuring the Interphase disk controller board. For the UNIX system on the Sun, the following options must be selected:

- Install the M3 jumper, selecting "increment by head" rather than "increment by cylinder".
- Switch #3 on DIP switch S2 must be *ON*, selecting level 2 interrupts.
- On DIP switch S1, set switch #8 *ON*, set #7 *OFF*, selecting 512 byte sectors, 34 sectors per track.
- Switches #1-6 on DIP switch S1 determine the address of the controller in Multibus I/O space. They supply the bits: 654321xx of the 8-bit Multibus I/O address. The board can be written to at four consecutive addresses, and read from at the lowest of the four. The Sun Workstation only uses addresses 0x40, 0x44, 0x48, or 0x4C. These addresses are obtained by setting the switches on DIP switch S1 as follows:

Selecting Disk Controller Base Address

Desired Address	*Switch Settings*					
	Switch 6	*Switch* 5	*Switch* 4	*Switch* 3	*Switch* 2	*Switch* 1
0x40	Off	On	Off	Off	Off	Off
0x44	Off	On	Off	Off	Off	On
0x48	Off	On	Off	Off	On	Off
0x4C	Off	On	Off	Off	On	On

Normally, the first Interphase controller in the system is configured at address 0x40 by setting switch #5 on DIP switch S1 *ON* and all others *OFF*. If you also have a Xylogics board in your system, configure the Interphase instead for address 0x48 by setting switch #2 *ON* and all others *OFF*.

- Finally, the Interphase controller board is normally configured to have lower Multibus priority than the CPU board by installing jumper E to F.

Sun recommends a maximum of one Interphase disk controller board per system.

For the full specification and detailed description of the disk controller, see the supplied document: *SMD 2180 Storage Module Controller/Formatter User's Guide.*

8.8. Nine-Track Tape Controller

If your Sun system is shipped with a nine-track tape unit, there should be a tape controller for that subsystem installed in the cardcage. The tape controller is a TAPEMASTER board from Computer Products Corporation. The TAPEMASTER nine-track tape controller board has two 50-pin cable connectors on its outside edge (viewed installed in the cardcage).

As shipped, the TAPEMASTER should be configured correctly for your Model 100U or 150U. If you are installing the board, please verify the following:

- For a 100U, locations 1 to 2 **are** jumpered; they must be jumpered for serial backplanes.
- For a 150U, locations 1 to 2 are **not** jumpered; they should be jumpered for serial backplanes only.

- For both models, locations 3 to 5 and locations 51 to 52 must be jumpered for 'proper' handling of CBRQ/: if these locations are jumpered, the tape controller will surrender the bus after each cycle. If these locations are not jumpered, you must configure your SUN-2 CPU so that it gives up Multibus after every cycle by installing the jumper at location J701 on the CPU board.

There are also board placement considerations involved in installation:

- The TAPEMASTER board should not be placed on the same P2 connector as the SUN-2 CPU and Memory Boards.
- The nine-track tape controller is a Multibus master. This means that the relative number of its slot determines its priority (slot 1 is the highest priority master). The board must be placed in a lower priority position than the CPU. If you have a Xylogics 450 SMD Controller Board in your system as well, place the TAPEMASTER Board in a higher priority position; otherwise, the Xylogics board will lock out the TAPEMASTER (due to its higher data transfer rate).

To cable up the tape controller, connect the 50-pin ribbon cable coming from the backpanel connector labelled "DISK/TAPE COMMAND B" to the "J2" connector in the *center* of the TAPEMASTER board. The 50-pin ribbon cable from the back panel connector labelled "TAPE COMMAND A" should go to the "J1" edge connector of the TAPEMASTER board.

As shipped, the tape controller board should be already set up to work correctly in a Sun system. Refer to the Computer Products Corporation TAPEMASTER Manual for details of configuring the board if required.

8.9. Quarter-Inch Tape Controller

The Sun tape interface board consists of an interface to an Archive 1/4" streaming tape unit and 256 KBytes of Multibus memory with parity. The tape controller is not a bus master, and boards which are Rev. C or 'greater' may actually be placed in a slot between bus masters (in the Model 100U), as they pass bus priority through.

Note that you must disable the Multibus memory on the Tape Controller Board if you have a SUN-2 CPU in your system. To do this, open all switches on switch package U50, as described below.

There are four 8-position DIP switches on the tape controller board. They are used to select the interrupt level at which the tape controller will interrupt the processor, select the base address for the Multibus memory on the board, and to select the base address for the Multibus I/O registers which communicate commands and status information between the tape controller and the central processor.

With the Multibus connectors facing you, the switches are all in the lower left hand corner of the board, at component locations U52, U50, U53, and U56. U52 is in the lower left-hand corner of the board; U50 is directly above U52; U53 is to the immediate right of U50; and U56 is to the immediate right of U53. For each package, switch #1 is on the right and switch #8 is on the left.

Switch package U52 selects the interrupt level; switch #1 selects interrupt level 0, switch #8 selects interrupt level 7. The desired switch should be closed (or ON); all others should be left open (or OFF). The table below shows the switch settings. The Sun system is configured to run with level 3 interrupts from the quarter-inch tape controller.

Required Interrupt Level	*Close Switch Number*	*Required Interrupt Level*	*Close Switch Number*
0	1	4	5
1	2	5	6
2	3	6	7
3	4 *	7	8

* Note that this is the standard setting.

Switch package U50 selects the base address for the Multibus memory. This memory must be disabled for proper operation with the SUN-2 CPU: all switches on U50 should be opened. If you are not using the tape controller board with a SUN-2 CPU, and need to make use of the Multibus memory, see the next subsection for switch settings.

Switch packages U53 and U56 select the base address of the I/O registers that the tape controller uses in Multibus I/O address space. The tape controller uses eight consecutive bytes of Multibus I/O space, and this block of eight registers must be located on an eight-byte boundary. DIP switch packages U53 and U56 decode address lines A15 through A3. Switch #8 on U53 corresponds to A15; switch #1 on U53 corresponds to A8; switch #8 on U56 corresponds to A7; and switch #4 on U56 corresponds to A3. Address lines A0 through A2 are not decoded through the switches.

Switch #1 on U56 enables/disables the tape section on the board.

Switches #2 and #3 on U56 are not used and their settings are irrelevant.

The table below illustrates the switch settings in a diagrammatic form. A *0* corresponds to an open (or OFF) switch, while a *1* corresponds to a closed (or ON) switch.

Required	*Switch Settings*	
Address	U53	U56
0xF000	11110000	00000000
0x1238	00010010	00111000
0x200 *	00000010	00000000
Disable Tape	xxxxxxxx	xxxxxxx1

* Note that this is the standard setting.

8.9.1. Tape Controller Multibus Memory

This subsection is irrelevant to you if you are using your 1/4" tape controller board with a SUN-2 CPU; the tape controller Multibus memory is unnecessary with such a configuration, and **must** be disabled as described above.

If you need to make use of the 256 KBytes of Multibus memory on the tape controller board, select the base address for the memory as follows.

Memory on the tape controller board comes in 128K byte chunks; there are two of these chunks. The chunk(s) of memory may be located on a 128K byte boundary anywhere within a one Megabyte address space by closing appropriate switches on package U50.

With the Multibus connectors facing you, the four switch packages are in the lower left hand corner of the board, at component locations U52, U50, U53, and U56. U52 is in the lower left-hand corner of the board; U50 is directly above U52. For each package, switch #1 is on the right and switch #8 is on the left.

Two adjacent switches on U50 should be closed. This places the 256K bytes of memory on the 0, 128K, 256K, 384K, 512K, 640K, or 768K boundaries. All other switches on package U50 should be opened.

If all switches on U50 are opened, all memory accesses to the tape controller board are disabled. The table below illustrates the switch settings in detail.

For 256K Byte Tape Controller Boards

Closing Switch Numbers	*Selects Address Range*	
	Decimal	*Hexadecimal*
8 and 7	0 — 256K-1	0x0 — 0x3FFFF
7 and 6	128K — 384K-1	0x20000 — 0x5FFFF
6 and 5	256K — 512K-1	0x40000 — 0x7FFFF
5 and 4	384K — 640K-1	0x60000 — 0x9FFFF
4 and 3	512K — 768K-1	0x80000 — 0xBFFFF
3 and 2	640K — 896K-1	0xA0000 — 0xDFFFF
2 and 1	768K — 1M-1	0xC0000 — 0xFFFFF

8.10. Ethernet Controller Board

A Sun Workstation in a networked environment is supplied with a 3COM 3C400 Ethernet Controller Board. The Ethernet controller is a memory mapped device in the Sun Workstation. There are several jumpers that should be already set correctly when the Controller leaves the factory. This section describes the settings of the jumpers and switches as they should be for a Sun Workstation.

The Ethernet controller should be set for 20-bit memory addressing. Look at the board with the components up and the Multibus edge connectors facing you. At the lower left corner of the board there are two jumpers labelled *JP1* and *JP2*. There should be a shorting jumper on *JP2*; there should not be any shorting jumper on *JP1*.

Just to the right of the address width selection jumpers, there is a row of jumper pins. The pins marked *MRDC* and *MWTC* should have shorting jumpers installed. The pins marked *IORC* and *IOWC* should not have any jumpers installed.

Then there is a row of jumpers marked *INT7* at one end, down to *INT0* at the other end; these select the interrupt level. For the Sun Workstation, select interrupt level 3 — install a shorting jumper on the fourth pair of jumper pins from the right hand end of the row. Remove the jumpers from all other INT(n) pins.

Finally, the switch marked *ADR17/* at the bottom right-hand corner of the board should have all its keyswitches set to *OFF*. The switch marked *ADR13/* in the lower left-hand corner of the board selects the memory base address for the board. The Ethernet controller board consumes 8K bytes of Multibus memory space. The Sun Workstation expects to find the Ethernet controller memory at address 0xE0000. This address is selected by setting keyswitches 1, 2, and 3 on the switch marked *ADR13/* to *ON* and the other switches to *OFF*. In all cases, keyswitch 8 on the switch marked *ADR13/* should always be *OFF*.

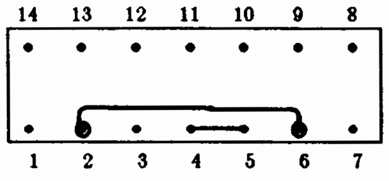
In component position I2 on the Ethernet controller board there is a PROM which contains the hardware ethernet address for this board.

8.11. Sky Floating Point Board

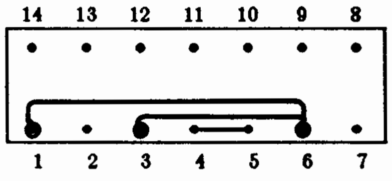
In both the Model 100U and the Model 150U cardcages, the Sky floating point processor may be placed in any available slot which does not share a P2 bus section with the CPU Board, as it uses the P2 in a manner which is incompatible with the CPU. The Sky board does IEEE format floating-point calculations much faster than they can be done in software, and so makes many compute-intensive programs run significantly faster.

To run in the Sun environment, the Sky board must be set to interrupt on level 2, and configured for Multibus I/O address 2000 (hexidecimal). If you purchased your Sky Board from Sun, the board should be shipped with these parameters already set. If you purchased your board from Sky, you must set these parameters by rewiring pins at jumper block locations JP02 and JP01, as follows:

- Holding the board with the Multibus edge connectors facing down (the location numbers on the board will be right side up), locate JP02 and JP01: they are the two blocks with exposed pins near the lower left-hand corner of the board. JP02 is the leftmost block. In the following diagrams, we assume that the board is in this position.
- To set the interrupt level to 2, replace the wire between pins 2 and 6 of JP02 by two wires: one between pins 1 and 6, and one between pins 3 and 6 (leave the shunt between pins 4 and 5 on):

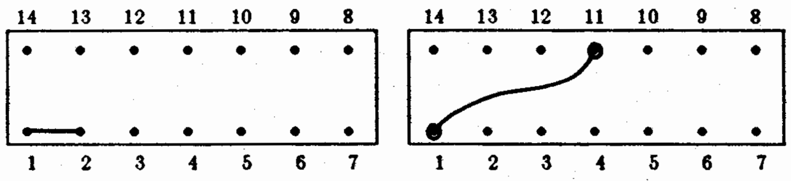


Before



After

- To set the address of the board to 2000 in Multibus I/O space, replace the shunt between pins 1 and 2 of JP01 with a wire between pins 1 and 11 of the same jumper block:



Before *After*

For more information, see the ***Installation Notes for SKY FFP*** document shipped with the board.

APPENDIX A – THE SUN WORKSTATION MONITOR

The central processor board (CPU) of the Sun Workstation has a set of ROMs which contain a program generally known as the "monitor". The monitor controls the operation of the system before the UNIX kernel takes control.

The first, second, and third major sections in this appendix cover the startup and bootstrap functions of the monitor. Under normal circumstances, the monitor automatically bootstraps the UNIX system after initial power-on; no manual intervention is required. These sections describe how it does that, and how to "boot" manually when necessary.

The fourth section lists messages which the monitor and boot program can display. These should be useful for troubleshooting. The fifth section covers PROM Monitor commands. The sixth treats the operational features of the two SUN-1 keyboards, the KeyTronic Model P2441 (VT100) and the Micro Switch 103SD30-2. The final section describes the Monitor routines which enable terminal emulation.

A.1. Power-On Self Test Procedures

When system power is first turned on, the monitor runs a quick self-test procedure. The test can have one of these results:

- Critical errors are found. The screen remains dark. The error is reported on eight LEDs on the CPU board in the card cage.
- No video board is found. The monitor sends its output to the serial port labelled "SIO-A" on the workstation backpanel. Connect an ASCII terminal to this RS-232 connector. Configure the terminal for 9600 baud, no parity, one stop bit. Then power on the workstation again and look for messages on the terminal.
- Non-critical errors are found. These are reported to the screen, and the system begins the automatic boot process.
- No errors are found. This is reported to the screen, and the system begins the automatic boot process.

A.1.1. Critical Errors from Self Test

Severe problems are reported using the eight miniature LEDs on the CPU board. These LEDs can be glimpsed thru the cooling slots on the left side of the SUN-1/100U, or by opening the front door of the SUN-1/150U. If your screen remains dark and your glimpse thru the slots shows more than one LED on in the middle of some board in the card cage, you'll probably have to slide the drawer out of your SUN-1/100U to really see the error code in the LEDs.

When power is first applied to the workstation, all eight LEDs light, then each lights quickly in sequence. Following this 'lamp test', the lights blink rapidly as each test is passed. The lights slow down as memory is tested; each of the two memory tests takes a few seconds per megabyte. Finally, three LEDs on the end light momentarily, then all the LEDs go off except for a middle LED which blinks about once a second. If your workstation follows this sequence, self-test has not found a critical problem. (Once UNIX or other programs have gained control of the system, they can use the LEDs in other ways. This description only applies to the power-on sequence.)

If at some point in the above sequence, the lights freeze (keep the same pattern for a long long time), or the sequence restarts from the beginning, there is a critical hardware problem with the workstation. The appropriate thing to do in this case is to contact Sun Microsystems Field

Service or your local Field Service organization. Copy down the pattern of lights (as well as you can, if it is repeating over and over); they contain important diagnostic information for Field Service.

A.1.2. Non-Critical Errors from Self Test

Non-critical errors result in a display like the following:

> Self Test found a problem in ***something***
>
> Wrote ***wdata*** at address ***addr***, but read ***rdata***.
> Damage found, ***damages***
>
> --> Give the above info to your service-person
>
> Sun Workstation, Model ***model number***, ***type of keyboard***.
> ROM Rev L, Serial number 12345, 2MB memory installed
>
> Auto-boot in progress . . .

Something shows what part of the system was most recently found to be malfunctioning. (If more than one error occurs, a summary of all is reported in the "damages" section, and details about the last one are reported here.)

Wdata is the data that was written into part of the system, or which was expected to be there if the system was functioning normally.

Addr is the address where the data was read and/or written. For memory errors, this is a physical memory address; for other errors, the interpretation of the address depends on ***something***.

Rdata is the data that was read back from ***addr*** and was found to be invalid because it was not the same as ***wdata***.

Damages is a list of all subsystems which were found to have errors. There is not enough room to save information about all of the errors that were found (only the last one), but this minimal information about each is recorded.

You should copy down the information from the screen, then call Sun Microsystems Field Service, or your local Field Service representative.

A.1.3. No Errors from Self Test

The following display results:

> Self Test completed successfully.
>
> Sun Workstation, Model ***model number***, ***type of keyboard***.
> ROM Rev L, Serial number 12345, 2MB memory installed
>
> Auto-boot in progress . . .

The monitor then begins auto-boot.

A.2. Automatic Boot Procedure

The monitor immediately tries to boot from a default device:

```
Auto-boot in progress

Boot: disk(0,0,0)vmunix
Load: disk(0,0,0)boot
Boot: disk(0,0,0)vmunix
Size: 215040+ 24576+ 30916 bytes
Sun UNIX 4.2, etc...
```

Disk is the device name of the "best" local or network disk the monitor could find. The file called *vmunix* is booted from it. This file does not have to contain a UNIX kernel; it can contain any program you like, as long as the disk is in 4.2BSD UNIX file system format. It is also possible to set up the disk to boot a small program which need not be in a UNIX file system. This discussion assumes that the disk is set up for UNIX.

A.3. Booting from Specific Devices

The Sun Workstation can be booted from:

- Any logical partition of a local disk.
- Any publicly available network disk partition on your Ethernet.
- The first file of a local tape drive.

As mentioned above, the monitor automatically attempts to boot *vmunix* from a default disk. If you want to boot a different program, or from a different device, you must stop the automatic boot process by aborting. The specific abort sequence depends on your keyboard type. For a SUN-1 Model 100U or 150U, the sequence is either 'SET-UP-A' (hold down the 'SET-UP key while typing the 'A' key) or 'ERASE-EOF-A'. When you abort, the monitor displays the address where it aborted and a ">" prompt, and waits for you to type a command.

The monitor's boot command looks like:

> b *device(parameters)pathname args*

where *device* is the type of hardware to boot from, *parameters* specify the address or partitioning of the device, *pathname* is the name of the actual file (in a UNIX file system on that device) to boot into memory, and *args* are optional arguments to the program.

To determine which devices your monitor ROMs are able to boot from, you can use the command:

> **b ?**

The devices are shown in order from "best" to "worst", as used by the automatic boot procedure to select a boot device.

Booting a program involves the cooperation of up to three different programs within your Sun Workstation. The first is the monitor, which reads in the "boot program" or "mini boot program", depending on the device. If the "mini boot" is used, it in turn reads in the "real" boot program. Once the "real" boot program is running, it finds and reads in the program you wanted to run. This process is more or less obvious depending what device you are booting from. Where there are minor differences between the commands you enter to the monitor and to the boot program, these will be noted. In general, the difference is that the monitor commands start with '**b**' and the boot program's commands don't.

A.3.1. Booting from Disk

For disk drives, the format of the boot command is:

> \> b *controller(address,drive,partition)pathname args*

Controller is the name of the disk controller which runs the specific disk; **ip** for the Interphase 2180 disk controller, **xy** for the Xylogics 440 or 450 controller, or **sd** for a SCSI disk controller. *Address* can either be a small number, indicating the *n*th standard controller board, or is the physical address of the controller on the Multibus. *Drive* is the unit number of the disk on that specific controller. *Partition* is a number corresponding to the logical partition on the disk where the file specified by *pathname* can be found. Zero corresponds to partition 'a', 1 to 'b', etc. *Pathname* is the name of the file to boot. *Args* are optional arguments.

A.3.2. Booting from Network Disk

To boot the system from network disk, use a command like:

> \> b **nd(***etheraddr, partition***)***pathname args*

where *etheraddr* is the lower 3 bytes of the Ethernet address of the desired network disk server machine, in hexadecimal. *Etheraddr* can be zero if the machine being booted is a registered client of the server machine.

Partition is the desired public partition number on the specified server. The correspondence between this number and a real disk partition is defined in */etc/nd.local* on the server machine.

A third parameter may be supplied, but it will be ignored.

Pathname is the name of the file to boot. *Args* are optional arguments.

A.3.3. Booting from Tape

The Sun Workstation can be booted from industry standard nine-track magnetic tape, from an Archive quarter-inch cartridge tape controlled by a Sun 1/4-inch tape controller, or from a quarter-inch tape controlled by a SCSI tape controller, using the following command:

> \> b *tape***(***controller, unit, filenum***)**

Tape is either **ar** for an Archive controlled by a Sun controller, **mt** for nine-track tape, or **st** for a SCSI tape controller. *Controller* is a small number indicating the nth standard magnetic tape controller in the system, or is the Multibus address of the controller. *Unit* specifies which tape drive on the controller is to be used. *Filenum* specifies which file of the tape is to be booted. By convention, boot commands number the first file on the tape file #0, the second #1, and so on.

The monitor ignores the supplied value of *filenum* and can only boot the first file on a tape. To boot a file further down the tape, use the monitor to boot the "boot" program. Sun-supplied UNIX distribution tapes always have the "boot" program on the first file of the tape.

A.3.4. Booting Files from the Default Device

To boot any file from the default device, enter:

> \> b *pathname args*

This is useful for booting standalone utility programs, once your disk is loaded, or for trying new versions of the UNIX kernel.

A.4. Messages from the Monitor and Boot Program

Abort at *aaaaaa*

The monitor has aborted execution of the current program because you entered the "abort sequence" (upper left key held while pressing "A") from the Sun keyboard, or pressed Break on a serial console. *Aaaaaa* is the address of the next instruction. You can continue the program from there by entering the "c" command.

Address Error, addr: *xxxxxx* at *aaaaaa*

The current program has stopped because it made an invalid memory access. *Xxxxxx* is the (invalid) address; *aaaaaa* is an address near to the instruction which failed (typically two to ten bytes beyond). There is no general way to recover from this error, except to debug the program.

ar: cartridge is write protected

The current program is trying to write on an Archive tape cartridge, but the "Safe" switch at the top left corner of the cartridge is set to prevent writing on the tape.

ar: *xxxx* error

The monitor or boot program is trying to boot from an Archive tape, and encountered an unexpected error. The status bytes *xxxx* can be decoded by looking under "Read Status Command" in the *Archive Product Manual.* This error could be caused by incorrect cables, a bad tape, or other problems.

ar: drive not responding

The monitor is trying to boot from an Archive tape, but can get no response from the tape drive. This can occur if your system contains an Archive controller board but no tape drive, or if the tape drive's cable is loose or disconnected, or if the tape drive's power is not on.

ar: invalid state *xx*

This message indicates that the standalone I/O system has a bug in its Archive driver.

ar: no cartridge
ar: no cartridge in drive

The monitor or boot program is trying to boot from an Archive tape, but there is no cartridge in the tape drive.

ar: no drive

The monitor or boot program is trying to boot from an Archive tape, but the specified drive does not exist. Typical Archive configurations only include drive 0.

ar: RDST gave Exception, retrying

The current program is trying to use the Archive tape drive, and encountered an error. The error is probably caused by hardware. Check the cable(s) that connect the tape drive to the system.

ar: triggerred at idle *xx*

This message indicates that the standalone I/O system has a bug in its Archive driver.

Auto-boot in progress...

The monitor has finished its power-on sequence and is looking for a good device to boot the Unix system from.

Bad device

The current program (possibly the boot program) has tried to open a file without a device name (eg *xy()*). This could mean that the boot command you typed had no device name.

Bad format

The boot program is trying to boot from a file which is not in a standard UNIX *a.out*(5) format. The boot program can only boot files which are in this format, which is generated by the *ld*(1) command.

bn negative

bn ovf *dd*

bn void *dd*

A standalone program (such as the boot program) is trying to read a file from disk or net disk, and the block number it is trying to read is invalid.

Boot:

The boot program is waiting for you to specify a device and file name to boot. The boot program accepts the same commands that the monitor would, without the initial 'b'. See the section *Booting From Specific Devices* above.

Boot: *dev(ctlr,unit,part)name options*

The monitor or boot program is preparing to boot the specified file from the specified device. Either you typed a boot command, or this is an auto-boot after power-on. *dev* is the device type; *ctlr, unit,* and *part* are the controller, unit-within-controller, and disk partition number. *Name* is the name of the file to boot from, if any; *options* are arguments for the booted program, such as '-s'. If you enter a boot command to the monitor, this message will be printed twice; once by the monitor and once by the boot program.

boot failed

The boot program has tried to boot the device and/or file you specified, but could not. A preceding message should give more details about why.

Boot syntax: b [!][dev(ctlr,unit,part)] name [options]

boot syntax: dev(ctlr,unit,part)name

You have entered an invalid boot command. This message describes the general format of the boot command to remind you. The first form is used by the monitor; the second (without the 'b') is used by the boot program. Don't type the brackets; they indicate optional parts of the command.

Break at *aaaaaa*

The current program has stopped because it tried to execute a breakpoint instruction at address *aaaaaa*. Hopefully, you set this breakpoint with the 'z' command. You can continue execution by pressing Return, or with the 'c' command.

Break *aaaaaa* installed

The monitor has installed a breakpoint at the specified address, because you typed a 'z' command.

Bus Error, addr: *xxxxxx* at *aaaaaa*

The current program has stopped because it tried to make an invalid memory access. The reason for the error is shown before this message. The memory location being accessed was *xxxxxx*, and the instruction which made the access is near location *aaaaaa*. There is no way to recover from this error, in general, except to debug the program.

Can't write files yet...Sorry

The current program is trying to write to a disk or network disk file thru the standalone I/O system. Writing on files (as opposed to writing on devices) is not supported when running standalone (i.e. before booting the Unix kernel).

Corrupt label

Corrupt label on head *h*

The monitor or boot program is trying to boot from a disk. The first sector of the disk appears to be a label (as it ought to be), but the checksum on the label is wrong. Try again a few times; if the problem recurs, you should probably re-label your disk. See sections "Using the Diag Utility" and "Labelling the Disk" in the chapter, *Installing UNIX for the First Time.* Before trying to re-label your disk, make sure that you know what ought to be in the label — writing the wrong label on the disk is highly likely to cause destruction of some or all files on the disk.

count=*ddd*?

A standalone program is trying to write to a device and has specified a block size which is not a multiple of 512. The write proceeds anyway, but may cause incorrect results.

Damage found, *damage...*

As part of the power-on self test procedure, the monitor has found damage in one or more parts of the system. This message should be reported to your local service representative or Sun Microsystems Field Service. *Damage* is a list of subsystem names, such as "memory" or "timer".

Exception *ee* at *aaaaaa*

The current program has stopped because it got an interrupt. The interrupt could have been caused either by hardware or software. *ee* is the hexadecimal address of the interrupt vector used; you can look it up on a Motorola 68010 or 68000 reference card or CPU manual to see what kind of interrupt has occurred. *Aaaaaa* is the address of the instruction where the interrupt occurred.

Extra chars in command

Your previous 'u' command had extra, unrecognized characters on the end.

FC*n* space

The address space being accessed by the monitor's memory reference commands is defined by Function Code number *n*. See the Motorola 68010 CPU manual for more information. This message is printed by the 's' command.

For phys part *p*, No label found.

The boot program is trying to boot from a nonzero "physical partition" on a disk, and can't find a label. Physical paritiitions are used for disk drives part of which are fixed and part of which are removable.

—> Give the above information to your service-person.

The monitor has found a hardware problem while executing its power-on self test procedure. The preceding messages describe the error in more detail. You should report the problem to your local service staff, or to Sun Microsystems Field Service.

Giving up...

See "Waiting for disk to spin up...". The monitor has given up on waiting for the disk to become ready.

ip: error *xx*

The monitor or boot program is trying to boot from an Interphase disk controller, and encountered an unexpected error. The error number *xx* can be decoded by looking in appendix B of the Interphase *SMD 2180 Storage Module Controller/Formatter User's Guide.* This problem is usually caused by loose or unplugged disk drive cables.

ID PROM INVALID

The monitor cannot find a valid ID PROM on the CPU board. The ID PROM contains the machine's serial number and other information specific to your system. If you have recently changed CPU boards, it is possible that you installed the ID PROM incorrectly. You should attempt to locate the correct ID PROM and install it in your CPU board.

Invalid Page Bus Error ...

See "Bus Error...". The attempted access was invalid because the virtual page containing the addressed data has been designated as invalid. It usually means that your program is using the wrong address.

Invalid selection

Your last 'u' command was not correct.

Keyboard error detected

The microprocessor on the keyboard has reported an error. This probably means that your keyboard hardware is broken and should be replaced.

Load: *dev(ctlr,unit,part)*boot

The monitor has loaded in the "mini" boot program from a disk drive or network disk. The mini boot is now reading in the "real" boot program from the disk. The "real" boot program will then read in the program you requested.

No controller at mbio *xxxx*

The monitor is trying to boot, but it can't find a device controller where you asked it to look. You should try another boot command, or make sure that your controller board is plugged in and has all its jumpers and switches set properly.

Lower Byte Parity Bus Error ...

See "Bus Error..." and "Parity...". The preceding access was to memory with a parity error in its lower byte.

Misplaced label on head *n*

The monitor or boot program is trying to boot from a disk. It has found a label which seems to identify itself as belonging to a different read/write head from the one where the label is written. See "Corrupt label" above.

mt: controller does not initialize

The monitor is trying to boot from nine-track tape, and could not get the tape controller to complete its initialization sequence. This might indicate a possible defect in the controller, or incorrect configuration of the controller board.

mt: error 0x*xx*

The monitor is trying to boot from nine-track tape, and encountered an unexpected error. The error number *xx* can be decoded by looking in appendix C of the *Tapemaster Product Specification*, which is supplied with your tape drive.

mt: unit not ready

The monitor is trying to boot from nine-track tape, but the tape drive is not ready. Check to see that the drive is on-line.

nd: no file server, giving up.

The monitor or boot program is trying to boot from a network disk server over the Ethernet. It has been retrying for a long time and there is no response from the server. Check the Ethernet address in the boot command; if it is zero, make sure your machine's Ethernet address is recorded in the server's */etc/nd.local* file. If that's OK, check your Ethernet cable connection, see whether the server is running correctly, and/or see whether other machines on the network can communicate.

No default boot devices

The monitor is trying to boot but it can't find a disk or Ethernet interface to boot from. To boot from a tape, you must specify the device name explicitly, as in '**b ar()**'.

No label found.

No label found -- attempting boot anyway.

The monitor or boot program is trying to boot from a disk, and can't find a valid label on the disk. This is best fixed by booting a copy of *diag*(8S) from a different device (eg, network disk or tape) and using the 'verify label' and 'label' commands. See the warning under "Corrupt label" above. This error might also be caused by missing or bad disk cables.

No more file slots

The current program is using the standalone I/O library and has opened too many devices or files.

not a directory

The current program (possibly the boot program) has tried to open a disk or netdisk file with a pathname, but one of the names in the path is not a directory.

name not found

The boot program has searched for the requested file, but cannot find it. You can retry your boot command, using "*" instead of *name*, to get a list of the names that exist in that directory.

xxxxxx now.

The monitor is displaying the current breakpoint address *xxxxxx* in response to your 'z' command.

null path

The current program (possibly the boot program) has tried to open a file whose name is empty.

PageMap *aaaaaa* [*ss*]: *xxxxxxxx*?

The monitor is displaying or modifying a page map entry because you entered a 'p' command. *Aaaaaa* is the virtual memory address whose map entry is being examined. *Ss* is the segment map entry which is being used to map this page map entry and page. *Xxxxxxxx* is the page map entry itself. You can enter a space and Return to get back to command mode.

Parity Bus Error ...

See "Bus Error...". The attempted access was probably valid, but was cancelled because the preceding access was to memory with bad parity. (Parity errors are reported on the memory cycle **after** the failing cycle.) If neither "Upper Byte" nor "Lower Byte" is reported, the parity on both bytes was invalid. The access address printed in the Bus Error message is probably not relevant to the parity error. There is no general way to recover from this error; a good starting point, though, is to boot *parscan*(8S), which will search all of memory for parity errors.

Please clear keyboard to begin

The monitor is trying to listen for your typing on the keyboard, but cannot tell which shift keys are down until you release all the locking keys (Caps Lock and Shift Lock). Once it has seen all the keys released, it can then track the movements of the keys and typing will work.

Please start it, if necessary, -OR- press any key to quit.
See "Waiting for disk to spin up...".

Possible boot devices:
You have asked for a list of boot devices with the '**b ?**' command.

Protection Bus Error ...
See "Bus Error...". The attempted access was invalid because your program is not permitted to access the addressed data in this way; for example, writing to that page is disallowed.

ROM Rev *x*, Serial number *sssss*, *mm* memory installed
The monitor is identifying its revision level and the system configuration as part of the power-on sequence. *x* is a letter or phrase indicating which particular version of the monitor is installed; *sssss* is your machine's serial number; *mm* shows how much memory was found during system configuration at power-on. If an even number of megabytes is installed, *mm* is displayed as "*nn*MB"; otherwise as "*nnnn*KB".

Retensing...
The monitor is attempting to boot from an Archive tape. Its first attempt failed, so it is retensing the tape (winding all the tape from one reel to the other), which makes it much more likely to succeed.

Seek not from beginning of file
The current program is using the standalone I/O library and has tried to do an unsupported seek operation.

SegMap *aaaaaa*: *xx*?
The monitor is examining or changing the segment map in response to your recent 'm' command. You can enter a space and Return to get back to command mode.

Self Test completed successfully
The monitor has completed its power-on self test without finding any hardware problems.

Self Test found a problem in *something*
The monitor has completed its power-on self test and found a problem in some subsystem. *Something* describes the general location of the error. Further messages give more details; see "Wrote ..." and "Damage found...".

Short read
The boot program is trying to boot a program from disk or net disk. It has located the program, but encountered an error while reading it into memory.

Size: *text* + *data* + *bss* bytes
The boot program is loading in the program you requested. *Text*, *data*, and *bss* are the sizes of the three sections of the program; they are printed as each is read into memory. After finishing display of this message, the boot program begins execution of your program; further messages can come from it instead of from the boot program or monitor.

Sun Workstation, Model Sun-1/100U or Sun-1/150U, *keyb* keyboard
The Model 100U or 150U workstation has just been powered on, or you entered a 'kb' command, and the monitor is identifying its configuration. *Keyb* is either VT100 or Two-tone, depending which keyboard your monitor ROMs support.

Sun Workstation, Model Sun-2/120 or Sun-2/170, Sun-2 keyboard
The Model 120 or 170 workstation has just been powered on, or you entered a 'kb' command, and the monitor is identifying its configuration.

Timeout Bus Error ...

See "Bus Error...". The attempted access was invalid because no device responded at the addressed location. This most often happens for Multibus references. The program was probably trying to access a device or section of memory which does not exist, or which has gotten into a hung state. If this occurs in response to a boot command, the device you are trying to boot from is not installed in your system.

tm: error *nn* during config of ctlr *cc*

tm hard err *nn*

tm: no response from ctlr *cc*

A standalone program (possibly the boot program) is trying to use the Tapemaster nine-track tape drive, and has encountered an error. This could be caused by a bad or missing tape, loose or misplugged cables, incorrect jumpers on the Tapemaster controller board, or hardware errors. *Nn* can be decoded by looking in the Tapemaster *Product Specification.*

Trace *iiii* at *aaaaaa*

The current program has taken a trace interrupt. *Iiii* is the hexadecimal representation of the next instruction to be executed; *aaaaaa* is its address. If the trace was started with the 'tc' command, execution does not stop; instead a long series of these messages occurs. You can stop it by pressing any key. If the trace was started with 'ty', execution stops. You can continue by pressing Return or by using the 'c' command. You can cancel the trace with the 'tn' command and then continue with 'c'.

Trace Off

The monitor has stopped a trace in response to your 'tn' command.

Tracing...

The monitor has begun a 'tc' or 'ty' command for you.

Unknown device

The current program (possibly the boot program) has tried to use a device which is unknown to the standalone I/O system.

Upper Byte Parity Bus Error ...

See "Bus Error..." and "Parity...". The preceding access was to memory with a parity error in its upper byte.

u*i* i, u*o*o, ua*abaud*, ub*bbaud*, uu*aaaaaa*, u*echo*

The monitor is describing its console and serial port configuration in response to a 'u' command. *I* is the input device (**k** for keyboard, or **a** or **b** for a serial port); *o* is the output device (**s** for screen, or **a** or **b**); *abaud* and *bbaud* are the baud rates on the serial ports; *aaaaaa* is the address of the Zilog 8530 chip which implements the serial ports, and *echo* is 'e' if input echoing is enabled ("full duplex") or 'ne' if disabled ("half duplex").

Using RS232 A input.

The monitor did not find the Sun keyboard, so it is taking input from one of the serial ports on the back of the Workstation, marked "RS232 A". If this is unexpected, make sure that the keyboard is plugged into the correct socket on the workstation. The keyboard must be plugged in before system power is turned on. If you connect a Sun keyboard after this message appears, you can let the monitor know about the keyboard by entering the Abort sequence (hold down the upper left key on the Sun keyboard, and press 'A'). The monitor will switch to using the Sun keyboard since that's where the Abort was typed. Then type 'c' to continue whatever program was running when you aborted.

If you don't want to use a Sun keyboard, connect a normal ASCII terminal to the "RS232 A" connector on the back panel. Configure the terminal for 9600 baud, no parity, one stop bit. Things that you type on the terminal will be displayed on the Sun video screen, if you have one, or on the terminal's screen.

Waiting for disk to spin up...

The monitor is trying to boot from a disk. The disk is not ready, so the monitor is waiting in the hope that the disk is just starting to spin and will become ready soon. If you get this message when the power has been on for a while, your disk cables are probably loose or misconnected.

Watchdog reset!

The current program has stopped executing with a "double bus fault". This is explained in detail in the Motorola 68010 manual; the two most common causes are that low memory (interrupt vectors) has been overwritten, or the system stack pointer is pointing to an invalid address. There is a serious bug in your program if this occurs.

What?

You typed a command that the monitor does not recognize. Try again.

Wrote *wdata* at address *addr*, but read *rdata*

The monitor has completed its power-on self test and found a problem in some subsystem. The preceding "Self Test found a problem..." message describes which part of the system was in error. This message gives more details about the error. *Wdata* is the data that was written into part of the system, or which was expected to be there if the system was functioning normally. *Addr* is the address where the data was read and/or written. For memory errors, this is a physical memory address; for other errors, the interpretation of this field depends on what subsystem was being tested. *Rdata* is the data that was read back from *addr* and was found to be invalid because it was not the same as *wdata*. This information should be written down and reported to your local Field Service organization, or to Sun Microsystems Field Service. See the section *Non-Critical Errors From Self Test* above.

xy: error *nn* cmd *xx*

xy: error *nn* bno *bbbbb*

xy: init error *xx*

The monitor is trying to boot from the Xylogics disk and has encountered an error. The command being executed at the time is defined by the hexadecimal value *xx* (if present); the block number is *bbbbb* (if present), and the particular error is encoded as *nn*. The error and command can be decoded by looking in the Xylogics manual.

xy: no bad block info

The boot program is trying to read from the Xylogics disk, but can't find the information about bad blocks on the disk. It continues, but if the program attempts to read any bad blocks (which have been remapped to elsewhere on the disk), the attempt will fail.

zero length directory

A standalone program (possibly the boot program) is trying to read a file from disk, but one of the directories in the path name has no files in it. The file system should be checked and fixed by using *fsck*(8).

A.5. The PROM Monitor Commands

A.5.1. Command Syntax

The command format understood by the monitor is quite simple. It is:

```
<verb><space>*[<argument>]<return>
```

The *<verb>* part is always one alphabetic character; case does not matter. *<Space>* * means that any number of spaces is skipped here. *<Argument>* is normally a hexadecimal number or a single letter; again, case does not matter. Square brackets "[ ]" indicate that the argument portion may be optional. When typing commands, *<backspace>* and *<delete>* (also called *<rubout>*, generated by the key labelled *<backtab>* on the non-VT100 Sun keyboard) erase one character; control-U erases the entire line. *<Return>* means that you should press the carriage return key.

A.5.2. Syntax for Memory and Register Access

Several of the commands *open* a memory location, map register, or processor register, permitting you to examine and/or modify the contents of the specified location. These commands include **a**, **d**, **e**, **l**, **m**, **o**, **p**, and **r**.

Each of these commands takes the form of a command letter, possibly followed by a memory address or register number, followed by a sequence of zero or more "action specifier" arguments. The various options are illustrated below, using the **e** command as an example. The user types the **boldface** parts, with a RETURN at the end of each command.

If no action specifier arguments are present, the address or register name is displayed along with its current contents. You may then type a new hexadecimal value, or simply <return> to go on the next address or register. Typing any non-hex character and RETURN will get you back to command level. For registers, "next" means within the sequence D0-D7, A0-A6, SS, US, SR, PC. For example, the following command sets consecutive locations 0x1234 and 0x1236 to the values 0x5678 and 0x0000 respectively:

```
> e1234
001234: 007F? 5678
001236: 51A4? 0
001238: C022? q
>
```

A non-hex character (such as question mark) on the command line means read-only:

```
> e1000 ?
001000: 007F
>
```

Multiple nonhex characters read multiple locations:

```
> e1000 ???
001000: 007F
001002: 0064
001004: 1234
>
```

A hex number on the command line does store-only:

```
> e1000 4567
001000 -> 4567
>
```

Multiple hex writes multiple locations:

```
> e1000 1 2 3
001000 -> 0001
001002 -> 0002
001004 -> 0003
>
```

Nonhex followed by hex reads, then stores.

```
> e1000 ? 346
001000: 007F -> 0346
>
```

Finally, reads and writes can be interspersed:

```
> e1000 ? 1 ? ? 3 4
001000: 007F -> 0001
001002: 0064
001004: 1234 -> 0003
001006 -> 0004
>
```

Spaces are optional except between two consecutive numbers. When actions are specified on the command line after the address, no further input is taken from the keyboard for that command; after executing the specified actions, a new command is prompted for. Note that these commands provide the ability to write to a location (such as an I/O register) without reading from it; and provide the ability to query a location without having to type a **q** or other non-hex character to exit the command.

A.5.3. Command Descriptions

A [*n*][*actions*]	Open A-register *n* ($0 \leq n \leq 7$, default zero). A7 is the System Stack Pointer; to see the User Stack Pointer, use the **r** command. For further explanation, see the section, "Syntax for Memory and Register Access" above.
B [*args*]	Boot. Resets appropriate parts of the system, then calls the boot routine, which is located in the second pair of PROMs on the processor board. Passes to the boot routine a pointer to the argument string following "b". This allows bootstrap loading of programs from various devices such as disk, tape, Ethernet, or serial ports. Simply typing "b" gives you a default boot, which is configuration dependent. For an explanation of the booting options, see the sections on "Booting," earlier in this appendix. See also the "User's Guide for the Sun Workstation".
C [*addr*]	Continue a program. The address *addr*, if given, is the address at which execution will begin; default is the current PC. The registers will be restored to the values shown by the A, D, and R commands, except for the system stack pointer.

D [*n*][*actions*] Open D-register *n* ($0 \leq n \leq 7$, default zero). For a detailed explanation, see the section, "Syntax for Memory and Register Access" above.

E [*addr*][*actions*] Open the word at memory address *addr* (default 0x0); odd addresses are rounded down. For a detailed explanation, see the section, "Syntax for Memory and Register Access" above.

G [*addr*][*param*] Start the program by executing a subroutine call to the address *addr* if given, or else to the current PC. The values of the address and data registers are undefined; the status register will contain 0x2700. One parameter is passed to the subroutine on the stack; it is the address of the remainder of the command line following the last digit of *addr* (and possible blanks).

K [*number*] If *number* is 0 (or not given), this does a "Soft Reset": it resets the system stack pointer to 0x1000. This can be useful after exceptions or other anomalous situations. If *number* is 1 this does a "Medium Reset", which re-initializes most of the system without clearing memory. If *number* is 2, a hard reset is done and memory is cleared. This is equivalent to a power-on reset and causes the PROM-based diagnostics to be run, which can take ten seconds or so (see chapter 6, "Diagnostics").

L [*addr*][*actions*] Open the longword at memory address *addr* (default 0x0); odd addresses are rounded down. For a detailed explanation, see the section, "Syntax for Memory and Register Access" above.

O [*addr*][*actions*] Opens the byte location specified (default zero). See the section, "Syntax for Memory and Register Access" above. The byte vs. word distinction can be a problem on the Multibus, since some Multibus boards follow the 8086 convention for byte ordering within words, which is the reverse of the 68000 convention.

R [*actions*] Opens the miscellaneous registers (in order) SS (Supervisor Stack Pointer), US (User Stack Pointer), SR (Status Register), and PC (Program counter). Alterations made to SS have no effect on the real SS, although the displayed value changes. For further explanation, see the section, "Syntax for Memory and Register Access" above.

T [*arg*][*cmds*] Trace. The possible arguments are **y**, **n**, or **c**. **Ty** (Trace Yes) turns on tracing and executes the first instruction. **Tn** (Trace No) turns off tracing. The Trace interrupt vector is saved by the **ty** command and restored to its previous value by the **tn** command. Continuous tracing may be enabled with the **tc** command, which acts like **ty** except that after each instruction is traced, a **c** command is simulated. This results in a display of the address and opcode of each instruction executed. Continuous tracing is interrupted by pressing any key; you can then type "**tn**" to turn off tracing.

Continuous tracing can also execute a given set of monitor commands after each instruction. This is enabled by typing

tc *cmd*;*cmd*;*cmd*;....

After each trace interrupt, the commands are executed in turn, then a **c** command is simulated. This provides a way to display the contents of memory or registers after each instruction. Again, continuous tracing is interrupted by pressing any key.

For more information see the section, "Trace Traps," below.

U [*arg*] The U command manipulates the on-board UARTs (serial ports) and switches the current input or output device. The argument may have the following values ("{ab}" means that either "a" or "b" is specified):

{ab}	Select UART a (or b) as input and output device
{ab}io	Select UART a (or b) as input and output device
{ab}i	Select UART a (or b) for input only
{ab}o	Select UART a (or b) for output only
k	Select keyboard for input
ki	Select keyboard for input
s	Select screen for output
so	Select screen for output
ks, sk	Select keyboard for input and screen for output
{ab}#	Set speed of UART a (or b) to # (such as 1200, 9600, ...)
u addr	Set UART address

If no argument is specified, the U command reports the current values of the settings. If no UART is specified when changing speeds, the "current" input device is changed.

At power-up, the following default settings are used: The default console input device is the Sun keyboard or, if the keyboard is unavailable, UART a. The default console output device is the Sun screen or, if the graphics board is unavailable, UART a. All serial ports are set to 9600 baud.

Z [*addr*] Display or set the breakpoint. If *addr* is omitted, the breakpoint is displayed. If an address of zero is specified, the breakpoint is removed. Otherwise, the breakpoint is set to the given address.

The interrupt vector for TRAP #1 (breakpoints) is saved when the breakpoint is set, and restored when the breakpoint is removed.

A.6. Tracing Programs

The monitor provides several facilities for tracing program execution. They are quite primitive, however, since addresses and instructions are represented only in hex, not symbolically†. If you have a symbol table listing of your program, you will at least know where each routine starts.

A.6.1. Breakpoint Traps

The use of a Breakpoint trap (BPT) allows you to run a program and regain control when execution reaches a certain location. The monitor currently can only maintain one breakpoint trap at a time. A breakpoint trap is set at address *x* by typing Z x. The current breakpoint address may be queried by typing Z alone. The current breakpoint may be cancelled by typing Z 0.

You can then start or resume your program with the C (or possibly G) command. Execution will proceed until the trap is reached, at which point you will get a message such as

```
Break 6708 at 00207C
```

where 6708 is the opcode of the instruction. At this point, you may press RETURN and the program will continue executing the "broken" instruction, leaving the breakpoint still set. Or you may examine registers or memory, clear the BPT, or set a new one. You may then

†Most operating systems provide a higher-level debugger, such as UNIX's *adb* or *dbx*.

continue by using the C command.

If you load a new program while a BPT is set, the monitor will normally be able to detect this and remove the breakpoint.

A.6.2. Trace Traps

Commands exist for beginning and ending tracing. **Ty** (Trace Yes) turns on tracing and executes the first instruction. **Tn** (Trace No) turns off tracing. After each instruction is executed, the message

Trace *opcode* at *pc*

appears, where *pc* and *opcode* indicate the *next* instruction to be executed. If you press RETURN right after this message appears, the next instruction will be executed. If you enter any other commands, you must use the C command to continue.

The Trace interrupt vector is saved by the **ty** command and restored by the **tn** command. Tracing is no longer turned off (as in monitor Revision C) by instructions which modify the Status Register, such as RTE, MOVE to SR, or AND to SR. Each of these instructions causes a trace trap to occur after execution, and the monitor turns the trace bit back on again. Tracing is only reset by the **tn** command or by explicitly resetting the bit in the status register using Monitor commands. (Note that in this case, the interrupt vector will not be restored.)

Continuous tracing may be enabled with the **tc** command, which acts like **ty** except that after each instruction is traced, a **c** command is simulated. This results in a display of the address and opcode of each instruction executed. Continuous tracing is turned off by pressing any key. Continuous tracing can also execute a set of commands after each instruction, by typing

tc cmd;cmd;cmd;....

After each trace interrupt, the commands are executed in turn, then a **c** command is simulated. This provides a way to display the contents of memory or registers after each instruction. Again, continuous tracing is halted by pressing any key.

A.7. The SUN-1 Keyboard

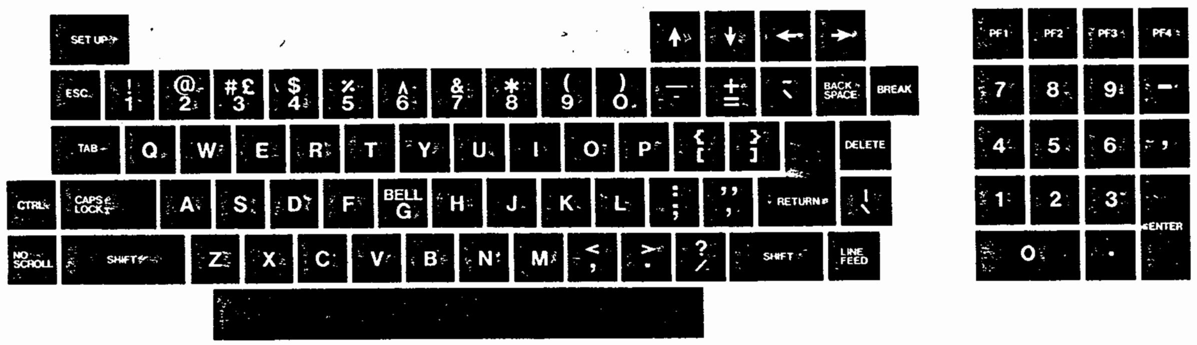
The SUN-1 Workstation (Model 100U or 150U) is supplied with the KeyTronic model P2441, a detached keyboard featuring the DEC VT100 key layout. A number of earlier-produced units were shipped with the Micro Switch 103SD30-2, which has a non-VT100 layout. These keyboard layouts are illustrated in Figure KEYS. The key mappings are table-driven in software and can be redefined.

This chapter describes the operational features of the SUN-1 keyboard.

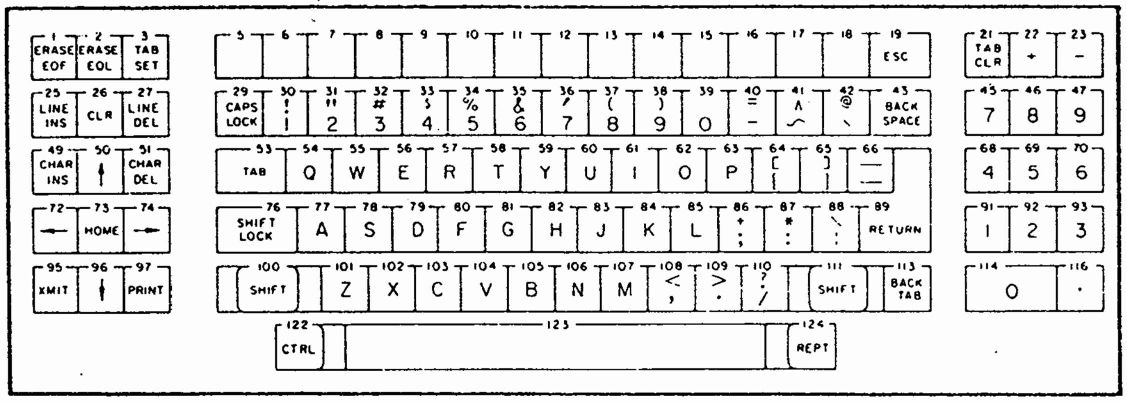
A.7.1. Interface

The Sun keyboard has been modified to produce up/down keycodes on 8 parallel output lines. Each keycode is held stable on these lines for a minimum of 2.5 ms in order that the main processor can read it during its refresh routine, which executes every 2 ms or so.

When no keys are depressed, the keyboard transmits a keycode of IDLEKEY (defined as 0x7F) to indicate that that is the case. Thus, when the last key is released, a keycode for its release is sent, followed by an IDLEKEY.


SET UP
ESC ! 1 @ 2 # £ 3 $ 4 % 5 ^ 6 & 7 * 8 (9) 0 — - + = ~ ` BACK SPACE BREAK
TAB Q W E R T Y U I O P { [}] DELETE
CTRL CAPS LOCK A S D F G H J K L : ; " ' RETURN | \
NO SCROLL SHIFT Z X C V B N M < , > . ? / SHIFT LINE FEED
PF1 PF2 PF3 PF4
7 8 9 -
4 5 6 ,
1 2 3 ENTER
0 .

VT100 Keyboard


ERASE EOF ERASE EOL TAB SET
LINE INS CLR LINE DEL
CHAR INS CHAR DEL
HOME
XMIT PRINT
ESC
CAPS LOCK ! 1 " 2 # 3 $ 4 % 5 & 6 ' 7 (8) 9 0 = - ^ ~ @ ` BACK SPACE
TAB Q W E R T Y U I O P [{] }
SHIFT LOCK A S D F G H J K L + ; * : | \ RETURN
SHIFT Z X C V B N M < , > . ? / SHIFT BACK TAB
CTRL REPT
TAB CLR + -
7 8 9
4 5 6
1 2 3
0 .

103SD30 Keyboard

Figure KEYS. Keyboard Layouts.

A.7.2. Shift Keys

The codes generated by the keyboard are modified by several shift and shift-like keys. These are:

left and right **SHIFT**

Cause the uppercase letter or symbol to be generated.

SHIFT LOCK (non-VT100 only)

Same as SHIFT keys. This key is unlocked by pressing either SHIFT key.

CAPS LOCK

Causes letter keys to generate uppercase letters; no other keys are affected.

CTRL
(on non-VT100, key labeled "REPT" is also a CTRL)

Causes control characters to be generated. See table at the end of this chapter. When this key is depressed, SHIFT, SHIFT LOCK, and CAPS LOCK are ignored.

BREAK (META; VT100 only)

This key serves as an additional kind of shift, called META. It sets the high-order bit (the 0x80 bit) of normal characters typed while it is held down. Like the shift keys, the act of pressing or releasing the BREAK key does not send any characters; it just affects what is sent by other keys.

SETUP

The SETUP key also acts like a META-style key, but it sets a different bit. It sets the 0x100 bit of the key code returned by getchar() or mayget(). Since most callers only use the low-order byte, this bit will typically be ignored. However, applications which look at the entire longword can notice this bit. NOTE that the sequence "SETUP goes down, A goes down" will cause a non-maskable abort back to monitor control; however, if any other keystrokes intervene (even keys going up), no abort occurs. See "Monitor Abort Sequence" below.

A.7.3. Monitor Abort Sequence

There is one special sequence which can be typed to generate an "Abort", which causes the currently executing program to be interrupted, allowing commands to be typed to the monitor. This is generated by holding down the upper-leftmost key (SETUP on the VT100 keyboard, ERASE-EOF on the non-VT100 keyboard) and then, without releasing it or pressing or releasing any other keys, pressing the "a" key. If the monitor's refresh routine is running, this will cause the message "Abort at xxxxxx" (where xxxxxx is the program counter value from the mainline program that was interrupted) and the monitor prompt character ">" to be output to the console. To resume, type "c".

Note: When the monitor is using serial line A or B for the console input, Abort is generated by a BREAK condition on the serial line.

A.7.4. Special Keys

A.7.4.1. No Scroll

The no-scroll key alternately sends ^S and ^Q each time it is pressed. If ^S or ^Q is typed manually, the monitor notices this; when NO-SCRL is next pressed, it sends the opposite character to the one typed manually. Thus, this key will alternately freeze and unfreeze the screen if the operating system or host system supports this form of flow control. This key is labeled "XMIT" on the non-VT100 keyboard.

A.7.4.2. CAPS LOCK

The effect of the CAPS LOCK key was already described under "Shift Keys" above for the case of ASCII-translated key codes. When raw up-down codes are being used, this key behaves somewhat differently on the two styles of keyboard. On the non-VT100 keyboard, this key alternately latches down and releases each time it is pressed; when getchar() and mayget() are returning raw up/down key codes, they simply return the normal up and down codes for this key.

However, on the VT100 keyboard, this key does not latch down. It is a press-on, press-off switch with an LED on the keycap that lights when it is "on". In addition to generating the normal up/down key codes, this key also generates separate key codes (with a different key number) for the LED: a key-down when it lights, and a key-up when it extinguishes. Thus, pressing or releasing this one key can cause two keycodes to be returned. Typically, one of the two key numbers related to this key is ignored by the receiving software; the ASCII translation in the monitor uses only the key number for the LED function (which corresponds to the latching function on the non-VT100 keyboard) and ignores the up-down encoded key number. For specific key encodings, consult Appendix D, "Sun Keyboard Translation Table Definitions".

A.7.4.3. String Keys

The monitor's keyboard tables permit arbitrary strings to be associated with particular keys, such that (under ASCII translation) pressing that key causes the specified string of characters to be returned. In the default mappings, this is used to translate certain keys into standard escape sequences.

On the VT100 keyboard, the cursor arrows and PF keys transmit their default VT100 escape sequences. There is no provision for "keypad application mode" or other VT100 modes. On the non-VT100 keyboard, the arrow and HOME keys generate ANSI standard control sequences (ESC [A, B, C, D, and H).

These sequences are summarized in the following table:

Strings Generated by Keys

↑	ESC [A	
↓	ESC [B	
→	ESC [C	
←	ESC [D	
PF1	ESC O P	*(VT100 only)*
PF2	ESC O Q	*(VT100 only)*
PF3	ESC O R	*(VT100 only)*
PF4	ESC O S	*(VT100 only)*
HOME	ESC [H	*(non-VT100 only)*

A.7.5. Generating Control Codes

The following table gives the keystrokes necessary to generate the ASCII control characters (0x0 through 0x1F). The notation CTRL-x means hold down the CTRL key (like a shift) and type "x". Where more than one keystroke entry is listed, any of the alternatives will produce the given control character. Note that the SHIFT, SHIFT-LOCK, and CAPS-LOCK keys have no effect on control characters. For example, on the VT100 keyboard, CTRL-a is equivalent to CTRL-A, CTRL-~ is equivalent to CTRL-`, and so on.

Sun Keyboard Control Characters

Hex value	*ASCII name*	*Keystrokes*	
		VT100	*non-VT100*
0	NUL	CTRL-space, CTRL-@	CTRL-space, CTRL-@
1	SOH	CTRL-A	CTRL-A
2	STX	CTRL-B	CTRL-B
3	ETX	CTRL-C	CTRL-C
4	EOT	CTRL-D	CTRL-D
5	ENQ	CTRL-E	CTRL-E
6	ACQ	CTRL-F	CTRL-F
7	BEL	CTRL-G	CTRL-G
8	BS	BACKSPACE, CTRL-H	BACKSPACE, CTRL-H
9	HT	TAB, CTRL-I	TAB, CTRL-I
A	LF	LINEFEED, CTRL-J	CTRL-J
B	VT	CTRL-K	CTRL-K
C	FF	CTRL-L	CLR, CTRL-L
D	CR	RETURN, CTRL-M	RETURN, CTRL-M
E	SO	CTRL-N	CTRL-N
F	SI	CTRL-O	CTRL-O
10	DLE	CTRL-P	CTRL-P
11	DC1	CTRL-Q	CTRL-Q
12	DC2	CTRL-R	CTRL-R
13	DC3	CTRL-S	CTRL-S
14	DC4	CTRL-T	CTRL-T
15	NAK	CTRL-U	CTRL-U
16	SYN	CTRL-V	CTRL-V
17	ETB	CTRL-W	CTRL-W
18	CAN	CTRL-X	CTRL-X
19	EM	CTRL-Y	CTRL-Y
1A	SUB	CTRL-Z	CTRL-Z
1B	ESC	ESC, CTRL-[	ESC, CTRL-[
1C	FS	CTRL-\	CTRL-\
1D	GS	CTRL-]	CTRL-]
1E	RS	CTRL-^, CTRL-~	CTRL-^
1F	US	CTRL-_, CTRL-?	CTRL-_
7F	DEL	DEL	BACKTAB

A.7.6. Mis-labeled and Undefined Keys

On the VT100 keyboard, the BREAK key does not generate an ANSI Break condition; instead, it serves as a META shift key. See "Shift Keys", above.

The following refer only to the non-VT100 keyboard:

REPT
: (Key #124) This key is a second CTRL key, not a repeat key.

BACK TAB
: (Key #113) This is a DEL key, generating 0x7F (Delete or Rubout).

ERASE EOF, ERASE EOL, TAB SET, TAB CLR, LINE INS, LINE DEL, CHAR INS, CHAR DEL
: (Keys #1, 2, 3, 21, 25, 27, 49, 51) Not implemented. These keys are ignored, except that ERASE EOF is used for the Abort sequence (ERASE EOF-a) and for escaping from Transparent Mode (ERASE EOF-e).

CLR
: (Key #26) Generates ^L (FF, Form Feed, 0xC).

unlabeled function keys
: (Keys #5-18) These keys are ignored.

A.8. Terminal Emulation

The monitor provides routines which enable the Sun Workstation to emulate a standard ANSI terminal and a Tektronix 4014 terminal. The workstation utilized in this fashion is referred to in this section as the "Sun terminal". The Sun terminal is used by the monitor as a "Console" output device.

The Sun terminal displays 34 lines of 80 ASCII characters per line, with scrolling, (x,y) cursor addressability, and a number of other ANSI standard* control functions. In addition it provides all of the standard and many of the enhanced vector capabilities of the 4014**.

A.8.1. ANSI Terminal Emulation

The Sun terminal displays a non-blinking block cursor which marks the current line and character position on the screen. ASCII characters between 0x20 and 0x7E inclusive are printing characters, which means that they have graphic renditions in the form of bit maps (see "Font Table", below). When a printing character is written to the Sun terminal and is not part of an escape sequence described below, it is displayed at the current cursor position and the cursor moves one position to the right on the current line. If the cursor is already at the right edge of the screen, it moves to the first character position on the next line. If the cursor is already at the right edge of the screen on the bottom line, the Line-feed function is performed (see CTRL-J below), which causes the screen to scroll up by one or more lines or wrap-around, before moving the cursor to the first character position on the next line.

*ANSI Standard X3.64, "Additional Controls for Use with ASCII", Secretariat: CBEMA, 1828 L St., N.W., Washington, D.C. 20036.

**4014 and 4014-1 Computer Display Terminal User's Manual, Tektronix, Inc., P.O. Box 500, Beaverton, Oregon 97077.

A.8.1.1. ANSI Control Sequence Syntax

The Sun terminal defines a number of control sequences which may occur in its input. When such a sequence is written to the Sun terminal, it will not be displayed on the screen, but will effect some control function as described below, for example, move the cursor or set a display mode.

Some of the control sequences consist of a single character. The alternate notations

CTRL-*X* or ^*X*

for some character *X*, represent a control character which is typed on the keyboard by holding down the CTRL key and typing *X*. On the Sun keyboard, the shift key has no effect on control characters; for example, CTRL-\ is equivalent to CTRL-|, also written as ^|. See section 4.5, "Generating Control Codes", for a complete list of control characters and their corresponding keystrokes.

Other ANSI control sequences are of the form

ESC [<params> <char>

Spaces are included only for readability; these characters must occur in the given sequence without the intervening spaces. ESC represents the ASCII Escape character (ESC, CTRL-[, 0x1B). The next character is a left square bracket "[" (0x5B). The <params> are a sequence of zero or more decimal numbers made up of digits between 0 and 9, separated by semicolons. <Char> represents a function character, which is different for each control sequence. Some examples of syntactically valid escape sequences are (again, ESC represent the single ASCII character "Escape"):

```
ESC[m
ESC[7m
ESC[33;54A
ESC[123;456;0;;3;B
```

Syntactically valid ANSI escape sequences which are not currently interpreted by the Sun terminal are ignored. Control characters which are not currently interpreted by the Sun terminal are also ignored.

Each control function requires a specified number of parameters, as noted below. If fewer parameters are supplied, then the remaining parameters are defaulted to 1, except as noted in the descriptions below. If more than the required number of parameters is supplied, then only the last n will be used, where n is the number required by that particular command character. Also, parameters which are omitted or set to zero are reset to the default value of 1 (except as noted below). Consider, for example, the command character M which requires one parameter. ESC[;M and ESC[0M and ESC[M and ESC[23;15;32;1M are all equivalent to ESC[1M and provide a parameter value of 1. Note that ESC[;5M (interpreted as "ESC[5M") is NOT equivalent to ESC[5;M (interpreted as "ESC[1M"). In the syntax descriptions below, parameters are represented as "#" or "#1;#2".

A.8.1.2. ANSI Control Functions

The following paragraphs specify the ANSI control functions implemented by the Sun terminal. Each description gives:

- the control sequence syntax

- the hex equivalent of control characters where applicable
- the ANSI standard control function name and two- or three-letter abbreviation where applicable
- description of parameters required, if any
- description of the control function
- for functions which set a mode, the initial setting of the mode. The initial settings are restored on k2 reset, or by calling finit( ) (see "PROM Vector Table").

CTRL-G (0x7)
Bell

The Sun Workstation is not equipped with an audible bell. It "rings the bell" by flashing the entire screen.

CTRL-H (0x8)
Backspace

The cursor moves one position to the left on the current line. If it is already at the left edge of the screen, nothing happens.

CTRL-I (0x9)
Tab

The cursor moves right on the current line to the next tab stop. The tab stops are fixed at every multiple of 8 columns. If the cursor is already at the right edge of the screen, nothing happens; otherwise the cursor moves right a minimum of one and a maximum of eight character positions.

CTRL-J (0xA)
Line-feed

The cursor moves down one line, remaining at the same character position on the line.

If the cursor is already at the bottom line, the screen either scrolls up or "wraps around" depending on the setting of an internal variable *S* (initially 1) which can be changed by the ESC[r control sequence. If *S* is greater than zero, the entire screen (including the cursor) is scrolled up by *S* lines before executing the Line-feed. The top *S* lines scroll off the screen and are lost. *S* new blank lines scroll onto the bottom of the screen. After scrolling, the line-feed is executed by moving the cursor down one line.

If *S* is zero, then "wrap-around" mode is entered. "ESC [1 r" will exit back to scroll mode. In wrap mode, if a linefeed occurs on the bottom line, the cursor goes to the same character position in the top line of the screen. When any linefeed occurs, the line that the cursor moves to is cleared. This means that the only scrolling that will occur is due to explicit escape sequences, not due to linefeeding off the bottom line.

The screen scrolls as fast as possible depending on how much data is backed up awaiting printing. Whenever a scroll must take place and the terminal is in normal scroll mode ("ESC [1 r"), it will scan the rest of the data awaiting printing to see how many linefeeds occur in it. This scan stops when any control character from the set {VT, FF, SO, SI, DLE, DC1, DC2, DC3, DC4, NAK, SYN, ETB, CAN, EM, SUB, ESC, FS, GS, RS, US} is found. At that point, the screen is scrolled by N lines (N at least 1) and processing continues. The scanned text is still processed normally to fill in the newly created lines. This behavior occurs with any program which outputs text thru the fwritestr( ) function, and results in much faster scrolling.

See also the discussion of the "Set scrolling" (ESC[r) control function below.

CTRL-L (0xC)
Form-feed

The cursor is postioned to the Home position (upper-left corner) and the entire screen is cleared.

CTRL-M (0xD)
Return

The cursor moves to the leftmost character position on the current line.

CTRL-[(0x1B)
ESC

This is the Escape character. It initiates a control sequence. The control sequence is ANSI-compatible if followed immediately by a left bracket "[". Otherwise it is a 4014 control sequence.

ESC[#@
Insert Character (ICH)

Takes one parameter, # (default 1). Inserts # blanks at the current cursor position. The tail of the current line starting at the current cursor position inclusive is shifted to the right by # character positions to make room for the blanks. The rightmost # character positions shift off the line and are lost. The position of the cursor is unchanged.

ESC[#A
Cursor Up (CUU)

Takes one parameter, # (default 1). Moves the cursor up # lines. If the cursor is fewer than # lines from the top of the screen, moves the cursor to the topmost line on the screen. The character position of the cursor on the line is unchanged.

ESC[#B
Cursor Down (CUD)

Takes one parameter, # (default 1). Moves the cursor down # lines. if the cursor is fewer than # lines from the bottom of the screen, moves the cursor to the last line on the screen. The character position of the cursor on the line is unchanged.

ESC[#C
Cursor Forward (CUF)

Takes one parameter, # (default 1). Moves the cursor to the right by # character positions on the current line. If the cursor is fewer than # positions from the right edge of the screen, moves the cursor to the rightmost position on the current line.

ESC[#D
Cursor Backward (CUB)

Takes one parameter, # (default 1). Moves the cursor to the left by # character positions on the current line. If the cursor is fewer than # positions from the left edge of the screen, moves the cursor to the leftmost position on the current line.

ESC[#E
Cursor Next Line (CNL)

Takes one parameter, # (default 1). Positions the cursor at the leftmost character position on the #-th line below the current line. If the current line is less than # lines from the bottom of the screen, postions the cursor at the leftmost character position on the bottom line.

ESC[#1;#2f
Horizontal And Vertical Position (HVP)
or
ESC[#1;#2H
Cursor Position (CUP)

These two escape sequences are equivalent. Takes two parameters, #1 and #2 (default 1, 1). Moves the cursor to the #2-th character position on the #1-th line. Character positions are numbered from 1 at the left edge of the screen; line positions are numbered from 1 at the top of the screen. Hence, if both parameters are omitted, the default action moves the cursor to the home position (upper left corner).

ESC[J
Erase in Display (ED)

Takes no parameters. Erases from the current cursor position inclusive to the end of the screen. In other words, erases from the current cursor position inclusive to the end of the current line and all lines below the current line. The cursor position is unchanged.

ESC[K
Erase in Line (EL)

Takes no parameters. Erases from the current cursor position inclusive to the end of the current line. The cursor position is unchanged.

ESC[#L
Insert Line (IL)

Takes one parameter, # (default 1). Makes room for # new lines starting at the current line by scrolling down by # lines the portion of the screen from the current line inclusive to the bottom. The # new lines at the cursor are filled with blanks; the bottom # lines shift off the bottom of the screen and are lost. The position of the cursor on the screen is unchanged.

ESC[#M
Delete Line (DL)

Takes one parameter, # (default 1). Deletes # lines beginning with the current line. The portion of the screen from the current line inclusive to the bottom is scrolled upward by # lines. The # new lines scrolling onto the bottom of the screen are filled with blanks; the # old lines beginning at the cursor line are deleted. The position of the cursor on the screen is unchanged.

ESC[#P
Delete Character (DCH)

Takes one parameter, # (default 1). Deletes # characters starting with the current cursor position. Shifts to the left by # character positions the tail of the current line from the current cursor position inclusive to the end of the line. Blanks are shifted into the rightmost # character positions. The position of the cursor on the screen is unchanged.

ESC[#m
Select Graphic Rendition (SGR)

Takes one parameter, # (default 0). Note that, unlike most escape sequences, the parameter defaults to zero if omitted. Invokes the graphic rendition specified by the parameter. All following printing characters in the data stream are rendered according to the parameter until the next occurrence of this escape sequence in the data stream. Currently only two graphic renditions are defined:

0	Normal rendition.
7	Negative (reverse) image.

Negative image will display characters as white-on-black if the screen mode is currently black-on white, and vice-versa. Any non-zero value of # is currently equivalent to 7 and will select the negative image rendition.

ESC[p
Black On White (SUNBOW)

Takes no parameters. Sets the screen mode to black-on-white. If the screen mode is already black-on-white, has no effect. In this mode blanks display as solid white, other characters as black-on-white. The cursor is a solid black block. Characters displayed in negative image rendition (see "Select Graphic Rendition" above) will be white-on-black in this mode. This is the initial setting of the screen mode on reset.

ESC[q
White On Black (SUNWOB)

Takes no parameters. Sets the screen mode to white-on-black. If the screen mode is already white-on-black, has no effect. In this mode blanks display as solid black, other characters as white-on-black. The cursor is a solid white block. Characters displayed in negative image rendition (see "Select Graphic Rendition" above) will be black-on-white in this mode. The initial setting of the screen mode on reset is the alternative mode, black on white.

ESC[#r
Set scrolling (SUNSCRL)

Takes one parameter, # (default 0). On power-up or k2 reset, this parameter is reset to 1. Sets to # an internal register which determines how many lines the screen will scroll up when a line-feed function is performed with the cursor on the bottom line. A parameter of 2 or 3 will introduce a small amount of "jump" when a scroll occurs. A parameter of 34 will clear the screen rather than scrolling.

A parameter of zero initiates "wrap mode" instead of scrolling. In wrap mode, if a linefeed occurs on the bottom line, the cursor goes to the same character position in the top line of the screen. When any linefeed occurs, the line that the cursor moves to is cleared. This means that the only scrolling that will occur is due to explicit escape sequences, not due to linefeeding off the bottom line. "ESC [1 r" will exit back to scroll mode.

For more information, see the description of the Line-feed (CTRL-J) control function above.

ESC[s
Reset terminal emulator (SUNRESET)

Takes no parameters. Resets all modes to default, restores current font from PROM. Screen and cursor position are unchanged.

A.8.2. Vector-Drawing Control Functions

The Sun monitor's terminal emulator includes vector-drawing capabilities which are a subset of those implemented by the Tektronix 4014 terminal with Enhanced Graphics Module option. The following modes are provided:

Alpha mode. ASCII characters are printed normally.

Graph mode. ASCII characters cause vectors to be drawn.

Point mode. ASCII characters cause points to be plotted.

Incremental mode. ASCII characters cause incremental "pen" motions.

A.8.2.1. Unimplemented Features

Due to hardware limitations, the Sun 4014 emulator currently does ***not*** emulate several 4014 features, including GIN mode (graphic input using a thumbwheel-controlled cursor), dotted and dashed vectors, 4K x 4K resolution, smaller character sizes, "Write-Thru", and grey scale. Escape codes for unimplemented features are ignored, with the following exceptions. Escape codes from **ESC `** through **ESC d** and **ESC h** through **ESC l** (0x1B followed by 0x60 through 0x64 or 0x68 through 0x6C) select normal (written) vectors in the Sun emulation; on the 4014, these codes select normal or defocused vectors with various combinations of dotted and dashed lines. Escape codes from **ESC p** through **ESC t** (0x1B followed by 0x70 through 0x74) select dark (unwritten) vectors in the Sun emulation; on the 4014, these codes select Write-Thru mode with various combinations of dotted and dashed lines.

A.8.2.2. Graph and Point Mode Address Format

In these modes a series of addresses are given. Each address causes the beam to move in a straight line from its current address to the given address. The first address of the series after entering graph mode is a dark vector (no vector is written), but all subsequent addresses cause vectors to be written. This is the only control there is over writing ("pen-up" versus "pen-down"). If the first address is preceded by ^G then the first vector is written rather than dark. In point mode, all vectors are dark, but the point at the given address is illuminated.

Addresses are transmitted either as full addresses or abbreviations thereof. A full address consists of two 10-bit coordinates Y,X transmitted in that order. Each coordinate is transmitted using two consecutive characters, with the high order 5 bits of the coordinate being extracted from the low order 5 bits of the first character and the the low order 5 bits of the coordinate being extracted from the low order 5 bits of the second character.

The high order 2 bits of each of the four characters transmitted for a full address must be respectively 01, 11, 01, 10. These denote respectively HI Y, LO Y, HI X, LO X. Whether 01 denotes HI Y or HI X is determined by whether LO X or LO Y respectively was transmitted most recently. On entering graph mode HI Y is assumed (as though LO X were transmitted most recently).

An address may be abbreviated merely by omitting characters, subject to the following two rules. LO X may never be omitted since it is the character that actually causes the vector to be drawn. HI X may occur only if immediately preceded by LO Y, since LO Y is used to enable the interpretation of 01 as HI X rather than as HI Y.

There is an extended address protocol to support two more low-order bits of precision. These bits are currently ignored by the Sun emulator since the Sun frame buffer does not have 4096x4096 resolution.

Three control characters are interpreted specially in these modes. CTRL-G (Bell) as the first character in graph mode causes subsequent vectors to be drawn (pen down); otherwise, the first vector would be dark. NUL (0) is ignored in all modes including graph and point modes. ESC in all modes may initiate an ANSI or 4014 control sequence. The 4014 control sequences are summarized in the section, "4014 Control Sequences," below.

A.8.2.3. Incremental plotting mode

This mode is entered with ^^ (control-caret, 0x1E). Each subsequent character causes action according to the following table:

space	raises pen
P	lowers pen
A	move right
E	move right and up
D	move up
F	move left and up
B	move left
J	move left and down
H	move down
I	move right and down
NUL (^@)	ignored

All other characters cause exit from incremental plotting mode and are interpreted as Alpha mode characters.

Note that each move occurs in increments of 1 in the 4014 "extended" 12-bit coordinate system. Since the Sun only uses 10-bit coordinates, no motion will be visible until at least four increments have occurred in the X or Y direction. In other words, the increments occur within 4x4 squares which are mapped to single pixels on the Sun frame buffer.

A.8.2.4. 4014 Control Sequences

The following table summarizes the control sequences which are relevant to vector drawing:

Control Sequence	*Hex*	*Function*
^]	1D	Enter graph mode, first vector dark, following vectors written.
^] ^G	1D 07	Enter graph mode, all following vectors written.
^\	1C	Enter point mode
^^	1E	Enter incremental plotting mode
^_ ^M (RETURN)	1F 0D	Enter alpha mode
ESC p	1B 61	Subsequent vectors "dark" (graph mode). (ESC followed by p, q, r, s, or t is equivalent.)
ESC a	1B 61	Subsequent vectors written (graph mode). (ESC followed by `, a, b, c, d, h, i, j, or k is equivalent.)
ESC ^L	1B 0C	Clears the screen.
ESC ?	1B 3F	Equivalent to DEL (0x7F).

NAME

intro – introduction to system maintenance and operation commands

DESCRIPTION

This section contains information related to system bootstrapping, operation and maintenance and describes all the server processes and daemons which run on the system.

Disk formatting and labelling is done by *diag*(8S) which participates in most system bootstraps. Bootstrapping of the system is described in *reboot*(8). The standard set of commands run by the system when it boots is described in *rc*(8). Related commands include ones to check the consistency of file systems *fsck*(8), mount and unmount file systems *mount*(8) and *umount*(8), add swap devices *swapon*(8), cause all outstanding disk I/O to complete *sync*(8), shutdown or reboot a running system *shutdown*(8), *halt*(8), and *reboot*(8), set the time on a machine from the time on another machine *rdate*(8).

Creation of file systems is discussed in *mkfs*(8) and *newfs*(8). File system performance parameters are adjustable with *tunefs*(8). File system saves and restores are described in *dump*(8) and *restore*(8).

Procedures for adding new users to a system are described in *adduser*(8) using *vipw*(8).

Other programs useful when the system crashes or hardware is broken include *gxtest*(8S) which tests the frame buffer on a workstation, *imemtest*(8S) which tests the memory, *crash*(8S) which describes what happens when the system crashes, *savecore*(8) and *analyze*(8) which can be used to analyze system crash dumps. Occasionally useful as adjuncts to the *fsck*(8) file system repair program are *clri*(8), *dcheck*(8), *icheck*(8), and *ncheck*(8).

Configuring a new version of the UNIX kernel requires using the program *config*(8); major system bootstraps often require the use of *mkproto*(8). New devices are made in the /dev directory when device drivers are added tp the system by using the *makedev*(8) and *mknod*(8) commands. If you have source, you will use the *install*(8) command to reinstall freshly compiled programs, and *catman*(8) to reformat the pre-formatted version of the manual.

Resource accounting is enabled by the *accton*(8) command, and summarized by *sa*(8). Login time accounting is performed by *ac*(8).

A number of service and daemon processes are described here. The *cron*(8) daemon forces delayed disk I/O to occur and runs periodic events (such as removing temporary files from the disk periodically). The *dmesg*(8) process is invoked by *cron* and keeps the system error log. The *init*(8) process is the initial process created when UNIX boots and manages the reboot process and creates the initial login prompts on the various system terminals through the services of *getty*(8). The Internet super-server *inetd*(8C) invokes all other internet servers as needed. These servers include the remote shell servers *rshd*(8C) and *rexecd*(8C) the remote login server *rlogind*(8C) the FTP and TELNET daemons *ftpd*(8C) and *telnetd*(8C), the TFTP daemon *tftpd*(8C) and the mail arrival notification daemon *comsat*(8C). Other network daemons include the 'load average/who is logged in' daemon *rwhod*(8C), the routing daemon *routed*(8C), and the mail daemon *sendmail*(8).

If network protocols are being debugged, then the protocol debugging trace program *trpt*(8C) is often useful. Remote magnetic tape access is provided by *rsh* and *rmt*(8C). Remote line printer access is provided by *lpd*(8) and control over the various print queues is had through *lpc*(8). Printer cost accounting is done through *pac*(8).

Network host tables may be gotten from the ARPA NIC using *gettable*(8C) and converted to UNIX usable format using *htable*(8).

LIST OF PROGRAMS

Program	*Appears on Page*	*Description*
MAKEDEV	makedev.8	make system special files
ac	ac.8	login accounting
accton	sa.8	system accounting
adduser	adduser.8	procedure for adding new users
analyze	analyze.8	Virtual UNIX postmortem crash analyzer
catman	catman.8	create the cat files for the manual
chown	chown.8	change owner
clri	clri.8	clear i-node
comsat	comsat.8c	biff server
config	config.8	build system configuration files
crash	crash.8s	what happens when the system crashes
cron	cron.8	clock daemon
dcheck	dcheck.8	file system directory consistency check
diag	diag.8s	General-purpose stand-alone utility package
dmesg	dmesg.8	collect system diagnostic messages to form error log
dump	dump.8	incremental file system dump
dumpfs	dumpfs.8	dump file system information
fastboot	fastboot.8	reboot/halt the system without checking the disks
fasthalt	fastboot.8	reboot/halt the system without checking the disks
fsck	fsck.8	file system consistency check and interactive repair
ftpd	ftpd.8c	DARPA Internet File Transfer Protocol server
gettable	gettable.8c	get NIC format host tables from a host
getty	getty.8	set terminal mode
gxtest	gxtest.8s	stand alone test for the Sun video graphics board
halt	halt.8	stop the processor
htable	htable.8	convert NIC standard format host tables
icheck	icheck.8	file system storage consistency check
ifconfig	ifconfig.8c	configure network interface parameters
imemtest	imemtest.8s	stand alone memory test
inetd	inetd.8c	internet services daemon
init	init.8	process control initialization
install	install.8	install binaries
iostat	iostat.8	report I/O statistics
kgmon	kgmon.8	generate a dump of the operating system's profile buffers
lpc	lpc.8	line printer control program
lpd	lpd.8	line printer daemon
makekey	makekey.8	generate encryption key
mkfs	mkfs.8	construct a file system
mknod	mknod.8	build special file
mkproto	mkproto.8	construct a prototype file system
mount	mount.8	mount and dismount file system
ncheck	ncheck.8	generate names from i-numbers
netstat	netstat.8	show network status
newaliases	newaliases.8	rebuild the data base for the mail aliases file
newfs	newfs.8	construct a new file system
pac	pac.8	printer/plotter accounting information
pstat	pstat.8	print system facts
rc	rc.8	command script for auto-reboot and daemons
rdate	rdate.8	set system date from a remote host
reboot	reboot.8	UNIX bootstrapping procedures
renice	renice.8	alter priority of running processes

restore	restore.8	incremental file system restore
rexecd	rexecd.8c	remote execution server
rlogind	rlogind.8c	remote login server
rmail	rmail.8	handle remote mail received via uucp
rmt	rmt.8c	remote magtape protocol module
route	route.8c	manually manipulate the routing tables
routed	routed.8c	network routing daemon
rshd	rshd.8c	remote shell server
rwhod	rwhod.8c	system status server
sa	sa.8	system accounting
savecore	savecore.8	save a core dump of the operating system
sendmail	sendmail.8	send mail over the internet
shutdown	shutdown.8	close down the system at a given time
sticky	sticky.8	executable files with persistent text
swapon	swapon.8	specify additional device for paging and swapping
sync	sync.8	update the super block
syslog	syslog.8	log systems messages
telnetd	telnetd.8c	DARPA TELNET protocol server
tftpd	tftpd.8c	DARPA Trivial File Transfer Protocol server
trpt	trpt.8c	transliterate protocol trace
tunefs	tunefs.8	tune up an existing file system
umount	mount.8	mount and dismount file system
update	update.8	periodically update the super block
uuclean	uuclean.8c	uucp spool directory clean-up
vipw	vipw.8	edit the password file
vmstat	vmstat.8	report virtual memory statistics

NAME

ac – login accounting

SYNOPSIS

/usr/etc/ac [**–w** wtmp] [**–p**] [**–d**] [people] ...

DESCRIPTION

Ac produces a printout giving connect time for each user who has logged in during the life of the current *wtmp* file. A total is also produced.

The accounting file */usr/adm/wtmp* is maintained by *init* and *login*. Neither of these programs creates the file, so if it does not exist no connect-time accounting is done. To start accounting, it should be created with length 0. On the other hand if the file is left undisturbed it will grow without bound, so periodically any information desired should be collected and the file truncated.

OPTIONS

–w specifies an alternate *wtmp* file.

–p prints individual totals; without this option, only totals are printed.

–d printout for each midnight to midnight period. Any *people* will limit the printout to only the specified login names. If no *wtmp* file is given, */usr/adm/wtmp* is used.

FILES

/usr/adm/wtmp

SEE ALSO

init(8), sa(8), login(1), utmp(5).

NAME

adduser – procedure for adding new users

DESCRIPTION

A new user must choose a login name, which must not already appear in */etc/passwd.* An account can be added by editing a line into the passwd file; this must be done with the password file locked, for example, by using *vipw*(8).

A new user is given a group and user id. User id's should be distinct across a system, since they are used to control access to files. Typically, users working on similar projects will be put in the same group. System staff is group '10' for historical reasons, and the super-user is in this group.

A skeletal account for a new user 'esmerelda' would look like:

```
esmerelda:*:235:20:& Featherstonehaugh:/usr/esmerelda:/bin/csh
```

Fields in the password file have the following meanings:

1. Login name 'esmerelda'.
2. Encrypted password. This is normally initialized to an asterisk, and the set using *passwd*(1).
3. User ID.
4. Group ID.
5. This field is called the 'GCOS' field (from earlier implementation of UNIX) and is traditionally used to hold the user's full name. Some installations have other information encoded in this field. From this information we can tell that esmerelda's real name is 'Esmerelda Featherstonehaugh'. The & here is a shorthand for the user's login name.
6. User's home directory.
7. Initial shell which this user will see on login. If this field is empty, *sh*(1) is used as the initial shell.

It is useful to give new users some help in getting started, supplying them with a few skeletal files such as *.profile* if they use '/bin/sh', or *.cshrc* and *.login* if they use '/bin/csh'. New users should be given copies of these files which, for instance, arrange to use *tset*(1) automatically at each login.

FILES

/etc/passwd	password file

SEE ALSO

passwd(1), chsh(1), passwd(5), vipw(8)

NAME

analyze – Virtual UNIX postmortem crash analyzer

SYNOPSIS

/usr/etc/analyze [**–s** swapfile] [**–f**] [**–m**] [**–d**] [**–D**] [**–v**] corefile [system]

DESCRIPTION

Analyze is the post-mortem analyzer for the state of the paging system. In order to use *analyze* you must arrange to get a image of the memory (and possibly the paging area) of the system after it crashes (see *crash*(8S)).

The *analyze* program reads the relevant system data structures from the core image file and indexing information from **/vmunix** (or the specified file) to determine the state of the paging subsystem at the point of crash. It looks at each process in the system, and the resources each is using in an attempt to determine inconsistencies in the paging system state. Normally, the output consists of a sequence of lines showing each active process, its state (whether swapped in or not), its *p0br*, and the number and location of its page table pages. Any pages which are locked while raw i/o is in progress, or which are locked because they are *intransit* are also printed. (Intransit text pages often diagnose as duplicated; you will have to weed these out by hand.)

The program checks that any pages in core which are marked as not modified are, in fact, identical to the swap space copies. It also checks for non-overlap of the swap space, and that the core map entries correspond to the page tables. The state of the free list is also checked.

Options to *analyze*:

–D	causes the diskmap for each process to be printed.
–d	causes the (sorted) paging area usage to be printed.
–f	which causes the free list to be dumped.
–m	causes the entire coremap state to be dumped.
–v	(long unused) which causes a hugely verbose output format to be used.

In general, the output from this program can be confused by processes which were forking, swapping, or exiting or happened to be in unusual states when the crash occurred. You should examine the flags fields of relevant processes in the output of a *pstat*(8) to weed out such processes.

It is possible to look at the core dump with *adb* if you do

```
adb -k /vmunix /vmcore
```

FILES

/vmunix	default system namelist

SEE ALSO

adb(1S), ps(1), crash(8S), pstat(8)

AUTHORS

Ozalp Babaoglu and William Joy

DIAGNOSTICS

Various diagnostics about overlaps in swap mappings, missing swap mappings, page table entries inconsistent with the core map, incore pages which are marked clean but differ from disk-image copies, pages which are locked or intransit, and inconsistencies in the free list.

It would be nice if this program analyzed the system in general, rather than just the paging system in particular.

NAME

catman – create the cat files for the manual

SYNOPSIS

/usr/etc/catman [**−p**] [**−n**] [**−w**] [sections]

DESCRIPTION

Catman creates the preformatted versions of the on-line manual from the nroff input files. Each manual page is examined and those whose preformatted versions are missing or out of date are recreated. If any changes are made, *catman* recreates the **/usr/lib/whatis** database.

If there is one parameter not starting with a '–', it is take to be a list of manual sections to look in. For example

catman 123

only updates manual sections 1, 2, and 3.

OPTIONS

−n	Do not create **/usr/lib/whatis**.
−p	Print what would be done instead of doing it.
−w	Only create the **/usr/lib/whatis** database. No manual reformatting is done.

FILES

/usr/man/man?/*.*	raw (nroff input) manual sections
/usr/man/cat?/*.*	preformatted manual pages
/usr/lib/makewhatis	commands to make whatis database

SEE ALSO

man(1)

NAME

chown – change owner

SYNOPSIS

/etc/chown –f owner file **...**

DESCRIPTION

Chown changes the owner of the *files* to *owner*. The owner may be either a decimal UID or a login name found in the password file.

Only the super-user can change owner, in order to simplify as yet unimplemented accounting procedures.

OPTIONS

–f Do not report errors.

FILES

/etc/passwd

SEE ALSO

chgrp(1), chown(2), passwd(5), group(5)

NAME

clri – clear i-node

SYNOPSIS

/etc/clri filesystem i-number ...

DESCRIPTION

N.B.: *Clri* is obsoleted for normal file system repair work by *fsck*(8).

Clri writes zeros on the i-nodes with the decimal *i-numbers* on the *filesystem*. After *clri,* any blocks in the affected file will show up as 'missing' in an *icheck*(8) of the *filesystem*.

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

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

SEE ALSO

icheck(8)

BUGS

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

NAME

comsat – biff server

SYNOPSIS

/usr/etc/in.comsat

DESCRIPTION

Comsat is the server process which listens for reports of incoming mail and notifies users who have requested to be told when mail arrives. It is invoked as needed by *inetd*(8C), and times out if inactive for a few minutes.

Comsat listens on a datagram port associated with the "biff" service specification (see *services*(5)) for one line messages of the form

user@mailbox-offset

If the *user* specified is logged in to the system and the associated terminal has the owner execute bit turned on (by a "biff y"), the *offset* is used as a seek offset into the appropriate mailbox file and the first 7 lines or 560 characters of the message are printed on the user's terminal. Lines which appear to be part of the message header other than the "From", "To", "Date", or "Subject" lines are not printed ignored when displaying the message.

FILES

/etc/utmp	to find out who's logged on and on what terminals

SEE ALSO

biff(1)

BUGS

The message header filtering is prone to error.

Users should be notified of mail which arrives on other machines than the one they are currently logged in to.

The notification should appear in a separate window so it does not mess up the screen.

NAME

config – build system configuration files

SYNOPSIS

/usr/etc/config [**–p**] *config_file*

DESCRIPTION

Config builds a set of system configuration files from a short file which describes the sort of system that is being configured. It also takes as input a file which tells *config* what files are needed to generate a system. This can be augmented by a configuration-specific set of files that give alternate files for a specific machine. (see the FILES section below) Specifying the **–p** option to *config* configures a system for profiling (see *kgmon*(8) and *gprof*(1)).

Run *config* from the *conf* subdirectory of the system source (in a Sun environment, from */sys/conf*). *Config* assumes that there is already a directory *../config_file* created and it places all its output files in there. The output of *config* consists of several files: *ioconf.c*, which contains a description of I/O devices attached to the system; a *makefile*, which is a file used by *make*(1) in building the system; a set of header files which contain the number of various devices that will be compiled into the system; and a set of swap configuration files which contain definitions for the disk areas to be used for swapping, the root file system, argument processing, and system dumps.

After running *config*, you must then change directory to the directory in which the new makefile was created, and use *make* to create the dependency tree for the new system: **# cd ../***System_Name* **# make depend** *Config* reminds you of this when it completes.

If you get any other error messages from *config,* you should fix the problems in your configuration file and try again. If you try to compile a system that had configuration errors, you will probably not succeed.

CONFIG FILE FORMAT

In the following descriptions, a number can be a decimal integer, a whole octal number or a whole hexadecimal number. Hex and octal are specified to *config* in the same way they are specified to the C compiler, a number starting with "0x" is a hex number and a number starting with just a "0" is an octal number. When specifying the timezone, you may also use floating point numbers.

Comments are specified in a config file with the character "#". All characters from a "#" to the end of a line are ignored.

Lines beginning with tabs are considered continuations of the previous line.

Lines of the configuration file can be one of two basic types. First, there are lines which describe general things about your system:

machine *type*
: This is system is to run on the machine type specified. Only one machine type can appear in the config file. The legal *type* for a Sun system is **sun**.

cpu *"type"*
: This system is to run on the cpu type specified. For a Sun system, *"type"* is **"SUN2"** (note that the double quotes are part of the designation).

ident *name*
: Gives the system identifier — a name for the machine or machines that run this kernel.

timezone *number* [**dst**]
: Specifies the timezone you are in. This is measured in the number of hours west of GMT you are. 5 is EST, 8 is PST. Negative numbers indicate hours east of GMT. If you specify **dst,** the system will operate under daylight savings time.

maxusers *number*
: The maximum expected number of simultaneously active user on this system is *number*. This number is used to size several system data structures.

options *optlist*
Compile the listed options into the system. Options in this list are separated by commas. There is a list of options that you may specify in the generic makefile. A line of the form "options FUNNY,HAHA" yields -DFUNNY -DHAHA to the C compiler. An option may be given a value, by following its name with "=" then the value enclosed in (double) quotes. None of the standard options use such a value.

config *sysname config_clauses...*
Generate a system with name *sysname and* configuration as specified in *config-clauses*. The *sysname* is used to name the resultant binary image and per-system swap configuration files. The *config_clauses* indicate the location for the root file system, one or more disk partitions for swapping and paging, a disk partition to which system dumps should be made, and a disk partition on which *execve* argument lists should be constructed. All but the root device specification may be omitted, *config* will fill them in with default values as described below.

root A root device specification is of the form **root** *on xy0d*. If a specific partition is omitted — for example, if only **root on xy0** is specified — the "a" partition is assumed. When a generic system is being built, no root specification should be given; the root device will be defined at boot time by prompting the console.

swap Swap device specifications have two possible forms. If a generic swap configuration is required, the clause **swap generic** should be specifed. Otherwise, if a single partition is to be used for swapping, one may specify **swap** *on xy0b*. If multiple partitions are to be interleaved one should specify something of the form **swap** *on xy0* **and** *xy1* **and** *xy1g*. If no swap specification is given, *config* assumes swapping should be done on the "b" partition of the root device. Swapping areas may be almost any size and multiple swap partitions of varying size may be interleaved. Partitions used for swapping are sized at boot time by the system; to override dynamic sizing of a swap area the number of sectors in the swap area can be specified in the config file. For example, **swap** *on xy0b size 99999* would configure a swap partition with 99999 sectors.

args The partition to be used for argument list processing may be specified with a clause of the form **args** *on xy1b*. If no argument device is specified, the first swap partition specified is used. If a device is specified without a particular partition, the "b" partition is assumed. If a generic system is being built, no argument device should be specified; the argument device will be assigned to the swap device dynamically configured at boot time.

dumps The location to which system dumps are sent may be specified with a clause of the form **dumps** *on xy1*. If no dump device is specified, the first swap partition specified is used. If a device is specified without a particular partition, the "b" partition is assumed. If a generic configuration is to be built, no dump device should be specified; the dump device will be assigned to the swap device dynamically configured at boot time.

Dumps are placed at the end of the partition specified. Their size and location is recorded in global kernel variables *dumpsize* and *dumplo*, respectively, for use by *savecore*(8).

Device names specified in configuration clauses are mapped to block device major numbers with a file */sys/conf/devices.machine,* where *machine* is the machine type previously specified in the configuration file. If a device name to block device major number mapping must be overridden, a device specification may be given in the form **major** *x* **minor** *y*.

The second group of lines in the configuration file describe which devices your system has and what they are connected to (for example, I have a Xylogics xy450 on the Multibus). These lines have the following format:

dev_type *dev_name* **at** *con_dev* *more_info*

Dev_type is either **tape, disk, controller, device,** or **pseudo-device.** These types have the following meanings:

controller
is a Multibus disk controller or a Multibus tape controller.

disk or **tape**
are devices connected to a controller.

device
is something that plugs into the Multibus, like a cartridge tape interface.

pseudo-device
is something which should be conditionally loaded, but is not really a device. Current examples are the pseudo-tty driver and various network subsystems. (For pseudo-devices, **more_info** may be specified as an integer, that gives the value of the symbol defined in the header file created for that device, and is generally used to indicate the number of instances of the pseudo-device to create. If you load a subsystem you will probably find it convenient to enable conditional code using an **options** specification.

dev_name is the name of the device you are specifying. If it is not a pseudo-device, you must give a number afterwards (for example, **xyc0** for a Xylogics controller, or **ar0** for an Archive quarter-inch tape controller).

Con_dev is what the device you are specifying is connected to. For example, disk **xy1** is connected to controller **xyc0**.

more_info is a sequence of the following:

csr *addr*
Specifies the csr (command and status registers) for a device. Must be specified for all Multibus tape and disk controllers and all devices connected to the Multibus. Make certain that you put a leading zero on the address so that it will be interpreted as an octal number.

drive *number*
For a disk or tape, specifies which drive this is.

flags *number*
These flags are passed to the device driver at system initialization time.

priority *level*
For devices which interrupt on the Multibus, specifies the interrupt level at which the device operates.

The easiest way to understand config files it to look at a working one and modify it to suit your system. A good sample is provided in the *Installing UNIX for the First Time* chapter of the *System Installation and Maintenance Guide* at the beginning of this manual. In the example, all lines are explained.

A **?** may be substituted for a number in two places and the system will figure out what to fill in for the **?** when it boots. You can put question marks on a *con_dev* (e.g. at mba?), or on a drive number (e.g. drive ?) This allows redundancy, as a single system can be built which will reboot on different hardware configurations.

FILES

/sys/conf/GENERIC	generic config file for Sun systems
/sys/conf/README	file describing how to make a new kernel
/sys/conf/makefile.sun	generic makefile for Sun systems
/sys/conf/files	list of common files system is built from

/sys/conf/files.sun	list of Sun-specific files
/sys/conf/devices.sun	name to major device mapping file for Sun systems

SEE ALSO

The SYNOPSIS portion of each device entry in the section 4 pages of the Sun *System Interface Manual.*

BUGS

The line numbers reported in error messages are usually off by one.

NAME

crash – what happens when the system crashes

DESCRIPTION

This section explains what happens when the system crashes and how you can analyze crash dumps.

When the system crashes voluntarily it prints a message of the form

```
panic: why i gave up the ghost
```

on the console, takes a dump on a mass storage peripheral, and then invokes an automatic reboot procedure as described in *reboot*(8). Unless some unexpected inconsistency is encountered in the state of the file systems due to hardware or software failure the system will then resume multi-user operations.

The system has a large number of internal consistency checks; if one of these fails, then it will panic with a very short message indicating which one failed.

The most common cause of system failures is hardware failure, which can reflect itself in different ways. Here are the messages which you are likely to encounter, with some hints as to causes. Left unstated in all cases is the possibility that hardware or software error produced the message in some unexpected way.

IO err in push
hard IO err in swap

The system encountered an error trying to write to the paging device or an error in reading critical information from a disk drive. You should fix your disk if it is broken or unreliable.

timeout table overflow

This really shouldn't be a panic, but until we fix up the data structure involved, running out of entries causes a crash. If this happens, you should make the timeout table bigger.

trap type %d, code=%d, pc=%x

A unexpected trap has occurred within the system; the trap types are:

0	reserved addressing fault
1	privileged instruction fault
2	reserved operand fault
3	bpt instruction fault
4	xfc instruction fault
5	system call trap
6	arithmetic trap
7	ast delivery trap
8	segmentation fault
9	protection fault
10	trace trap
11	compatibility mode fault
12	page fault
13	page table fault

The favorite trap types in system crashes are trap types 8 and 9, indicating a wild reference. The code is the referenced address, and the pc at the time of the fault is printed. These problems tend to be easy to track down if they are kernel bugs since the processor stops cold, but random flakiness seems to cause this sometimes.

init died

The system initialization process has exited. This is bad news, as no new users will then be able to log in. Rebooting is the only fix, so the system just does it right away.

That completes the list of panic types you are likely to see.

When the system crashes it writes (or at least attempts to write) an image of memory into the back end of the primary swap area. After the system is rebooted, the program *savecore*(8) runs and preserves a copy of this core image and the current system in a specified directory for later perusal. See *savecore*(8) for details.

To analyze a dump you should begin by running *adb*(1S) with the **–k** flag on the core dump. Normally the command "*(intstack-4)$c" will provide a stack trace from the point of the crash and this will provide a clue as to what went wrong. A more complete discussion of system debugging is impossible here. See, however, "Using ADB to Debug the UNIX Kernel".

SEE ALSO

adb(1S), analyze(8), reboot(8)
Using ADB to Debug the UNIX Kernel

NAME

cron – clock daemon

SYNOPSIS

/etc/cron

DESCRIPTION

Cron executes commands at specified dates and times according to the instructions in the file */usr/lib/crontab*. Since *cron* never exits, it should only be executed once. This is best done by running *cron* from the initialization process through the file */etc/rc*; see *init*(8).

/usr/lib/crontab consists of lines of six fields each. The fields are separated by spaces or tabs. The first five are integer patterns to specify the minute (0-59), hour (0-23), day of the month (1-31), month of the year (1-12), and day of the week (1-7 with 1=Monday). Each of these patterns may contain a number in the range above; two numbers separated by a minus meaning a range inclusive; a list of numbers separated by commas meaning any of the numbers; or an asterisk meaning all legal values. The sixth field is a string that is executed by the Shell at the specified times. A percent character in this field is translated to a new-line character. Only the first line (up to a % or end of line) of the command field is executed by the Shell. The other lines are made available to the command as standard input.

Cron examines */usr/lib/crontab* under the following conditions:

- At least once per hour (on the hour).
- WHen the next command is to be run — *cron* looks ahead untii the next command and sleeps until then.
- When *cron*'s process is sent a SIGHUP. This means that comeone who changes */usr/lib/crontab* can get *cron* to look at it right away.

FILES

/usr/lib/crontab

SEE ALSO

crontab(5)

NAME

dcheck – file system directory consistency check

SYNOPSIS

/usr/etc/dcheck [–i *numbers*] [filesystem]

DESCRIPTION

N.B.: *Dcheck* is obsoleted for normal consistency checking by *fsck*(8).

Dcheck reads the directories in a file system and compares the link-count in each i-node with the number of directory entries by which it is referenced. If the file system is not specified, *dcheck* checks a set of default file systems.

Dcheck is fastest if the raw version of the special file is used, since the i-list is read in large chunks.

OPTIONS

–i *numbers*

Numbers is a list of i-numbers; when one of those i-numbers turns up in a directory, the number, the i-number of the directory, and the name of the entry are reported.

FILES

Default file systems vary with installation.

SEE ALSO

fsck(8), icheck(8), fs(5), clri(8), ncheck(8)

DIAGNOSTICS

When a file turns up for which the link-count and the number of directory entries disagree, the relevant facts are reported. Allocated files which have 0 link-count and no entries are also listed. The only dangerous situation occurs when there are more entries than links; if entries are removed, so the link-count drops to 0, the remaining entries point to thin air. They should be removed. When there are more links than entries, or there is an allocated file with neither links nor entries, some disk space may be lost but the situation will not degenerate.

BUGS

Since *dcheck* is inherently two-pass in nature, extraneous diagnostics may be produced if applied to active file systems.

Dcheck is obsoleted by *fsck* and remains for historical reasons.

NAME

diag – General-purpose stand-alone utility package

SYNOPSIS

b stand/diag

DESCRIPTION

Diag is a general-purpose stand-alone utility package containing a grab-bag of the kinds of tools needed for disk initialization, testing, and file transfer. *Diag* supports the various SMD and ST-506 disk controllers.

Note: that *diag* can only be called from the Sun Workstation PROM monitor — *diag* is not a system utility.

The most common use of *diag* is formatting and labelling a disk — see the *format, label,* and *partition* commands.

Diag is interactive — it prompts for options and arguments. There are two phases to using *diag*: in the first phase, *diag* prompts for information about the type of disk drive and controller that it is working with, and essentially 'configures' itself to work with that disk and controller; in the second phase, *diag* waits for the user to type one of the commands from the list below. A specific *diag* command is called up by typing enough characters to uniquely identify the command. The commands that *diag* currently recognizes are listed here:

clear	Sends a restore command to a disk. This is needed to manually reset disk errors.
diag	Re-initializes the *diag* program itself — goes back to phase one of the inititialization process described above.
errors	Toggles an option to report all errors as they occur.
fix	Formats a single track of a disk. If the track is already formatted, you are asked for confirmation in order to avoid destroying the data on the track.
format	Formats a disk.
help or **?**	Displays a list of the available commands.
info	Toggles an option to report all disk activity as it completes.
label	Labels the disk.
map	Explicitly maps one track to a different track. Usually required for bad track mapping. The *format* command usually does this automatically.
partition	Creates, assigns, or modifies logical partition tables for a disk. The UNIX operating system requires logical partitions. The *label* command writes the partition map to the disk. There are standard partition tables for each type of disk that *diag* knows about.
position	Tests the disk by reading sectors from random positions on the disk. This command runs until the user aborts it by typing the letter 'a'.
quit	Quits from *diag* and returns to whoever called it up.
read	Reads specified blocks from the disk. The *read* command prompts for the starting block number, number of blocks, and the block increment. The *read* command doesn't report the data it reads — it is intended for verifying that blocks are readable.
seek	Performs a seek test on the disk such that a seek is made to every cylinder and a seek is made to every possible cylinder distance.
status	Reports the ready status of each drive on the current controller.
test	Does a general disk test by writing random data to random positions on the disk and then verifying that the correct data can be read back. The *test* command destroys data on the disk. It runs until the user aborts the process by typing the letter 'a'.

time Toggles an option to turn timing on and off. When timing is on, *diag* reports on how long things take — *diag* is less verbose in this state so it doesn't waste time displaying messages.

verify Reads and displays the label from the disk. Shows the logical partition assignments. This is usually done automatically when the *format* command has labelled the disk.

write Writes garbage data to specified blocks on the disk. The *write* command prompts for the starting block number, number of blocks, and the block increment. The *write* is intended for verifying that blocks are writeable.

+ Adds two numbers and reports the result in decimal, hexadecimal, and as a disk address.

– Subtracts two numbers and reports the result in decimal, hexadecimal, and as a disk address.

Block numbers may be entered either as an absolute decimal block number, or as a disk address of the form cylinder/head/sector.

Any *diag* command may be aborted by typing a ^C (control-C).

NAME

dmesg – collect system diagnostic messages to form error log

SYNOPSIS

/usr/etc/dmesg [–]

DESCRIPTION

Dmesg looks in a system buffer for recently printed diagnostic messages and prints them on the standard output. The messages are those printed by the system when device (hardware) errors occur and (occasionally) when system tables overflow non-fatally. If the – flag is given, then *dmesg* computes (incrementally) the new messages since the last time it was run and places these on the standard output. This is typically used with *cron*(8) to produce the error log */usr/adm/messages* by running the command

```
/etc/dmesg – >> /usr/adm/messages
```

every 10 minutes.

FILES

/usr/adm/messages	error log (conventional location)
/usr/adm/msgbuf	scratch file for memory of – option

BUGS

The system error message buffer is of small finite size. As *dmesg* is run only every few minutes, not all error messages are guaranteed to be logged. This can be construed as a blessing rather than a curse.

Error diagnostics generated immediately before a system crash will never get logged.

NAME

dump – incremental file system dump

SYNOPSIS

/etc/dump [key [*argument* ...] filesystem]

DESCRIPTION

Dump copies to magnetic tape all files changed after a certain date in the *filesystem.* The *key* specifies the date and other options about the dump. *Key* consists of characters from the set **0123456789fusdWnc.**

0–9 This number is the 'dump level'. All files modified since the last date stored in the file */etc/dumpdates* for the same filesystem at lesser levels will be dumped. If no date is determined by the level, the beginning of time is assumed; thus the entire filesystem is dumped if the **0** option is used.

f Place the dump on the next *argument* file instead of the tape. If file is of the form machine:device, dump to a remote machine.

u If the dump completes successfully, write the date of the beginning of the dump on file */etc/dumpdates.* This file records a separate date for each filesystem and each dump level. The format of */etc/dumpdates* is readable by people, consisting of one free format record per line: filesystem name, increment level and *ctime(3)* format dump date. */etc/dumpdates* may be edited to change any of the fields, if necessary.

s The size of the dump tape is specified in feet. The number of feet is taken from the next *argument.* When the specified size is reached, *dump* waits for reels to be changed. The default tape size is 2300 feet.

d The density of the tape, expressed in BPI, is taken from the next *argument.* This is used in calculating the amount of tape used per reel. The default is 1600.

W *Dump* tells the operator what file systems need to be dumped. This information is gleaned from the files */etc/dumpdates* and */etc/fstab.* The **W** option causes *dump* to print out, for each file system in */etc/dumpdates* the most recent dump date and level, and highlights those file systems that should be dumped. If the **W** option is set, all other options are ignored, and *dump* exits immediately.

w Is like W, but prints only those filesystems which need to be dumped.

n Whenever *dump* requires operator attention, notify by means similar to a *wall*(1) all of the operators in the group "operator".

c Dump to cartridge tape instead of standard half-inch magnetic tape.

b Specifies the blocking factor for the dump. The blocking factor is taken from the next argument on the command line. The default blocking factor is 10.

If no arguments are given, the *key* is assumed to be **9u** and a default file system is dumped to the default tape.

Dump requires operator intervention on these conditions: end of tape, end of dump, tape write error, tape open error or disk read error (if there are more than a threshold of 32). In addition to alerting all operators implied by the **n** key, *dump* interacts with the operator on *dump's* control terminal at times when *dump* can no longer proceed, or if something is grossly wrong. All questions *dump* poses **must** be answered by typing "yes" or "no", appropriately.

Since making a dump involves a lot of time and effort for full dumps, *dump* checkpoints itself at the start of each tape volume. If writing that volume fails for some reason, *dump* will, with operator permission, restart itself from the checkpoint after the old tape has been rewound and removed, and a new tape has been mounted.

Dump tells the operator what is going on at periodic intervals, including usually low estimates of the number of blocks to write, the number of tapes it will take, the time to completion, and the time to the tape change. The output is verbose, so that others know that the terminal controlling *dump* is busy, and will be for some time.

Now a short suggestion on how to perform dumps. Start with a full level 0 dump

```
dump 0un
```

Next, dumps of active file systems are taken on a daily basis, using a modified Tower of Hanoi algorithm, with this sequence of dump levels:

3 2 5 4 7 6 9 8 9 9 ...

For the daily dumps, a set of 10 tapes per dumped file system is used on a cyclical basis. Each week, a level 1 dump is taken, and the daily Hanoi sequence repeats with 3. For weekly dumps, a set of 5 tapes per dumped file system is used, also on a cyclical basis. Each month, a level 0 dump is taken on a set of fresh tapes that is saved forever.

FILES

/dev/rrp1g	default filesystem to dump from
/dev/rmt8	default tape unit to dump to
/etc/dumpdates	new format dump date record
/etc/fstab	dump table: file systems and frequency
/etc/group	to find group *operator*

SEE ALSO

restore(8), dump(5), fstab(5)

DIAGNOSTICS

Many, and verbose.

BUGS

Sizes are based on 1600 BPI blocked tape; the raw magtape device has to be used to approach these densities. Fewer than 32 read errors on the filesystem are ignored. Each reel requires a new process, so parent processes for reels already written just hang around until the entire tape is written.

It would be nice if *dump* knew about the dump sequence, kept track of the tapes scribbled on, told the operator which tape to mount when, and provided more assistance for the operator running *restor*.

NAME
dumpfs – dump file system information

SYNOPSIS
/usr/etc/dumpfs *device*

DESCRIPTION
Dumpfs prints out the super block and cylinder group information for the file system or special device specified. The listing is very long and detailed. This command is useful mostly for finding out certain file system information such as the file system block size and minimum free space percentage.

SEE ALSO
fs(5), tunefs(8), newfs(8), fsck(8)

NAME

fastboot, fasthalt – reboot/halt the system without checking the disks

SYNOPSIS

/etc/fastboot [*boot-options*]
/etc/fasthalt [*halt-options*]

DESCRIPTION

Fastboot and *fasthalt* are shell scripts which reboot and halt the system without checking the file systems. This is done by creating a file */fastboot*, then invoking the *reboot* program. The system startup script, */etc/rc*, looks for this file and, if present, skips the normal invocation of *fsck*(8).

SEE ALSO

halt(8), reboot(8), rc(8)

NAME

fsck – file system consistency check and interactive repair

SYNOPSIS

/etc/fsck –p [filesystem ...]
/etc/fsck [**–b** block#] [**–y**] [**–n**] [filesystem] ...

DESCRIPTION

The first form of *fsck* preens a standard set of filesystems or the specified file systems. It is normally used in the script **/etc/rc** during automatic reboot. In this case *fsck* reads the table **/etc/fstab** to determine which file systems to check. It uses the information there to inspect groups of disks in parallel taking maximum advantage of i/o overlap to check the file systems as quickly as possible. Normally, the root file system will be checked on pass 1, other "root" ("a" partition) file systems on pass 2, other small file systems on separate passes (e.g. the "d" file systems on pass 3 and the "e" file systems on pass 4), and finally the large user file systems on the last pass, e.g. pass 5. A pass number of 0 in fstab causes a disk to not be checked; similarly partitions which are not shown as to be mounted "rw" or "ro" are not checked.

The system takes care that only a restricted class of innocuous inconsistencies can happen unless hardware or software failures intervene. These are limited to the following:

Unreferenced inodes

Link counts in inodes too large

Missing blocks in the free list

Blocks in the free list also in files

Counts in the super-block wrong

These are the only inconsistencies which *fsck* with the **–p** option will correct; if it encounters other inconsistencies, it exits with an abnormal return status and an automatic reboot will then fail. For each corrected inconsistency one or more lines will be printed identifying the file system on which the correction will take place, and the nature of the correction. After successfully correcting a file system, *fsck* will print the number of files on that file system and the number of used and free blocks.

Without the **–p** option, *fsck* audits and interactively repairs inconsistent conditions for file systems. If the file system is inconsistent the operator is prompted for concurrence before each correction is attempted. It should be noted that a number of the corrective actions which are not fixable under the **–p** option will result in some loss of data. The amount and severity of data lost may be determined from the diagnostic output. The default action for each consistency correction is to wait for the operator to respond **yes** or **no**. If the operator does not have write permission *fsck* will default to a **–n** action.

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

The following flags are interpreted by *fsck.*

–b Use the block specified immediately after the flag as the super block for the file system. Block 32 is always an alternate super block.

–y Assume a yes response to all questions asked by *fsck;* this should be used with great caution as this is a free license to continue after essentially unlimited trouble has been encountered.

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

If no filesystems are given to *fsck* then a default list of file systems is read from the file **/etc/fstab**.

Inconsistencies checked are as follows:

1. Blocks claimed by more than one inode or the free list.
2. Blocks claimed by an inode or the free list outside the range of the file system.
3. Incorrect link counts.
4. Size checks:
   Directory size not of proper format.
5. Bad inode format.
6. Blocks not accounted for anywhere.
7. Directory checks:
   File pointing to unallocated inode.
   Inode number out of range.
8. Super Block checks:

   More blocks for inodes than there are in the file system.
9. Bad free block list format.
10. Total free block and/or free inode count incorrect.

Orphaned files and directories (allocated but unreferenced) are, with the operator's concurrence, reconnected by placing them in the **lost+found** directory. The name assigned is the inode number. The only restriction is that the directory **lost+found** must preexist in the root of the filesystem being checked and must have empty slots in which entries can be made. This is accomplished by making **lost+found**, copying a number of files to the directory, and then removing them (before *fsck* is executed).

Checking the raw device is almost always faster.

FILES

/etc/fstab	contains default list of file systems to check.

DIAGNOSTICS

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

SEE ALSO

fstab(5), fs(5), newfs(8), mkfs(8), crash(8S), reboot(8)

BUGS

Inode numbers for **.** and **..** in each directory should be checked for validity.

There should be some way to start a **fsck -p** at pass *n*.

NAME

ftpd – DARPA Internet File Transfer Protocol server

SYNOPSIS

/usr/etc/in.ftpd host.socket

DESCRIPTION

Ftpd is the DARPA Internet File Transfer Prototocol server process. The server is invoked by the Internet daemon *inetd*(8C) each time a connection to the ftp service (see *services*(5)) is made, with the connection available as descriptor 0 and the host and socket the connection originated from (in hex and decimal respectively) as argument.

Inactive connections are timed out after 60 seconds.

The ftp server currently supports the following ftp requests; case is not distinguished.

Request	**Description**
ACCT	specify account (ignored)
ALLO	allocate storage (vacuously)
APPE	append to a file
CWD	change working directory
DELE	delete a file
HELP	give help information
LIST	give list files in a directory ("ls -lg")
MODE	specify data transfer *mode*
NLST	give name list of files in directory ("ls")
NOOP	do nothing
PASS	specify password
PORT	specify data connection port
QUIT	terminate session
RETR	retrieve a file
RNFR	specify rename-from file name
RNTO	specify rename-to file name
STOR	store a file
STRU	specify data transfer *structure*
TYPE	specify data transfer *type*
USER	specify user name
XCUP	change to parent of current working directory
XCWD	change working directory
XMKD	make a directory
XPWD	print the current working directory
XRMD	remove a directory

The remaining ftp requests specified in Internet RFC 765 are recognized, but not implemented.

Ftpd interprets file names according to the "globbing" conventions used by *csh*(1). This allows users to utilize the metacharacters "*?[]{}~".

Ftpd authenticates users according to three rules.

1) The user name must be in the password data base, */etc/passwd*, and not have a null password. In this case a password must be provided by the client before any file operations may be performed.

2) The user name must not appear in the file */etc/ftpusers*.

3) If the user name is "anonymous" or "ftp", an anonymous ftp account must be present in the password file (user "ftp"). In this case the user is allowed to log in by specifying any password (by convention this is given as the client host's name).

In the last case, *ftpd* takes special measures to restrict the client's access privileges. The server performs a *chroot*(2) command to the home directory of the "ftp" user. In order that system security is not breached, it is recommended that the "ftp" subtree be constructed with care; the following rules are recommended.

~ftp) Make the home directory owned by "ftp" and unwritable by anyone.

~ftp/bin)
Make this directory owned by the super-user and unwritable by anyone. The program *ls*(1) must be present to support the list commands. This program should have mode 111.

~ftp/etc)
Make this directory owned by the super-user and unwritable by anyone. The files *passwd*(5) and *group*(5) must be present for the *ls* command to work properly. These files should be mode 444.

~ftp/pub)
Make this directory mode 777 and owned by "ftp". Users should then place files which are to be accessible via the anonymous account in this directory.

SEE ALSO

ftp(1C),

BUGS

There is no support for aborting commands.

The anonymous account is inherently dangerous and should avoided when possible.

The server must run as the super-user to create sockets with privileged port numbers. It maintains an effective user id of the logged in user, reverting to the super-user only when binding addresses to sockets. The possible security holes have been extensively scrutinized, but are possibly incomplete.

NAME

gettable – get NIC format host tables from a host

SYNOPSIS

/etc/gettable *host*

DESCRIPTION

Gettable is a simple program used to obtain the NIC standard host tables from a "nicname" server. The indicated *host* is queried for the tables. The tables, if retrieved, are placed in the file *hosts.txt*.

Gettable operates by opening a TCP connection to the port indicated in the service specification for "nicname". A request is then made for "ALL" names and the resultant information is placed in the output file.

Gettable is best used in conjunction with the *htable*(8) program which converts the NIC standard file format to that used by the network library lookup routines.

SEE ALSO

intro(3N), htable(8)

BUGS

Should allow requests for only part of the database.

NAME

getty – set terminal mode

SYNOPSIS

/etc/getty [char]

DESCRIPTION

Getty is invoked by *init*(8) immediately after a terminal is opened, following the making of a connection. While reading the name *getty* attempts to adapt the system to the speed and type of terminal being used.

Init calls *getty* with an argument specified by the *ttys* file entry for the terminal line. Arguments other than '0' can be used to make *getty* treat the line specially. Refer to *ttys*(5) and *gettytab*(5) for descriptions of how *getty* uses the data in *gettytab* to set up the terminal. Normally, it sets the speed of the interface to 300 baud, specifies that raw mode is to be used (break on every character), that echo is to be suppressed, and either parity allowed. It types a banner identifying the system (from gethostname(2)) and the 'login:' message. Then the user's name is read, a character at a time. If a null character is received, it is assumed to be the result of the user pushing the 'break' ('interrupt') key. The speed is then changed to 1200 baud and the 'login:' is typed again; a second 'break' changes the speed to 150 baud and the 'login:' is typed again. Successive 'break' characters cycle through the speeds 300, 1200, and 150 baud.

The user's name is terminated by a new-line or carriage-return character. The latter results in the system being set to treat carriage returns appropriately (see *tty*(4)).

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

Finally, *login* is called with the user's name as argument.

SEE ALSO

init(8), login(1), ioctl(2), tty(4), ttys(5), gettytab(5)

NAME

gxtest – stand alone test for the Sun video graphics board

SYNOPSIS

b /stand/gxtest

DESCRIPTON

Gxtest runs stand alone, not under control of the UNIX operating system. With the PROM resident monitor in control of the system, type the command:

```
> b /stand/gxtest
```

and the monitor boots the video test program into memory. *Gxtest* is completely self-explanatory and runs under its own steam. It reports any errors it finds on the screen.

NAME

halt – stop the processor

SYNOPSIS

/etc/halt [**–n**] [**–q**] [**–y**]

DESCRIPTION

Halt writes out sandbagged information to the disks and then stops the processor.

OPTIONS

–n	Prevents the *sync* before stopping.
–q	Do a quick halt, no graceful shutdown is attempted.
–y	Needed if you are trying to halt the system from a dialup.

SEE ALSO

reboot(8), shutdown(8)

NAME

htable – convert NIC standard format host tables

SYNOPSIS

/usr/etc/htable *file*

DESCRIPTION

Htable is used to convert host files in the format specified in Internet RFC 810 to the format used by the network library routines. Three files are created as a result of running *htable*: *hosts, networks,* and *gateways*. The *hosts* file is used by the *gethostent*(3N) routines in mapping host names to addresses. The *networks* file is used by the *getnetent*(3N) routines in mapping network names to numbers. The *gateways* file is used by the routing daemon in identifying "passive" Internet gateways; see *routed*(8C) for an explanation.

If any of the files *localhosts, localnetworks,* or *localgateways* are present in the current directory, the file's contents is prepended to the output file without interpretation. This allows sites to maintain local aliases and entries which are not normally present in the master database.

Htable is best used in conjunction with the *gettable*(8C) program which retrieves the NIC database from a host.

SEE ALSO

intro(3N), gettable(8C)

BUGS

Does not properly calculate the *gateways* file.

NAME

icheck – file system storage consistency check

SYNOPSIS

/usr/etc/icheck [**–s**] [**–b** numbers] [filesystem]

DESCRIPTION

N.B.: *Icheck* is obsoleted for normal consistency checking by *fsck*(8).

Icheck examines a file system, builds a bit map of used blocks, and compares this bit map against the free list maintained on the file system. If the file system is not specified, a set of default file systems is checked. The normal output of *icheck* includes a report of

> The total number of files and the numbers of regular, directory, block special and character special files.
>
> The total number of blocks in use and the numbers of single-, double-, and triple-indirect blocks and directory blocks.
>
> The number of free blocks.
>
> The number of blocks missing; i.e. not in any file nor in the free list.

The **–s** option causes *icheck* to ignore the actual free list and reconstruct a new one by rewriting the super-block of the file system. The file system should be dismounted while this is done; if this is not possible (for example if the root file system has to be salvaged) care should be taken that the system is quiescent and that it is rebooted immediately afterwards so that the old, bad in-core copy of the super-block will not continue to be used. Notice also that the words in the super-block which indicate the size of the free list and of the i-list are believed. If the super-block has been curdled these words will have to be patched. The **–s** option causes the normal output reports to be suppressed.

Following the **–b** option is a list of block numbers; whenever any of the named blocks turns up in a file, a diagnostic is produced.

Icheck is faster if the raw version of the special file is used, since it reads the i-list many blocks at a time.

FILES

Default file systems vary with installation.

SEE ALSO

fsck(8), dcheck(8), ncheck(8), fs(5), clri(8)

DIAGNOSTICS

For duplicate blocks and bad blocks (which lie outside the file system) *icheck* announces the difficulty, the i-number, and the kind of block involved. If a read error is encountered, the block number of the bad block is printed and *icheck* considers it to contain 0. 'Bad freeblock' means that a block number outside the available space was encountered in the free list. '*n* dups in free' means that *n* blocks were found in the free list which duplicate blocks either in some file or in the earlier part of the free list.

BUGS

Since *icheck* is inherently two-pass in nature, extraneous diagnostics may be produced if applied to active file systems.

It believes even preposterous super-blocks and consequently can get core images.

The system should be fixed so that the reboot after fixing the root file system is not necessary.

NAME

ifconfig – configure network interface parameters

SYOPNSIS

/etc/ifconfig interface [*address*] [*parameters*]

DESCRIPTION

Ifconfig is used to assign an address to a network interface and/or to configure network interface parameters. *Ifconfig* must be used at boot time to define the network address of each interface present on a machine; it may also be used at a later time to redefine an interface's address. The *interface* parameter is a string of the form "name unit", for example "en0", while the address is either a host name present in the host name data base, *hosts*(5), or a DARPA Internet address expressed in the Internet standard "dot notation".

The following parameters may be set with *ifconfig*:

up	Mark an interface "up".
down	Mark an interface "down". When an interface is marked "down", the system will not attempt to transmit messages through that interface.
trailers	Enable the use of a "trailer" link level encapsulation when sending (default). If a network interface supports *trailers*, the system will, when possible, encapsulate outgoing messages in a manner which minimizes the number of memory to memory copy operations performed by the receiver.
–trailers	Disable the use of a "trailer" link level encapsulation.
arp	Enable the use of the Address Resolution Protocol in mapping between network level addresses and link level addresses (default). This is currently implemented for mapping between DARPA Internet addreses and 10Mb/s Ethernet addresses.
–arp	Disable the use of the Address Resolution Protocol.

Ifconfig displays the current configuration for a network interface when no optional parameters are supplied.

Only the super-user may modify the configuration of a network interface.

DIAGNOSTICS

Messages indicating the specified interface does not exit, the requested address is unknown, the user is not privileged and tried to alter an interface's configuration.

SEE ALSO

rc(8), intro(4N), netstat(1)

NAME

imemtest – stand alone memory test

SYNOPSIS

b /stand/imemtest

DESCRIPTON

Imemtest runs stand alone, not under control of the UNIX operating system. With the PROM resident monitor in control of the system, type the command:

```
> b /stand/imemtest
```

and the monitor boots the memory test program into memory. *Imemtest* is completely self-explanatory. It prompts for all start and end addresses, and after that it runs under its own steam. It reports any errors it finds on the screen.

NAME

inetd – internet services daemon

SYNOPSIS

/etc/inetd [**–d**]

DESCRIPTION

Inetd is the internet super-server which invokes all internet server processes as needed. Connection-oriented services are invoked each time a connection is made, by creating a process. This process is passed the connection as file descriptor 0 and an argument of the form "sourcehost.sourceport" where sourcehost is hex and sourceport is decimal.

Datagram oriented services are invoked when a datagram arrives; a process is created and passed the connection as file descriptor 0. Inetd will look at the socket where datagrams arrive again only after this process completes. The paradigms for such proceses are either to read off the incoming datagram and then fork and exit, or to process the arriving datagram and then time out.

Services supported by *inetd* are ftp *ftpd*(8C), *telnetd*(8C), *rshd*(8C), *rlogind*(8C) and *rexecd*(8C) which are conection based and *tftpd*(8C) and *comsat*(8C) which are datagram based.

OPTIONS

–d Specifies that debugging traces are to be turned on for connection-oriented (TCP) services.

SEE ALSO

comsat(8C), ftpd(8C), rexecd(8C), rlogind(8C), rshd(8C), telnetd(8C), tftpd(8C)

BUGS

The services supported by the program are hard-wired.

There is no provision for selectively invoking TCP debugging packet tracing per-service.

NAME

init – process control initialization

SYNOPSIS

/etc/init

DESCRIPTION

Init is invoked inside UNIX as the last step in the boot procedure. It normally then runs the automatic reboot sequence as described in *reboot*(8), and if this succeeds, begins multi-user operation. If the reboot fails, it commences single user operation by giving the super-user a shell on the console. It is possible to pass parameters from the boot program to *init* so that single user operation is commenced immediately. When such single user operation is terminated by killing the single-user shell (i.e. by hitting ^D), *init* runs */etc/rc* without the reboot parameter. This command file performs housekeeping operations such as removing temporary files, mounting file systems, and starting daemons.

In multi-user operation, *init's* role is to create a process for each terminal port on which a user may log in. To begin such operations, it reads the file */etc/ttys* and forks several times to create a process for each terminal specified in the file. Each of these processes opens the appropriate terminal for reading and writing. These channels thus receive file descriptors 0, 1 and 2, the standard input and output and the diagnostic output. Opening the terminal will usually involve a delay, since the *open* is not completed until someone is dialed up and carrier established on the channel. If a terminal exists but an error occurs when trying to open the terminal *init* complains by writing a message to the system console; the message is repeated every 10 minutes for each such terminal until the terminal is shut off in /etc/ttys and init notified (by a hangup, as described below), or the terminal becomes accessible (init checks again every minute). After an open succeeds, */etc/getty* is called with argument as specified by the second character of the *ttys* file line. *Getty* reads the user's name and invokes *login* to log in the user and execute the Shell.

Ultimately the Shell will terminate because of an end-of-file either typed explicitly or generated as a result of hanging up. The main path of *init*, which has been waiting for such an event, wakes up and removes the appropriate entry from the file *utmp*, which records current users, and makes an entry in */usr/adm/wtmp*, which maintains a history of logins and logouts. The *wtmp* entry is made only if a user logged in successfully on the line. Then the appropriate terminal is reopened and *getty* is reinvoked.

Init catches the *hangup* signal (signal SIGHUP) and interprets it to mean that the file */etc/ttys* should be read again. The Shell process on each line which used to be active in *ttys* but is no longer there is terminated; a new process is created for each added line; lines unchanged in the file are undisturbed. Thus it is possible to drop or add phone lines without rebooting the system by changing the *ttys* file and sending a *hangup* signal to the *init* process: use 'kill -HUP 1.'

Init will terminate multi-user operations and resume single-user mode if sent a terminate (TERM) signal, i.e. "kill -TERM 1". If there are processes outstanding which are deadlocked (due to hardware or software failure), *init* will not wait for them all to die (which might take forever), but will time out after 30 seconds and print a warning message.

Init will cease creating new *getty*'s and allow the system to slowly die away, if it is sent a terminal stop (TSTP) signal, i.e. "kill -TSTP 1". A later hangup will resume full multi-user operations, or a terminate will initiate a single user shell. This hook is used by *reboot*(8) and *halt*(8).

Init's role is so critical that if it dies, the system will reboot itself automatically. If, at bootstrap time, the *init* process cannot be located, the system will loop in user mode at location 0x13.

DIAGNOSTICS

init: ***tty*****: cannot open**. A terminal which is turned on in the *rc* file cannot be opened, likely because the requisite lines are either not configured into the system or the associated device was not attached during boot-time system configuration.

WARNING: Something is hung (wont die); ps axl advised. A process is hung and could not be killed when the system was shutting down. This is usually caused by a process which is stuck in a device driver due to a persistent device error condition.

FILES

/dev/console, /dev/tty*, /etc/utmp, /usr/adm/wtmp, /etc/ttys, /etc/rc

SEE ALSO

login(1), kill(1), sh(1), ttys(5), getty(8), rc(8), reboot(8), halt(8), shutdown(8)

NAME

install – install binaries

SYNOPSIS

install [**–c**] [**–m***mode*] [**–o***owner*] [**–s**] binary destination

DESCRIPTION

Binary is moved (or copied if **–c** is specified) to *destination*. If *destination* already exists, it is removed before *binary* is moved. If the destination is a directory then *binary* is moved into the *destination* directory with its original file-name.

OPTIONS

–m *mode*
: Set the permission mode for *destination* to *mode*. The mode is set to 755 in the absence of the **–m** option.

–o *owner*
: Change owner of *destination* to *owner*. *Destination* is changed to owner root in the absence of the **–o** option.

–s Strip the binary is stripped after it is installed.

Install refuses to move a file onto itself.

SEE ALSO

chmod(1), cp(1), mv(1), strip(1), chown(8)

NAME

iostat – report I/O statistics

SYNOPSIS

iostat [interval [count]]

DESCRIPTION

Iostat iteratively reports the number of characters read and written to terminals, and, for each disk, the number of seeks and transfers per second, and the milliseconds per average seek. It also gives the percentage of time the system has spent in user mode, in user mode running low priority (niced) processes, in system mode, and idling.

To compute this information, for each disk, seeks and data transfer completions and number of words transferred are counted; for terminals collectively, the number of input and output characters are counted. Also, each sixtieth of a second, the state of each disk is examined and a tally is made if the disk is active. From these numbers and given the transfer rates of the devices it is possible to determine average seek times for each device.

The optional *interval* argument causes *iostat* to report once each *interval* seconds. The first report is for all time since a reboot and each subsequent report is for the last interval only.

The optional *count* argument restricts the number of reports.

FILES

/dev/kmem
/vmunix

SEE ALSO

vmstat(8)

NAME

kgmon – generate a dump of the operating system's profile buffers

SYNOPSIS

/usr/etc/kgmon [**–b**] [**–h**] [**–r**] [**–p**] [system] [memory]

DESCRIPTION

Kgmon is a tool used when profiling the operating system. When no arguments are supplied, *kgmon* indicates the state of operating system profiling as running, off, or not configured (see *config*(8)). If the **–p** flag is specified, *kgmon* extracts profile data from the operating system and produces a *gmon.out* file suitable for later analysis by *gprof*(1).

OPTIONS

–b Resume the collection of profile data.

–h Stop the collection of profile data.

–p Dump the contents of the profile buffers into a *gmon.out* file.

–r Reset all the profile buffers. If the **–p** flag is also specified, the *gmon.out* file is generated before the buffers are reset.

If neither **–b** nor **–h** is specified, the state of profiling collection remains unchanged. For example, if the **–p** flag is specified and profile data is being collected, profiling is momentarily suspended, the operating system profile buffers are dumped, and profiling is immediately resumed.

FILES

/vmunix – the default system
/dev/kmem – the default memory

SEE ALSO

gprof (1), config(8)

DIAGNOSTICS

Users with only read permission on /dev/kmem cannot change the state of profiling collection. They can get a *gmon.out* file with the warning that the data may be inconsistent if profiling is in progress.

NAME

lpc – line printer control program

SYNOPSIS

/usr/etc/lpc [command [argument ...]]

DESCRIPTION

Lpc is used by the system administrator to control the operation of the line printer system. For each line printer configured in /etc/printcap, *lpc* may be used to:

- disable or enable a printer,
- disable or enable a printer's spooling queue,
- rearrange the order of jobs in a spooling queue,
- find the status of printers, and their associated spooling queues and printer dameons.

Without any arguments, *lpc* will prompt for commands from the standard input. If arguments are supplied, *lpc* interprets the first argument as a command and the remaining arguments as parameters to the command. The standard input may be redirected so that *lpc* reads commands from a file. Commands may be abreviated; the following is the list of recognized commands.

? [command ...]

help [command ...]
: Print a short description of each command specified in the argument list, or, if no arguments are given, a list of the recognized commands.

abort { all | printer ... }
: Terminate an active spooling daemon on the local host immediately and then disable printing (preventing new daemons from being started by *lpr*) for the specified printers.

clean { all | printer ... }
: Remove all files beginning with "cf", "tf", or "df" from the specified printer queue(s) on the local machine.

enable { all | printer ... }
: Enable spooling on the local queue for the listed printers. This will allow *lpr* to put new jobs in the spool queue.

exit

quit
: Exit from lpc.

disable { all | printer ... }
: Turn the specified printer queues off. This prevents new printer jobs from being entered into the queue by *lpr*.

restart { all | printer ... }
: Attempt to start a new printer daemon. This is useful when some abnormal condition causes the daemon to die unexpectedly leaving jobs in the queue. *Lpq* will report that there is no daemon present when this condition occurs.

start { all | printer ... }
: Enable printing and start a spooling daemon for the listed printers.

status [all] [printer ...]
: Display the status of daemons and queues on the local machine.

stop { all | printer ... }
: Stop a spooling daemon after the current job completes and disable printing.

FILES

/etc/printcap	printer description file
/usr/spool/*	spool directories
/usr/spool/*/lock	lock file for queue control

SEE ALSO

lpd(8), lpr(1), lpq(1), lprm(1), printcap(5)

DIAGNOSTICS

?Ambiguous command	abreviation matches more than one command
?Invalid command	no match was found
?Privileged command	command can be executed by root only

NAME

lpd – line printer daemon

SYNOPSIS

/usr/lib/lpd [-l] [-L logfile] [port #]

DESCRIPTION

Lpd is the line printer daemon (spool area handler) and is normally invoked at boot time from the *rc*(8) file. It makes a single pass through the *printcap*(5) file to find out about the existing printers and prints any files left after a crash. It then uses the system calls *listen*(2) and *accept*(2) to receive requests to print files in the queue, transfer files to the spooling area, display the queue, or remove jobs from the queue. In each case, it forks a child to handle the request so the parent can continue to listen for more requests. The Internet port number used to rendezvous with other processes is normally obtained with *getservent*(3N) but can be changed with the *port#* argument. The –L option changes the file used for writing error conditions from the system console to *logfile*. The –l flag causes *lpd* to log valid requests received from the network. This can be useful for debugging purposes.

Access control is provided by two means. First, all requests must come from one of the machines listed in the file */etc/hosts.equiv*. Second, if the "rs" capability is specified in the *printcap* entry for the printer being accessed, *lpr* requests will only be honored for those users with accounts on the machine with the printer.

The file *lock* in each spool directory is used to prevent multiple daemons from becoming active simultaneously, and to store information about the daemon process for *lpr*(1), *lpq*(1), and *lprm*(1). After the daemon has successfully set the lock, it scans the directory for files beginning with *cf*. Lines in each *cf* file specify files to be printed or non-printing actions to be performed. Each such line begins with a key character to specify what to do with the remainder of the line.

J	Job Name. String to be used for the job name on the burst page.
C	Classification. String to be used for the classification line on the burst page.
L	Literal. The line contains identification info from the password file and causes the banner page to be printed.
T	Title. String to be used as the title for *pr*(1).
H	Host Name. Name of the machine where *lpr* was invoked.
P	Person. Login name of the person who invoked *lpr*. This is used to verify ownership by *lprm*.
M	Send mail to the specified user when the current print job completes.
f	Formatted File. Name of a file to print which is already formatted.
l	Like "f" but passes control characters and does not make page breaks.
p	Name of a file to print using *pr*(1) as a filter.
t	Troff File. The file contains *troff*(1) output (C/A/T phototypesetter commands).
d	DVI File. The file contains $\TeX$ output (DVI format from Stanford).
g	Graph File. The file contains data produced by *plot*(3X).
c	Cifplot File. The file contains data produced by *cifplot*.
v	The file contains a raster image.
r	The file contains text data with FORTRAN carriage control characters.
1	Troff Font R. Name of the font file to use instead of the default.
2	Troff Font I. Name of the font file to use instead of the default.
3	Troff Font B. Name of the font file to use instead of the default.

4	Troff Font S. Name of the font file to use instead of the default.
W	Width. Changes the page width (in characters) used by *pr*(1) and the text filters.
I	Indent. The number of characters to indent the output by (in ascii).
U	Unlink. Name of file to remove upon completion of printing.
N	File name. The name of the file which is being printed, or a blank for the standard input (when *lpr* is invoked in a pipeline).

If a file can not be opened, a message will be placed in the log file (normally the console). *Lpd* will try up to 20 times to reopen a file it expects to be there, after which it will skip the file to be printed.

Lpd uses *flock*(2) to provide exclusive access to the lock file and to prevent multiple deamons from becoming active simultaneously. If the daemon should be killed or die unexpectedly, the lock file need not be removed. The lock file is kept in a readable ASCII form and contains two lines. The first is the process id of the daemon and the second is the control file name of the current job being printed. The second line is updated to reflect the current status of *lpd* for the programs *lpq*(1) and *lprm*(1).

FILES

/etc/printcap	printer description file
/usr/spool/*	spool directories
/dev/lp*	line printer devices
/etc/hosts.equiv	lists machine names allowed printer access

SEE ALSO

lpc(8), pac(8), lpr(1), lpq(1), lprm(1), printcap(5)
4.2BSD Line Printer Spooler Manual

NAME

MAKEDEV – make system special files

SYNOPSIS

/dev/MAKEDEV *device...*

DESCRIPTION

MAKEDEV is a shell script normally used to install special files. It resides in the */dev* directory, as this is the normal location of special files. Arguments to *MAKEDEV* are usually of the form *device-name*? where *device-name* is one of the supported devices listed in section 4 of the manual and '?' is a logical unit number (0-9). A few special arguments create assorted collections of devices and are listed below.

std Create the *standard* devices for the system; for example, /dev/console, /dev/tty.

local Create those devices specific to the local site. This request runs the shell file */dev/MAKEDEV.local*. Site specific commands, such as those used to setup dialup lines as 'ttyd?' should be included in this file.

Since all devices are created using *mknod*(8), this shell script is useful only to the super-user.

DIAGNOSTICS

Either self-explanatory, or generated by one of the programs called from the script. Use **sh -x MAKEDEV** in case of trouble.

SEE ALSO

intro(4), config(8), mknod(8)

NAME

makekey – generate encryption key

SYNOPSIS

/usr/lib/makekey

DESCRIPTION

Makekey improves the usefulness of encryption schemes depending on a key by increasing the amount of time required to search the key space. It reads 10 bytes from its standard input, and writes 13 bytes on its standard output. The output depends on the input in a way intended to be difficult to compute (that is, to require a substantial fraction of a second).

The first eight input bytes (the *input key*) can be arbitrary ASCII characters. The last two (the *salt*) are best chosen from the set of digits, upper- and lower-case letters, and '.' and '/'. The salt characters are repeated as the first two characters of the output. The remaining 11 output characters are chosen from the same set as the salt and constitute the *output key.*

The transformation performed is essentially the following: the salt is used to select one of 4096 cryptographic machines all based on the National Bureau of Standards DES algorithm, but modified in 4096 different ways. Using the input key as key, a constant string is fed into the machine and recirculated a number of times. The 64 bits that come out are distributed into the 66 useful key bits in the result.

Makekey is intended for programs that perform encryption (for instance, *ed* and *crypt*(1)). Usually makekey's input and output will be pipes.

SEE ALSO

crypt(1), ed(1)

NAME

mkfs – construct a file system

SYNOPSIS

/etc/mkfs special size [nsect] [ntrack] [blksize] [fragsize] [ncpg] [minfree] [rps]

DESCRIPTION

N.B.: file system are normally created with the *newfs*(8) command.

Mkfs constructs a file system by writing on the special file *special.* The numeric size specifies the number of sectors in the file system. *Mkfs* builds a file system with a root directory and a *lost+found* directory (see *fsck*(8)). The number of i-nodes is calculated as a function of the file system size. No boot program is initialized by *mkfs* (see *newfs*(8)).

OPTIONS

The optional arguments allow fine tune control over the parameters of the file system.

Nsect specify the number of sectors per track on the disk.

Ntrack
specify the number of tracks per cylinder on the disk.

Blksize
gives the primary block size for files on the file system. It must be a power of two, currently selected from 4096 or 8192.

Fragsize
gives the fragment size for files on the file system. The **fragsize** represents the smallest amount of disk space that will be allocated to a file. It must be a power of two currently selected from the range 512 to 8192.

Ncpg specifies the number of disk cylinders per cylinder group. This number must be in the range 1 to 32.

Minfree
specifies the minimum percentage of free disk space allowed. Once the file system capacity reaches this threshold, only the super-user is allowed to allocate disk blocks. The default value is 10%.

rps If a disk does not revolve at 60 revolutions per second, this parameter may be specified.

Users with special demands for their file systems are referred to the paper cited below for a discussion of the tradeoffs in using different configurations.

SEE ALSO

fs(5), dir(5), fsck(8), newfs(8), tunefs(8)

McKusick, Joy, Leffler; *A Fast File System for Unix,* Computer Systems Research Group, Dept of EECS, Berkeley, CA 94720;

NAME

mknod – build special file

SYNOPSIS

/etc/mknod name [**c**] [**b**] major minor

DESCRIPTION

Mknod makes a special file. The first argument is the *name* of the entry. The second is **b** if the special file is block-type (disks, tape) or **c** if it is character-type (other devices). The last two arguments are numbers specifying the *major* device type and the *minor* device (for example, unit, drive, or line number).

Mknod is only for use by system configuration people. Normally you should use */dev/MAKEDEV* instead when making special files.

SEE ALSO

mknod(2), makedev(8)

NAME

mkproto – construct a prototype file system

SYNOPSIS

/usr/etc/mkproto special proto

DESCRIPTION

Mkproto is used to bootstrap a new file system. First a new file system is created using *newfs*(8). *Mkproto* is then used to copy files from the old file system into the new file system according to the directions found in the prototype file *proto*. The prototype file contains tokens separated by spaces or new lines. The first tokens comprise the specification for the root directory. File specifications consist of tokens giving the mode, the user-id, the group id, and the initial contents of the file. The syntax of the contents field depends on the mode.

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

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

If the file is a regular file, the next token is a pathname whence the contents and size are copied.

If the file is a block or character special file, two decimal number tokens follow which give the major and minor device numbers.

If the file is a directory, *mkproto* makes the entries **.** and **..** and then reads a list of names and (recursively) file specifications for the entries in the directory. The scan is terminated with the token **$**.

A sample prototype specification follows:

```
d—777 3 1
usr     d—777 3 1
        sh      —–755 3 1 /bin/sh
        ken     d—755 6 1
                $
        b0      b—644 3 1 0 0
        c0      c—644 3 1 0 0
        $
$
```

SEE ALSO

fs(5), dir(5), fsck(8), newfs(8)

BUGS

There should be some way to specify links.

There should be some way to specify bad blocks.

Mkproto can only be run on virgin file systems. It should be possible to copy files into existent file systems.

NAME

mount, umount – mount and dismount file system

SYNOPSIS

/etc/mount [special name [**–r**]]

/etc/mount –a

/etc/umount [**–v**] special

/etc/umount [**–v**] **–a**

DESCRIPTION

Mount announces to the system that a removable file system is present on the device *special*. The file *name* must exist already and it must be a directory (unless the root of the mounted file system is not a directory). It becomes the name of the newly mounted root. The optional argument **–r** indicates that the file system is to be mounted read-only.

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

OPTIONS

–r Mount the specified file system read-only. This is only applicable to the *mount* command.

–a Attempt to mount or unmount all the file systems described in */etc/fstab*. In this case, *special* and *name* are taken from */etc/fstab*. The *special* file name from */etc/fstab* is the block special name.

–v The verbose option for *umount*: *umount* displays a message indicating the file system being unmounted.

Mount and *umount* maintain a table of mounted devices in */etc/mtab*. If invoked without an argument, *mount* prints the table.

Physically write-protected and magnetic tape file systems must be mounted read-only or errors will occur when access times are updated, whether or not any explicit write is attempted.

FILES

/etc/mtab	mount table
/etc/fstab	file system table

SEE ALSO

mount(2), mtab(5), fstab(5)

BUGS

Mounting file systems full of garbage will crash the system.
Mounting a root directory on a non-directory makes some apparently good pathnames invalid.

NAME

ncheck – generate names from i-numbers

SYNOPSIS

/usr/etc/ncheck [–**i***numbers*] [**–a**] [**–s**] [filesystem]

DESCRIPTION

N.B.: For most normal file system maintenance, the function of *ncheck* is subsumed by *fsck*(8).

Ncheck with no argument generates a pathname versus i-number list of all files on a set of default file systems. Names of directory files are followed by '/.'.

A file system may be specified by the optional *filesystem* argument.

The report is in no useful order, and probably should be sorted.

OPTIONS

–**i***numbers*
: Report only those files whose i-*numbers* follow.

–a
: Print the names '.' and '..', which are ordinarily suppressed.

–s
: Report only special files and files with set-user-ID mode. This is intended to discover concealed violations of security policy.

SEE ALSO

sort(1), dcheck(8), fsck(8), icheck(8)

DIAGNOSTICS

When the filesystem structure is improper, '??' denotes the 'parent' of a parentless file and a pathname beginning with '...' denotes a loop.

NAME

netstat – show network status

SYNOPSIS

netstat [**–Aahimnrs**] [**–p** *protocol*] [**–a**] [*interval*] [*system*] [*core*]

DESCRIPTION

The *netstat* command symbolically displays the contents of various network-related data structures. The options have the following meaning:

–A show the address of any associated protocol control blocks; used for debugging

–a show the state of all sockets; normally sockets used by server processes are not shown

–h show the state of the IMP host table

–i show the state of interfaces which have been auto-configured (interfaces statically configured into a system, but not located at boot time are not shown)

–m show statistics recorded by the memory management routines (the network manages a "private share" of memory)

–n show network addresses as numbers (normally *netstat* interprets addresses and attempts to display them symbolically)

–p *proto*
show the state of sockets utilizing protocol *proto*; the protocol is specified symbolically, and may be any protocol listed in the file */etc/protocols.*

–s show per-protocol statistics

–r show the routing tables

The arguments, *system* and *core* allow substitutes for the defaults "/vmunix" and "/dev/kmem".

If an *interval* is specified, *netstat* will continuously display the information regarding packet traffic on the configured network interfaces, pausing *interval* seconds before refreshing the screen.

There are a number of display formats, depending on the information presented. The default display, for active sockets, shows the local and remote addresses, send and receive queue sizes (in bytes), protocol, and, optionally, the internal state of the protocol.

Address formats are of the form "host.port" or "network.port" if a socket's address specifies a network but no specific host address. When known the host and network addresses are displayed symbolically according to the data bases */etc/hosts* and */etc/networks*, respectively. If a symbolic name for an address is unknown, or if the **–n** option is specified, the address is printed in the Internet "dot format"; refer to *inet*(3N) for more information regarding this format. Unspecified, or "wildcard", addresses and ports appear as "*".

The interface display provides a table of cumulative statistics regarding packets transferred, errors, and collisions. The network address (currently Internet specific) of the interface and the maximum transmission unit ("mtu") are also displayed.

The routing table display indicates the available routes and their status. Each route consists of a destination host or network and a gateway to use in forwarding packets. The flags field shows the state of the route ("U" if "up"), and whether the route is to a gateway ("G"). Direct routes are created for each interface attached to the local host. The refcnt field gives the current number of active uses of the route. Connection oriented protocols normally hold on to a single route for the duration of a connection while connectionless protocols obtain a route then discard it. The use field provides a count of the number of packets sent using that route. The interface entry indicates the network interface utilized for the route.

When *netstat* is invoked with an *interval* argument, it displays a running count of statistics related to network interfaces. This display consists of a column summarizing information for all interfaces, and a column for the interface with the most traffic since the system was last rebooted.

The first line of each screen of information contains a summary since the system was last rebooted. Subsequent lines of output show values accumulated over the preceding interval.

SEE ALSO

iostat(8), vmstat(8), hosts(5), networks(5), protocols(5), services(5), trpt(8C)

BUGS

The notion of errors is ill-defined. Collisions mean something else for the IMP.

NAME

newaliases – rebuild the data base for the mail aliases file

SYNOPSIS

newaliases

DESCRIPTION

Newaliases rebuilds the random access data base for the mail aliases file */usr/lib/aliases*. It must be run each time */usr/lib/aliases* is changed in order for the change to take effect.

SEE ALSO

aliases(5), sendmail(8)

NAME

newfs – construct a new file system

SYNOPSIS

/etc/newfs [–v] [–n] [mkfs-options] special [disk-type]

DESCRIPTION

Newfs is a "friendly" front-end to the *mkfs*(8) program. On the VAX, *newfs* will look up the type of disk a file system is being created on in the disk description file */etc/disktab*; on the Sun the disk type is determined by reading the disk label. *Newfs* then calculate the appropriate parameters to use in calling *mkfs*, then build the file system by forking *mkfs* and, if the file system is a root partition, install the necessary bootstrap programs in the initial 16 sectors of the device.

OPTIONS

–n Do not instal the bootstrap programs.

–v *newfs* prints out its actions, including the parameters passed to *mkfs*.

Options which may be used to override default parameters passed to *mkfs* are:

–s size The size of the file system in sectors.

–b block-size
The block size of the file system in bytes.

–f frag-size
The fragment size of the file system in bytes.

–t #tracks/cylinder

–c #cylinders/group
The number of cylinders per cylinder group in a file system. The default value used is 16.

–m free space %
The percentage of space reserved from normal users; the minimum free space threshhold. The default value used is 10%.

–r revolutions/minute
The speed of the disk in revolutions per minute (normally 3600).

FILES

/etc/disktab	for disk geometry and file partition information (VAX only)
/etc/mkfs	to actually build the file system
/usr/mdec	for boot strapping programs

SEE ALSO

fs(5), fsck(8), tunefs(8)

McKusick, Joy, Leffler; "A Fast File System for Unix", Computer Systems Research Group, Dept of EECS, Berkeley, CA 94720; TR #7, September 1982.

NAME

pac – printer/plotter accounting information

SYNOPSIS

/usr/etc/pac [**–P***printer*] [**–p***price*] [**–s**] [**–r**] [**–c**] [name ...]

DESCRIPTION

Pac reads the printer/plotter accounting files, accumulating the number of pages (the usual case) or feet (for raster devices) of paper consumed by each user, and printing out how much each user consumed in pages or feet and dollars. If any *names* are specified, then statistics are only printed for those users; usually, statistics are printed for every user who has used any paper.

OPTIONS

–P*printer*
: Do accounting for the named *printer*. Normally, accounting is done for the default printer (site dependent) or the value of the PRINTER environment variable is used.

–p*price*
: Use the value *price* for the cost in dollars instead of the default value of 0.02.

–c Sorted the output by cost; usually the output is sorted alphabetically by name.

–r Reverse the sorting order.

–s Summarize the accounting information on the summary accounting file; this summary is necessary since on a busy system, the accounting file can grow by several lines per day.

FILES

/usr/adm/?acct	raw accounting files
/usr/adm/?_sum	summary accounting files

BUGS

The relationship between the computed price and reality is as yet unknown.

NAME
pstat - print system facts

SYNOPSIS
/etc/pstat -a 5ixptufT [**suboptions**] [**system**] [**corefile**]

DESCRIPTION
Pstat interprets the contents of certain system tables. If *corefile* is given, the tables are sought there, otherwise in */dev/kmem.* The required namelist is taken from */vmunix* unless *system* is specified.

OPTIONS

-a Under **-p**, describe all process slots rather than just active ones.

-i Print the inode table with the these headings:

LOC	The core location of this table entry.	
FLAGS	Miscellaneous state variables encoded thus:	
	L	locked
	U	update time (*fs*(5)) must be corrected
	A	access time must be corrected
	M	file system is mounted here
	W	wanted by another process (L flag is on)
	T	contains a text file
	C	changed time must be corrected
	S	shared lock applied
	E	exclusive lock applied
	Z	someone waiting for an exclusive lock
CNT	Number of open file table entries for this inode.	
DEV	Major and minor device number of file system in which this inode resides.	
RDC	Reference count of shared locks on the inode.	
WRC	Reference count of exclusive locks on the inode (this may be > 1 if, for example, a file descriptor is inherited across a fork).	
INO	I-number within the device.	
MODE	Mode bits, see *chmod*(2).	
NLK	Number of links to this inode.	
UID	User ID of owner.	
SIZ/DEV	Number of bytes in an ordinary file, or major and minor device of special file.	

-x Print the text table with these headings:

LOC	The core location of this table entry.	
FLAGS	Miscellaneous state variables encoded thus:	
	T	*ptrace*(2) in effect
	W	text not yet written on swap device
	L	loading in progress
	K	locked
	w	wanted (L flag is on)
	P	resulted from demand-page-from-inode exec format (see *execve*(2))
DADDR	Disk address in swap, measured in multiples of 512 bytes.	
CADDR	Head of a linked list of loaded processes using this text segment.	
SIZE	Size of text segment, measured in multiples of 512 bytes.	
IPTR	Core location of corresponding inode.	
CNT	Number of processes using this text segment.	

CCNT	Number of processes in core using this text segment.

−p Print process table for active processes with these headings:

LOC	The core location of this table entry.
S	Run state encoded thus:
	0 no process
	1 (unused)
	3 runnable
	4 being created
	5 being terminated
	6 stopped under trace
F	Miscellaneous state variables, or-ed together (hexadecimal):
	0000001 loaded
	0000002 a system process (scheduler or page-out daemon)
	0000004 locked for swap out
	0000008 swapped out during process creation
	0000010 traced
	0000020 used in tracing
	0000040 (unused)
	0000080 in page-wait
	0000100 prevented from swapping during *fork*(2)
	0000200 restore old sigmask after taking signal
	0000400 exiting
	0001000 process resulted from a *vfork*(2) which is not yet complete
	0002000 another flag for *vfork*(2)
	0004000 process has no virtual memory, as it is a parent in the context of *vfork*(2)
	0008000 process is demand paging data pages from its text inode.
	0010000 process has advised of sequential behavior with *vadvise*(2).
	0020000 process has advised of anomalous behavior with *vadvise*(2).
	0040000 process is in a sleep which will timeout.
	0080000 (unused)
	0100000 using old signal mechanism
	0200000 process is owed a profiling tick.
	0400000 process is setting up a *select*(2) call
	800000 Process is a login process
	1000000 Page tables for this process have changed (tlb is dirty)
POIP	number of pages currently being pushed out from this process.
PRI	Scheduling priority, see *setpriority*(2).
SIGNAL	Signals received (signals 1-32 coded in bits 0-31),
UID	Real user ID.
SLP	Amount of time process has been blocked.
TIM	Time resident in seconds; times over 127 coded as 127.
CPU	Weighted integral of CPU time, for scheduler.
NI	Nice level, see *setpriority*(2).
PGRP	Process number of root of process group (the opener of the controlling terminal).
PID	The process ID number.
PPID	The process ID of parent process.
ADDR	If in core, the page frame number of the first page of the 'u-area' of the process. If swapped out, the position in the swap area measured in multiples of 512 bytes.
RSS	Resident set size – the number of physical page frames allocated to this process.
SRSS	RSS at last swap (0 if never swapped).
SIZE	Virtual size of process image (data+stack) in multiples of 512 bytes.

WCHAN	Wait channel number of a waiting process.
LINK	Link pointer in list of runnable processes.
TEXTP	If text is pure, pointer to location of text table entry.
CLKT	Countdown for real interval timer, *setitimer*(2) measured in clock ticks (10 milliseconds).

–t Print table for terminals with these headings:

RAW	Number of characters in raw input queue.
CAN	Number of characters in canonicalized input queue.
OUT	Number of characters in putput queue.
MODE	See *tty*(4).
ADDR	Physical device address.
DEL	Number of delimiters (newlines) in canonicalized input queue.
COL	Calculated column position of terminal.
STATE	Miscellaneous state variables encoded thus:

W	waiting for open to complete
O	open
S	has special (output) start routine
C	carrier is on
B	busy doing output
A	process is awaiting output
X	open for exclusive use
H	hangup on close

PGRP	Process group for which this is controlling terminal.
DISC	Line discipline; blank is old tty OTTYDISC or "new tty" for NTTYDISC or "net" for NETLDISC (see *bk*(4)).

–u print information about a user process; the next argument is its address as given by *ps*(1). The process must be in main memory, or the file used can be a core image and the address 0.

–f Print the open file table with these headings:

LOC	The core location of this table entry.
TYPE	The type of object the file table entry points to.
FLG	Miscellaneous state variables encoded thus:

R	open for reading
W	open for writing
A	open for appending

CNT	Number of processes that know this open file.
INO	The location of the inode table entry for this file.
OFFS/SOCK	The file offset (see *lseek*(2)), or the core address of the associated socket structure.

–s print information about swap space usage: the number of (1k byte) pages used and free is given as well as the number of used pages which belong to text images.

–T prints the number of used and free slots in the several system tables and is useful for checking to see how full system tables have become if the system is under heavy load.

FILES

/vmunix	namelist
/dev/kmem	default source of tables

SEE ALSO

ps(1), stat(2), fs(5)
K. Thompson, *UNIX Implementation*

BUGS

It would be very useful if the system recorded "maximum occupancy" on the tables reported by **–T**; even more useful if these tables were dynamically allocated.

NAME

rc – command script for auto-reboot and daemons

SYNOPSIS

/etc/rc
/etc/rc.local

DESCRIPTION

Rc is the command script which controls the automatic reboot and *rc.local* is the script holding commands which are pertinent only to a specific site.

When an automatic reboot is in progress, *rc* is invoked with the argument *autoboot* and runs a *fsck* with option **-p** to "preen" all the disks of minor inconsistencies resulting from the last system shutdown and to check for serious inconsistencies caused by hardware or software failure. If this auto-check and repair succeeds, then the second part of *rc* is run.

The second part of *rc,* which is run after a auto-reboot succeeds and also if *rc* is invoked when a single user shell terminates (see *init*(8)), starts all the daemons on the system, preserves editor files and clears the scratch directory **/tmp.** *Rc.local* is executed immediately before any other commands after a successful *fsck*. Normally, the first commands placed in the *rc.local* file define the machine's name, using *hostname*(1), and save any possible core image that might have been generated as a result of a system crash, *savecore*(8). The latter command is included in the *rc.local* file because the directory in which core dumps are saved is usually site specific.

SEE ALSO

init(8), reboot(8), savecore(8)

BUGS

NAME

rdate – set system date from a remote host

SYNOPSIS

/usr/etc/rdate hostname

DESCRIPTION

Rdate is a shell script to set the local date and time from the *hostname* given as argument. You must be super-user on the local system. Typically *rdate* can be inserted as part of your */etc/rc.local* startup script.

BUGS

Prints 'bad conversion' if remote host unavailable. Could be modified to accept a list of hostnames and try each until a valid date returned. Better yet would be to write a real date server that accepted broadcast requests.

NAME

reboot – UNIX bootstrapping procedures

SYNOPSIS

/etc/reboot [**-n**] [**-q**]

DESCRIPTION

The UNIX operating system is started by placing it in memory at location zero and transferring to zero. Since the system is not reenterable, it is necessary to read it in from disk or tape each time it is to be bootstrapped.

Rebooting a running system. When a UNIX system is running and a reboot is desired, *shutdown*(8) is normally used. If there are no users then **/etc/reboot** can be used. Reboot performs a *sync* operation on the disks, and then a multi-user reboot (as described below) is initiated. This causes a system to be booted and an automatic disk check to be performed. If all this succeeds without incident, the system is then brought up for many users.

OPTIONS

-n	option avoids the sync. It can be used if a disk or the processor is on fire.
-q	reboots quickly and ungracefully, without first shutting down running processes.

Power fail and crash recovery. Normally, the system will reboot itself at power-up or after crashes. Provided the auto-restart is enabled on the machine front panel, an automatic consistency check of the file systems will be performed then and unless this fails the system will resume multi-user operations.

On both processors, the *boot* program finds the corresponding file on the given device, loads that file into memory location zero, and starts the program at the entry address specified in the program header (after clearing off the high bit of the specified entry address.) Normal line editing characters can be used in specifying the pathname.

For tapes, the minor device number gives a file offset.

FILES

/vmunix	system code
/boot	system bootstrap

SEE ALSO

crash(8S), fsck(8), init(8), rc(8), shutdown(8), halt(8), newfs(8)

NAME

renice – alter priority of running processes

SYNOPSIS

/etc/renice [**–g**] [**–u**] priority who ...

DESCRIPTION

Renice can be used to alter the scheduling priority of one or more running processes. By default, the processes to be affected are specified by their process id's. If the **–g** option is specified, the *who* parameters are interpreted as process groups and all the processes in the specified process groups have their scheduling priority altered. If the **–u** option is indicated, the *who* parameters are interpreted as user names and all process owned by the user are affected.

Users other than the super-user may only alter the priority of processes they own, and can only monotonically increase their "nice value" within the range 0 to PRIO_MIN (20). (This prevents overriding administrative fiats.) The super-user may alter the priority of any process and set the priority to any value in the range PRIO_MAX (–20) to PRIO_MIN. Useful priorities are: 19 (the affected processes will run only when nothing else in the system wants to), 0 (the "base" scheduling priority), anything negative (to make things go very fast).

If no *who* parameter is specified, the current process (alternatively, process group or user) is used.

FILES

/etc/passwd	to map user names to user id's

SEE ALSO

getpriority(2)

BUGS

If you make the priority very negative, then the process cannot be interrupted. To regain control you make the priority greater than zero. Non super-users can not increase scheduling priorities of their own processes, even if they were the ones that decreased the priorities in the first place.

NAME

restore – incremental file system restore

SYNOPSIS

/etc/restore key [name ...]

DESCRIPTION

Restore reads tapes dumped with the *dump*(8) command. Its actions are controlled by the *key* argument. The *key* is a string of characters containing at most one function letter and possibly one or more function modifiers. Other arguments to the command are file or directory names specifying the files that are to be restored. Unless the **h** key is specified (see below), the appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

The function portion of the key is specified by one of the following letters:

r The tape is read and loaded into the current directory. This should not be done lightly; the **r** key should only be used to restore a complete dump tape onto a clear file system or to restore an incremental dump tape after a full level zero restore. Thus

```
/etc/newfs /dev/rxy0g eagle
/etc/mount /dev/xy0g /mnt
cd /mnt
restore r
```

is a typical sequence to restore a complete dump. Another *restore* can be done to get an incremental dump in on top of this.
A *dump*(8) followed by a *newfs*(8) and a *restore* is used to change the size of a file system.

R *Restore* requests a particular tape of a multi volume set on which to restart a full restore (see the **r** key above). This allows *restore* to be interrupted and then restarted.

x The named files are extracted from the tape. If the named file matches a directory whose contents had been written onto the tape, and the **h** key is not specified, the directory is recursively extracted. The owner, modification time, and mode are restored (if possible). If no file argument is given, then the root directory is extracted, which results in the entire content of the tape being extracted, unless the **h** key has been specified.

t The names of the specified files are listed if they occur on the tape. If no file argument is given, then the root directory is listed, which results in the entire content of the tape being listed, unless the **h** key has been specified. Note that the **t** key replaces the function of the old *dumpdir* program.

i This mode allows interactive restoration of files from a dump tape. After reading in the directory information from the tape, *restore* provides a shell like interface that allows the user to move around the directory tree selecting files to be extracted. The available commands are given below; for those commands that require an argument, the default is the current directory.

ls [arg] – List the current or specified directory. Entries that are directories are appended with a "/". Entries that have been marked for extraction are prepended with a "*". If the verbose key is set the inode number of each entry is also listed.

cd arg – Change the current working directory to the specified argument.

pwd – Print the full pathname of the current working directory.

add [arg] – The current directory or specified argument is added to the list of files to be extracted. If a directory is specified, then it and all its descendents are added to the extraction list (unless the **h** key is specified on the command line). Files that are on the extraction list are prepended with a "*" when they are listed by **ls**.

delete [arg] – The current directory or specified argument is deleted from the list of files to be extracted. If a directory is specified, then it and all its descendents are deleted from the extraction list (unless the **h** key is specified on the command line). The most expedient way to extract most of the files from a directory is to add the directory to the extraction list and then delete those files that are not needed.

extract – All the files that are on the extraction list are extracted from the dump tape. *Restore* will ask which volume the user wishes to mount. The fastest way to extract a few files is to start with the last volume, and work towards the first volume.

verbose – The sense of the **v** key is toggled. When set, the verbose key causes the **ls** command to list the inode numbers of all entries. It also causes *restore* to print out information about each file as it is extracted.

help – List a summary of the available commands.

quit – Restore immediately exits, even if the extraction list is not empty.

The following characters may be used in addition to the letter that selects the function desired.

v Normally *restore* does its work silently. The **v** (verbose) key causes it to type the name of each file it treats preceded by its file type.

f The next argument to *restore* is used as the name of the archive instead of /dev/rmt?. If the name of the file is "–", *restore* reads from standard input. If the name of the file is "machine:device" the restore is done from the specified machine through the internet using *rmt*(8C). Thus, *dump*(8) and *restore* can be used in a pipeline to dump and restore a file system with the command

```
dump 0f - /usr | (cd /mnt; restore xf -)
```

y *Restore* will not ask whether it should abort the restore if gets a tape error. It will always try to skip over the bad tape block(s) and continue as best it can.

m *Restore* will extract by inode numbers rather than by file name. This is useful if only a few files are being extracted, and one wants to avoid regenerating the complete pathname to the file.

h *Restore* extracts the actual directory, rather than the files that it references. This prevents hierarchical restoration of complete subtrees from the tape.

DIAGNOSTICS

Complaints about bad key characters.

Complaints if it gets a read error. If **y** has been specified, or the user responds "y", *restore* will attempt to continue the restore.

If the dump extends over more than one tape, *restore* will ask the user to change tapes. If the **x** or **i** key has been specified, *restore* will also ask which volume the user wishes to mount. The fastest way to extract a few files is to start with the last volume, and work towards the first volume.

There are numerous consistency checks that can be listed by *restore*. Most checks are self-explanatory or can "never happen". Common errors are given below.

Converting to new file system format.
A dump tape created from the old file system has been loaded. It is automatically converted to the new file system format.

<filename>: not found on tape
: The specified file name was listed in the tape directory, but was not found on the tape. This is caused by tape read errors while looking for the file, and from using a dump tape created on an active file system.

expected next file <inumber>, got <inumber>
: A file that was not listed in the directory showed up. This can occur when using a dump tape created on an active file system.

Incremental tape too low
: When doing incremental restore, a tape that was written before the previous incremental tape, or that has too low an incremental level has been loaded.

Incremental tape too high
: When doing incremental restore, a tape that does not begin its coverage where the previous incremental tape left off, or that has too high an incremental level has been loaded.

Tape read error while restoring <filename>
Tape read error while skipping over inode <inumber>
Tape read error while trying to resynchronize
: A tape read error has occurred. If a file name is specified, then its contents are probably partially wrong. If an inode is being skipped or the tape is trying to resynchronize, then no extracted files have been corrupted, though files may not be found on the tape.

resync restore, skipped <num> blocks
: After a tape read error, *restore* may have to resynchronize itself. This message lists the number of blocks that were skipped over.

FILES

/dev/rmt?	the default tape drive
/tmp/rstdir*	file containing directories on the tape.
/tmp/rstmode*	owner, mode, and time stamps for directories.
./restoresymtab	information passed between incremental restores.

SEE ALSO

dump(8), newfs(8), mount(8), mkfs(8), rmt(8C)

BUGS

Restore can get confused when doing incremental restores from dump tapes that were made on active file systems.

A level zero dump must be done after a full restore. Because restore runs in user mode, it has no control over inode allocation; this means that *restore* re-positions the files, although it does change their contents. Thus, a full dump must be done to get a new set of directories reflecting the new file positions, so that later incremental dumps will be correct.

NAME

rexecd – remote execution server

SYNOPSIS

/usr/etc/in.rexecd host.port

DESCRIPTION

Rexecd is the server for the *rexec*(3N) routine. The server provides remote execution facilities with authentication based on user names and encrypted passwords. It is invoked automatically as needed by *inetd*(8C), and then executes the following protocol:

1) The server reads characters from the socket up to a null ('\0') byte. The resultant string is interpreted as an ASCII number, base 10.

2) If the number received in step 1 is non-zero, it is interpreted as the port number of a secondary stream to be used for the **stderr**. A second connection is then created to the specified port on the client's machine.

3) A null terminated user name of at most 16 characters is retrieved on the initial socket.

4) A null terminated, encrypted, password of at most 16 characters is retrieved on the initial socket.

5) A null terminated command to be passed to a shell is retrieved on the initial socket. The length of the command is limited by the upper bound on the size of the system's argument list.

6) *Rexecd* then validates the user as is done at login time and, if the authentication was successful, changes to the user's home directory, and establishes the user and group protections of the user. If any of these steps fail the connection is aborted with a diagnostic message returned.

7) A null byte is returned on the connection associated with the **stderr** and the command line is passed to the normal login shell of the user. The shell inherits the network connections established by *rexecd*.

DIAGNOSTICS

All diagnostic messages are returned on the connection associated with the **stderr**, after which any network connections are closed. An error is indicated by a leading byte with a value of 1 (0 is returned in step 7 above upon successful completion of all the steps prior to the command execution).

"username too long"
The name is longer than 16 characters.

"password too long"
The password is longer than 16 characters.

"command too long "
The command line passed exceeds the size of the argument list (as configured into the system).

"Login incorrect."
No password file entry for the user name existed.

"Password incorrect."
The wrong was password supplied.

"No remote directory."
The *chdir* command to the home directory failed.

"Try again."
A *fork* by the server failed.

"/bin/sh: ..."
The user's login shell could not be started.

BUGS

Indicating "Login incorrect" as opposed to "Password incorrect" is a security breach which allows people to probe a system for users with null passwords.

A facility to allow all data exchanges to be encrypted should be present.

SEE ALSO

inetd(8C)

NAME

rlogind – remote login server

SYNOPSIS

/etc/in.rlogind host.port

DESCRIPTION

Rlogind is the server for the *rlogin*(1C) program. The server provides a remote login facility with authentication based on privileged port numbers.

Rlogind is invoked by *inetd*(8C) when a remote login connection is established, and executes the following protocol:

1) The server checks the client's source port. If the port is not in the range 0-1023, the server aborts the connection. The client's address and port number are passed as arguments to *rlogind* by *inetd* in the form "host.port" with host in hex and port in decimal.

2) The server checks the client's source address. If the address is associated with a host for which no corresponding entry exists in the host name data base (see *hosts*(5)), the server aborts the connection.

Once the source port and address have been checked, *rlogind* allocates a pseudo terminal (see *pty*(4)), and manipulates file descriptors so that the slave half of the pseudo terminal becomes the **stdin** , **stdout** , and **stderr** for a login process. The login process is an instance of the *login*(1) program, invoked with the **–r** option. The login process then proceeds with the authentication process as described in *rshd*(8C), but if automatic authentication fails, it reprompts the user to login as one finds on a standard terminal line.

The parent of the login process manipulates the master side of the pseduo terminal, operating as an intermediary between the login process and the client instance of the *rlogin* program. In normal operation, the packet protocol described in *pty*(4) is invoked to provide ^S/^Q type facilities and propagate interrupt signals to the remote programs. The login process propagates the client terminal's baud rate and terminal type, as found in the environment variable, "TERM"; see *environ*(5).

DIAGNOSTICS

All diagnostic messages are returned on the connection associated with the **stderr**, after which any network connections are closed. An error is indicated by a leading byte with a value of 1.

"Hostname for your address unknown."
No entry in the host name database existed for the client's machine.

"Try again."
A *fork* by the server failed.

"/bin/sh: ..."
The user's login shell could not be started.

BUGS

The authentication procedure used here assumes the integrity of each client machine and the connecting medium. This is insecure, but is useful in an "open" environment.

A facility to allow all data exchanges to be encrypted should be present.

SEE ALSO

inetd(8C)

NAME

rmail – handle remote mail received via uucp

SYNOPSIS

rmail user ...

DESCRIPTION

Rmail interprets incoming mail received via *uucp*(1C), collapsing "From" lines in the form generated by *binmail*(1) into a single line of the form "return-path!sender", and passing the processed mail on to *sendmail*(8).

Rmail is explicitly designed for use with *uucp* and *sendmail.*

SEE ALSO

binmail(1), uucp(1C), sendmail(8)

BUGS

Rmail should not reside in /bin.

NAME

rmt – remote magtape protocol module

SYNOPSIS

/etc/rmt

DESCRIPTION

Rmt is a program used by the remote dump and restor programs in manipulating a magnetic tape drive through an interprocess communication connection. *Rmt* is normally started up with an *rexec*(3N) or *rcmd*(3N) call.

The *rmt* program accepts requests specific to the manipulation of magnetic tapes, performs the commands, then responds with a status indication. All responses are in ASCII and in one of two forms. Successful commands have responses of

A*number*\n

where *number* is an ASCII representation of a decimal number. Unsuccessful commands are responded to with

E*error-number*\n*error-message*\n,

where *error-number* is one of the possible error numbers described in *intro*(2) and *error-message* is the corresponding error string as printed from a call to *perror*(3). The protocol is comprised of the following commands (a space is present between each token).

O device mode
Open the specified *device* using the indicated *mode*. *Device* is a full pathname and *mode* is an ASCII representation of a decimal number suitable for passing to *open*(2). If a device had already been opened, it is closed before a new open is performed.

C device Close the currently open device. The *device* specified is ignored.

L whence offset
Perform an *lseek*(2) operation using the specified parameters. The response value is that returned from the *lseek* call.

W count Write data onto the open device. *Rmt* reads *count* bytes from the connection, aborting if a premature end-of-file is encountered. The response value is that returned from the *write*(2) call.

R count Read *count* bytes of data from the open device. If *count* exceeds the size of the data buffer (10 kilobytes), it is truncated to the data buffer size. *Rmt* then performs the requested *read*(2) and responds with **A***count-read*\n if the read was successful; otherwise an error in the standard format is returned. If the read was successful, the data read is then sent.

I operation count
Perform a MTIOCOP *ioctl*(2) command using the specified parameters. The parameters are interpreted as the ASCII representations of the decimal values to place in the *mt_op* and *mt_count* fields of the structure used in the *ioctl* call. The return value is the *count* parameter when the operation is successful.

S Return the status of the open device, as obtained with a MTIOCGET *ioctl* call. If the operation was successful, an "ack" is sent with the size of the status buffer, then the status buffer is sent (in binary).

Any other command causes *rmt* to exit.

DIAGNOSTICS

All responses are of the form described above.

SEE ALSO

rcmd(3N), rexec(3N), mtio(4), dump(8), restore(8)

BUGS

People tempted to use this for a remote file access protocol are discouraged.

NAME

route – manually manipulate the routing tables

SYNOPSIS

/usr/etc/route [**–f**] [*command args*]

DESCRIPTION

Route is a program used to manually manipulate the network routing tables. It normally is not needed, as the system routing table management daemon, *routed*(8C), should tend to this task.

Route accepts three commands: *add*, to add a route; *delete*, to delete a route; and *change*, to modify an existing route.

All commands have the following syntax:

/usr/etc/route *command* **destination gateway** [**metric**]

where *destination* is a host or network for which the route is "to", *gateway* is the gateway to which packets should be addressed, and *metric* is an optional count indicating the number of hops to the *destination*. If no metric is specified, *route* assumes a value of 0. Routes to a particular host are distinguished from those to a network by interpreting the Internet address associated with *destination*. If the *destination* has a "local address part" of INADDR_ANY, then the route is assumed to be to a network; otherwise, it is presumed to be a route to a host. If the route is to a destination connected via a gateway, the *metric* should be greater than 0. All symbolic names specified for a *destination* or *gateway* are looked up first in the host name database, *hosts*(5). If this lookup fails, the name is then looked for in the network name database, *networks*(5).

Route uses a raw socket and the SIOCADDRT and SIOCDELRT *ioctl*'s to do its work. As such, only the super-user may modify the routing tables.

If the **–f** option is specified, *route* will "flush" the routing tables of all gateway entries. If this is used in conjunction with one of the commands described above, the tables are flushed prior to the command's application.

DIAGNOSTICS

"add %s: gateway %s flags %x"
The specified route is being added to the tables. The values printed are from the routing table entry supplied in the *ioctl* call.

"delete %s: gateway %s flags %x"
As above, but when deleting an entry.

"%s %s done"
When the **–f** flag is specified, each routing table entry deleted is indicated with a message of this form.

"not in table"
A delete operation was attempted for an entry which wasn't present in the tables.

"routing table overflow"
An add operation was attempted, but the system was low on resources and was unable to allocate memory to create the new entry.

SEE ALSO

routing(4N), routed(8C)

BUGS

The change operation is not implemented, one should add the new route, then delete the old one.

NAME

routed – network routing daemon

SYNOPSIS

/usr/etc/in.routed [**–s**] [**–q**] [**–t**] [*logfile*]

DESCRIPTION

Routed is invoked at boot time to manage the network routing tables. The routing daemon uses a variant of the Xerox NS Routing Information Protocol in maintaining up to date kernel routing table entries.

In normal operation *routed* listens on *udp*(4P) socket 520 (decimal) for routing information packets. If the host is an internetwork router, it periodically supplies copies of its routing tables to any directly connected hosts and networks.

When *routed* is started, it uses the SIOCGIFCONF *ioctl* to find those directly connected interfaces configured into the system and marked "up" (the software loopback interface is ignored). If multiple interfaces are present, it is assumed the host will forward packets between networks. *Routed* then transmits a *request* packet on each interface (using a broadcast packet if the interface supports it) and enters a loop, listening for *request* and *response* packets from other hosts.

When a *request* packet is received, *routed* formulates a reply based on the information maintained in its internal tables. The *response* packet generated contains a list of known routes, each marked with a "hop count" metric (a count of 16, or greater, is considered "infinite"). The metric associated with each route returned provides a metric *relative to the sender*.

Request packets received by *routed* are used to update the routing tables if one of the following conditions is satisfied:

(1) No routing table entry exists for the destination network or host, and the metric indicates the destination is "reachable" (that is, the hop count is not infinite).

(2) The source host of the packet is the same as the router in the existing routing table entry. That is, updated information is being received from the very internetwork router through which packets for the destination are being routed.

(3) The existing entry in the routing table has not been updated for some time (defined to be 90 seconds) and the route is at least as cost effective as the current route.

(4) The new route describes a shorter route to the destination than the one currently stored in the routing tables; the metric of the new route is compared against the one stored in the table to decide this.

When an update is applied, *routed* records the change in its internal tables and generates a *response* packet to all directly connected hosts and networks. *Routed* waits a short period of time (no more than 30 seconds) before modifying the kernel's routing tables to allow possible unstable situations to settle.

In addition to processing incoming packets, *routed* also periodically checks the routing table entries. If an entry has not been updated for 3 minutes, the entry's metric is set to infinity and marked for deletion. Deletions are delayed an additional 60 seconds to insure the invalidation is propagated throughout the internet.

Hosts acting as internetwork routers gratuitously supply their routing tables every 30 seconds to all directly connected hosts and networks.

Supplying the **–s** option forces *routed* to supply routing information whether it is acting as an internetwork router or not. The **–q** option is the opposite of the **–s** option. If the **–t** option is specified, all packets sent or received are printed on the standard output. In addition, *routed* will not divorce itself from the controlling terminal so that interrupts from the keyboard will kill the process. Any other argument supplied is interpreted as the name of file in which *routed*'s actions should be logged. This log contains information about any changes to the routing tables and a history of recent messages sent and received which are related to the changed route.

In addition to the facilities described above, *routed* supports the notion of "distant" *passive* and *active* gateways. When *routed* is started up, it reads the file */etc/gateways* to find gateways which may not be identified using the SIOGIFCONF *ioctl*. Gateways specified in this manner should be marked passive if they are not expected to exchange routing information, while gateways marked active should be willing to exchange routing information (that is, they should have a *routed* process running on the machine). Passive gateways are maintained in the routing tables forever and information regarding their existence is included in any routing information transmitted. Active gateways are treated equally to network interfaces. Routing information is distributed to the gateway and if no routing information is received for a period of the time, the associated route is deleted.

The */etc/gateways* is comprised of a series of lines, each in the following format:

< **net** | **host** > *name1* **gateway** *name2* **metric** *value* < **passive** | **active** >

The **net** or **host** keyword indicates if the route is to a network or specific host.

Name1 is the name of the destination network or host. This may be a symbolic name located in */etc/networks* or */etc/hosts*, or an Internet address specified in "dot" notation; see *inet*(3N).

Name2 is the name or address of the gateway to which messages should be forwarded.

Value is a metric indicating the hop count to the destination host or network.

The keyword **passive** or **active** indicates if the gateway should be treated as *passive* or *active* (as described above).

FILES

/etc/gateways for distant gateways

SEE ALSO

Internet Transport Protocols, XSIS 028112, Xerox System Integration Standard. (Sun 800-1066-01)
udp(4P)

BUGS

The kernel's routing tables may not correspond to those of *routed* for short periods of time while processes utilizing existing routes exit; the only remedy for this is to place the routing process in the kernel.

Routed should listen to intelligent interfaces, such as an IMP, and to error protocols, such as ICMP, to gather more information.

NAME

rshd – remote shell server

SYNOPSIS

/etc/in.rshd host.port

DESCRIPTION

Rshd is the server for the *rcmd*(3N) routine and, consequently, for the *rsh*(1C) program. The server provides remote execution facilities with authentication based on privileged port numbers.

Rshd is invoked by *inetd*(8C) each time a shell service is requested, and executes the following protocol:

1) The server checks the client's source port. If the port is not in the range 0-1023, the server aborts the connection. The clients host address (in hex) and port number (in decimal) are the argument passed to *rshd.*

2) The server reads characters from the socket up to a null ('\0') byte. The resultant string is interpreted as an ASCII number, base 10.

3) If the number received in step 1 is non-zero, it is interpreted as the port number of a secondary stream to be used for the **stderr**. A second connection is then created to the specified port on the client's machine. The source port of this second connection is also in the range 0-1023.

4) The server checks the client's source address. If the address is associated with a host for which no corresponding entry exists in the host name data base (see *hosts*(5)), the server aborts the connection.

5) A null terminated user name of at most 16 characters is retrieved on the initial socket. This user name is interpreted as a user identity to use on the **server**'s machine.

6) A null terminated user name of at most 16 characters is retrieved on the initial socket. This user name is interpreted as the user identity on the **client**'s machine.

7) A null terminated command to be passed to a shell is retrieved on the initial socket. The length of the command is limited by the upper bound on the size of the system's argument list.

8) *Rshd* then validates the user according to the following steps. The remote user name is looked up in the password file and a *chdir* is performed to the user's home directory. If either the lookup or *chdir* fail, the connection is terminated. If the user is not the super-user, (user id 0), the file */etc/hosts.equiv* is consulted for a list of hosts considered "equivalent". If the client's host name is present in this file, the authentication is considered successful. If the lookup fails, or the user is the super-user, then the file *.rhosts* in the home directory of the remote user is checked for the machine name and identity of the user on the client's machine. If this lookup fails, the connection is terminated.

9) A null byte is returned on the connection associated with the **stderr** and the command line is passed to the normal login shell of the user. The shell inherits the network connections established by *rshd.*

DIAGNOSTICS

All diagnostic messages are returned on the connection associated with the **stderr**, after which any network connections are closed. An error is indicated by a leading byte with a value of 1 (0 is returned in step 9 above upon successful completion of all the steps prior to the command execution).

"locuser too long"
The name of the user on the client's machine is longer than 16 characters.

"remuser too long"
The name of the user on the remote machine is longer than 16 characters.

"command too long "
The command line passed exceeds the size of the argument list (as configured into the system).

"Hostname for your address unknown."
No entry in the host name database existed for the client's machine.

"Login incorrect."
No password file entry for the user name existed.

"No remote directory."
The *chdir* command to the home directory failed.

"Permission denied."
The authentication procedure described above failed.

"Can't make pipe."
The pipe needed for the **stderr**, wasn't created.

"Try again."
A *fork* by the server failed.

"/bin/sh: ..."
The user's login shell could not be started.

SEE ALSO

rsh(1C), rcmd(3N)

BUGS

The authentication procedure used here assumes the integrity of each client machine and the connecting medium. This is insecure, but is useful in an "open" environment.

A facility to allow all data exchanges to be encrypted should be present.

NAME
rwhod – system status server

SYNOPSIS
/etc/rwhod

DESCRIPTION
Rwhod is the server which maintains the database used by the *rwho*(1C) and *ruptime*(1C) programs. Its operation is predicated on the ability to *broadcast* messages on a network.

Rwhod operates as both a producer and consumer of status information. As a producer of information it periodically queries the state of the system and constructs status messages which are broadcast on a network. As a consumer of information, it listens for other *rwhod* servers' status messages, validating them, then recording them in a collection of files located in the directory */usr/spool/rwho.*

The *rwho* server transmits and receives messages at the port indicated in the "rwho" service specification, see *services*(5). The messages sent and received, are of the form:

```
struct  outmp {
        char    out_line[8];    /* tty name */
        char    out_name[8];    /* user id */
        long    out_time;       /* time on */
};

struct  whod {
        char    wd_vers;
        char    wd_type;
        char    wd_fill[2];
        int     wd_sendtime;
        int     wd_recvtime;
        char    wd_hostname[32];
        int     wd_loadav[3];
        int     wd_boottime;
        struct  whoent {
                struct          outmp we_utmp;
                int             we_idle;
        } wd_we[1024 / sizeof (struct whoent)];
};
```

All fields are converted to network byte order prior to transmission. The load averages are as calculated by the *w*(1) program, and represent load averages over the 5, 10, and 15 minute intervals prior to a server's transmission. The host name included is that returned by the *gethostname*(2) system call. The array at the end of the message contains information about the users logged in to the sending machine. This information includes the contents of the *utmp*(5) entry for each non-idle terminal line and a value indicating the time since a character was last received on the terminal line.

Messages received by the *rwho* server are discarded unless they originated at a *rwho* server's port. In addition, if the host's name, as specified in the message, contains any unprintable ASCII characters, the message is discarded. Valid messages received by *rwhod* are placed in files named *whod.hostname* in the directory */usr/spool/rwho.* These files contain only the most recent message, in the format described above.

Status messages are generated approximately once every 60 seconds. *Rwhod* performs an *nlist*(3) on /vmunix every 10 minutes to guard against the possibility that this file is not the system image currently operating.

SEE ALSO

rwho(1C), ruptime(1C)

BUGS

Should relay status information between networks. People often interpret the server dying as a machine going down.

NAME

sa, accton – system accounting

SYNOPSIS

/usr/etc/sa [**–abcdDfijkKlnrstuv**] [file]

/usr/etc/accton [file]

DESCRIPTION

With an argument naming an existing *file, accton* causes system accounting information for every process executed to be placed at the end of the file. If no argument is given, accounting is turned off.

Sa reports on, cleans up, and generally maintains accounting files.

Sa is able to condense the information in */usr/adm/acct* into a summary file */usr/adm/savacct* which contains a count of the number of times each command was called and the time resources consumed. This condensation is desirable because on a large system */usr/adm/acct* can grow by 500K bytes per day. The summary file is normally read before the accounting file, so the reports include all available information.

If a file name is given as the last argument, that file will be treated as the accounting file; */usr/adm/acct* is the default.

Output fields are labelled: 'cpu' for the sum of user+system time (in minutes), 're' for real time (also in minutes), 'k' for cpu-time averaged core usage (in 1k units), 'avio' for average number of I/O operations per execution. With options fields labelled 'tio' for total I/O operations, 'k*sec' for cpu storage integral (kilo-core seconds), 'u' and 's' for user and system cpu time alone (both in minutes) will sometimes appear.

OPTIONS

a	Place all command names containing unprintable characters and those used only once under the name '***other.'
b	Sort output by sum of user and system time divided by number of calls. Default sort is by sum of user and system times.
c	Besides total user, system, and real time for each command print percentage of total time over all commands.
d	Sort by average number of disk I/O operations.
D	Print and sort by total number of disk I/O operations.
f	Force no interactive threshold compression with –v flag.
i	Don't read in summary file.
j	Instead of total minutes time for each category, give seconds per call.
k	Sort by cpu-time average memory usage.
K	Print and sort by cpu-storage integral.
l	Separate system and user time; normally they are combined.
m	Print number of processes and number of CPU minutes for each user.
n	Sort by number of calls.
r	Reverse order of sort.
s	Merge accounting file into summary file */usr/adm/savacct* when done.
t	For each command report ratio of real time to the sum of user and system times.
u	Superseding all other flags, print for each record in the accounting file the user ID and command name.

v	Followed by a number *n*, types the name of each command used *n* times or fewer. Await a reply from the terminal; if it begins with 'y', add the command to the category '**junk**.' This is used to strip out garbage.

FILES

/usr/adm/acct	raw accounting

SEE ALSO

ac(8), acct(2), acct(5), savacct(5), usracct(5)

NAME

savecore – save a core dump of the operating system

SYNOPSIS

/usr/etc/savecore *dirname* [*system*]

DESCRIPTION

Savecore is meant to be called near the end of the */etc/rc* file after the system boots. *Savecore*'s function is to save the core dump of the system (assuming one was made) and to write a reboot message in the shutdown log.

Savecore checks the core dump to be certain it corresponds with the current running version of the operating system. If it does, *savecore* saves the core image in the file *dirname*/vmcore.n and its brother, the namelist, *dirname*/vmunix.n The trailing *.n* in the pathnames is replaced by a number which grows every time *savecore* is run in that directory.

Before *savecore* writes out a core image, it reads a number from the file *dirname*/minfree. If there less free space on the filesystem which contains *dirname* than the number obtained from the minfree file, the core dump is not done. If the *minfree* file does not exist, *savecore* always writes out the core file (assuming that a core dump was taken).

Savecore also writes a reboot message in the shut down log. If the system crashed as a result of a panic, *savecore* records the panic string in the shut down log too.

If the core dump was from a system other than /vmunix, the name of that system must be supplied as *sysname*.

FILES

/usr/adm/shutdownlog	shut down log
/usr/crash/bounds	number to assign to the core images
/vmunix	current UNIX

BUGS

Can be fooled into thinking a core dump is the wrong size.

NAME

sendmail – send mail over the internet

SYNOPSIS

/usr/lib/sendmail [**–ba**] [**–bd**] [**–bi**] [**–bm**] [**–bs**] [**–bt**] [**–bv**] [**–bz**] [**–C***file*] [**–d***X*] [**–F***fullname*] [**–f***name*] [**–h***N*] [**–n**] [**–o***x value*] [**–q**[*time*]] [**–r***name*] [**–t**] [**–v**] [address **...**]

DESCRIPTION

Sendmail sends a message to one or more people, routing the message over whatever networks are necessary. *Sendmail* does internetwork forwarding as necessary to deliver the message to the correct place.

Sendmail is not intended as a user interface routine; other programs provide user-friendly front ends; *sendmail* is used only to deliver pre-formatted messages.

With no flags, *sendmail* reads its standard input up to a control-D or a line with a single dot and sends a copy of the letter found there to all of the addresses listed. It determines the network to use based on the syntax and contents of the addresses.

Local addresses are looked up in a file and aliased appropriately. Aliasing can be prevented by preceding the address with a backslash. Normally the sender is not included in any alias expansions, for example, if 'john' sends to 'group', and 'group' includes 'john' in the expansion, then the letter will not be delivered to 'john'.

OPTIONS

–ba	Go into ARPANET mode. All input lines must end with a CR-LF, and all messages will be generated with a CR-LF at the end. Also, the "From:" and "Sender:" fields are examined for the name of the sender.
–bd	Run as a daemon. This requires Berkeley IPC.
–bi	Initialize the alias database.
–bm	Deliver mail in the usual way (default).
–bs	Use the SMTP protocol as described in RFC821. This flag implies all the operations of the **–ba** flag that are compatible with SMTP.
–bt	Run in address test mode. This mode reads addresses and shows the steps in parsing; it is used for debugging configuration tables.
–bv	Verify names only – do not try to collect or deliver a message. Verify mode is normally used for validating users or mailing lists.
–bz	Create the configuration freeze file.
–C*file*	Use alternate configuration file.
–d*X*	Set debugging value to *X*.
–F*fullname*	Set the full name of the sender.
–f*name*	Sets the name of the "from" person (that is, the sender of the mail). **–f** can only be used by the special users *root*, *daemon*, and *network*, or if the person you are trying to become is the same as the person you are.
–h*N*	Set the hop count to *N*. The hop count is incremented every time the mail is processed. When it reaches a limit, the mail is returned with an error message, the victim of an aliasing loop.
–n	Don't do aliasing.
–o*x value*	Set option *x* to the specified *value*. Options are described below.
–q[*time*]	Processed saved messages in the queue at given intervals. If is omitted, process the queue once. is given as a tagged number, with 's' being seconds, 'm'

	being minutes, 'h' being hours, 'd' being days, and 'w' being weeks. For example, "–q1h30m" or "–q90m" would both set the timeout to one hour thirty minutes.
–r*name*	An alternate and obsolete form of the –f flag.
–t	Read message for recipients. To:, Cc:, and Bcc: lines will be scanned for people to send to. The Bcc: line will be deleted before transmission. Any addresses in the argument list will be suppressed.
–v	Go into verbose mode. Alias expansions will be announced, etc.

PROCESSING OPTIONS

There are also a number of processing options that may be set. Normally these will only be used by a system administrator. Options may be set either on the command line using the **–o** flag or in the configuration file. These are described in detail in the *Installation and Operation Guide*. The options are:

A*file*	Use alternate alias file.
c	On mailers that are considered "expensive" to connect to, don't initiate immediate connection. This requires queueing.
d*x*	Set the delivery mode to *x*. Delivery modes are 'i' for interactive (synchronous) delivery, 'b' for background (asynchronous) delivery, and 'q' for queue only – that is, actual delivery is done the next time the queue is run.
D	Try to automatically rebuild the alias database if necessary.
e*x*	Set error processing to mode *x*. Valid modes are 'm' to mail back the error message, 'w' to "write" back the error message (or mail it back if the sender is not logged in), 'p' to print the errors on the terminal (default), 'q' to throw away error messages (only exit status is returned), and 'e' to do special processing for the BerkNet. If the text of the message is not mailed back by modes 'm' or 'w' and if the sender is local to this machine, a copy of the message is appended to the file "dead.letter" in the sender's home directory.
F*mode*	The mode to use when creating temporary files.
f	Save UNIX-style From lines at the front of messages.
g*N*	The default group id to use when calling mailers.
H*file*	The SMTP help file.
i	Do not take dots on a line by themselves as a message terminator.
L*n*	The log level.
m	Send to "me" (the sender) also if I am in an alias expansion.
o	If set, this message may have old style headers. If not set, this message is guaranteed to have new style headers (that is, commas instead of spaces between addresses). If set, an adaptive algorithm is used that will correctly determine the header format in most cases.
Q*queuedir*	Select the directory in which to queue messages.
r*timeout*	The timeout on reads; if none is set, *sendmail* will wait forever for a mailer.
S*file*	Save statistics in the named file.
s	Always instantiate the queue file, even under circumstances where it is not strictly necessary.
T*time*	Set the timeout on messages in the queue to the specified time. After sitting in the queue for this amount of time, they will be returned to the sender. The

	default is three days.
t*stz,dtz*	Set the name of the time zone.
u*N*	Set the default user id for mailers.

If the first character of the user name is a vertical bar, the rest of the user name is used as the name of a program to pipe the mail to. It may be necessary to quote the name of the user to keep *sendmail* from suppressing the blanks from between arguments.

Sendmail returns an exit status describing what it did. The codes are defined in <*sysexits.h*>

EX_OK	Successful completion on all addresses.
EX_NOUSER	User name not recognized.
EX_UNAVAILABLE	Catchall meaning necessary resources were not available.
EX_SYNTAX	Syntax error in address.
EX_SOFTWARE	Internal software error, including bad arguments.
EX_OSERR	Temporary operating system error, such as "cannot fork".
EX_NOHOST	Host name not recognized.
EX_TEMPFAIL	Message could not be sent immediately, but was queued.

If invoked as *newaliases, sendmail* rebuilds the alias database. If invoked as *mailq, sendmail* prints the contents of the mail queue.

FILES

Except for */usr/lib/sendmail.cf,* these pathnames are all specified in */usr/lib/sendmail.cf.* Thus, these values are only approximations.

/usr/lib/aliases	raw data for alias names
/usr/lib/aliases.pag	
/usr/lib/aliases.dir	data base of alias names
/usr/lib/sendmail.cf	configuration file
/usr/lib/sendmail.fc	frozen configuration
/usr/lib/sendmail.hf	help file
/usr/lib/sendmail.st	collected statistics
/usr/bin/uux	to deliver uucp mail
/usr/net/bin/v6mail	to deliver local mail
/usr/net/bin/sendberkmail	to deliver Berknet mail
/usr/lib/mailers/arpa	to deliver ARPANET mail
/usr/spool/mqueue/*	temp files

SEE ALSO

biff(1), binmail(1), mail(1), aliases(5),
DARPA Internet Request For Comments RFC819, RFC821, RFC822,
Sendmail - An Internetwork Mail Router,
Sendmail Installation and Operation Guide.

BUGS

Sendmail converts blanks in addresses to dots. This is incorrect according to the old ARPANET mail protocol RFC733 (NIC 41952), but is consistent with the new protocols (RFC822).—

NAME

shutdown – close down the system at a given time

SYNOPSIS

/etc/shutdown [**–k**] [**–r**] [**–h**] time [warning-message ...]

DESCRIPTION

Shutdown provides an automated shutdown procedure which a super-user can use to notify users nicely when the system is shutting down, saving them from system administrators, hackers, and gurus, who would otherwise not bother with niceties.

Time is the time at which *shutdown* will bring the system down and may be the word **now** (indicating an immediate shutdown) or specify a future time in one of two formats: +number and hour:min. The first form brings the system down in *number* minutes and the second brings the system down at the time of day indicated (as a 24–hour clock).

At intervals which get closer together as apocalypse approaches, warning messages are displayed at the terminals of all users on the system. Five minutes before shutdown, or immediately if shutdown is in less than 5 minutes, logins are disabled by creating */etc/nologin* and writing a message there. If this file exists when a user attempts to log in, *login*(1) prints its contents and exits. The file is removed just before *shutdown* exits.

At shutdown time a message is written in the file */usr/adm/shutdownlog,* containing the time of shutdown, who ran shutdown and the reason. Then a terminate signal is sent at *init* to bring the system down to single-user state.

The time of the shutdown and the warning message are placed in */etc/nologin* and should be used to inform the users about when the system will be back up and why it is going down (or anything else).

OPTIONS

As an alternative to the above procedure, these options can be specified:

–r	Execute *reboot*(8).
–h	Execute *halt*(8).
–k	If it isn't obvious, **–k** is to make people *think* the system is going down!

FILES

/etc/nologin	tells login not to let anyone log in
/usr/adm/shutdownlog	log file for succesful shutdowns.

SEE ALSO

login(1), reboot(8)

BUGS

Only allows you to kill the system between now and 23:59 if you use the absolute time for shutdown.

NAME

sticky – executable files with persistent text

DESCRIPTION

While the 'sticky bit', mode 01000 (see *chmod*(2)), is set on a sharable executable file, the text of that file will not be removed from the system swap area. Thus the file does not have to be fetched from the file system upon each execution. As long as a copy remains in the swap area, the original text cannot be overwritten in the file system, nor can the file be deleted. Directory entries can be removed so long as one link remains.

Sharable files are made by the **-z** option of *ld*(1).

To replace a sticky file that has been used do: (1) Clear the sticky bit with *chmod*(1). (2) Execute the old program to flush the swapped copy. This can be done safely even if others are using it. (3) Overwrite the sticky file. If the file is being executed by any process, writing will be prevented; it suffices to simply remove the file and then rewrite it, being careful to reset the owner and mode with *chmod* and *chown*(2). (4) Set the sticky bit again.

Only the super-user can set the sticky bit.

NAME

swapon – specify additional device for paging and swapping

SYNOPSIS

/usr/etc/swapon -a
/usr/etc/swapon name ...

DESCRIPTION

Swapon is used to specify additional devices on which paging and swapping are to take place. The system begins by swapping and paging on only a single device so that only one disk is required at bootstrap time. Calls to *swapon* normally occur in the system multi-user initialization file */etc/rc* making all swap devices available, so that the paging and swapping activity is interleaved across several devices.

Normally, the **-a** argument is given, causing all devices marked as "sw" swap devices in **/etc/fstab** to be made available.

The second form gives individual block devices as given in the system swap configuration table. The call makes only this space available to the system for swap allocation.

SEE ALSO

swapon(2), init(8)

FILES

/dev/[ru][pk]?b normal paging devices

BUGS

There is no way to stop paging and swapping on a device. It is therefore not possible to make use of devices which may be dismounted during system operation.

NAME

sync – update the super block

SYNOPSIS

sync

DESCRIPTION

Sync executes the *sync* system primitive. *Sync* can be called to insure all disk writes have been completed before the processor is halted in a way not suitably done by *reboot*(8) or *halt*(8).

See *sync*(2) for details on the system primitive.

SEE ALSO

sync(2), fsync(2), halt(8), reboot(8)

NAME

syslog – log systems messages

SYNOPSIS

/etc/syslog [**-m***N*] [**-f***name*] [**-d**] [**-p***port*]

DESCRIPTION

Syslog reads a datagram socket and logs each line it reads into a set of files described by the configuration file /etc/syslog.conf. *Syslog* configures when it starts up and whenever it receives a hangup signal.

Each message is one line. A message can contain a priority code, marked by a digit in angle braces at the beginning of the line. Priorities are defined in <syslog.h>, as defined in the list below. *LOG_ALERT* is prioity 1 (the highest priority) while *LOG_DEBUG* is priority 9 (the lowest priority).

LOG_ALERT	this priority should essentially never be used. It applies only to messages that are so important that every user should be aware of them, for example, a serious hardware failure.
LOG_SALERT	messages of this priority should be issued only when immediate attention is needed by a qualified system person, for example, when some valuable system resource disappears. They get sent to a list of system people.
LOG_EMERG	Emergency messages are not sent to users, but represent major conditions. An example might be hard disk failures. These could be logged in a separate file so that critical conditions could be easily scanned.
LOG_ERR	these represent error conditions, such as soft disk failures, etc.
LOG_CRIT	such messages contain critical information, but which can not be classed as errors, for example, 'su' attempts. Messages of this priority and higher are typically logged on the system console.
LOG_WARNING	issued when an abnormal condition has been detected, but recovery can take place.
LOG_NOTICE	something that falls in the class of "important information"; this class is informational but important enough that you don't want to throw it away casually. Messages without any priority assigned to them are typically mapped into this priority.
LOG_INFO	information level messages. These messages could be thrown away without problems, but should be included if you want to keep a close watch on your system.
LOG_DEBUG	it may be useful to log certain debugging information. Normally this will be thrown away.

It is expected that the kernel will not log anything below LOG_ERR priority.

The configuration file is in two sections separated by a blank line. The first section defines files that *syslog* will log into. Each line contains a single digit which defines the lowest priority (highest numbered priority) that this file will receive, an optional asterisk which guarantees that something gets output at least every 20 minutes, and a pathname. The second part of the file contains a list of users that will be informed on SALERT level messages. For example, the configuration file:

```
5*/dev/tty8
8/usr/spool/adm/syslog
3/usr/adm/critical
```

```
eric
kridle
kalash
```

logs all messages of priority 5 or higher onto the system console, including timing marks every 20 minutes; all messages of priority 8 or higher into the file /usr/spool/adm/syslog; and all messages of priority 3 or higher into /usr/adm/critical. The users 'eric', 'kridle', and 'kalash' will be informed on any subalert messages.

OPTIONS

−m *N* Set the mark interval to *N* (default 20 minutes).

−f *name*
Specify an alternate configuration file.

−d Turn on debugging (if compiled in).

−p *port* Port number where *syslog* listens for incoming datagrams. The default port is defined in the 'syslog/udp' entry in the */etc/services* file.

To bring *syslog* down, it should be sent a terminate signal. It logs that it is going down and then waits approximately 30 seconds for any additional messages to come in.

There are some special messages that cause control functions. '<*>N' sets the default message priority to *N*. '<$>' causes *syslog* to reconfigure (equivalent to a hangup signal). This can be used in a shell file run automatically early in the morning to truncate the log.

Syslog creates the file /etc/syslog.pid if possible containing a single line with its process id. This can be used to kill or reconfigure *syslog*.

FILES

/etc/syslog.conf – the configuration file
/etc/syslog.pid – the process id
/etc/services – to find the *syslog* server's port number.

BUGS

LOG_ALERT and LOG_SUBALERT messages should only be allowed to privileged programs.

Actually, *syslog* is not clever enough to deal with kernel error messages in the current implementation.

SEE ALSO

syslog(3)

NAME

telnetd – DARPA TELNET protocol server

SYNOPSIS

/usr/etc/in.telnetd host.port

DESCRIPTION

Telnetd is a server which supports the DARPA standard TELNET virtual terminal protocol. The TELNET server is invoked by *inetd*(8C) each time there is a connection to the telnet service; see *services*(5).

Telnetd operates by allocating a pseudo-terminal device (see *pty*(4)) for a client, then creating a login process which has the slave side of the pseudo-terminal as **stdin**, **stdout**, and **stderr**. *Telnetd* manipulates the master side of the pseudo terminal, implementing the TELNET protocol and passing characters between the client and login process.

When a TELNET session is started up, *telnetd* sends a TELNET option to the client side indicating a willingness to do "remote echo" of characters. The pseudo terminal allocated to the client is configured to operate in "cooked" mode, and with XTABS and CRMOD enabled (see *tty*(4)). Aside from this initial setup, the only mode changes *telnetd* will carry out are those required for echoing characters at the client side of the connection.

Telnetd supports binary mode, and most of the common TELNET options, but does not, for instance, support timing marks.

SEE ALSO

telnet(1C)

BUGS

A complete list of the options supported should be given here.

NAME

tftpd – DARPA Trivial File Transfer Protocol server

SYNOPSIS

/usr/etc/tftpd [**–d**] [*port*]

DESCRIPTION

Tftpd is a server which supports the DARPA Trivial File Transfer Protocol. The TFTP server operates at the port indicated in the "tftp" service description; see *services*(5), and is invoked each time a datagram reaches this port by the internet server *inetd*(8C).

Due to the lack of authentication information, *tftpd* will allow only publicly readable files to be accessed.

SEE ALSO

tftp(1C)

BUGS

This server is known only to be self consistent (i.e. it operates with the user TFTP program, *tftp*(1C)).

NAME

trpt – transliterate protocol trace

SYNOPSIS

/usr/etc/trpt [**–a**] [**–s**] [**–t**] [**–j**] [**–p***hex-address*] [system [core]]

DESCRIPTION

Trpt interrogates the buffer of TCP trace records created when a socket is marked for 'debugging' (see *setsockopt*(2)), and prints a readable description of these records. When no options are supplied, *trpt* prints all the trace records found in the system grouped according to TCP connection protocol control block (PCB).

OPTIONS

–s Print a detailed description of the packet sequencing information, in addition to the normal output.

–t Print the values for all timers at each point in the trace, in addition to the normal output.

–j Just give a list of the protocol control block addresses for which there are trace records.

–p*hex-address*
Show only trace records associated with the protocol control block who's address follows.

–a in addition to the normal output, print the values of the source and destination addresses for each packet recorded.

The recommended use of *trpt* is as follows. Isolate the problem and enable debugging on the socket(s) involved in the connection. Find the address of the protocol control blocks associated with the sockets using the **–A** option to *netstat*(8). Then run *trpt* with the **–p** option, supplying the associated protocol control block addresses. If there are many sockets using the debugging option, the **–j** option may be useful in checking to see if any trace records are present for the socket in question.

If debugging is being performed on a system or core file other than the default, the last two arguments may be used to supplant the defaults.

FILES

/vmunix
/dev/kmem

SEE ALSO

setsockopt(2), netstat(8)

DIAGNOSTICS

'no namelist' when the system image doesn't contain the proper symbols to find the trace buffer; others which should be self explanatory.

BUGS

Should also print the data for each input or output, but this is not saved in the race record.

The output format is inscrutable, and should be described here.

NAME

tunefs – tune up an existing file system

SYNOPSIS

/etc/tunefs *tuneup-options special|filesys*

DESCRIPTION

Tunefs is designed to change the dynamic parameters of a file system which affect the layout policies. The parameters which are to be changed are indicated by the flags given below:

–a maxcontig
: This specifies the maximum number of contiguous blocks that will be laid out before forcing a rotational delay (see –d below). The default value is one, since most device drivers require an interrupt per disk transfer. Device drivers that can chain several buffers together in a single transfer should set this to the maximum chain length.

–d rotdelay
: This specifies the expected time (in milliseconds) to service a transfer completion interrupt and initiate a new transfer on the same disk. It is used to decide how much rotational spacing to place between successive blocks in a file.

–e maxbpg
: This indicates the maximum number of blocks any single file can allocate out of a cylinder group before it is forced to begin allocating blocks from another cylinder group. Typically this value is set to about one quarter of the total blocks in a cylinder group. The intent is to prevent any single file from using up all the blocks in a single cylinder group, thus degrading access times for all files subsequently allocated in that cylinder group. The effect of this limit is to cause big files to do long seeks more frequently than if they were allowed to allocate all the blocks in a cylinder group before seeking elsewhere. For file systems with exclusively large files, this parameter should be set higher.

–m minfree
: This value specifies the percentage of space held back from normal users; the minimum free space threshold. The default value used is 10%. This value can be set to zero, however up to a factor of three in throughput will be lost over the performance obtained at a 10% threshold. Note that if the value is raised above the current usage level, users will be unable to allocate files until enough files have been deleted to get under the higher threshold.

SEE ALSO

fs(5), newfs(8), mkfs(8)

McKusick, Joy, Leffler; "A Fast File System for Unix", Computer Systems Research Group, Dept of EECS, Berkeley, CA 94720; TR #7, September 1982.

BUGS

This program should work on mounted and active file systems. Because the super-block is not kept in the buffer cache, the program will only take effect if it is run on dismounted file systems. (if run on the root file system, the system must be rebooted)

NAME
update – periodically update the super block

SYNOPSIS
/etc/update

DESCRIPTION
Update is a program that executes the *sync*(2) primitive every 30 seconds. This insures that the file system is fairly up to date in case of a crash. This command should not be executed directly, but should be executed out of the initialization shell command file.

SEE ALSO
sync(2), sync(8), init(8)

NAME

uuclean – uucp spool directory clean-up

SYNOPSIS

/usr/lib/uucp/uuclean [**–p***pre*] [**–n***time*] [**–m**]

DESCRIPTION

Uuclean scans the spool directory for files with the specified prefix and deletes all those which are older than the specified number of hours.

OPTIONS

–p*pre*	Scan for files with *pre* as the file prefix. Up to 10 **–p** arguments may be specified. A **–p** without any *pre* following deletes all files older than the specified time.
–n*time*	Files whose age is more than *time* hours are deleted if the prefix test is satisfied (default time is 72 hours).
–m	Send mail to the owner of the file when it is deleted.

Uuclean will typically be started by *cron*(8).

FILES

/usr/lib/uucp	directory with commands used by uuclean internally
/usr/lib/uucp/spool	spool directory

SEE ALSO

uucp(1C), uux(1C)

NAME

vipw – edit the password file

SYNOPSIS

/etc/vipw

DESCRIPTION

Vipw edits the password file while setting the appropriate locks, and does any necessary processing after the password file is unlocked. If the password file is already being edited, then you will be told to try again later. The *vi* editor will be used unless the environment variable EDITOR indicates an alternate editor. *Vipw* performs a number of consistency checks on the password entry for *root*, and will not allow a password file with a "mangled" root entry to be installed.

SEE ALSO

chsh(1), passwd(1), passwd(5), adduser(8)

FILES

/etc/ptmp

NAME

vmstat – report virtual memory statistics

SYNOPSIS

vmstat [**–fs**] [interval [count]]

DESCRIPTION

Vmstat delves into the system and normally reports certain statistics kept about process, virtual memory, disk, trap and cpu activity.

If none of these options are given, *vmstat* will report in the first line a summary of the virtual memory activity since the system has been booted. If *interval* is specified, then successive lines are summaries over the last *interval* seconds. "vmstat 5" will print what the system is doing every five seconds; this is a good choice of printing interval since this is how often some of the statistics are sampled in the system; others vary every second, running the output for a while will make it apparent which are recomputed every second. If a *count* is given, the statistics are repeated *count* times. The format fields are:

Procs: information about numbers of processes in various states.

r	in run queue
b	blocked for resources (i/o, paging, etc.)
w	runnable or short sleeper (< 20 secs) but swapped

Memory: information about the usage of virtual and real memory. Virtual pages are considered active if they belong to processes which are running or have run in the last 20 seconds. A "page" here is 2048 bytes.

avm	active virtual pages
fre	size of the free list

Page: information about page faults and paging activity. These are averaged each five seconds, and given in units per second.

re	page reclaims (simulating reference bits)
pi	pages paged in
po	pages paged out
fr	pages freed per second
de	anticipated short term memory shortfall
sr	pages scanned by clock algorithm, per-second

up/hp/rk: Disk operations per second (this field is system dependent). Typically paging will be split across several of the available drives. The number under each of these is the unit number.

Faults: trap/interrupt rate averages per second over last 5 seconds.

in	(non clock) device interrupts per second
sy	system calls per second
cs	cpu context switch rate (switches/sec)

Cpu: breakdown of percentage usage of CPU time

us	user time for normal and low priority processes
sy	system time
id	cpu idle

OPTIONS

–f Report on the number of *forks* and *vforks* since system startup and the number of pages of virtual memory involved in each kind of fork.

–s Display the contents of the *sum* structure, giving the total number of several kinds of paging related events which have occurred since boot.

FILES

/dev/kmem, /vmunix

BUGS

There should be a screen oriented program which combines *vmstat* and *ps*(1) in real time as well as reporting on other system activity.

Building UNIX† Systems with Config

September, 1983

ABSTRACT

This document describes the use of the *config*(8) program to configure and create bootable UNIX kernels. It discusses the structure of kernel configuration files, and how to configure kernels with non-standard hardware configurations. An appendix contains a summary of the rules used by the kernel in calculating the size of kernel data structures, and also indicates some of the standard kernel size limitations (and how to change them).

†UNIX is a Trademark of Bell Laboratories.
‡ This document is based on *Building 4.2BSD UNIX Systems with Config* by Samuel J. Leffler, Computer Systems Research Group, U.C. Berkeley.

1. INTRODUCTION

Config is a tool used in building new Sun UNIX kernel images. It takes a file describing a kernel's tunable parameters and hardware support, and generates a collection of files which are then used to build a copy of UNIX appropriate to that configuration. *Config* simplifies kernel maintenance by isolating configuration dependencies in a single, easy to understand, file.

Section 2 of this document describes building kernels. Section 3 describes the syntax of the configuration file.

Appendix A gives the grammar for config files. Appendix B describes defaulting rules used during the bootstrap process. Appendix C give sample configuration files. A final Appendix D gives the default rules used in calculating sizes for the most important kernel data structures. It also lists kernel data structure size limitations and indicates how you can modify these limits.

If you are planning to write a device-driver for UNIX, consult the *Device Driver Manual* in the Sun *System Internals Manual* for more information.

2. KERNEL BUILDING PROCESS

2.1. Quick summary

Assuming the kernel source is located in the */sys* directory, there are seven steps in building, installing and running a new kernel:

1) Choose a name for your configuration of the system; for example GAIA. Note that — by convention — the name should be in all uppercase letters.

2) In the */sys/conf* directory, create the config file for the system and the directory to contain the kernel image:

```
# cp GENERIC GAIA
# mkdir ../GAIA
```

3) Edit GAIA to reflect your system, e.g. change the timezone from 8 to 5 (Pacific to Eastern time), delete devices and options which you don't need from the description, and add devices which you have which are not in the configuration.

4) Run config:

```
# /usr/etc/config GAIA
```

5) Build the new kernel:

```
# cd ../GAIA
# make depend
# make
 ... lots of output ...
```

6) Install the new kernel and try it out

```
# cp vmunix /newvmunix
# /etc/halt
>b newvmunix – s
```

7) If the new kernel appears to work, save the old kernel and install the new one in **/vmunix.**

```
# cd /
# mv vmunix ovmunix
# mv newvmunix vmunix
# /etc/reboot
```

Steps 1 and 2 are usually done only once. When a kernel configuration changes, it usually suffices to just run *config* on the modified configuration file, rebuild the source code dependencies, and remake the kernel.†

2.2. Creating a configuration file

Configuration files normally reside in the subdirectory */sys/conf*. A configuration file is most easily constructed by copying an existing configuration file and modifying it. This distribution contains a GENERIC configuration file which you can edit to suit your particular system configuration.

† In rare cases invisible configuration dependencies may exist, requiring the kernel to be rebuilt from scratch by removing the kernel building directory and starting again from step 2. This will be discussed later.

The configuration file must have the same name as the directory in which the configured kernel is to be built. Further, *config* assumes this directory is located in the parent directory of the directory in which it is run. For example, the generic kernel has a configuration file */sys/conf/GENERIC* and an accompanying directory named */sys/GENERIC*. In general it is unwise to move your configuration directories out of */sys*, as most of the kernel code and the files created by *config* use pathnames of the form "../". If you are running out of space on the file system where the configuration directories are located, there is a mechanism (for source distributions only) for sharing relocatable object files between kernels. This is described later.

When building your configuration file, be sure to include the items described in section 3. In particular, the machine type, cpu type, timezone, system identifier, maximum users, and root device must be specified. The specification of the hardware present may take a bit of work, particularly if your hardware is configured at non-standard places (for example, device registers located at funny places or devices not supported by the kernel). If the devices to be configured are not already described in one of the existing configuration files, check the section 4 manual pages in the Sun *System Interface Manual*. For each supported device, the manual page synopsis entry gives a sample configuration line.

Once the configuration file is complete, run it through *config* and look for any errors. Never use a kernel which *config* has complained about; the results are unpredictable. For the most part, *config*'s error diagnostics are self-explanatory. It may be the case that the line numbers given with the error messages are off by one.

A successful run of *config* on your configuration file will generate a number of files in the configuration directory. These files are:

- A file to be used by *make*(1) in compiling and loading the kernel.
- One file for each possible kernel image for your machine which describes where the root and swap devices are located.
- A collection of header files, one per possible device the kernel supports, which define the hardware configured.
- A file containing the i/o configuration tables used by the kernel during its *autoconfiguration* phase.

Unless you suspect that there is a bug in *config*, or are curious about how the kernel's autoconfiguration scheme works, you should never have to look at any of these files.

2.3. Constructing source code dependencies

When *config* is done generating the files needed to compile and link your kernel it will terminate with a message of the form "Don't forget to run make depend". This is a reminder that you should change your working directory to the configuration directory for the kernel just configured, and type "make depend" to build the rules used by *make* to recognize interdependencies in the kernel source code. This will insure that any changes to a piece of the kernel source code will result in the proper modules being compiled the next time *make* is run.

This step is particularly important if your site makes changes to the kernel include files. The rules generated specify which source code files are dependent on which include files. Without these rules, *make* will not recognize when it must rebuild modules due to a kernel header file being modified. Note that dependency rules created by this step only reflect directly included files. That is, if file "a" includes another file "b", which includes yet another, say "c", and then "c" is modified, *make* will not recognize that "a" should be recompiled. It is best to keep include file dependencies only one level deep.

2.4. Building the kernel

The makefile constructed by *config* should allow a new kernel to be rebuilt by simply typing "make image-name". For example, if you have named your bootable kernel image "vmunix", then "make vmunix" will generate a bootable image named "vmunix". Alternate kernel image names are used when the root file system location and/or swapping configuration is done in more than one way. The makefile which *config* creates has entry points for each kernel image defined in the configuration file. Thus, if you have configured "vmunix" to be a kernel with the root file system on an "xy" device and "ipvmunix" to be a kernel with the root file system on an "ip" device, then "make vmunix ipvmunix" will generate binary images for both.

Note that the name of a bootable image is different from the system identifier. All bootable images are configured for the same devices; only the information about the root file system and paging devices differ.

The last step in the kernel building process is to rearrange certain commonly used symbols in the symbol table of the kernel image; the makefile generated by *config* does this automatically for you. This is advantageous for programs such as *ps*(1) and *vmstat*(8), which run much faster when the symbols they need are located at the front of the symbol table. Remember also that programs such as *ps* and *vmstat* expect the currently executing kernel to be named "/vmunix". If you install a new kernel and name it something other than "/vmunix", these programs are likely to produce incorrect results.

2.5. Sharing object modules

This is only effective if you have a source distribution.

If you have many kernels which are all built on a single machine there are at least two approaches to saving time in building kernel images. The best way is to have a single kernel image which is run on all machines. This is attractive since it minimizes disk space used and time required to rebuild kernels after making changes. However, it is often the case that one or more systems will require a separately configured kernel image. This may be due to limited memory (building a kernel with many unused device drivers wastes core), or to configuration requirements (one machine may be a development machine where disk quotas are not needed, while another is a production machine where they are), etc. In these cases it is possible for kernels to share relocatable object modules which are not configuration dependent; most of the modules in the directory */sys/sys* are of this sort.

To share object modules, first build a GENERIC kernel. Then, for each kernel, configure as before, but before recompiling and linking, type "make links". This will cause the kernel source to be searched for source modules which are safe to share between kernels and generate symbolic links in the current directory to the appropriate object modules in the directory *../GENERIC*. A shell script, "makelinks" is generated with this request and may be checked for correctness. The file *sys/conf/defines* contains a list of symbols which we believe are safe to ignore when checking the source code for modules which may be shared. Note that this list includes the definitions used to conditionally compile in the virtual memory tracing facilities, and the trace point support used only rarely. It may be necessary to modify this list to reflect local needs. Also, as described previously, interdependencies which are not directly visible in the source code are not caught. Thus if you place per-system dependencies in an include file, they will not be recognized.

2.6. Building profiled kernels

This is effective with source distributions only.

It is simple to configure a kernel which will automatically collect profiling information as it operates. The profiling data may be collected with *kgmon*(8) and processed with *gprof*(1) to obtain information regarding the kernel's operation. Profiled kernels maintain histograms of the program counter as well as the number of invocations of each routine. The *gprof*(1) command will generate a dynamic call graph of the executing kernel and propagate time spent in each routine along the arcs of the call graph.

To configure a profiled kernel, use the **− p** option with *config*. A profiled kernel is about 5-10% larger in its text space due to the calls to count the subroutine invocations. When the kernel executes, the profiling data is stored in a buffer which is 1.2 times the size of the text space. The overhead for running a profiled kernel varies; under normal load we see anywhere from 5-25% of the kernel time spent in the profiling code.

Note that kernels configured for profiling should not be shared as described above unless all the other shared kernels are also to be profiled.

2.7. Configuring systems without source

Object only releases have binaries for standard system modules in the directory */sys/OBJ.* Using these binaries you can create new configurations and add new device drivers to the kernel. The following lines from the GENERIC config file must be in every config file for object-only distributions:

```
machine sun
cpu     "SUN2"
options INET

pseudo-device   inet
pseudo-device   ether
pseudo-device   loop
controller      mb0 at nexus ?
```

If you include these lines you can make any changes you wish to the configuration file, provided you do not configure in more devices of a particular type than are allowed by the distributed object code in *../OBJ.* Attempting to do so will not be detected and may cause the kernel to appear to work but have only occasional failures. Double check the *.h* files in *../OBJ* if you change the number of devices configured for any standard drivers.

2.8. Adding new device drivers

New device drivers require entries in the files */sys/sun/conf.c*, */sys/conf/files.sun*, and possibly */sys/sun/swapgeneric.c* and *sun/devices.sun.* New devices also require one or more new special files to be added to the */dev* directory. See the *Device Driver Manual* in the *System Internals Manual for the Sun UNIX System.*

3. CONFIGURATION FILE SYNTAX

In this section we consider the specific rules used in writing a configuration file. Appendix A gives a complete grammar for the input language; use it if you have problems with syntax errors.

A configuration file has three logical parts:

- configuration parameters global to all kernel images specified in the configuration file,
- parameters specific to each kernel image to be generated, and
- device specifications.

3.1. Global configuration parameters

The global configuration parameters are the type of machine, cpu types, options, timezone, system identifier, and maximum users. Each is specified with a separate line in the configuration file.

machine *type*

The system is to run on the machine type specified. No more than one machine type can appear in the configuration file. The appropriate line for the sun is **machine sun**.

cpu "*type*"

This system is to run on the cpu type specified. The appropriate line is **cpu "SUN2"**.

options *optionlist*

Compile the listed optional code into the system. Options in this list are separated by commas. Possible options are listed at the top of the generic makefile. A line of the form "options FUNNY,HAHA" generates global "#define"s – DFUNNY – DHAHA in the resultant makefile. An option may be given a value by following its name with "=", then the value enclosed in (double) quotes.

timezone *number* [**dst** [*number*]]

Specifies the timezone you are in. This is measured in the number of hours your timezone is west of GMT. EST is 5 hours west of GMT, PST is 8. Negative numbers indicate hours east of GMT. If you specify **dst**, the kernel will operate under daylight savings time. An optional integer or floating point number may be included to specify a particular daylight saving time correction algorithm; the default value is 1, indicating the United States. Other values are: 2 (Australian style), 3 (Western European), 4 (Middle European), and 5 (Eastern European). See *gettimeofday*(2) and *ctime*(3) for more information.

ident *name*

Gives the system identifier – a name for the machine or machines to run this kernel.

maxusers *number*

The maximum expected number of simultaneously active users on this kernel is *number*. This number is used to size several kernel data structures. Typical values are 2 for 1 Megabyte memory single-user Sun Workstation, 4 for Sun Workstations with 2 Megabytes, and 8 for Sun servers.

3.2. Kernel image parameters

Multiple bootable images may be specified in a single configuration file. The kernels will have the same global configuration parameters and devices, but the location of the root file system and other kernel-specific devices may be different. A kernel image is specified with a "config" line:

config *kername config-clauses*

The *kername* field is the name given to the loaded kernel image; the standard kernel image is "vmunix". The configuration clauses are one or more specifications indicating where the root file system is located, how many paging devices there are and where they go.

A configuration clause is one of the following

root [**on**] *root-device*
swap [**on**] *swap-device* [**and** *swap-device*]*
dumps [**on**] *dump-device*
args [**on**] *arg-device*

(the "on" is optional.) Multiple configuration clauses are separated by white space; *config* allows specifications to be continued across multiple lines by beginning the continuation line with a tab character. The "root" clause specifies where the root file system is located, the "swap" clause indicates swapping and paging area(s), and the "dumps" clause can be used to force crash dumps to be taken on a particular device, and the "args" clause can be used to specify that argument list processing for *execve* should be done on a particular disk.

The device names supplied in the clauses may be fully specified — as a device, unit, and file system partition — or underspecified. If underspecified, *config* will use builtin rules to select default unit numbers and file system partitions. The defaulting rules are dependent on the overall system configuration. For example, the swap area need not be specified at all if the root device is specified; in this case the swap area is placed in the "b" partition of the same disk where the root file system is located. Appendix B contains a complete list of the defaulting rules used in selecting devices.

The device names are translated to the appropriate major and minor device numbers on a per-machine basis. A file, */sys/conf/devices.machine* (where "machine" is the machine type specified in the configuration file), is used to map a device name to its major block device number. The minor device number is calculated using the standard disk partitioning rules.

If the default mapping of device name to major/minor device number is incorrect for your configuration, it can be replaced by an explicit specification of the major/minor device. This is done by substituting

major *x* **minor** *y*

where the device name would normally be found. For example,

config vmunix **root on major** 99 **minor** 1

Normally, the areas configured for swap space are sized by the kernel at boot time. If a non-standard partition size is to be used for one or more swap areas, this can also be specified. To do this, the device name specified for a swap area should have a "size" specification appended. For example,

config vmunix **root on** xy0 **swap on** xy0b **size** 12000

would force swapping to be done in partition "b" of "xy0" and the swap partition size would be set to 1200 sectors. A swap area sized larger than the associated disk partition is trimmed to the partition size.

To create a generic configuration, only the clause "swap generic" should be specified; any extra clauses cause an error. Note that if you use the "swap generic" clause, your system identification line **must** read "ident GENERIC".

3.3. Device specifications

Each device attached to a machine must be specified to *config* so that the system generated will know to probe for it during the autoconfiguration process carried out at boot time. Hardware specified in the configuration need not actually be present on the machine where the generated system is to be run. Only the hardware actually found at boot time will be used by the system.

A device specification takes one of the following forms:

> **controller** *device-name device-info* [*interrupt-spec*]
> **device** *device-name device-info interrupt-spec*
> **disk** *device-name device-info*
> **tape** *device-name device-info*

A ''controller'' is a disk or tape controller. A ''device'' is an autonomous device which connects directly to the Multibus. ''Disk'' and ''tape'' identify disk drives and tape drives connected to a ''controller''.

device-name is a standard UNIX device name (see the section 4 pages of the *System Interface Manual*) concatenated with the *logical* unit number to be assigned the device. The *logical* unit number may be different than the *physical* unit number indicated on the front of something like a disk; the *logical* unit number is used to refer to the UNIX device, not the physical unit number). The *device-info* clause specifies how the hardware is connected in the interconnection hierarchy. On a Sun Workstation, the following specification should be used:

> **controller** mb0 **at nexus** *?*

The remaining interconnections on the Sun Workstation are:

- a controller may be connected to the Multibus (mb0),
- a disk or tape is always attached to a controller, and
- devices may be attached to controllers or to the Multibus.

For controllers, the control status register must be given explicitly, as well the interrupt priority level of the device. The following lines give an example of each of these interconnections:

> **controller** xyc0 **at** mb0 **csr** 0xee40 **priority** 2
> **disk** xy0 **at** xyc0 **drive** 0
> **device** cg0 **at** mb0 **csr** 0xe8000 **priority** 3

Certain device drivers require extra information passed to them at boot time to tailor their operation to the actual hardware present. For example, the drivers for the terminal multiplexors need to know which lines are attached to modem lines so that no one will be allowed to use them unless a connection is present. For this reason, one last parameter may be specified to a *device*, a *flags* field. It has the syntax

> **flags** *number*

and is usually placed after the *csr* specification. The *number* is passed directly to the associated driver. The manual pages in section 4 should be consulted to determine how each driver uses this value (if at all).

3.4. Pseudo-devices

A number of drivers and software subsystems are treated like device drivers without any associated hardware. To include any of these pieces, a ''pseudo-device'' specification must be used. A specification for a pseudo device takes the form

> **pseudo-device** *device-name* [*howmany*]

Examples of pseudo devices are the pseudo terminal driver (see *pty*(4)), where the optional *howmany* value indicates the number of pseudo terminals to configure, and the Sun Window System (see *win*(4)).

APPENDIX A. CONFIGURATION FILE GRAMMAR

The following grammar is a compressed form of the actual *yacc*(1) grammar used by *config* to parse configuration files. Terminal symbols are shown all in upper case, literals are emboldened; optional clauses are enclosed in brackets, "[" and "]"; zero or more instantiations are denoted with "*".

Configuration ::= [Spec **;**]*

Spec ::= Config_spec
　|Device_spec
　|**trace**
　|/* lambda */

/* configuration specifications */

Config_spec ::= **machine** ID
　|**cpu** ID
　|**options** Opt_list
　|**ident** ID
　|System_spec
　|**timezone** [**–**] NUMBER [**dst** [NUMBER]]
　|**timezone** [**–**] FPNUMBER [**dst** [NUMBER]]
　|**maxusers** NUMBER

/* system configuration specifications */

System_spec ::= **config** ID System_parameter [System_parameter]*

System_parameter ::= swap_spec |root_spec |dump_spec |arg_spec

swap_spec ::= **swap** [**on**] swap_dev [**and** swap_dev]*

swap_dev ::= dev_spec [**size** NUMBER]

root_spec ::= **root** [**on**] dev_spec

dump_spec ::= **dumps** [**on**] dev_spec

arg_spec ::= **args** [**on**] dev_spec

dev_spec ::= dev_name |major_minor

major_minor ::= **major** NUMBER **minor** NUMBER

dev_name ::= ID [NUMBER [ID]]

/* option specifications */

Opt_list ::= Option [**,** Option]*

Option ::= ID [**=** Opt_value]

Opt_value ::= ID |NUMBER

/* device specifications */

Device_spec ::= **device** Dev_name Dev_info Int_spec
    |**disk** Dev_name Dev_info
    |**tape** Dev_name Dev_info
    |**controller** Dev_name Dev_info [Int_spec]
    |**pseudo-device** Dev [NUMBER]

Dev_name ::= Dev NUMBER

Dev ::= **uba** |**mba** |ID

Dev_info ::= Con_info [Info]*

Con_info ::= **at** Dev NUMBER
    |**at nexus** NUMBER

Info ::= **csr** NUMBER
    |**drive** NUMBER
    |**slave** NUMBER
    |**flags** NUMBER

Int_spec ::= **vector** ID [ID]*
    |**priority** NUMBER

Lexical Conventions

The terminal symbols are loosely defined as:

ID

One or more alphabetics, either upper or lower case, and underscore, ''_''.

NUMBER

Approximately the C language specification for an integer number. That is, a leading ''0x'' indicates a hexadecimal value, a leading ''0'' indicates an octal value, otherwise the number is expected to be a decimal value. Hexadecimal numbers may use either upper or lower case alphabetics.

FPNUMBER

A floating point number without exponent. That is a number of the form ''nnn.ddd'', where the fractional component is optional.

In special instances a question mark, ''?'', can be substituted for a ''NUMBER'' token. This is used to effect wildcarding in device interconnection specifications.

Comments in configuration files are indicated by a ''#'' character at the beginning of the line; the remainder of the line is discarded.

A specification is interpreted as a continuation of the previous line if the first character of the line is tab.

APPENDIX B. RULES FOR DEFAULTING SYSTEM DEVICES

When *config* processes a "config" rule which does not fully specify the location of the root file system, paging area(s), device for system dumps, and device for argument list processing it applies a set of rules to define those values left unspecified. The following list of rules are used in defaulting system devices.

1) If a root device is not specified, the swap specification must indicate a "generic" system is to be built.
2) If the root device does not specify a unit number, it defaults to unit 0.
3) If the root device does not include a partition specification, it defaults to the "a" partition.
4) If no swap area is specified, it defaults to the "b" partition of the root device.
5) If no device is specified for processing argument lists, the first swap partition is selected.
6) If nc device is chosen for system dumps, the first swap partition is selected (see below to find out where dumps are placed within the partition).

The following table summarizes the default partitions selected when a device specification is incomplete, e.g. "hp0".

Type	Partition
root	"a"
swap	"b"
args	"b"
dumps	"b"

Multiple swap/paging areas

When multiple swap partitions are specified, the system treats the first specified as a "primary" swap area which is always used. The remaining partitions are then interleaved into the paging system at the time a *swapon*(2) system call is made. This is normally done at boot time with a call to *swapon*(8) from the **/etc/rc** file.

System dumps

System dumps are automatically taken after a system crash, provided the device driver for the "dumps" device supports this. The dump contains the contents of memory, but not the swap areas. Normally the dump device is a disk in which case the information is copied to a location near the back of the partition. The dump is placed in the back of the partition because the primary swap and dump device are commonly the same device and this allows the system to be rebooted without immediately overwriting the saved information. When a dump has occurred, the system variable *dumpsize* is set to a non-zero value indicating the size (in bytes) of the dump. The *savecore*(8) program then copies the information from the dump partition to a file in a "crash" directory and also makes a copy of the system which was running at the time of the crash (usually **/vmunix).**

APPENDIX C. SAMPLE CONFIGURATION FILES

The following few pages present three sample configuration files.

The following GENERIC configuration file is used to build the release kernel. Lines are interpreted in #comments#. Lines marked #mandatory# must be included in every configuration file; lines which describe devices which your system does not have should be deleted when you produce your system configuration file.

```
#
# GENERIC SUN
#
machine         sun             #**Mandatory**#
cpu             "SUN2"          #**Mandatory**#
ident           GENERIC         #**Mandatory** Use "GENERIC" only if
                                    config line, below, has only "swap generic" clause#
timezone        8 dst           #**Mandatory** Number and "dst" are variable#
maxusers        2               #**Mandatory** Number may vary#
options         INET            #**Mandatory** INET means include Internet code#
options         SYSACCT         #Optional; include only with pseudo-device sysacct.
                                 Controls inclusion of code to do process accounting — see acct(2) and acct(5).#

config          vmunix  swap generic            #**Mandatory** Specify root and swap devices#

pseudo-device   pty             #Pseudo-tty's. Needed for network or window system.#
pseudo-device   bk              #Berknet line discipline for high speed tty input -  see bk(4).#
pseudo-device   sysacct         #Include only with SYSACCT options clause, above.#
pseudo-device   inet            #**Mandatory** Internet code -  see inet(4).#
pseudo-device   ether           #ARP code. Must include if using Ethernet — see arp(4).#
pseudo-device   loop            #**Mandatory** Software loop back network
                                    device driver; must include if inet — see lo(4).#
pseudo-device   nd              #Network disk.  Needed if server or diskless -  see nd(4).#
pseudo-device   win             #Window system.#
pseudo-device   ms              #Mouse; required for window system -  see ms(4).#
pseudo-device   kb              #Optional; required if using any Sun keyboard;
                                  omit if using serial terminal for console.#
controller      mb0 at nexus ?  #**Mandatory** Multibus code.#
controller      ipc0 at mb0 csr 0x40 priority 2         #1st 2180 controller -  see ip(4).#
controller      ipc1 at mb0 csr 0x44 priority 2         #2nd 2180 controller.#
disk            ip0 at ipc0 drive 0                     #1st disk on 1st 2180 controller#
disk            ip1 at ipc0 drive 1                     #2nd disk on 1st 2180 controller#
disk            ip2 at ipc1 drive 0                     #1st disk on 2nd 2180 controller#
disk            ip3 at ipc1 drive 1                     #2nd disk on 2nd 2180 controller#
controller      xyc0 at mb0 csr 0xee40 priority 2       #1st Xylogics controller -  see xy(4).#
controller      xyc1 at mb0 csr 0xee48 priority 2       #2nd Xylogics controller#
disk            xy0 at xyc0 drive 0                     #1st disk on 1st Xylogics controller#
disk            xy1 at xyc0 drive 1                     #2nd disk on 1st Xylogics controller#
disk            xy2 at xyc1 drive 0                     #1st disk on 2nd Xylogics controller#
disk            xy3 at xyc1 drive 1                     #2nd disk on 2nd Xylogics controller#
controller      sc0 at mb0 csr 0x80000 priority 2       #1st SCSI controller#
disk            sd0 at sc0 drive 0 flags 0              #1st disk on 1st SCSI controller#
disk            sd1 at sc0 drive 1 flags 0              #2nd disk on 1st SCSI controller#
tape            st0 at sc0 drive 32 flags 1             #SCSI tape#
controller      sc1 at mb0 csr 0x84000 priority 2       #2nd SCSI controller#
```

```
disk        sd2 at sc1 drive 0 flags 0                  #1st disk on 2nd SCSI controller#
disk        sd3 at sc1 drive 1 flags 0                  #2nd disk on 2nd SCSI controller#
tape        st1 at sc1 drive 32 flags 1                 #SCSI tape#
device      ropc0 at mb0 csr 0xee0800 priority 1        #**Mandatory** Raster Op chip - see ropc(4).#
device      sky0 at mb0 csr 0x2000 priority 2           #Sky Floating Point board.#
device      zs0 at mb0 csr 0xeec800 flags 3 priority 6#Optional. UART (tty) driver - see zs(4).#
device      zs1 at mb0 csr 0xeec000 flags 3 priority 6#Optional; required if using Sun-2 keybd and mouse
                                                       on Model 120. UARTs on Sun 2 Video board.#
device      zs2 at mb0 csr 0x80800 flags 3 priority 6 #Optional. UARTs on first SCSI board.#
device      zs3 at mb0 csr 0x81000 flags 3 priority 6 #Optional. UARTs on first SCSI board.#
device      zs4 at mb0 csr 0x84800 flags 3 priority 6 #Optional. UARTs on second SCSI board.#
device      zs5 at mb0 csr 0x85000 flags 3 priority 6 #Optional. UARTs on second SCSI board.#
device      oct0 at mb0 csr 0x520 flags 0xff priority 4#Central Data Octal Card - see oct(4).#
device      ec0 at mb0 csr 0xe0000 priority 3           #1st 3COM Ethernet Controller - see ec(4)#
device      ec1 at mb0 csr 0xe2000 priority 3           #2nd 3COM Ethernet Controller#
device      en0 at mb0 csr 0x100 flags 126 priority 3 #Sun 3Mbit Ethernet Controller — see en(4).#
controller  tm0 at mb0 csr 0xa0 priority 3              #1st TAPEMASTER tape controller - see tm(4).#
controller  tm1 at mb0 csr 0xa2 priority 3              #2nd TAPEMASTER tape controller#
tape        mt0 at tm0 drive 0 flags 1                  #1st 1/2" tape drive on 1st controller.#
tape        mt1 at tm1 drive 0 flags 1                  #1st 1/2" tape drive on 2nd controller.#
device      ar0 at mb0 csr 0x200 priority 3             #1st 1/4" tape drive - see ar(4).#
device      ar1 at mb0 csr 0x208 priority 3             #2nd 1/4" tape drive.#
device      cg0 at mb0 csr 0xe8000 priority 3           #Sun Color Board - see cg(4).#
device      vp0 at mb0 csr 0x400 priority 2             #Ikon Versatec Board - see vp(4).#
device      pi0 at mb0 csr 0xee2000 priority 1          #Parallel input. Only used on Sun Models 100U and 150U,
                                                            for keyboard and mouse. Not needed if using terminal
                                                            for console.#
```

The following configuration M120 is a stripped down version of the GENERIC system for a standard Model 120 system with only a SCSI tape and disk. It's a good example of a standard configuration file:

```
#
# M120
#
machine         sun
cpu             "SUN2"
ident           M120
timezone        8 dst
maxusers        2
options         INET

config          vmunix  root on sd

pseudo-device   pty
pseudo-device   inet
pseudo-device   ether
pseudo-device   loop
pseudo-device   win
pseudo-device   ms
pseudo-device   kb
controller      mb0 at nexus ?
controller      sc0 at mb0 csr 0x80000 priority 2
disk            sd0 at sc0 drive 0 flags 0
tape            st0 at sc0 drive 32 flags 1
device          zs0 at mb0 csr 0xeec800 flags 3 priority 6
device          zs1 at mb0 csr 0xeec000 flags 3 priority 6
device          ec0 at mb0 csr 0xe0000 priority 3
device          ropc0 at mb0 csr 0xee0800 priority 1
```

The following configuration SMALLXY is a stripped down version of the GENERIC system for a Model 100U or 150U Sun system with only a single local peripheral: a Xylogics disk controller. It's another good example of a standard configuration file:

```
#
# SMALL SUN WITH ONE XY DISK
#
machine         sun
cpu             "SUN2"
ident           SMALLXY
timezone        8 dst
maxusers        2
options         INET

config          vmunix  root on xy

pseudo-device   pty
pseudo-device   inet
pseudo-device   ether
pseudo-device   loop
pseudo-device   win
pseudo-device   ms
pseudo-device   kb
controller      mb0 at nexus ?
controller      xyc0 at mb0 csr 0xee40 priority 2
disk            xy0 at xyc0 drive 0
device          zs0 at mb0 csr 0xeec800 flags 3 priority 6
device          ec0 at mb0 csr 0xe0000 priority 3
device          ropc0 at mb0 csr 0xee0800 priority 1
device          pi0 at mb0 csr 0xee2000 priority 1
```

APPENDIX D. DATA STRUCTURE SIZING RULES

Certain kernel data structures are sized at compile time according to the maximum number of simultaneous users expected, while others are calculated at boot time based on the physical resources present; e.g. memory. This appendix lists both sets of rules and also includes some hints on changing built-in limitations on certain data structures.

Compile time rules

The file *sys/conf/param.c* contains the definitions of almost all data structures sized at compile time. This file is copied into the directory of each configured kernel to allow configuration-dependent rules and values to be maintained. The rules implied by its contents are summarized below (here MAXUSERS refers to the value defined in the configuration file in the "maxusers" rule).

nproc

The maximum number of processes which may be running at any time. It is defined to be 20 + 8 * MAXUSERS and referred to in other calculations as NPROC.

ntext

The maximum number of active shared text segments. Defined as 24 + MAXUSERS.

ninode

The maximum number of files in the file system which may be active at any time. This includes files in use by users, as well as directory files being read or written by the system and files associated with bound sockets in the UNIX ipc domain. This is defined as (NPROC + 16 + MAXUSERS) + 32.

nfile

The number of "file table" structures. One file table structure is used for each open, unshared, file descriptor. Multiple file descriptors may reference a single file table entry when they are created through a *dup* call, or as the result of a *fork*. This is defined to be

16 * (NPROC + 16 + MAXUSERS) / 10 + 32

ncallout

The number of "callout" structures. One callout structure is used per internal system event handled with a timeout. Timeouts are used for terminal delays, watchdog routines in device drivers, protocol timeout processing, etc. This is defined as 16 + NPROC.

nclist

The number of "c-list" structures. C-list structures are used in terminal i/o. This is defined as 100 + 16 * MAXUSERS.

nmbclusters

The maximum number of pages which may be allocated by the network. This is defined as 256 (a quarter megabyte of memory) in /sys/h/mbuf.h. In practice, the network rarely uses this much memory. It starts off by allocating 64 kilobytes of memory, then requesting more as required. This value represents an upper bound.

nquota

The number of "quota" structures allocated. Quota structures are present only when disc quotas are configured in the system. One quota structure is kept per user. This is defined to be (MAXUSERS * 9) / 7 + 3.

ndquot

The number of "dquot" structures allocated. Dquot structures are present only when disc quotas are configured in the system. One dquot structure is required per user, per active file system quota. That is, when a user manipulates a file on a file system on which quotas are enabled, the information regarding the user's quotas on that file system must be in-core. This information is cached, so that not all information must be present in-core all the time. This is defined as (MAXUSERS * NMOUNT) / 4 + NPROC, where NMOUNT is the maximum number of mountable file systems.

Run-time calculations

The most important data structure sized at run-time is the file system buffer cache. The system allocates 10% of each half-megabyte after the first half-megabyte to the cache. Thus on a 1 Megabyte machine, 50 kilobytes is allocated to the cache, while on a 2 Megabyte machine, 150 kilobytes is allocated to the cache. In any case, not less than 16 pages of file system buffers is allocated.

The number of buffers to be allocated can be forced to a specific value by patching the kernel variable *nbuf* with *adb*:

```
# adb -w /vmunix
nbuf?W 0t32
nbuf:   0 =     20
$q
#
```

sets the number of buffers to be 32 (decimal) independent of the amount of main memory available. Reboot after performing this series.

System size limitations

Because the file system block numbers are stored in page table *pg_blkno* entries, the maximum size of a file system is limited to 2^19 1024 byte blocks. Thus no file system can be larger than 512M bytes.

The count of mountable file systems is limited to 15. This should be sufficient. If you have many disks it makes sense to make some of them single file systems, and the paging areas don't count in this total. To increase this, you must change the core-map (only if you have source) */sys/h/cmap.h*, since there is a 4 bit field used here. The size of the core-map will then expand to 16 bytes per 2048 byte page. Don't forget to change MSWAPX and NMOUNT in */sys/h/param.h* also.

The maximum value NOFILE (open files per process limit) can be raised to is 30 because of a bit field in the page table entry in */sys/machine/pte.h.*

Fsck – The UNIX† File System Check Program

Revised July 28, 1983

Marshall Kirk McKusick

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

T. J. Kowalski

Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

This document reflects the use of *fsck* with the revised file system organization implemented in release 0.1 of the Sun UNIX operating system. This is a revision of the original paper written by T. J. Kowalski.

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

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

†UNIX is a trademark of Bell Laboratories

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

TABLE OF CONTENTS

1. Introduction

This document reflects the use of *fsck* with the new file system organization implemented in the 0.1 release of the Sun UNIX system. This is a revision of the original paper written by T. J. Kowalski.

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

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

2. Overview of the file system

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

2.1. Superblock

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

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

Sun's block size is 4K; fragment size is 1K.

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.

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

2.2. Summary information

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

2.3. Cylinder groups

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

All the cylinder group bookkeeping information could be placed at the beginning of each cylinder group. However if this approach were used, all the redundant information would be on the top platter. A single hardware failure that destroyed the top platter could cause the loss of all copies of the redundant super-blocks. Thus the cylinder group bookkeeping information begins at a floating offset from the beginning of the cylinder group. The offset for the *i+1*st cylinder group is about one track further from the beginning of the cylinder group than it was for the *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.

2.4. Fragments

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

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

2.5. Updates to the file system

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

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

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

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

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

3. Fixing corrupted file systems

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

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

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

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

3.1. Detecting and correcting corruption

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

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

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

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

3.2. Super-block checking

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

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

3.3. Free block checking

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

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

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

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

3.4. Checking the inode state

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

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

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

3.5. Inode links

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

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

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

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

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

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

3.6. Inode data size

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

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

3.7. Checking the data associated with an inode

An inode can directly or indirectly reference three kinds of data blocks. All referenced blocks must be the same kind. The three types of data blocks are: plain data blocks, symbolic link data blocks, and directory data blocks. Plain data blocks 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.

3.8. File system connectivity

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

Acknowledgements

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

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

References

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

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

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

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

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

4. Appendix A – Fsck Error Conditions

4.1. Conventions

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

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

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

4.2. Initialization

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

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

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

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

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

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

Can't open *F*
Fsck's request attempt to open the file system *F* failed. When running manually, it ignores this file system and continues checking the next file system given. Check access modes of *F*.

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

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

Possible responses to the OK prompt are:

YES Ignore this error condition.

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

One of the following messages will appear:
MAGIC NUMBER WRONG
NCG OUT OF RANGE
CPG OUT OF RANGE
NSECT < 1
NTRAK < 1
SPC DOES NOT JIVE w/NTRAK*NSECT
INODES NOT MULTIPLE OF A BLOCK
IMPLIES MORE INODE THAN DATA BLOCKS
NCYL DOES NOT JIVE WITH NCG*CPG
FPG DOES NOT JIVE WITH CPG & SPC
SIZE PREPOSTEROUSLY SMALL
SIZE PREPOSTEROUSLY LARGE
CGSIZE INCORRECT
CSSIZE INCORRECT

and will be followed by the message:
F: BAD SUPER BLOCK: B
USE -b OPTION TO FSCK TO SPECIFY LOCATION OF AN ALTERNATE SUPER-BLOCK TO SUPPLY NEEDED INFORMATION; SEE fsck(8).
The super block has been corrupted. An alternative super block must be selected from among those listed by *newfs* (8) when the file system was created. For file systems with a blocksize less than 32K, specifying –b32 is a good first choice.

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

Possible responses to the CONTINUE prompt are:

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

NO terminate the program.

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

Possible responses to the CONTINUE prompt are:

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

NO terminate the program.

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

Possible responses to the CONTINUE prompt are:

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

NO terminate the program.

4.3. Phase 1 – Check Blocks and Sizes

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

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

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

Possible responses to the CLEAR prompt are:

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

NO ignore this error condition.

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

Possible responses to the CONTINUE prompt are:

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

NO terminate the program.

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

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

Possible responses to the CONTINUE prompt are:

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

NO terminate the program.

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

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

Possible responses to the CONTINUE prompt are:

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

NO terminate the program.

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

Possible responses to the CONTINUE prompt are:

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

NO terminate the program.

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

Possible responses to the CLEAR prompt are:

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

NO ignore this error condition.

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

Possible responses to the CORRECT prompt are:

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

NO ignore this error condition.

4.4. Phase 1B: Rescan for More Dups

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

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

4.5. Phase 2 – Check Pathnames

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

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

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

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

Possible responses to the FIX prompt are:

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

NO terminate the program.

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

Possible responses to the CONTINUE prompt are:

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

NO terminate the program.

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

Possible responses to the REMOVE prompt are:

YES the directory entry *F* is removed.

NO ignore this error condition.

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

Possible responses to the REMOVE prompt are:

YES the directory entry *F* is removed.

NO ignore this error condition.

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

Possible responses to the REMOVE prompt are:

YES the directory entry *F* is removed.

NO ignore this error condition.

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

Possible responses to the REMOVE prompt are:

YES the directory entry *F* is removed.

NO ignore this error condition.

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

Possible responses to the REMOVE prompt are:

YES the directory entry *F* is removed.

NO ignore this error condition.

4.6. Phase 3 – Check Connectivity

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

DIRECTORY *D* CORRUPTED (SALVAGE)
A directory with an inconsistent internal state has been found. This error is fatal if the file system is being preen'ed.

Possible responses to the SALVAGE prompt are:

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

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

UNREF DIR I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* (RECONNECT)
The directory inode *I* was not connected to a directory entry when the file system was traversed. The owner *O*, mode *M*, size *S*, and modify time *T* of directory inode *I* are printed. When

preen'ing, the directory is reconnected if its size is non-zero, otherwise it is cleared.

Possible responses to the RECONNECT prompt are:

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

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

SORRY. NO lost+found DIRECTORY

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

SORRY. NO SPACE IN lost+found DIRECTORY

There is no space to add another entry to the *lost+found* directory in the root directory of the file system; *fsck* ignores the request to link a directory in *lost+found*. This will always invoke the UNREF error condition in Phase 4. Clean out unnecessary entries in *lost+found* or make *lost+found* larger. See *fsck*(8) manual entry for further detail. This error is fatal if the file system is being preen'ed.

DIR I=*I1* CONNECTED. PARENT WAS I=*I2*

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

4.7. Phase 4 – Check Reference Counts

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

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

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

Possible responses to the RECONNECT prompt are:

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

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

(CLEAR)

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

Possible responses to the CLEAR prompt are:

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

NO ignore this error condition.

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

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

LINK COUNT FILE I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* COUNT=*X* SHOULD BE *Y* (ADJUST)
The link count for inode *I* which is a file, is *X* but should be *Y*. The owner *O*, mode *M*, size *S*, and modify time *T* are printed. When preen'ing the link count is adjusted.

Possible responses to the ADJUST prompt are:

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

NO ignore this error condition.

LINK COUNT DIR I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* COUNT=*X* SHOULD BE *Y* (ADJUST)
The link count for inode *I* which is a directory, is *X* but should be *Y*. The owner *O*, mode *M*, size *S*, and modify time *T* of directory inode *I* are printed. When preen'ing the link count is adjusted.

Possible responses to the ADJUST prompt are:

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

NO ignore this error condition.

LINK COUNT *F* I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* COUNT=*X* SHOULD BE *Y* (ADJUST)
The link count for *F* inode *I* is *X* but should be *Y*. The name *F*, owner *O*, mode *M*, size *S*, and modify time *T* are printed. When preen'ing the link count is adjusted.

Possible responses to the ADJUST prompt are:

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

NO ignore this error condition.

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

Possible responses to the CLEAR prompt are:

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

NO ignore this error condition.

UNREF DIR I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* (CLEAR)
Inode *I* which is a directory, was not connected to a directory entry when the file system was

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

Possible responses to the CLEAR prompt are:

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

NO ignore this error condition.

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

Possible responses to the CLEAR prompt are:

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

NO ignore this error condition.

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

Possible responses to the CLEAR prompt are:

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

NO ignore this error condition.

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

Possible responses to the FIX prompt are:

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

NO ignore this error condition.

4.8. Phase 5 - Check Cyl groups

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

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

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

Possible responses to the CONTINUE prompt are:

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

NO terminate the program.

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

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

BAD CYLINDER GROUPS (SALVAGE)
Phase 5 has found bad blocks in the free-block maps, duplicate blocks in the free-block maps, or blocks missing from the file system. When preen'ing, the cylinder groups are reconstructed.

Possible responses to the SALVAGE prompt are:

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

NO ignore this error condition.

FREE BLK COUNT WRONG IN SUPERBLOCK (FIX)
The actual count of free blocks does not match the count in the super-block of the file system. When preen'ing, the counts are fixed.

Possible responses to the FIX prompt are:

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

NO ignore this error condition.

4.9. Phase 6 - Salvage Cylinder Groups

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

4.10. Cleanup

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

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

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

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

SENDMAIL – An Internetwork Mail Router

Eric Allman†

Britton-Lee, Inc.
1919 Addison Street, Suite 105.
Berkeley, California 94704.

ABSTRACT

Routing mail through a heterogenous internet presents many new problems. Among the worst of these is that of address mapping. Historically, this has been handled on an *ad hoc* basis. However, this approach has become unmanageable as internets grow.

***Sendmail* acts a unified "post office" to which all mail can be submitted. Address interpretation is controlled by a production system, which can parse both domain-based addressing and old-style *ad hoc* addresses. The production system is powerful enough to rewrite addresses in the message header to conform to the standards of a number of common target networks, including old (NCP/RFC733) Arpanet, new (TCP/RFC822) Arpanet, UUCP, and Phonenet. Sendmail also implements an SMTP server, message queueing, and aliasing.**

***Sendmail* implements a general internetwork mail routing facility, featuring aliasing and forwarding, automatic routing to network gateways, and flexible configuration.**

In a simple network, each node has an address, and resources can be identified with a host-resource pair; in particular, the mail system can refer to users using a host-username pair. Host names and numbers have to be administered by a central authority, but usernames can be assigned locally to each host.

In an internet, multiple networks with different characterstics and managements must communicate. In particular, the syntax and semantics of resource identification change. Certain special cases can be handled trivially by *ad hoc* techniques, such as providing network names that appear local to hosts on other networks, as with the Ethernet at Xerox PARC. However, the general case is extremely complex. For example, some networks require point-to-point routing, which simplifies the database update problem since only adjacent hosts must be entered into the system tables, while others use end-to-end addressing. Some networks use a left-associative syntax and others use a right-associative syntax, causing ambiguity in mixed addresses.

Internet standards seek to eliminate these problems. Initially, these proposed expanding the address pairs to address triples, consisting of {network, host, resource} triples. Network numbers must be universally agreed upon, and hosts can be assigned locally on each network. The user-level presentation was quickly expanded to address domains, comprised of a local resource identification and a hierarchical domain specification with a common static root. The domain technique separates the issue of physical versus logical addressing. For example, an address of the form "eric@a.cc.berkeley.arpa" describes only the logical organization of the address space.

***Sendmail* is intended to help bridge the gap between the totally *ad hoc* world of networks that know nothing of each other and the clean, tightly-coupled world of unique network numbers. It can accept old arbitrary address syntaxes, resolving ambiguities using heuristics specified by the**

†A considerable part of this work was done while under the employ of the INGRES Project at the University of California at Berkeley.

system administrator, as well as domain-based addressing. It helps guide the conversion of message formats between disparate networks. In short, *sendmail* is designed to assist a graceful transition to consistent internetwork addressing schemes.

Section 1 discusses the design goals for *sendmail.* Section 2 gives an overview of the basic functions of the system. In section 3, details of usage are discussed. Section 4 compares *sendmail* to other internet mail routers, and an evaluation of *sendmail* is given in section 5, including future plans.

1. DESIGN GOALS

Design goals for *sendmail* include:

(1) Compatibility with the existing mail programs, including Bell version 6 mail, Bell version 7 mail [UNIX83], Berkeley *Mail* [Shoens79], BerkNet mail [Schmidt79], and hopefully UUCP mail [Nowitz78a, Nowitz78b]. ARPANET mail [Crocker77a, Postel77] was also required.

(2) Reliability, in the sense of guaranteeing that every message is correctly delivered or at least brought to the attention of a human for correct disposal; no message should ever be completely lost. This goal was considered essential because of the emphasis on mail in our environment. It has turned out to be one of the hardest goals to satisfy, especially in the face of the many anomalous message formats produced by various ARPANET sites. For example, certain sites generate improperly formated addresses, occasionally causing error-message loops. Some hosts use blanks in names, causing problems with UNIX mail programs that assume that an address is one word. The semantics of some fields are interpreted slightly differently by different sites. In summary, the obscure features of the ARPANET mail protocol really *are* used and are difficult to support, but must be supported.

(3) Existing software to do actual delivery should be used whenever possible. This goal derives as much from political and practical considerations as technical.

(4) Easy expansion to fairly complex environments, including multiple connections to a single network type (such as with multiple UUCP or Ether nets [Metcalfe76]). This goal requires consideration of the contents of an address as well as its syntax in order to determine which gateway to use. For example, the ARPANET is bringing up the TCP protocol to replace the old NCP protocol. No host at Berkeley runs both TCP and NCP, so it is necessary to look at the ARPANET host name to determine whether to route mail to an NCP gateway or a TCP gateway.

(5) Configuration should not be compiled into the code. A single compiled program should be able to run as is at any site (barring such basic changes as the CPU type or the operating system). We have found this seemingly unimportant goal to be critical in real life. Besides the simple problems that occur when any program gets recompiled in a different environment, many sites like to "fiddle" with anything that they will be recompiling anyway.

(6) *Sendmail* must be able to let various groups maintain their own mailing lists, and let individuals specify their own forwarding, without modifying the system alias file.

(7) Each user should be able to specify which mailer to execute to process mail being delivered for him. This feature allows users who are using specialized mailers that use a different format to build their environment without changing the system, and facilitates specialized functions (such as returning an "I am on vacation" message).

(8) Network traffic should be minimized by batching addresses to a single host where possible, without assistance from the user.

These goals motivated the architecture illustrated in Figure 1. The user interacts with a mail generating and sending program. When the mail is created, the generator calls *sendmail*,

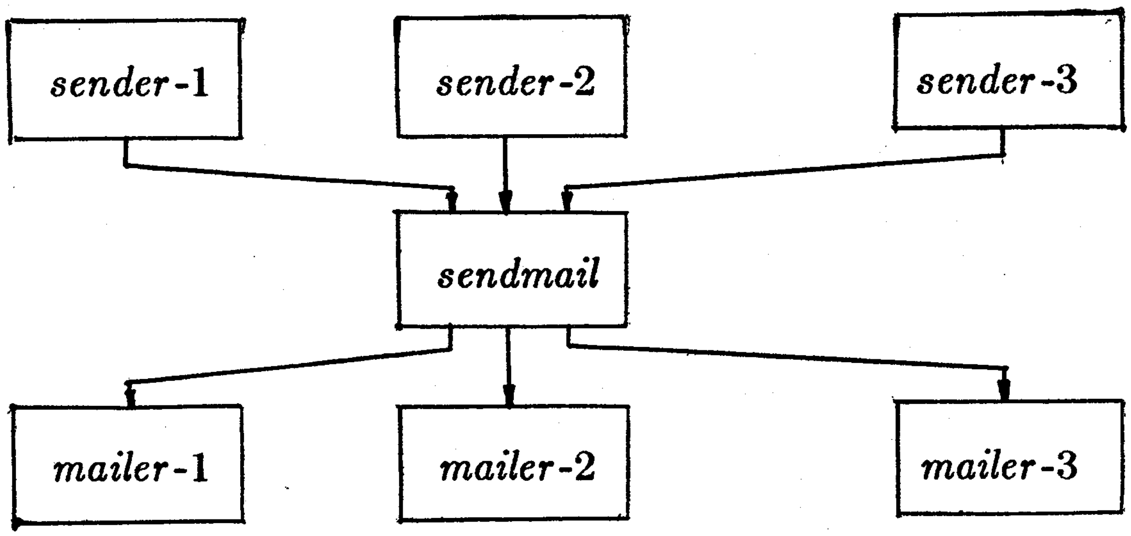


Figure 1 – Sendmail System Structure.

which routes the message to the correct mailer(s). Since some of the senders may be network servers and some of the mailers may be network clients, *sendmail* may be used as an internet mail gateway.

2. OVERVIEW

2.1. System Organization

Sendmail neither interfaces with the user nor does actual mail delivery. Rather, it collects a message generated by a user interface program (UIP) such as Berkeley *Mail*, MS [Crocker77b], or MH [Borden79], edits the message as required by the destination network, and calls appropriate mailers to do mail delivery or queueing for network transmission[1]. This discipline allows the insertion of new mailers at minimum cost. In this sense *sendmail* resembles the Message Processing Module (MPM) of [Postel79b].

2.2. Interfaces to the Outside World

There are three ways *sendmail* can communicate with the outside world, both in receiving and in sending mail. These are using the conventional UNIX argument vector/return status, speaking SMTP over a pair of UNIX pipes, and speaking SMTP over an interprocess(or) channel.

2.2.1. Argument vector/exit status

This technique is the standard UNIX method for communicating with the process. A list of recipients is sent in the argument vector, and the message body is sent on the standard input. Anything that the mailer prints is simply collected and sent back to the sender if there were any problems. The exit status from the mailer is collected after the message is sent, and a diagnostic is printed if appropriate.

[1]except when mailing to a file, when *sendmail* does the delivery directly.

2.2.2. SMTP over pipes

The SMTP protocol [Postel82] can be used to run an interactive lock-step interface with the mailer. A subprocess is still created, but no recipient addresses are passed to the mailer via the argument list. Instead, they are passed one at a time in commands sent to the processes standard input. Anything appearing on the standard output must be a reply code in a special format.

2.2.3. SMTP over an IPC connection

This technique is similar to the previous technique, except that it uses a 4.2BSD IPC channel [UNIX83]. This method is exceptionally flexible in that the mailer need not reside on the same machine. It is normally used to connect to a sendmail process on another machine.

2.3. Operational Description

When a sender wants to send a message, it issues a request to *sendmail* using one of the three methods described above. *Sendmail* operates in two distinct phases. In the first phase, it collects and stores the message. In the second phase, message delivery occurs. If there were errors during processing during the second phase, *sendmail* creates and returns a new message describing the error and/or returns an status code telling what went wrong.

2.3.1. Argument processing and address parsing

If *sendmail* is called using one of the two subprocess techniques, the arguments are first scanned and option specifications are processed. Recipient addresses are then collected, either from the command line or from the SMTP RCPT command, and a list of recipients is created. Aliases are expanded at this step, including mailing lists. As much validation as possible of the addresses is done at this step: syntax is checked, and local addresses are verified, but detailed checking of host names and addresses is deferred until delivery. Forwarding is also performed as the local addresses are verified.

Sendmail appends each address to the recipient list after parsing. When a name is aliased or forwarded, the old name is retained in the list, and a flag is set that tells the delivery phase to ignore this recipient. This list is kept free from duplicates, preventing alias loops and duplicate messages deliverd to the same recipient, as might occur if a person is in two groups.

2.3.2. Message collection

Sendmail then collects the message. The message should have a header at the beginning. No formatting requirements are imposed on the message except that they must be lines of text (in other words, binary data is not allowed). The header is parsed and stored in memory, and the body of the message is saved in a temporary file.

To simplify the program interface, the message is collected even if no addresses were valid. The message will be returned with an error.

2.3.3. Message delivery

For each unique mailer and host in the recipient list, *sendmail* calls the appropriate mailer. Each mailer invocation sends to all users receiving the message on one host. Mailers that only accept one recipient at a time are handled properly.

The message is sent to the mailer using one of the same three interfaces used to submit a message to sendmail. Each copy of the message is prepended by a customized header. The mailer status code is caught and checked, and a suitable error message given as appropriate. The exit code must conform to a system standard or a generic message ("Service unavailable") is given.

2.3.4. Queueing for retransmission

If the mailer returned an status that indicated that it might be able to handle the mail later, *sendmail* will queue the mail and try again later.

2.3.5. Return to sender

If errors occur during processing, *sendmail* returns the message to the sender for retransmission. The letter can be mailed back or written in the file "dead.letter" in the sender's home directory[2].

2.4. Message Header Editing

Certain editing of the message header occurs automatically. Header lines can be inserted under control of the configuration file. Some lines can be merged; for example, a "From:" line and a "Full-name:" line can be merged under certain circumstances.

2.5. Configuration File

Almost all configuration information is read at runtime from an ASCII file, encoding macro definitions (defining the value of macros used internally), header declarations (telling sendmail the format of header lines that it will process specially, i.e., lines that it will add or reformat), mailer definitions (giving information such as the location and characteristics of each mailer), and address rewriting rules (a limited production system to rewrite addresses which is used to parse and rewrite the addresses).

To improve performance when reading the configuration file, a memory image can be provided. This provides a "compiled" form of the configuration file.

3. USAGE AND IMPLEMENTATION

3.1. Arguments

Arguments may be flags and addresses. Flags set various processing options. Following flag arguments, address arguments may be given, unless we are running in SMTP mode. Addresses follow the syntax in RFC822 [Crocker82] for ARPANET address formats. In brief, the format is:

(1) Anything in parentheses is thrown away (as a comment).

(2) Anything in angle brackets ("< >") is preferred over anything else. This rule implements the ARPANET standard that addresses of the form

    user name <machine-address>

will send to the electronic "machine-address" rather than the human "user name."

(3) Double quotes (") quote phrases; backslashes quote characters. Backslashes are more powerful in that they will cause otherwise equivalent phrases to compare differently – for example, *user* and "*user*" are equivalent, but \user is different from either of them.

Parentheses, angle brackets, and double quotes must be properly balanced and nested. The rewriting rules control remaining parsing[3].

[2]Obviously, if the site giving the error is not the originating site, the only reasonable option is to mail back to the sender. Also, there are many more error disposition options, but they only effect the error message – the "return to sender" function is always handled in one of these two ways.

[3]Disclaimer: Some special processing is done after rewriting local names; see below.

3.2. Mail to Files and Programs

Files and programs are legitimate message recipients. Files provide archival storage of messages, useful for project administration and history. Programs are useful as recipients in a variety of situations, for example, to maintain a public repository of systems messages (such as the Berkeley *msgs* program, or the MARS system [Sattley78]).

Any address passing through the initial parsing algorithm as a local address (i.e, not appearing to be a valid address for another mailer) is scanned for two special cases. If prefixed by a vertical bar ("|") the rest of the address is processed as a shell command. If the user name begins with a slash mark ("/") the name is used as a file name, instead of a login name.

Files that have setuid or setgid bits set but no execute bits set have those bits honored if *sendmail* is running as root.

3.3. Aliasing, Forwarding, Inclusion

Sendmail reroutes mail three ways. Aliasing applies system wide. Forwarding allows each user to reroute incoming mail destined for that account. Inclusion directs *sendmail* to read a file for a list of addresses, and is normally used in conjunction with aliasing.

3.3.1. Aliasing

Aliasing maps names to address lists using a system-wide file. This file is indexed to speed access. Only names that parse as local are allowed as aliases; this guarantees a unique key (since there are no nicknames for the local host).

3.3.2. Forwarding

After aliasing, recipients that are local and valid are checked for the existence of a ".forward" file in their home directory. If it exists, the message is *not* sent to that user, but rather to the list of users in that file. Often this list will contain only one address, and the feature will be used for network mail forwarding.

Forwarding also permits a user to specify a private incoming mailer. For example, forwarding to:

```
"|/usr/local/newmail myname"
```

will use a different incoming mailer.

3.3.3. Inclusion

Inclusion is specified in RFC 733 [Crocker77a] syntax:

```
:Include: pathname
```

An address of this form reads the file specified by *pathname* and sends to all users listed in that file.

The intent is *not* to support direct use of this feature, but rather to use this as a subset of aliasing. For example, an alias of the form:

```
project: :include:/usr/project/userlist
```

is a method of letting a project maintain a mailing list without interaction with the system administration, even if the alias file is protected.

It is not necessary to rebuild the index on the alias database when a :include: list is changed.

3.4. Message Collection

Once all recipient addresses are parsed and verified, the message is collected. The message comes in two parts: a message header and a message body, separated by a blank line.

The header is formatted as a series of lines of the form

field-name: field-value

Field-value can be split across lines by starting the following lines with a space or a tab. Some header fields have special internal meaning, and have appropriate special processing. Other headers are simply passed through. Some header fields may be added automatically, such as time stamps.

The body is a series of text lines. It is completely uninterpreted and untouched, except that lines beginning with a dot have the dot doubled when transmitted over an SMTP channel. This extra dot is stripped by the receiver.

3.5. Message Delivery

The send queue is ordered by receiving host before transmission to implement message batching. Each address is marked as it is sent so rescanning the list is safe. An argument list is built as the scan proceeds. Mail to files is detected during the scan of the send list. The interface to the mailer is performed using one of the techniques described in section 2.2.

After a connection is established, *sendmail* makes the per-mailer changes to the header and sends the result to the mailer. If any mail is rejected by the mailer, a flag is set to invoke the return-to-sender function after all delivery completes.

3.6. Queued Messages

If the mailer returns a "temporary failure" exit status, the message is queued. A control file is used to describe the recipients to be sent to and various other parameters. This control file is formatted as a series of lines, each describing a sender, a recipient, the time of submission, or some other salient parameter of the message. The header of the message is stored in the control file, so that the associated data file in the queue is just the temporary file that was originally collected.

3.7. Configuration

Configuration is controlled primarily by a configuration file read at startup. *Sendmail* should not need to be recomplied except

(1) To change operating systems (V6, V7/32V, 4BSD).

(2) To remove or insert the DBM (UNIX database) library.

(3) To change ARPANET reply codes.

(4) To add headers fields requiring special processing.

Adding mailers or changing parsing (i.e., rewriting) or routing information does not require recompilation.

If the mail is being sent by a local user, and the file ".mailcf" exists in the sender's home directory, that file is read as a configuration file after the system configuration file. The primary use of this feature is to add header lines.

The configuration file encodes macro definitions, header definitions, mailer definitions, rewriting rules, and options.

3.7.1. Macros

Macros can be used in three ways. Certain macros transmit unstructured textual information into the mail system, such as the name *sendmail* will use to identify itself in error messages. Other macros transmit information from *sendmail* to the configuration file for use in creating other fields (such as argument vectors to mailers): the name of the sender, and the host and user of the recipient. Other macros are unused internally, and can be used as shorthand in the configuration file.

3.7.2. Header declarations

Header declarations inform *sendmail* of the format of known header lines. Knowledge of a few header lines is built into *sendmail*, such as the "From:" and "Date:" lines.

Most configured headers will be automatically inserted in the outgoing message if they don't exist in the incoming message. Certain headers are suppressed by some mailers.

3.7.3. Mailer declarations

Mailer declarations tell *sendmail* of the various mailers available to it. The definition specifies the internal name of the mailer, the pathname of the program to call, some flags associated with the mailer, and an argument vector to be used on the call; this vector is macro-expanded before use.

3.7.4. Address rewriting rules

The heart of address parsing in *sendmail* is a set of rewriting rules. These are an ordered list of pattern-replacement rules, (somewhat like a production system, except that order is critical), which are applied to each address. The address is rewritten textually until it is either rewritten into a special canonical form (i.e., a (mailer, host, user) 3-tuple, such as {arpanet, usc-isif, postel} representing the address "postel@usc-isif"), or it falls off the end. When a pattern matches, the rule is reapplied until it fails.

The configuration file also supports the editing of addresses into different formats. For example, an address of the form:

ucsfcgl!tef

might be mapped into:

tef@ucsfcgl.UUCP

to conform to the domain syntax. Translations can also be done in the other direction.

3.7.5. Option setting

There are several options that can be set from the configuration file. These include the pathnames of various support files, timeouts, default modes, etc.

4. COMPARISON WITH OTHER MAILERS

4.1. Delivermail

Sendmail is an outgrowth of *delivermail*. The primary differences are:

(1) Configuration information is not compiled in. This change simplifies many of the problems of moving to other machines. It also allows easy debugging of new mailers.

(2) Address parsing is more flexible. For example, *delivermail* only supported one gateway to any network, whereas *sendmail* can be sensitive to host names and reroute to different gateways.

(3) Forwarding and :include: features eliminate the requirement that the system alias file be writable by any user (or that an update program be written, or that the system administration make all changes).

(4) *Sendmail* supports message batching across networks when a message is being sent to multiple recipients.

(5) A mail queue is provided in *sendmail*. Mail that cannot be delivered immediately but can potentially be delivered later is stored in this queue for a later retry. The queue also provides a buffer against system crashes; after the message has been collected it may be reliably redelivered even if the system crashes during the initial delivery.

(6) *Sendmail* uses the networking support provided by 4.2BSD to provide a direct interface networks such as the ARPANET and/or Ethernet using SMTP (the Simple Mail Transfer Protocol) over a TCP/IP connection.

4.2. MMDF

MMDF [Crocker79] spans a wider problem set than *sendmail*. For example, the domain of MMDF includes a "phone network" mailer, whereas *sendmail* calls on preexisting mailers in most cases.

MMDF and *sendmail* both support aliasing, customized mailers, message batching, automatic forwarding to gateways, queueing, and retransmission. MMDF supports two-stage timeout, which *sendmail* does not support.

The configuration for MMDF is compiled into the code[4].

Since MMDF does not consider backwards compatibility as a design goal, the address parsing is simpler but much less flexible.

It is somewhat harder to integrate a new channel[5] into MMDF. In particular, MMDF must know the location and format of host tables for all channels, and the channel must speak a special protocol. This allows MMDF to do additional verification (such as verifying host names) at submission time.

MMDF strictly separates the submission and delivery phases. Although *sendmail* has the concept of each of these stages, they are integrated into one program, whereas in MMDF they are split into two programs.

4.3. Message Processing Module

The Message Processing Module (MPM) discussed by Postel [Postel79b] matches *sendmail* closely in terms of its basic architecture. However, like MMDF, the MPM includes the network interface software as part of its domain.

MPM also postulates a duplex channel to the receiver, as does MMDF, thus allowing simpler handling of errors by the mailer than is possible in *sendmail*. When a message queued by *sendmail* is sent, any errors must be returned to the sender by the mailer itself. Both MPM and MMDF mailers can return an immediate error response, and a single error processor can create an appropriate response.

MPM prefers passing the message as a structured object, with type-length-value tuples[6]. Such a convention requires a much higher degree of cooperation between mailers than is required by *sendmail*. MPM also assumes a universally agreed upon internet name space (with each address in the form of a net-host-user tuple), which *sendmail* does not.

5. EVALUATIONS AND FUTURE PLANS

Sendmail is designed to work in a nonhomogeneous environment. Every attempt is made to avoid imposing unnecessary constraints on the underlying mailers. This goal has driven much of the design. One of the major problems has been the lack of a uniform address space, as postulated in [Postel79a] and [Postel79b].

[4]Dynamic configuration tables are currently being considered for MMDF; allowing the installer to select either compiled or dynamic tables.

[5]The MMDF equivalent of a *sendmail* "mailer."

[6]This is similar to the NBS standard.

A nonuniform address space implies that a path will be specified in all addresses, either explicitly (as part of the address) or implicitly (as with implied forwarding to gateways). This restriction has the unpleasant effect of making replying to messages exceedingly difficult, since there is no one "address" for any person, but only a way to get there from wherever you are.

Interfacing to mail programs that were not initially intended to be applied in an internet environment has been amazingly successful, and has reduced the job to a manageable task.

Sendmail has knowledge of a few difficult environments built in. It generates ARPANET FTP/SMTP compatible error messages (prepended with three-digit numbers [Neigus73, Postel74, Postel82]) as necessary, optionally generates UNIX-style "From" lines on the front of messages for some mailers, and knows how to parse the same lines on input. Also, error handling has an option customized for BerkNet.

The decision to avoid doing any type of delivery where possible (even, or perhaps especially, local delivery) has turned out to be a good idea. Even with local delivery, there are issues of the location of the mailbox, the format of the mailbox, the locking protocol used, etc., that are best decided by other programs. One surprisingly major annoyance in many internet mailers is that the location and format of local mail is built in. The feeling seems to be that local mail is so common that it should be efficient. This feeling is not born out by our experience; on the contrary, the location and format of mailboxes seems to vary widely from system to system.

The ability to automatically generate a response to incoming mail (by forwarding mail to a program) seems useful ("I am on vacation until late August....") but can create problems such as forwarding loops (two people on vacation whose programs send notes back and forth, for instance) if these programs are not well written. A program could be written to do standard tasks correctly, but this would solve the general case.

It might be desirable to implement some form of load limiting. I am unaware of any mail system that addresses this problem, nor am I aware of any reasonable solution at this time.

The configuration file is currently practically inscrutable; considerable convenience could be realized with a higher-level format.

It seems clear that common protocols will be changing soon to accommodate changing requirements and environments. These changes will include modifications to the message header (e.g., [NBS80]) or to the body of the message itself (such as for multimedia messages [Postel80]). Experience indicates that these changes should be relatively trivial to integrate into the existing system.

In tightly coupled environments, it would be nice to have a name server such as Grapvine [Birrell82] integrated into the mail system. This would allow a site such as "Berkeley" to appear as a single host, rather than as a collection of hosts, and would allow people to move transparently among machines without having to change their addresses. Such a facility would require an automatically updated database and some method of resolving conflicts. Ideally this would be effective even without all hosts being under a single management. However, it is not clear whether this feature should be integrated into the aliasing facility or should be considered a "value added" feature outside *sendmail* itself.

As a more interesting case, the CSNET name server [Solomon81] provides an facility that goes beyond a single tightly-coupled environment. Such a facility would normally exist outside of *sendmail* however.

ACKNOWLEDGEMENTS

Thanks are due to Kurt Shoens for his continual cheerful assistance and good advice, Bill Joy for pointing me in the correct direction (over and over), and Mark Horton for more advice, prodding, and many of the good ideas. Kurt and Eric Schmidt are to be credited for using *delivermail* as a server for their programs (*Mail* and BerkNet respectively) before any sane person should have, and making the necessary modifications promptly and happily. Eric gave me considerable advice about the perils of network software which saved me an unknown amount of work and grief. Mark did the original implementation of the DBM version of aliasing, installed the VFORK code, wrote the current version of *rmail*, and was the person who really convinced me to put the work into *delivermail* to turn it into *sendmail*. Kurt deserves accolades for using *sendmail* when I was myself afraid to take the risk; how a person can continue to be so enthusiastic in the face of so much bitter reality is beyond me.

Kurt, Mark, Kirk McKusick, Marvin Solomon, and many others have reviewed this paper, giving considerable useful advice.

Special thanks are reserved for Mike Stonebraker at Berkeley and Bob Epstein at Britton-Lee, who both knowingly allowed me to put so much work into this project when there were so many other things I really should have been working on.

REFERENCES

[Birrell82] Birrell, A. D., Levin, R., Needham, R. M., and Schroeder, M. D., "Grapevine: An Exercise in Distributed Computing." In *Comm. A.C.M. 25*, 4, April 82.

[Borden79] Borden, S., Gaines, R. S., and Shapiro, N. Z., *The MH Message Handling System: Users' Manual.* R-2367-PAF. Rand Corporation. October 1979.

[Crocker77a] Crocker, D. H., Vittal, J. J., Pogran, K. T., and Henderson, D. A. Jr., *Standard for the Format of ARPA Network Text Messages.* RFC 733, NIC 41952. In [Feinler78]. November 1977.

[Crocker77b] Crocker, D. H., *Framework and Functions of the MS Personal Message System.* R-2134-ARPA, Rand Corporation, Santa Monica, California. 1977.

[Crocker79] Crocker, D. H., Szurkowski, E. S., and Farber, D. J., *An Internetwork Memo Distribution Facility - MMDF.* 6th Data Communication Symposium, Asilomar. November 1979.

[Crocker82] Crocker, D. H., *Standard for the Format of Arpa Internet Text Messages.* RFC 822. Network Information Center, SRI International, Menlo Park, California. August 1982.

[Metcalfe76] Metcalfe, R., and Boggs, D., "Ethernet: Distributed Packet Switching for Local Computer Networks", *Communications of the ACM 19*, 7. July 1976.

[Feinler78] Feinler, E., and Postel, J. (eds.), *ARPANET Protocol Handbook.* NIC 7104, Network Information Center, SRI International, Menlo Park, California. 1978.

[NBS80] National Bureau of Standards, *Specification of a Draft Message Format Standard.* Report No. ICST/CBOS 80-2. October 1980.

[Neigus73] Neigus, N., *File Transfer Protocol for the ARPA Network.* RFC 542, NIC 17759. In [Feinler78]. August, 1973.

[Nowitz78a] Nowitz, D. A., and Lesk, M. E., *A Dial-Up Network of UNIX Systems.* Bell Laboratories. In UNIX Programmer's Manual, Seventh Edition, Volume 2. August, 1978.

[Nowitz78b] Nowitz, D. A., *Uucp Implementation Description.* Bell Laboratories. In UNIX Programmer's Manual, Seventh Edition, Volume 2. October, 1978.

[Postel74] Postel, J., and Neigus, N., Revised FTP Reply Codes. RFC 640, NIC 30843. In [Feinler78]. June, 1974.

[Postel77] Postel, J., *Mail Protocol.* NIC 29588. In [Feinler78]. November 1977.

[Postel79a]	Postel, J., *Internet Message Protocol.* RFC 753, IEN 85. Network Information Center, SRI International, Menlo Park, California. March 1979.
[Postel79b]	Postel, J. B., *An Internetwork Message Structure.* In *Proceedings of the Sixth Data Communications Symposium,* IEEE. New York. November 1979.
[Postel80]	Postel, J. B., *A Structured Format for Transmission of Multi-Media Documents.* RFC 767. Network Information Center, SRI International, Menlo Park, California. August 1980.
[Postel82]	Postel, J. B., *Simple Mail Transfer Protocol.* RFC821 (obsoleting RFC788). Network Information Center, SRI International, Menlo Park, California. August 1982.
[Schmidt79]	Schmidt, E., *An Introduction to the Berkeley Network.* University of California, Berkeley California. 1979.
[Shoens79]	Shoens, K., *Mail Reference Manual.* University of California, Berkeley. In UNIX Programmer's Manual, Seventh Edition, Volume 2C. December 1979.
[Sluizer81]	Sluizer, S., and Postel, J. B., *Mail Transfer Protocol.* RFC 780. Network Information Center, SRI International, Menlo Park, California. May 1981.
[Solomon81]	Solomon, M., Landweber, L., and Neuhengen, D., "The Design of the CSNET Name Server." CS-DN-2, University of Wisconsin, Madison. November 1981.
[Su82]	Su, Zaw-Sing, and Postel, Jon, *The Domain Naming Convention for Internet User Applications.* RFC819. Network Information Center, SRI International, Menlo Park, California. August 1982.
[UNIX83]	*The UNIX Programmer's Manual, Seventh Edition,* Virtual VAX-11 Version, Volume 1. Bell Laboratories, modified by the University of California, Berkeley, California. March, 1983.

SENDMAIL

INSTALLATION AND OPERATION GUIDE

Version 4.2

TABLE OF CONTENTS

SENDMAIL

INSTALLATION AND OPERATION GUIDE

Eric Allman
Britton-Lee, Inc.

Version 4.2

Sendmail implements a general purpose internetwork mail routing facility under the UNIX* operating system. It is not tied to any one transport protocol – its function may be likened to a crossbar switch, relaying messages from one domain into another. In the process, it can do a limited amount of message header editing to put the message into a format that is appropriate for the receiving domain. All of this is done under the control of a configuration file.

Due to the requirements of flexibility for *sendmail*, the configuration file can seem somewhat unapproachable. However, there are only a few basic configurations for most sites, for which standard configuration files have been supplied. Most other configurations can be built by adjusting an existing configuration file incrementally.

Although *sendmail* is intended to run without the need for monitoring, it has a number of features that may be used to monitor or adjust the operation under unusual circumstances. These features are described.

Section one describes how to do a basic *sendmail* installation. Section two explains the day-to-day information you should know to maintain your mail system. If you have a relatively normal site, these two sections should contain sufficient information for you to install *sendmail* and keep it happy. Section three describes some parameters that may be safely tweaked. Section four has information regarding the command line arguments. Section five contains the nitty-gritty information about the configuration file. This section is for masochists and people who must write their own configuration file. The appendices give a brief but detailed explanation of a number of features not described in the rest of the paper.

The references in this paper are actually found in the companion paper *Sendmail – An Internetwork Mail Router*. Read that paper before this one to gain a basic understanding of how the pieces fit together.

1. BASIC INSTALLATION

There are two basic steps to installing *sendmail*. The hard part is to build the configuration table. This is a file that *sendmail* reads when it starts up that describes the mailers it knows about, how to parse addresses, how to rewrite the message header, and the settings of various options. Although the configuration table is quite complex, a configuration can usually be built by adjusting an existing off-the-shelf configuration. The second part is actually doing the installation, i.e., creating the necessary files, etc.

The remainder of this section will describe the installation of *sendmail* assuming you can use one of the existing configurations and that the standard installation parameters are acceptable. All pathnames and examples are given from the root of the *sendmail* subtree.

*UNIX is a trademark of Bell Laboratories.

1.1. Off-The-Shelf Configurations

You can produce your own *sendmail.cf* file by editing the generic configuration file provided with this release: */usr/lib/sendmail.generic.cf*. Procedures are given in the *Setting Up the Mail System* section of the *System Set-up and Operation* chapter.

1.2. Installation Using the Makefile

A makefile exists in the root of the *sendmail* directory that will do all of these steps for a 4.2BSD system. It may have to be slightly tailored for use on other systems.

Before using this makefile, you should already have created your configuration file and left it in the file "cf/*system*.cf" where *system* is the name of your system (i.e., what is returned by *hostname*(1)). If you do not have *hostname* you can use the declaration "HOST=*system*" on the *make*(1) command line. You should also examine the file *md/config.m4* and change the *m4* macros there to reflect any libraries and compilation flags you may need.

The basic installation procedure is to type:

```
make
make install
```

in the root directory of the *sendmail* distribution. This will make all binaries and install them in the standard places. The second *make* command must be executed as the superuser (root).

1.3. Installation by Hand

Along with building a configuration file, you will have to install the *sendmail* startup into your UNIX system. If you are doing this installation in conjunction with a regular Berkeley UNIX install, these steps will already be complete. Many of these steps will have to be executed as the superuser (root).

1.3.1. /usr/lib/sendmail

The binary for *sendmail* is located in */usr/lib*.

1.3.2. /usr/lib/sendmail.cf

The configuration file that you created earlier should be installed in */usr/lib/sendmail.cf*; see the *Setting up the Mail System* section in the *System Set-up and Operation* chapter.

1.3.3. /usr/ucb/newaliases

If you are running delivermail, it is critical that the *newaliases* command be replaced. This can just be a link to *sendmail*:

```
rm -f /usr/ucb/newaliases
ln /usr/lib/sendmail /usr/ucb/newaliases
```

1.3.4. /usr/lib/sendmail.cf

The configuration file must be installed in */usr/lib*. This is described above.

1.3.5. /usr/spool/mqueue

The directory */usr/spool/mqueue* should be created to hold the mail queue. This directory should be mode 777 unless *sendmail* is run setuid, when *mqueue* should be owned by the *sendmail* owner and mode 755.

1.3.6. /usr/lib/aliases*

The file "/usr/lib/aliases" is the master file for system aliases. This file is a symbolic link to */private/aliases*, used for clients.

1.3.7. /usr/lib/sendmail.fc

If you intend to install the frozen version of the configuration file (for quick startup) you should create the file */usr/lib/sendmail.fc* and initialize it. This step may be safely skipped.

```
cp /dev/null /usr/lib/sendmail.fc
/usr/lib/sendmail -bz
```

1.3.8. /etc/rc

It will be necessary to start up the *sendmail* daemon when your system reboots. This daemon performs two functions: it listens on the SMTP socket for connections (to receive mail from a remote system) and it processes the queue periodically to insure that mail gets delivered when hosts come up.

Add the following lines to */etc/rc* (or */etc/rc.local* as appropriate) in the area where it is starting up the daemons:

```
if [ -f /usr/lib/sendmail ]; then
        (cd /usr/spool/mqueue; rm -f [lnx]f*)
        /usr/lib/sendmail -bd -q30m &
        echo -n ' sendmail' >/dev/console
fi
```

The *cd* and *rm* commands insure that all lock files have been removed; extraneous lock files may be left around if the system goes down in the middle of processing a message. The line that actually invokes *sendmail* has two flags: **-bd** causes it to listen on the SMTP port, and **-q30m** causes it to run the queue every half hour.

If you are not running a version of UNIX that supports Berkeley TCP/IP, do not include the **-bd** flag.

1.3.9. /usr/lib/sendmail.hf

This is the help file used by the SMTP **HELP** command. The file is already installed in the distribution.

1.3.10. /usr/lib/sendmail.st

If you wish to collect statistics about your mail traffic, create the file */usr/lib/sendmail.st*:

```
cp /dev/null /usr/lib/sendmail.st
chmod 666 /usr/lib/sendmail.st
```

This file does not grow. It is printed with the program *aux/mailstats.*

1.3.11. /etc/syslog

You may want to run the *syslog* program (to collect log information about *sendmail*). This program normally resides in */etc/syslog,* with support files */etc/syslog.conf* and */etc/syslog.pid*. The file */etc/syslog.conf* describes the file(s) that *sendmail* will log in. For a complete description of syslog, see the manual page for *syslog*(8).

1.3.12. /usr/ucb/newaliases

If *sendmail* is invoked as "newaliases," it will simulate the **-bi** flag (that is, will rebuild the alias database; see below). This should be a link to */usr/lib/sendmail.*

1.3.13. /usr/ucb/mailq

If *sendmail* is invoked as *mailq,* it will simulate the **–bp** flag (that is, *sendmail* will print the contents of the mail queue; see below). This should be a link to */usr/lib/sendmail*.

2. NORMAL OPERATIONS

2.1. Quick Configuration Startup

A fast version of the configuration file may be set up by using the **–bz** flag:

```
/usr/lib/sendmail -bz
```

This creates the file */usr/lib/sendmail.fc* ("frozen configuration"). This file is an image of *sendmail*'s data space after reading in the configuration file. If this file exists, it is used instead of */usr/lib/sendmail.cf.sendmail.fc* time *sendmail.cf* is changed.

The frozen configuration file will be ignored if a **–C** flag is specified or if *sendmail* detects that it is out of date. However, the heuristics are not strong so this should not be trusted.

2.2. The System Log

The system log is supported by the *syslog*(8) program.

2.2.1. Format

Each line in the system log consists of a timestamp, the name of the machine that generated it (for logging from several machines over the ethernet), the word "sendmail:", and a message.

2.2.2. Levels

If you have *syslog*(8) or an equivalent installed, you will be able to do logging. There is a large amount of information that can be logged. The log is arranged as a succession of levels. At the lowest level only extremely strange situations are logged. At the highest level, even the most mundane and uninteresting events are recorded for posterity. As a convention, log levels under ten are considered "useful;" log levels above ten are usually for debugging purposes.

A complete description of the log levels is given in section 4.3.

2.3. The Mail Queue

The mail queue should be processed transparently. However, you may find that manual intervention is sometimes necessary. For example, if a major host is down for a period of time the queue may become clogged. Although *sendmail* ought to recover gracefully when the host comes up, you may find performance unacceptably bad in the meantime.

2.3.1. Printing the queue

The contents of the queue can be printed using the *mailq* command (or by specifying the **–bp** flag to *sendmail*):

```
mailq
```

This will produce a listing of the queue id's, the size of the message, the date the message entered the queue, and the sender and recipients.

2.3.2. Format of queue files

All queue files have the form *xfAA99999* where *AA99999* is the *id* for this file and the *x* is a type. The types are:

d	The data file. The message body (excluding the header) is kept in this file.
l	The lock file. If this file exists, the job is currently being processed, and a queue run will not process the file. For that reason, an extraneous *lf* file can cause a job to apparently disappear (it will not even time out!).
n	This file is created when an id is being created. It is a separate file to insure that no mail can ever be destroyed due to a race condition. It should exist for no more than a few milliseconds at any given time.
q	The queue control file. This file contains the information necessary to process the job.
t	A temporary file. These are an image of the *qf* file when it is being rebuilt. It should be renamed to a *qf* file very quickly.
x	A transcript file, existing during the life of a session showing everything that happens during that session.

The *qf* file is structured as a series of lines each beginning with a code letter. The lines are as follows:

D	The name of the data file. There may only be one of these lines.
H	A header definition. There may be any number of these lines. The order is important: they represent the order in the final message. These use the same syntax as header definitions in the configuration file.
R	A recipient address. This will normally be completely aliased, but is actually realiased when the job is processed. There will be one line for each recipient.
S	The sender address. There may only be one of these lines.
T	The job creation time. This is used to compute when to time out the job.
P	The current message priority. This is used to order the queue. Higher numbers mean lower priorities. The priority increases as the message sits in the queue. The initial priority depends on the message class and the size of the message.
M	A message. This line is printed by the *mailq* command, and is generally used to store status information. It can contain any text.

As an example, the following is a queue file sent to "mckusick@calder" and "wnj":

```
DdfA13557
Seric
T404261372
P132
Rmckusick@calder
Rwnj
H?D?date: 23-Oct-82 15:49:32-PDT (Sat)
H?F?from: eric (Eric Allman)
H?x?full-name: Eric Allman
Hsubject: this is an example message
Hmessage-id: <8209232249.13557@UCBARPA.BERKELEY.ARPA>
Hreceived: by UCBARPA.BERKELEY.ARPA (3.227 [10/22/82])
        id A13557; 23-Oct-82 15:49:32-PDT (Sat)
Hphone: (415) 548-3211
HTo: mckusick@calder, wnj
```

This shows the name of the data file, the person who sent the message, the submission time (in seconds since January 1, 1970), the message priority, the message class, the recipients, and the headers for the message.

2.3.3. Forcing the queue

Sendmail should run the queue automatically at intervals. The algorithm is to read and sort the queue, and then to attempt to process all jobs in order. When it attempts to run the job, *sendmail* first checks to see if the job is locked. If so, it ignores the job.

There is no attempt to insure that only one queue processor exists at any time, since there is no guarantee that a job cannot take forever to process. Due to the locking algorithm, it is impossible for one job to freeze the queue. However, an uncooperative recipient host or a program recipient that never returns can accumulate many processes in your system. Unfortunately, there is no way to resolve this without violating the protocol.

In some cases, you may find that a major host going down for a couple of days may create a prohibitively large queue. This will result in *sendmail* spending an inordinate amount of time sorting the queue. This situation can be fixed by moving the queue to a temporary place and creating a new queue. The old queue can be run later when the offending host returns to service.

To do this, it is acceptable to move the entire queue directory:

```
cd /usr/spool
mv mqueue omqueue; mkdir mqueue; chmod 777 mqueue
```

You should then kill the existing daemon (since it will still be processing in the old queue directory) and create a new daemon.

To run the old mail queue, run the following command:

```
/usr/lib/sendmail -oQ/usr/spool/omqueue -q
```

The **-oQ** flag specifies an alternate queue directory and the **-q** flag says to just run every job in the queue. If you have a tendency toward voyeurism, you can use the **-v** flag to watch what is going on.

When the queue is finally emptied, you can remove the directory:

```
rmdir /usr/spool/omqueue
```

2.4. The Alias Database

The alias database exists in two forms. One is a text form, maintained in the file */usr/lib/aliases.* The aliases are of the form

```
name: name1, name2, ...
```

Only local names may be aliased; e.g.,

```
eric@mit-xx: eric@berkeley
```

will not have the desired effect. Aliases may be continued by starting any continuation lines with a space or a tab. Blank lines and lines beginning with a sharp sign ("#") are comments.

The second form is processed by the *dbm*(3) library. This form is in the files */usr/lib/aliases.dir* and */usr/lib/aliases.pag.* This is the form that *sendmail* actually uses to resolve aliases. This technique is used to improve performance.

2.4.1. Rebuilding the alias database

The DBM version of the database may be rebuilt explicitly by executing the command

```
newaliases
```

This is equivalent to giving *sendmail* the **-bi** flag:

```
/usr/lib/sendmail -bi
```

If the "D" option is specified in the configuration, *sendmail* will rebuild the alias database automatically if possible when it is out of date. The conditions under which it will do this are:

(1) The DBM version of the database is mode 666. --or--

(2) *Sendmail* is running setuid to root.

Auto-rebuild can be dangerous on heavily loaded machines with large alias files; if it might take more than five minutes to rebuild the database, there is a chance that several processes will start the rebuild process simultaneously.

2.4.2. Potential problems

There are a number of problems that can occur with the alias database. They all result from a *sendmail* process accessing the DBM version while it is only partially built. This can happen under two circumstances: One process accesses the database while another process is rebuilding it, or the process rebuilding the database dies (due to being killed or a system crash) before completing the rebuild.

Sendmail has two techniques to try to relieve these problems. First, it ignores interrupts while rebuilding the database; this avoids the problem of someone aborting the process leaving a partially rebuilt database. Second, at the end of the rebuild it adds an alias of the form

```
@: @
```

(which is not normally legal). Before *sendmail* will access the database, it checks to insure that this entry exists[1]. It will wait up to five minutes for this entry to appear, at which point it will force a rebuild itself[2].

2.4.3. List owners

If an error occurs on sending to a certain address, say "*x*", *sendmail* will look for an alias of the form "owner-*x*" to receive the errors. This is typically useful for a mailing list where the submitter of the list has no control over the maintanence of the list itself; in this case the list maintainer would be the owner of the list. For example:

```
unix-wizards: eric@ucbarpa, wnj@monet, nosuchuser,
        sam@matisse
owner-unix-wizards: eric@ucbarpa
```

would cause "eric@ucbarpa" to get the error that will occur when someone sends to unix-wizards due to the inclusion of "nosuchuser" on the list.

2.5. Per-User Forwarding (.forward Files)

As an alternative to the alias database, any user may put a file with the name ".forward" in his or her home directory. If this file exists, *sendmail* redirects mail for that user to the list of addresses listed in the .forward file. For example, if the home directory for user "mckusick" has a .forward file with contents:

[1]The "a" option is required in the configuration for this action to occur. This should normally be specified unless you are running *delivermail* in parallel with *sendmail.*

[2]Note: the "D" option must be specified in the configuration file for this operation to occur.

```
mckusick@ernie
kirk@calder
```

then any mail arriving for "mckusick" will be redirected to the specified accounts.

2.6. Special Header Lines

Several header lines have special interpretations defined by the configuration file. Others have interpretations built into *sendmail* that cannot be changed without changing the code. These builtins are described here.

2.6.1. Return-Receipt-To:

If this header is sent, a message will be sent to any specified addresses when the final delivery is complete. if the mailer has the **l** flag (local delivery) set in the mailer descriptor.

2.6.2. Errors-To:

If errors occur anywhere during processing, this header will cause error messages to go to the listed addresses rather than to the sender. This is intended for mailing lists.

2.6.3. Apparently-To:

If a message comes in with no recipients listed in the message (in a To:, Cc:, or Bcc: line) then *sendmail* will add an "Apparently-To:" header line for any recipients it is aware of. This is not put in as a standard recipient line to warn any recipients that the list is not complete.

At least one recipient line is required under RFC 822.

3. ARGUMENTS

The complete list of arguments to *sendmail* is described in detail in Appendix A. Some important arguments are described here.

3.1. Queue Interval

The amount of time between forking a process to run through the queue is defined by the **–q** flag. If you run in mode **f** or **a** this can be relatively large, since it will only be relevant when a host that was down comes back up. If you run in **q** mode it should be relatively short, since it defines the maximum amount of time that a message may sit in the queue.

3.2. Daemon Mode

If you allow incoming mail over an IPC connection, you should have a daemon running. This should be set by your */etc/rc* file using the **–bd** flag. The **–bd** flag and the **–q** flag may be combined in one call:

```
/usr/lib/sendmail -bd -q30m
```

3.3. Forcing the Queue

In some cases you may find that the queue has gotten clogged for some reason. You can force a queue run using the **–q** flag (with no value). It is entertaining to use the **–v** flag (verbose) when this is done to watch what happens:

```
/usr/lib/sendmail -q -v
```

3.4. Debugging

There are a fairly large number of debug flags built into *sendmail*. Each debug flag has a number and a level, where higher levels means to print out more information. The convention is that levels greater than nine are "absurd," because they print out so much information that you wouldn't normally want to see them except for debugging that particular piece of code. Debug flags are set using the **–d** option; the syntax is:

```
debug-flag:   –d debug-list
debug-list:   debug-option [ , debug-option ]
debug-option: debug-range [ . debug-level ]
debug-range:  integer | integer – integer
debug-level:  integer
```

where spaces are for reading ease only. For example,

–d12	Set flag 12 to level 1
–d12.3	Set flag 12 to level 3
–d3-17	Set flags 3 through 17 to level 1
–d3-17.4	Set flags 3 through 17 to level 4

For a complete list of the available debug flags you will have to look at the code (they are too dynamic to keep this documentation up to date).

3.5. Trying a Different Configuration File

An alternative configuration file can be specified using the **–C** flag; for example,

```
/usr/lib/sendmail –Ctest.cf
```

uses the configuration file *test.cf* instead of the default */usr/lib/sendmail.cf*. If the **–C** flag has no value it defaults to *sendmail.cf* in the current directory.

3.6. Changing the Values of Options

Options can be overridden using the **–o** flag. For example,

```
/usr/lib/sendmail –oT2m
```

sets the **T** (timeout) option to two minutes for this run only.

4. TUNING

There are a number of configuration parameters you may want to change, depending on the requirements of your site. Most of these are set using an option in the configuration file. For example, the line "OT3d" sets option **T** to the value "3d" (three days).

4.1. Timeouts

All time intervals are set using a scaled syntax. For example, "10m" represents ten minutes, whereas "2h30m" represents two and a half hours. The full set of scales is:

s	seconds
m	minutes
h	hours
d	days
w	weeks

4.1.1. Queue interval

The argument to the **–q** flag specifies how often a subdaemon will run the queue. This is typically set to between five minutes and one half hour.

4.1.2. Read timeouts

It is possible to time out when reading the standard input or when reading from a remote SMTP server. Technically, this is not acceptable within the published protocols. However, it might be appropriate to set it to something large in certain environments (such as an hour). This will reduce the chance of large numbers of idle daemons piling up on your system. This timeout is set using the **r** option in the configuration file.

4.1.3. Message timeouts

After sitting in the queue for a few days, a message will time out. This is to insure that at least the sender is aware of the inability to send a message. The timeout is typically set to three days. This timeout is set using the **T** option in the configuration file.

The time of submission is set in the queue, rather than the amount of time left until timeout. As a result, you can flush messages that have been hanging for a short period by running the queue with a short message timeout. For example,

```
/usr/lib/sendmail -oT1d -q
```

will run the queue and flush anything that is one day old.

4.2. Delivery Mode

There are a number of delivery modes that *sendmail* can operate in, set by the "d" configuration option. These modes specify how quickly mail will be delivered. Legal modes are:

i	deliver interactively (synchronously)
b	deliver in background (asynchronously)
q	queue only (don't deliver)

There are tradeoffs. Mode "i" passes the maximum amount of information to the sender, but is hardly ever necessary. Mode "q" puts the minimum load on your machine, but means that delivery may be delayed for up to the queue interval. Mode "b" is probably a good compromise. However, this mode can cause large numbers of processes if you have a mailer that takes a long time to deliver a message.

4.3. Log Level

The level of logging can be set for *sendmail.* The default using a standard configuration table is level 9. The levels are as follows:

0	No logging.
1	Major problems only.
2	Message collections and failed deliveries.
3	Successful deliveries.
4	Messages being defered (due to a host being down, etc.).
5	Normal message queueups.
6	Unusual but benign incidents, e.g., trying to process a locked queue file.
9	Log internal queue id to external message id mappings. This can be useful for tracing a message as it travels between several hosts.
12	Several messages that are basically only of interest when debugging.
16	Verbose information regarding the queue.

4.4. File Modes

There are a number of files that may have a number of modes. The modes depend on what functionality you want and the level of security you require.

4.4.1. To suid or not to suid?

Sendmail can safely be made setuid to root. At the point where it is about to *exec*(2) a mailer, it checks to see if the userid is zero; if so, it resets the userid and groupid to a default (set by the **u** and **g** options). (This can be overridden by setting the **S** flag to the mailer for mailers that are trusted and must be called as root.) However, this will cause mail processing to be accounted (using *sa*(8)) to root rather than to the user sending the mail.

4.4.2. Temporary file modes

The mode of all temporary files that *sendmail* creates is determined by the **F** option. Reasonable values for this option are 0600 and 0644. If the more permissive mode is selected, it will not be necessary to run *sendmail* as root at all (even when running the queue).

4.4.3. Should my alias database be writable?

At Sun Microsystems, we have the alias database (*/usr/lib/aliases**) mode 666. There are some dangers inherent in this approach: any user can add him-/her-self to any list, or can read any other user's mail. However, we have found users to be basically trustworthy, and the cost of having a read-only database greater than the expense of finding and eradicating the rare nasty person.

5. THE WHOLE SCOOP ON THE CONFIGURATION FILE

This section describes the configuration file in detail, including hints on how to write one of your own if you have to.

There is one point that should be made clear immediately: the syntax of the configuration file is designed to be reasonably easy to parse, since this is done every time *sendmail* starts up, rather than easy for a human to read or write. On the "future project" list is a configuration-file compiler.

An overview of the configuration file is given first, followed by details of the semantics.

5.1. The Syntax

The configuration file is organized as a series of lines, each of which begins with a single character defining the semantics for the rest of the line. Lines beginning with a space or a tab are continuation lines (although the semantics are not well defined in many places). Blank lines and lines beginning with a sharp symbol ('#') are comments.

5.1.1. R and S – rewriting rules

The core of address parsing are the rewriting rules. These are an ordered production system. *Sendmail* scans through the set of rewriting rules looking for a match on the left hand side (LHS) of the rule. When a rule matches, the address is replaced by the right hand side (RHS) of the rule.

There are several sets of rewriting rules. Some of the rewriting sets are used internally and must have specific semantics. Other rewriting sets do not have specifically assigned semantics, and may be referenced by the mailer definitions or by other rewriting sets.

The syntax of these two commands are:

S*n*

Sets the current ruleset being collected to *n*. If you begin a ruleset more than once it deletes the old definition.

R*lhs rhs comments*

The fields must be separated by at least one tab character; there may be embedded spaces in the fields. The *lhs* is a pattern that is applied to the input. If it matches, the input is rewritten to the *rhs*. The *comments* are ignored.

5.1.2. D – define macro

Macros are named with a single character. These may be selected from the entire ASCII set, but user-defined macros should be selected from the set of upper case letters only. Lower case letters and special symbols are used internally.

The syntax for macro definitions is:

D*x val*

where *x* is the name of the macro and *val* is the value it should have. Macros can be interpolated in most places using the escape sequence **$***x*.

5.1.3. C and F – define classes

Classes of words may be defined to match on the left hand side of rewriting rules. For example a class of all local names for this site might be created so that attempts to send to oneself can be eliminated. These can either be defined directly in the configuration file or read in from another file. Classes may be given names from the set of upper case letters. Lower case letters and special characters are reserved for system use.

The syntax is:

C*c word1 word2...*
F*c file* [*format*]

The first form defines the class *c* to match any of the named words. It is permissible to split them among multiple lines; for example, the two forms:

```
CHmonet ucbmonet
```

and

```
CHmonet
CHucbmonet
```

are equivalent. The second form reads the elements of the class *c* from the named *file*; the *format* is a *scanf*(3) pattern that should produce a single string.

5.1.4. M – define mailer

Programs and interfaces to mailers are defined in this line. The format is:

M*name, {field=value}**

where *name* is the name of the mailer (used internally only) and the "field=name" pairs define attributes of the mailer. Fields are:

Path	The pathname of the mailer
Flags	Special flags for this mailer
Sender	A rewriting set for sender addresses
Recipient	A rewriting set for recipient addresses
Argv	An argument vector to pass to this mailer
Eol	The end-of-line string for this mailer
Maxsize	The maximum message length to this mailer

Only the first character of the field name is checked.

5.1.5. H – define header

The format of the header lines that *sendmail* inserts into the message are defined by the **H** line. The syntax of this line is:

H[**?***mflags***?**]*hname***:** *htemplate*

Continuation lines in this spec are reflected directly into the outgoing message. The *htemplate* is macro expanded before insertion into the message. If the *mflags* (surrounded by question marks) are specified, at least one of the specified flags must be stated in the mailer definition for this header to be automatically output. If one of these headers is in the input it is reflected to the output regardless of these flags.

Some headers have special semantics that will be described below.

5.1.6. O – set option

There are a number of "random" options that can be set from a configuration file. Options are represented by single characters. The syntax of this line is:

O*o value*

This sets option *o* to be *value.* Depending on the option, *value* may be a string, an integer, a boolean (with legal values "t", "T", "f", or "F"; the default is TRUE), or a time interval.

5.1.7. T – define trusted users

Trusted users are those users who are permitted to override the sender address using the **–f** flag. These typically are "root," "uucp," and "network," but on some users it may be convenient to extend this list to include other users, perhaps to support a separate UUCP login for each host. The syntax of this line is:

T*user1 user2...*

There may be more than one of these lines.

5.1.8. P – precedence definitions

Values for the "Precedence:" field may be defined using the **P** control line. The syntax of this field is:

P*name***=***num*

When the *name* is found in a "Precedence:" field, the message class is set to *num.* Higher numbers mean higher precedence. Numbers less than zero have the special property that error messages will not be returned. The default precedence is zero. For example, our list of precedences is:

```
Pfirst-class=0
Pspecial-delivery=100
Pjunk=-100
```

5.2. The Semantics

This section describes the semantics of the configuration file.

5.2.1. Special macros, conditionals

Macros are interpolated using the construct **$***x*, where *x* is the name of the macro to be interpolated. In particular, lower case letters are reserved to have special semantics, used to pass information in or out of *sendmail,* and some special characters are reserved to provide conditionals, etc.

The following macros *must* be defined to transmit information into *sendmail:*

e	The SMTP entry message
j	The "official" domain name for this site
l	The format of the UNIX from line
n	The name of the daemon (for error messages)
o	The set of "operators" in addresses
q	default format of sender address

The **$e** macro is printed out when SMTP starts up. The first word must be the **$j** macro. The **$j** macro should be in RFC821 format. The **$l** and **$n** macros can be considered constants except under terribly unusual circumstances. The **$o** macro consists of a list of characters which will be considered tokens and which will separate tokens when doing parsing. For example, if "r" were in the **$o** macro, then the input "address" would be scanned as three tokens: "add," "r," and "ess." Finally, the **$q** macro specifies how an address should appear in a message when it is defaulted. For example, on our system these definitions are:

```
De$j Sendmail $v ready at $b
DnMAILER-DAEMON
DlFrom $g  $d
Do.:%@!^=/
Dq$g$?x ($x)$.
Dj$H.$D
```

An acceptable alternative for the **$q** macro is "$?x$x $.<$g>". These correspond to the following two formats:

```
eric@Berkeley (Eric Allman)
Eric Allman <eric@Berkeley>
```

Some macros are defined by *sendmail* for interpolation into argv's for mailers or for other contexts. These macros are:

a	The origination date in Arpanet format
b	The current date in Arpanet format
c	The hop count
d	The date in UNIX (ctime) format
f	The sender (from) address
g	The sender address relative to the recipient
h	The recipient host
i	The queue id
p	Sendmail's pid
r	Protocol used
s	Sender's host name
t	A numeric representation of the current time
u	The recipient user
v	The version number of *sendmail*
w	The hostname of this site
x	The full name of the sender
y	The id of the sender's tty
z	The home directory of the recipient

There are three types of dates that can be used. The **$a** and **$b** macros are in Arpanet format; **$a** is the time as extracted from the "Date:" line of the message (if there was one), and **$b** is the current date and time (used for postmarks). If no "Date:" line is found in the incoming message, **$a** is set to the current time also. The **$d** macro is equivalent to the **$a** macro in UNIX (ctime) format.

The **$f** macro is the id of the sender as originally determined; when mailing to a specific host the **$g** macro is set to the address of the sender *relative to the recipient.* For example, if I send to "bollard@matisse" from the machine "ucbarpa" the **$f** macro will be "eric" and the **$g** macro will be "eric@ucbarpa."

The **$x** macro is set to the full name of the sender. This can be determined in several ways. It can be passed as flag to *sendmail.* The second choice is the value of the "Full-name:" line in the header if it exists, and the third choice is the comment field of a "From:" line. If all of these fail, and if the message is being originated locally, the full name is looked up in the */etc/passwd* file.

When sending, the **$h**, **$u**, and **$z** macros get set to the host, user, and home directory (if local) of the recipient. The first two are set from the **$@** and **$:** part of the rewriting rules, respectively.

The **$p** and **$t** macros are used to create unique strings (e.g., for the "Message-Id:" field). The **$i** macro is set to the queue id on this host; if put into the timestamp line it can be extremely useful for tracking messages. The **$y** macro is set to the id of the terminal of the sender (if known); some systems like to put this in the Unix "From" line. The **$v** macro is set to be the version number of *sendmail;* this is normally put in timestamps and has been proven extremely useful for debugging. The **$w** macro is set to the name of this host if it can be determined. The **$c** field is set to the "hop count," i.e., the number of times this message has been processed. This can be determined by the **–h** flag on the command line or by counting the timestamps in the message.

The **$r** and **$s** fields are set to the protocol used to communicate with *sendmail* and the sending hostname; these are not supported in the current version.

Conditionals can be specified using the syntax:

$**?x text1 **$| text2 **$.**

This interpolates *text1* if the macro **$x** is set, and *text2* otherwise. The "else" (**$|**) clause may be omitted.

5.2.2. Special classes

The class **$=w** is set to be the set of all names this host is known by. This can be used to delete local hostnames.

5.2.3. The left hand side

The left hand side of rewriting rules contains a pattern. Normal words are simply matched directly. Metasyntax is introduced using a dollar sign. The metasymbols are:

$*	Match zero or more tokens
$+	Match one or more tokens
$–	Match exactly one token
$=*x*	Match any token in class *x*
$˜*x*	Match any token not in class *x*

If any of these match, they are assigned to the symbol **$***n* for replacement on the right hand side, where *n* is the index in the LHS. For example, if the LHS:

$–:$+

is applied to the input:

UCBARPA:eric

the rule will match, and the values passed to the RHS will be:

$1	UCBARPA
$2	eric

5.2.4. The right hand side

When the right hand side of a rewriting rule matches, the input is deleted and replaced by the right hand side. Tokens are copied directly from the RHS unless they are begin with a dollar sign. Metasymbols are:

$*n*	Substitute indefinite token *n* from LHS
$>*n*	"Call" ruleset *n*
$#*mailer*	Resolve to *mailer*
$@*host*	Specify *host*
$:*user*	Specify *user*

The $*n* syntax substitutes the corresponding value from a $+, $–, $*, $=, or $~ match on the LHS. It may be used anywhere.

The $>*n* syntax causes the remainder of the line to be substituted as usual and then passed as the argument to ruleset *n*. The final value of ruleset *n* then becomes the substitution for this rule.

The $# syntax should *only* be used in ruleset zero. It causes evaluation of the ruleset to terminate immediately, and signals to *sendmail* that the address has completely resolved. The complete syntax is:

$#*mailer*$@*host*$:*user*

This specifies the {mailer, host, user} 3-tuple necessary to direct the mailer. If the mailer is local the host part may be omitted. The *mailer* and *host* must be a single word, but the *user* may be multi-part.

A RHS may also be preceeded by a $@ or a $: to control evaluation. A $@ prefix causes the ruleset to return with the remainder of the RHS as the value. A $: prefix causes the rule to terminate immediately, but the ruleset to continue; this can be used to avoid continued application of a rule. The prefix is stripped before continuing.

The $@ and $: prefixes may preceed a $> spec; for example:

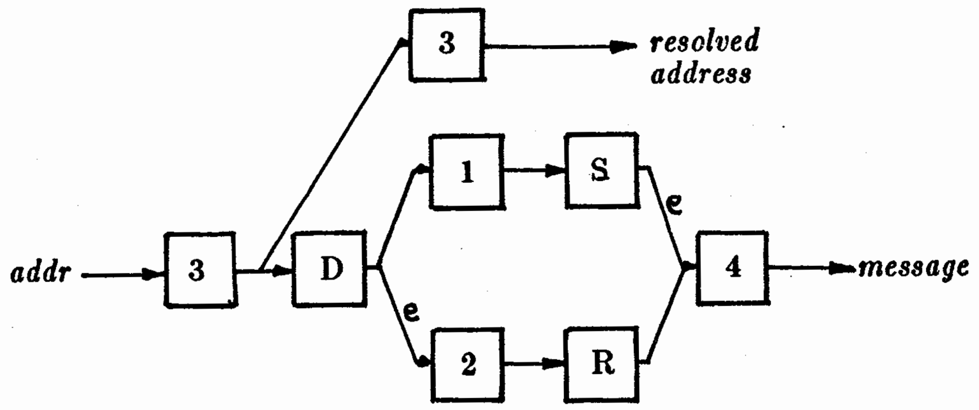
```
R$+      $:$>7$1
```

matches anything, passes that to ruleset seven, and continues; the $: is necessary to avoid an infinite loop.

5.2.5. Semantics of rewriting rule sets

There are five rewriting sets that have specific semantics. These are related as depicted by Figure 2.

Ruleset three should turn the address into "canonical form." This form should have the basic syntax:



local-part@host-domain-spec

If no "@" sign is specified, then the host-domain-spec *may* be appended from the sender address (if the **C** flag is set in the mailer definition corresponding to the *sending* mailer). Ruleset three is applied by *sendmail* before doing anything with any address.

Ruleset zero is applied after ruleset three to addresses that are going to actually specify recipients. It must resolve to a {*mailer, host, user*} triple. The *mailer* must be defined in the mailer definitions from the configuration file. The *host* is defined into the **$h** macro for use in the argv expansion of the specified mailer.

Rulesets one and two are applied to all sender and recipient addresses respectively. They are applied before any specification in the mailer definition. They must never resolve.

Ruleset four is applied to all addresses in the message. It is typically used to translate internal to external form.

5.2.6. Mailer flags etc.

There are a number of flags that may be associated with each mailer, each identified by a letter of the alphabet. Many of them are assigned semantics internally. These are detailed in Appendix C. Any other flags may be used freely to conditionally assign headers to messages destined for particular mailers.

5.2.7. The "error" mailer

The mailer with the special name "error" can be used to generate a user error. The (optional) host field is a numeric exit status to be returned, and the user field is a message to be printed. For example, the entry:

$**#error**$:Host unknown in this domain

on the RHS of a rule will cause the specified error to be generated if the LHS matches. This mailer is only functional in ruleset zero.

5.3. Building a Configuration File From Scratch

Building a configuration table from scratch is an extremely difficult job. Fortunately, it is almost never necessary to do so; nearly every situation that may come up may be resolved by changing an existing table. In any case, it is critical that you understand what it is that you are trying to do and come up with a philosophy for the configuration table. This section is intended to explain what the real purpose of a configuration table is and to give you some ideas for what your philosophy might be.

5.3.1. What you are trying to do

The configuration table has three major purposes. The first and simplest is to set up the environment for *sendmail*. This involves setting the options, defining a few critical macros, etc. Since these are described in other places, we will not go into more detail here.

The second purpose is to rewrite addresses in the message. This should typically be done in two phases. The first phase maps addresses in any format into a canonical form. This should be done in ruleset three. The second phase maps this canonical form into the syntax appropriate for the receiving mailer. *Sendmail* does this in three subphases. Rulesets one and two are applied to all sender and recipient addresses respectively. After this, you may specify per-mailer rulesets for both sender and recipient addresses; this allows mailer-specific customization. Finally, ruleset four is applied to do any default conversion to external form.

The third purpose is to map addresses into the actual set of instructions necessary to get the message delivered. Ruleset zero must resolve to the internal form, which is in turn used as a pointer to a mailer descriptor. The mailer descriptor describes the interface requirements of the mailer.

5.3.2. Philosophy

The particular philosophy you choose will depend heavily on the size and structure of your organization. I will present a few possible philosophies here.

One general point applies to all of these philosophies: it is almost always a mistake to try to do full name resolution. For example, if you are trying to get names of the form "user@host" to the Arpanet, it does not pay to route them to "xyzvax!decvax!ucbvax!c70:user@host" since you then depend on several links not under your control. The best approach to this problem is to simply forward to "xyzvax!user@host" and let xyzvax worry about it from there. In summary, just get the message closer to the destination, rather than determining the full path.

5.3.2.1. Large site, many hosts – minimum information

Berkeley is an example of a large site; it has more than two or three hosts. Berkeley has decided that the only reasonable philosophy for their environment is to designate one host as site guru. It must be able to resolve any piece of mail it receives. The other sites should have the minimum amount of information they can get away with, and even this minimum should be hints rather than solid information.

For example, a typical site on the Berkeley local ether network is "monet." Monet has a list of known ethernet hosts; if it receives mail for any of them, it can do direct delivery. If it receives mail for any unknown host, it just passes it directly to "ucbvax," the Berkeley master host. Ucbvax may determine that the host name is illegal and reject the message, or may be able to do delivery. However, it is important to note that when a new ethernet host is added, the only host that *must* have its tables updated is ucbvax; the others *may* be updated as convenient, but this is not critical.

This picture is slightly muddied due to network connections that are not actually located on ucbvax. For example, the Berkeley TCP connection is currently on "ucbarpa." However, monet *does not* know about this; the information is hidden between ucbvax and ucbarpa. Mail going from monet to a TCP host is transferred via the ethernet from monet to ucbvax, then via the ethernet from ucbvax to ucbarpa, and then is submitted to the Arpanet. Although this involves some extra hops, Berkeley feels this is an acceptable tradeoff.

An interesting point is that it would be possible to update monet to send TCP mail directly to ucbarpa if the load got too high: if monet failed to note a host as a TCP host, the mail would go via ucbvax as before; and if monet incorrectly sent a message to ucbarpa, it would still be sent by ucbarpa to ucbvax as before. The only problem that might occur in this scenario is 'looping': if ucbarpa thought that ucbvax had the TCP connection and vice versa. For this reason, updates should *always* happen to the master host first.

This philosophy results as much from the need to have a single source for the configuration files (typically built using *m4*(1) or some similar tool) as any logical need. Maintaining more than three separate tables by hand is essentially an impossible job.

5.3.2.2. Small site – complete information

A small site (two or three hosts) may find it more reasonable to have complete information at each host. This would require that each host know exactly where each network connection is, possibly including the names of each host on that network. As long as the site remains small and the the configuration remains relatively static, the update problem will probably not be too great.

5.3.2.3. Single host

This is in some sense the trivial case. The only major issue is trying to insure that you don't have to know too much about your environment. For example, if you have a UUCP connection you might find it useful to know about the names of hosts connected directly to you, but this is really not necessary since this may be determined from the syntax.

5.3.3. Relevant issues

The canonical form you use should almost certainly be as specified in the Arpanet protocols RFC819 and RFC822. Copies of these RFC's are included on the *sendmail* tape as *doc/rfc819.lpr* and *doc/rfc822.lpr.*

RFC822 describes the format of the mail message itself. *Sendmail* follows this RFC closely, to the extent that many of the standards described in this document can not be changed without changing the code. In particular, the following characters have special interpretations:

< > ( ) " \

Any attempt to use these characters for other than their RFC822 purpose in addresses is probably doomed to disaster.

RFC819 describes the specifics of the domain-based addressing. This is touched on in RFC822 as well. Essentially each host is given a name which is a right-to-left dot qualified pseudo-path from a distinguished root. The elements of the path need not be physical hosts; the domain is logical rather than physical. For example, at Berkeley one legal host is "a.cc.berkeley.arpa". Reading from right to left, "arpa" is a top level domain (related to, but not limited to, the physical Arpanet), "berkeley" is both an Arpanet host and a logical domain which is actually interpreted by a host called ucbvax (the "major" host for this domain), "cc" represents the Computer Center, (in this case a strictly logical entity), and "a" is a host in the Computer Center; this particular host happens to be connected via berknet, the Berkeley network, but other hosts might be connected via one of two ethernets or some other network.

Be aware when reading RFC819 that there are a number of errors in it.

5.3.4. How to proceed

Once you have decided on a philosophy, it is worth examining the available configuration tables to decide if any of them are close enough to steal major parts of. Even under the worst of conditions, there is a fair amount of boiler plate that can be collected safely.

The next step is to build ruleset three. This will be the hardest part of the job. Beware of doing too much to the address in this ruleset, since anything you do will reflect through to the message. In particular, stripping of local domains is best deferred, since this can leave you with addresses with no domain spec at all. Since *sendmail* likes to append the sending domain to addresses with no domain, this can change the semantics of addresses. Also try to avoid fully qualifying domains in this ruleset. Although technically legal, this can lead to unpleasantly and unnecessarily long addresses reflected into messages. The Berkeley configuration files define ruleset nine to qualify domain names and strip local domains. This is called from ruleset zero to get all addresses into a cleaner form.

Once you have ruleset three finished, the other rulesets should be relatively trivial. If you need hints, examine the supplied configuration tables.

5.3.5. Testing the rewriting rules – the –bt flag

When you build a configuration table, you can do a certain amount of testing using the "test mode" of *sendmail.* For example, you could invoke *sendmail* as:

```
sendmail -bt -Ctest.cf
```

which would read the configuration file *test.cf* and enter test mode. In this mode, you enter lines of the form:

```
rwset address
```

where *rwset* is the rewriting set you want to use and *address* is an address to apply the set to. Test mode shows you the steps it takes as it proceeds, finally showing you the address it ends up with. You may use a comma separated list of rwsets for sequential application of rules to an input; ruleset three is always applied first. For example:

```
1,21,4 monet:bollard
```

first applies ruleset three to the input "monet:bollard." Ruleset one is then applied to the output of ruleset three, followed similarly by rulesets twenty-one and four.

If you need more detail, you can also use the **–d21** flag to turn on more debugging. For example,

```
sendmail -bt -d21.99
```

turns on an incredible amount of information; a single word address is probably going to print out several pages worth of information.

5.3.6. Building mailer descriptions

To add an outgoing mailer to your mail system, you will have to define the characteristics of the mailer.

Each mailer must have an internal name. This can be arbitrary, except that the names "local" and "prog" must be defined.

The pathname of the mailer must be given in the P field. If this mailer should be accessed via an IPC connection, use the string "[IPC]" instead.

The F field defines the mailer flags. You should specify an "f" or "r" flag to pass the name of the sender as a **–f** or **–r** flag respectively. These flags are only passed if they were passed to *sendmail,* so that mailers that give errors under some circumstances can be placated. If the mailer is not picky you can just specify "–f **$g**" in the argv template. If the mailer must be called as **root** the "S" flag should be given; this will not reset the userid before calling the mailer[3]. If this mailer is local (i.e., will perform final delivery rather than another network hop) the "l" flag should be given. Quote characters (backslashes and " marks) can be stripped from addresses if the "s" flag is specified; if this is not given they are passed through. If the mailer is capable of sending to more than one user on the same host in a single transaction the "m" flag should be stated. If this flag is on, then the argv template containing **$u** will be repeated for each unique user on a given host. The "e" flag will mark the mailer as being "expensive," which will cause *sendmail* to defer connection until a queue run[4].

An unusual case is the "C" flag. This flag applies to the mailer that the message is received from, rather than the mailer being sent to; if set, the domain spec of the sender (i.e., the "@host.domain" part) is saved and is appended to any addresses in the message that do not already contain a domain spec. For example, a message of the form:

[3]*Sendmail* must be running setuid to root for this to work.

[4]The "c" configuration option must be given for this to be effective.

```
From: eric@ucbarpa
To: wnj@monet, mckusick
```

will be modified to:

```
From: eric@ucbarpa
To: wnj@monet, mckusick@ucbarpa
```

if and only if the "C" flag is defined in the mailer corresponding to "eric@ucbarpa."

Other flags are described in Appendix C.

The S and R fields in the mailer description are per-mailer rewriting sets to be applied to sender and recipient addresses respectively. These are applied after the sending domain is appended and the general rewriting sets (numbers one and two) are applied, but before the output rewrite (ruleset four) is applied. A typical use is to append the current domain to addresses that do not already have a domain. For example, a header of the form:

```
From: eric
```

might be changed to be:

```
From: eric@ucbarpa
```

or

```
From: ucbvax!eric
```

depending on the domain it is being shipped into. These sets can also be used to do special purpose output rewriting in cooperation with ruleset four.

The E field defines the string to use as an end-of-line indication. A string containing only newline is the default. The usual backslash escapes (\r, \n, \f, \b) may be used.

Finally, an argv template is given as the E field. It may have embedded spaces. If there is no argv with a **$u** macro in it, *sendmail* will speak SMTP to the mailer. If the pathname for this mailer is "[IPC]," the argv should be

```
IPC $h [ port ]
```

where *port* is the optional port number to connect to.

For example, the specifications:

```
Mlocal, P=/bin/mail, F=rlsm  S=10, R=20, A=mail -d $u
Mether,P=[IPC],      F=meC, S=11, R=21, A=IPC $h, M=100000
```

specifies a mailer to do local delivery and a mailer for ethernet delivery. The first is called "local," is located in the file "/bin/mail," takes a picky **-r** flag, does local delivery, quotes should be stripped from addresses, and multiple users can be delivered at once; ruleset ten should be applied to sender addresses in the message and ruleset twenty should be applied to recipient addresses; the argv to send to a message will be the word "mail," the word "-d," and words containing the name of the receiving user. If a -r flag is inserted it will be between the words "mail" and "-d." The second mailer is called "ether," it should be connected to via an IPC connection, it can handle multiple users at once, connections should be deferred, and any domain from the sender address should be appended to any receiver name without a domain; sender addresses should be processed by ruleset eleven and recipient addresses by ruleset twenty-one. There is a 100,000 byte limit on messages passed through this mailer.

APPENDIX A

COMMAND LINE FLAGS

Arguments must be presented with flags before addresses. The flags are:

–f *addr*	The sender's machine address is *addr*. This flag is ignored unless the real user is listed as a "trusted user" or if *addr* contains an exclamation point (because of certain restrictions in UUCP).
–r *addr*	An obsolete form of **–f**.
–h *cnt*	Sets the "hop count" to *cnt*. This represents the number of times this message has been processed by *sendmail* (to the extent that it is supported by the underlying networks). *Cnt* is incremented during processing, and if it reaches MAXHOP (currently 30) *sendmail* throws away the message with an error.
–F*name*	Sets the full name of this user to *name*.
–n	Don't do aliasing or forwarding.
–t	Read the header for "To:", "Cc:", and "Bcc:" lines, and send to everyone listed in those lists. The "Bcc:" line will be deleted before sending. Any addresses in the argument vector will be deleted from the send list.
–b*x*	Set operation mode to *x*. Operation modes are:

m	Deliver mail (default)
a	Run in arpanet mode (see below)
s	Speak SMTP on input side
d	Run as a daemon
t	Run in test mode
v	Just verify addresses, don't collect or deliver
i	Initialize the alias database
p	Print the mail queue
z	Freeze the configuration file

The special processing for the ARPANET includes reading the "From:" line from the header to find the sender, printing ARPANET style messages (preceded by three digit reply codes for compatibility with the FTP protocol [Neigus73, Postel74, Postel77]), and ending lines of error messages with <CRLF>.

–q*time*	Try to process the queued up mail. If the time is given, a *sendmail* will run through the queue at the specified interval to deliver queued mail; otherwise, it only runs once.
–C*file*	Use a different configuration file.
–d*level*	Set debugging level.
–o*x value*	Set option *x* to the specified *value*. These options are described in Appendix B.

There are a number of options that may be specified as primitive flags (provided for compatibility with *delivermail*). These are the e, i, m, and v options. Also, the f option may be specified as the **–s** flag.

APPENDIX B

CONFIGURATION OPTIONS

The following options may be set using the **–o** flag on the command line or the **O** line in the configuration file:

A*file*	Use the named *file* as the alias file. If no file is specified, use *aliases* in the current directory.
a	If set, wait for an "@:@" entry to exist in the alias database before starting up. If it does not appear in five minutes, rebuild the database.
c	If an outgoing mailer is marked as being expensive, don't connect immediately. This requires that queueing be compiled in, since it will depend on a queue run process to actually send the mail.
d*x*	Deliver in mode *x*. Legal modes are: i Deliver interactively (synchronously); b Deliver in background (asynchronously); q Just queue the message (deliver during queue run)
D	If set, rebuild the alias database if necessary and possible. If this option is not set, *sendmail* will never rebuild the alias database unless explicitly requested using **–bi**.
e*x*	Dispose of errors using mode *x*. The values for *x* are: p Print error messages (default); q No messages, just give exit status; m Mail back errors; w Write back errors (mail if user not logged in); e Mail back errors and give zero exit stat always
F*n*	The temporary file mode, in octal. 644 and 600 are good choices.
f	Save Unix-style "From" lines at the front of headers. Normally they are assumed redundant and discarded.
g*n*	Set the default group id for mailers to run in to *n*.
H*file*	Specify the help file for SMTP.
i	Ignore dots in incoming messages.
L*n*	Set the default log level to *n*.
M*x value*	Set the macro *x* to *value*. This is intended only for use from the command line.
m	Send to me too, even if I am in an alias expansion.
o	Assume that the headers may be in old format, i.e., spaces delimit names. This actually turns on an adaptive algorithm: if any recipient address contains a comma, parenthesis, or angle bracket, it will be assumed that commas already exist. If this flag is not on, only commas delimit names. Headers are always output with commas between the names.
Q*dir*	Use the named *dir* as the queue directory.
r*time*	Timeout reads after *time* interval.

S*file*	Log statistics in the named *file*.
s	Be super-safe when running things, i.e., always instantiate the queue file, even if you are going to attempt immediate delivery. *Sendmail* always instantiates the queue file before returning control the the client under any circumstances.
T*time*	Set the queue timeout to *time*. After this interval, messages that have not been successfully sent will be returned to the sender.
t*S,D*	Set the local timezone name to *S* for standard time and *D* for daylight time; this is only used under version six.
u*n*	Set the default userid for mailers to *n*. Mailers without the *S* flag in the mailer definition will run as this user.
v	Run in verbose mode.

APPENDIX C

MAILER FLAGS

The following flags may be set in the mailer description.

f	The mailer wants a **–f** *from* flag, but only if this is a network forward operation (i.e., the mailer will give an error if the executing user does not have special permissions).
r	Same as **f**, but sends a **–r** flag.
S	Don't reset the userid before calling the mailer. This would be used in a secure environment where *sendmail* ran as root. This could be used to avoid forged addresses. This flag is suppressed if given from an "unsafe" environment (e.g, a user's mail.cf file).
n	Do not insert a UNIX-style "From" line on the front of the message.
l	This mailer is local (i.e., final delivery will be performed).
s	Strip quote characters off of the address before calling the mailer.
m	This mailer can send to multiple users on the same host in one transaction. When a **$u** macro occurs in the *argv* part of the mailer definition, that field will be repeated as necessary for all qualifying users.
F	This mailer wants a "From:" header line.
D	This mailer wants a "Date:" header line.
M	This mailer wants a "Message-Id:" header line.
x	This mailer wants a "Full-Name:" header line.
P	This mailer wants a "Return-Path:" line.
u	Upper case should be preserved in user names for this mailer.
h	Upper case should be preserved in host names for this mailer.
A	This is an Arpanet-compatible mailer, and all appropriate modes should be set.
U	This mailer wants Unix-style "From" lines with the ugly UUCP-style "remote from <host>" on the end.
e	This mailer is expensive to connect to, so try to avoid connecting normally; any necessary connection will occur during a queue run.
X	This mailer want to use the hidden dot algorithm as specified in RFC821; basically, any line beginning with a dot will have an extra dot prepended (to be stripped at the other end). This insures that lines in the message containing a dot will not terminate the message prematurely.
L	Limit the line lengths as specified in RFC821.
P	Use the return-path in the SMTP "MAIL FROM:" command rather than just the return address; although this is required in RFC821, many hosts do not process return paths properly.
I	This mailer will be speaking SMTP to another *sendmail* – as such it can use special protocol features. This option is not required (i.e., if this option is omitted the transmission will still operate successfully, although perhaps not as efficiently as possible).
C	If mail is *received* from a mailer with this flag set, any addresses in the header that do not have an at sign ("@") after being rewritten by ruleset three will have the "@domain" clause from the sender tacked on. This allows mail with headers of the form:

```
From: usera@hosta
To: userb@hostb, userc
```

to be rewritten as:

```
From: usera@hosta
To: userb@hostb, userc@hosta
```

automatically.

APPENDIX D

SUMMARY OF SUPPORT FILES

This is a summary of the support files that *sendmail* creates or generates.

/usr/lib/sendmail	The binary of *sendmail.*
/usr/bin/newaliases	A link to /usr/lib/sendmail; causes the alias database to be rebuilt. Running this program is completely equivalent to giving *sendmail* the **–bi** flag.
/usr/lib/sendmail.cf	The configuration file, in textual form.
/usr/lib/sendmail.fc	The configuration file represented as a memory image.
/usr/lib/sendmail.hf	The SMTP help file.
/usr/lib/sendmail.st	A statistics file; need not be present.
/usr/lib/aliases	The textual version of the alias file.
/usr/lib/aliases.{pag,dir}	The alias file in *dbm*(3) format.
/etc/syslog	The program to do logging.
/etc/syslog.conf	The configuration file for syslog.
/etc/syslog.pid	Contains the process id of the currently running syslog.
/usr/spool/mqueue	The directory in which the mail queue and temporary files reside.
/usr/spool/mqueue/qf*	Control (queue) files for messages.
/usr/spool/mqueue/df*	Data files.
/usr/spool/mqueue/lf*	Lock files
/usr/spool/mqueue/tf*	Temporary versions of the qf files, used during queue file rebuild.
/usr/spool/mqueue/nf*	A file used when creating a unique id.
/usr/spool/mqueue/xf*	A transcript of the current session.

Uucp Implementation Description

D. A. Nowitz

Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

Uucp is a series of programs designed to permit communication between UNIX† systems using either dial-up or hardwired communication lines. This document gives a detailed implementation description of the current implementation of uucp. It is designed for use by an administrator/installer of the system. It is not meant as a user's guide.

†UNIX is a Trademark of Western Electric.

INTRODUCTION

Uucp is a series of programs designed such that UNIX systems can communicate with each other using either dial-up or hardwired communication lines. *Uucp* transfers files between UNIX systems, and can also run commands on remote machines. The first version of the system was designed and implemented by M. E. Lesk. This paper describes the current (second) implementation of the system. This document gives a detailed description of the current implementation of *uucp*. It is designed for use by an administrator or installer of the system. It is not meant as a user's guide.

Uucp is a batch operation. Files are created in a spool directory for processing by the *uucp* daemons. There are three types of files used for the execution of work: *Data files* contain data for transfer to remote systems; *Work files* contain directions for file transfers between systems; *Execute files* are scripts for UNIX commands that involve the resources of one or more systems.

There are four primary programs involved in *uucp*'s operation:

uucp	builds *work files* and gathers *data files* in the spool directory for data transmission.
uux	creates *work files* and *execute files*, and gathers *data files* for the remote execution of UNIX commands.
uucico	executes the work files for data transmission.
uuxqt	executes the scripts for UNIX command execution.

There are two administrative programs:

uulog	gathers temporary log files that may occur due to lockout of the *uucp* log file and reports some information such as copy requests and completion status.
uuclean	removes old files from the spool directory.

The remaining sections of this manual describes the operation of each program, security, installation details, files required for execution, and administration of the *uucp* system.

UUCP – UNIX TO UNIX FILE COPY

The *uucp* command was is designed to look like *cp*(1) to the user. The syntax is:

uucp [option] ... source ... destination

where the source and destination may contain the prefix *system-name!*, which indicates the system where the file or files reside or where they will be copied.

Uucp has several options:

–d	Make directories when necessary for copying the file.
–c	Don't copy source files to the spool directory, but use the specified source when the actual transfer takes place. Note that the files, and all directories leading to them, must be accessible by everybody.
–e*sys*	Send this job to system *sys* to execute. Note that this only works when the system *sys* allows *uuxqt* to execute a *uucp* command. See the sections entitled *Uuxqt* and *Security*.
–C	Force the source files to be copied to the spool directory.
–f	Do not make directories.

–g*letter*	Put *letter* in as the grade in the name of the work file. This can be used to change the order of work for a particular machine.
–m	Send mail to the requester on completion of the work.
–n*user*	Notify *user* on the remote machine that a file has been sent.

Then there are options available for debugging:

–s*dir*	Use *dir* as the spool directory.
–r	Queue the job but do not start *uucico* program.
–x*num*	*Num* is a level number between 1 and 9; higher numbers give more debugging output.

The destination may be a directory name, in which case the file name is taken from the last part of the source's name. If the directory exists, it must be writable by everybody. Note that if the destination is a directory name and the –d option is specified to create the directory, the directory name must be followed by "/". The source name can contain special shell characters such as ?, *, and [and]. These are expanded on the appropriate system.

The command

```
uucp  *.c  usg!/usr/dan
```

sets up the transfer of all files whose names end with *.c* to the */usr/dan* directory on the *usg* machine.

The source and/or destination names may also contain a ˜*user* prefix. This translates to the login directory of *user* on the specified system. File names beginning with "˜/" translate into the public directory (usually */usr/spool/uucppublic*) on the remote system. For names with partial path-names, the current directory is prepended to the file name. File names beginning with ../ are not permitted for security reasons.

The command

```
uucp  usg!˜dan/*.h  ˜dan
```

sets up the transfer of files whose names end with *.h* in dan's login directory on system *usg* to dan's local login directory.

For each source file, *uucp* checks the source and destination file-names, the system-part of each argument, and the options to classify the work into several types:

[1] Copy source to destination on local system.

[2] Receive files from other systems.

[3] Send files to a remote system.

[4] Send files from remote systems to another remote system.

[5] Receive files from remote systems when the source contains special shell characters as mentioned above.

[6] Request that the *uucp* command be executed by a remote system.

After the work has been set up in the spool directory, the *uucico* program is started to try to contact the other machine and execute the work (unless the –r option was specified).

Type 1 – Local Copy

The copy is done locally. The *–m* and *–n* options are not honored in this case.

Type 2 – Receive Files

A *work file* is created or appended with a one line entry for each request. The upper limit to the number of files per *work file* is set in *uucp.h* The default setting is 20. After the limit has been reached, a new *work file* is created.

All *work files* and *execute files* use a blank as the field separator. The fields for these entries are given below.

[1] R

[2] The full path-name of the source or a ~something/path-name. The ~*something* part is expanded on the remote system.

[3] The full path-name of the destination file. If the ~*something* notation is used, it is immediately expanded.

[4] The user's login name.

[5] A "–" followed by an option list. The options –m and –d may appear.

Type 3 – Send Files

Each source file is copied into a *data file* in the spool directory. (A –c option on the *uucp* command prevents the *data file* from being made. In this case, the file is transmitted from the indicated source.) The fields for these entries are given below.

[1] S

[2] The full-path name of the source file.

[3] The full-path name of the destination or ~something/file-name.

[4] The user's login name.

[5] A "–" followed by an option list. The options –d, –m, and –n may appear.

[6] The name of the *data file* in the spool directory. A dummy name, "D.0" is used when the –c option is specified.

[7] The file mode bits of the source file in octal print format (666 for instance).

[8] The user on the remote system who should be notified upon completion of the file copy when the –n option is specified.

Type 4 and Type 5 – Remote UUCP Required

Uucp generates a *uucp* command and sends it to the remote machine; the remote *uucico* executes the *uucp* command.

Type 6 – Remote Execution

Remote execution occurs when the –e option is used. In this case, the *uux* facility is used to create and send the request. This requires that the remote *uuxqt* program allows the *uucp* command to be executed.

UUX – UNIX TO UNIX EXECUTION

The *uux* command sets up the execution of a UNIX command where the execution machine and/or some of the files are remote. The syntax of the *uux* command is

uux [–] [option] ... command-string

where the command-string is made up of one or more arguments. All special shell characters such as <, >, |, and ^, must be quoted either by quoting the entire command-string or quoting the special character as a separate argument.

Within the command-string, the command and file names may contain a *system-name!* prefix. Arguments which do not contain a ! character are assumed to be part of the command string and are not treated as files (that is, *uux* does not copy them to the execution machine).

An argument containing a ! but should not be treated as a file at the present time, can be escaped by surrounding the argument in parentheses (and). Note that the (and) characters themselves must usually be escaped with a \ character.

The "–" indicates that the standard input for *command-string* should be inherited from the standard input of the *uux* command.

The following options are available for *uux*:

–	Read from the standard input.
–x	Read from the standard input.
–n	Do not notify the requestor (by mail) of completion status.
–z	Only notify the requestor (by mail) of non-zero completion status.

The following options are available for debugging:

–r	Don't start *uucico* or *uuxqt* after queuing the job.
–x*num*	*Num* is a level number between 1 and 9; higher numbers give more debugging output.

The command:

```
pr abc | uux - usg!lpr
```

sets up the output of the

```
pr abc
```

command as standard input to an *lpr* command to be executed on system *usg*.

Uux generates an *execute file* containing the names of the files required for execution (including standard input), the user's login name, the destination of the standard output, and the command to be executed. This *execute file* file is either put in the spool directory for local execution or sent to the remote system using a send command (*uucp* type 3 command, described previously).

For required files that are not on the execution machine, *uux* generates receive command files (*uucp* type 2 command, described previously). The *uucico* program puts these command-files on the execution machine for execution.

The *execute file* contains a script that is processed by the *uuxqt* program. It is made up of several lines, each of which contains an identification character and one or more arguments. The lines are described in the subsections below. Here is a summary of the types of lines that appear in the file. They are described in detail in the sections following.

User Line	Identifies the requestor's login name and system.
File Line	Identifies a filename for transmission.
Standard Input Line	Specifies a standard input file.
Standard Output Line	Specifies a standard output file.
Command Line	Identifies a UNIX system command for *uuxqt* to execute.

User Line

    U user system

where the *user* and *system* are the requester's login name and system.

Required File Line

    F file-name real-name

where *file-name* is a unique name used for file transmission and *real-name* is the last part of the actual file name (contains no path information).

Zero or more of these lines may be present. The *uuxqt* program checks for the existence of all these files before the command is executed.

Standard Input Line

    I file-name

The standard input is either specified by a < in the command-string or inherited from the standard input of the *uux* command if the "–" option is used. If a standard input is not specified, */dev/null* is used. Note that if there is a standard input specified, it will also appear in an "F" line.

Standard Output Line

    O file-name system-name

The standard output is specified by a > within the command-string. If a standard output is not specified, */dev/null* is used. Note that the appending to a file by using >> is not implemented.

Command Line

    C command [arguments] ...

The arguments are those specified in the command-string. The standard input and standard output will not appear on this line. All *required files* are moved to the execution directory (usually */usr/lib/uucp/.XQTDIR*) and the UNIX command is executed using the shell specified in the *uucp.h* header file. In addition, a shell "PATH" statement is prepended to the command line as specified in the *uuxqt* program.

Note that a check is made to see that the command is allowed as specified in the *uuxqt* program. After execution, the standard output is copied or sent to the proper place.

UUXQT – UUCP COMMAND EXECUTION

The *uuxqt* program executes scripts generated by *uux.*

The *uuxqt* program may be started by either the *uucico* or *uux* programs or a daemon specified by a *crontab* entry. *Uuxqt* scans the spool directory for *execute files* (prefix "X."). Each *execute file* is checked to see if all the required files are available and if so, the command line is verified and executed.

The *execute file* is described in the section entitled *uux,* above.

The execution is accomplished by executing a

    sh –c

of the command line after appropriate standard input and standard output have been opened. If a standard output is specified, the program creates a send command, or copies the output file as appropriate.

Uuxqt accepts the standard debugging option:

–x*num*	*Num* is a level number between 1 and 9; higher numbers give more debug output.

UUCICO – COPY IN, COPY OUT

The *uucico* program performs several major functions:

- Scan the spool directory for work.
- Place a call to a remote system.
- Negotiate a line protocol to be used.
- Execute all requests from both systems.
- Log work requests and work completions.

Uucico may be started in several ways:

a. by a system daemon specified in a crontab entry,

b. by one of the *uucp, uux, uuxqt* or *uucico* programs,

c. directly by the user (this is usually for testing),

d. by a remote system. The uucico program should be specified as the "shell" field in the */etc/passwd* file for the logins used by remote systems to access uucp.

When started by method a, b or c, *uucico* is considered to be in *MASTER* mode. In this mode, a connection is made to a remote system. If started by a remote system (method d), *uucico* is considered to be in *SLAVE* mode.

The *MASTER* mode operates in one of two ways. If no system name is specified (–s option not specified) *uucico* scans the spool directory for systems to call. If a system name is specified, that system is called, and work is only done for that system.

Uucico is generally started by another program. There are several options used for execution:

–r1	Start *uucico* in *MASTER* mode. This is used when *uucico* is started by a program or "cron" shell.
–s*sys*	Do work only for system *sys*. If *–s* is specified, a call to the specified system is made even if there is no work for system *sys* in the spool directory. This is useful for polling systems that do not have the hardware to initiate a connection.

The following options are used primarily for debugging:

–d*dir*	Use directory *dir* for the spool directory.
–x*num*	*Num* is a level number between 1 and 9; higher numbers give more debugging output.

The next part of this section describes the major steps within the *uucico* program.

Scan For Work

The names of the work related files in the spool directory have format

type . system-name grade number

type	is an upper case letter (*C* – copy command file, *D* – data file, *X* – execute file),

system-name
is the remote system, *truncated to seven characters.*

grade is a character,

number is a four digit, hexadecimal, zero padded sequence number.

The file

C.res45n0031

would be a *work file* for a file transfer between the local machine and the "res45" machine.

The scan for work is done by looking through the spool directory for *work files* (files with prefix "C."). A list is made of all systems to be called. *Uucico* then calls each system and processes all *work files.*

Call Remote System

The call is made using information from several files that reside in the *uucp* program directory (usually */usr/lib/uucp*). At the start of the call process, a lock is set to prevent multiple conversations between the same two systems.

The *L.sys* file contains information required to make the remote connection:

[1] system name,

[2] times to call the system (days-of-week and times-of-day) and the minimum time delay before retry,

[3] device or device type to be used for call,

[4] line class (this is the line speed on almost all systems),

[5] phone number if field [3] is *ACU* or the device if not *ACU,*

[6] login information (zero or more fields),

The time field is checked against the present time to see if the call should be made. The *phone number* may contain abbreviations (for example, mh, py, boston) that get translated into dial sequences using the *L-dialcodes* file.

The *L-devices* file is scanned using fields [3] and [4] from the *L.sys* file to find an available device for the call. The program tries each devices that satisfy [3] and [4] until a call is made, or no more devices can be tried. If a device is successfully opened, a lock file is created. If the call is completed, the login information (field [6] of *L.sys*) is used to login.

The conversation between the two *uucico* programs begins with a handshake started by the called (*SLAVE*) system. The *SLAVE* sends a message to let the *MASTER* know it is ready to receive the system identification and conversation sequence number. The *SLAVE* verifies the response from the *MASTER* and if acceptable, protocol selection begins. The *SLAVE* can also reply with a "call-back required" message, in which case the current conversation is terminated.

Line Protocol Selection

The remote system sends a message:

P*proto-list*

where *proto-list* is a string of characters, each representing a line protocol.

The calling program checks *proto-list* for a letter corresponding to an available line protocol and returns a *use-protocol* message. The *use-protocol* message is

U*code*

where *code* is either a one character protocol letter or "N", which means there is no common protocol. The only protocol which is currently implemented is "g", which uses the packet driver.

Work Processing

The *MASTER* program does a work search similar to the one used in the *Scan For Work* section described above (the *MASTER* has been specified by the "-r1" uucico option). Each message used during the work processing is specified by the first character of the message:

S send a file,

R receive a file,

C copy complete,

X execute a *uucp* command,

H hangup.

The *MASTER* sends *R*, *S* or *X* messages until all work for the remote system is complete, at which point an *H* message is sent. The *SLAVE* replies with *SY*, *SN*, *RY*, *RN*, *HY*, *HN*, *XY*, or *XN*, corresponding to *yes* or *no* for each request.

An *N* response can be followed by a number giving the reson for the failure:

N0 Copy failed (reason not given by remote system).

N1 Local access to file denied.

N2 Remote access to path or file denied.

N3 System error - bad *uucp* command generated.

N4 Remote system cannot create temporary file.

N5 Cannot copy to file or directory - file left in *pubdir/user/file.*

N6 Cannot copy to file or directory - file left in *pubdir/user/file.*

The send and receive replies are based on permission to access the requested file or directory using the *USERFILE* and read/write permissions of the file or directory.

After each file is copied into the spool directory of the receiving system, a copy-complete message is sent by the receiver of the file. The message *CY* is sent if the file has successfully been moved from the spool directory to the destination. If the file was not successfully moved from the spool directory to the destination, a *CN* message is sent, the file is put in the public directory (usually */usr/spool/uucppublic*), and the requester is notified by mail.

The requests and results are logged on both systems.

The hangup response is determined by a work scan of the *SLAVE*'s spool directory. If work for the remote system exists an *HN* message is sent and the programs switch roles. If no work exists, an *HY* response is sent.

Conversation Termination

When the *MASTER* receives a *HY* message, the *MASTER* echoes the message back to the *SLAVE* and the protocols are turned off. Each program sends a final "OO" (Over and Out) message to the other. The original *SLAVE* program cleans up and terminate. The *MASTER* then proceeds to call other systems unless a "-s" option was specified.

UULOG - UUCP LOG INQUIRY

When a *uucp* program can not make a log entry directly into the *LOGFILE* an individual log file is created: a file with prefix *LOG.* This sometimes occurs when more than one *uucp* process is running. Periodically, *uulog* may be executed to append these files to the *LOGFILE.*

The *uulog* program may also be used to request the output of *LOGFILE* entries. The request is specified by the use of the options:

–s*sys*	Print entries where *sys* is the remote system name,
–u*user*	Print entries for user *user*.

The intersection of lines satisfying the two options is output. A null *sys* or *user* means all system names or users respectively.

Debugging options available are:

–x*num*	*Num* is a level number between 1 and 9; higher numbers give more debug output.
–n*secs*	Time out lock on log file if older than *secs* seconds.

UUCLEAN – UUCP SPOOL DIRECTORY CLEANUP

Uuclean is typically started by the *uucp* daily daemon. Its function is to remove files from the spool directory that are more than three days old. These are usually files for work that can not be completed. The requester of this work is notified that the files have been deleted.

There are several options:

–d*dir*	The directory to be scanned is *dir*.
–m	Send mail to the owner of each file being removed. Note that most files put into the spool directory are owned by the owner of the *uucp* programs since the setuid bit will be set on these programs. This mail is sometimes useful for administration.
–n*hours*	Change the aging time from 72 hours to *hours* hours.
–p*pre*	Examine files with prefix *pre* for deletion. Up to 10 of these options may be specified.
–x*num*	This is the level of debugging output desired.

SECURITY

⇒ *The uucp system, left unrestricted, will let any outside user execute any commands and copy out/in any file that is readable/writable by a uucp login user. It is up to the individual sites to be aware of this and apply the protections that they feel are necessary.*

There are several security features available aside from the normal file mode protections. These must be set up by the administrator of the *uucp* system.

- The login for *uucp* does not get a standard shell. Instead, the *uucico* program is started so that all work is done through *uucico*.
- The owner of the *uucp* programs should be an administrative login. It should not be one of the logins used for remote system access to *uucp*.
- All *uucp* logins should have passwords.
- A path check is done on file names that are to be sent or received. The *USERFILE* supplies the information for these checks. The *USERFILE* can also be set up to require call-back for certain login-ids. See the "Files Required For Execution" section for the file description.
- A conversation sequence count can be set up so that the called system can be more confident of the caller's identity.
- The *uuxqt* program reads a file containing a list of commands that it will execute. A "PATH" shell statement is prepended to the command line as specified in the *uuxqt* program. The installer may modify the list or remove the restrictions as desired.

- The *L.sys* file should be owned by the *uucp* administrative login and have mode 0400 to protect the phone numbers and login information for remote sites.
- The programs *uucp, uucico, uux, uuxqt, uulog,* and *uuclean* should be owned by the *uucp* administrative login, have the setuid bit set, and have only execute permissions.

UUCP INSTALLATION

It is assumed that the *login name* used by a remote computer to call into a local computer is not the same as the login name of a normal user or the *uucp* administrative login. However, several remote computers may use the same login name. It is suggested that the installer follow the convention of using the letter "U" followed by the system name as the login name for each system. For example, use login name *Uusg* for the *usg* system.

Each computer should be given a unique *system name* that is transmitted at the start of each call. This name identifies the calling machine to the called machine. The *login/system* names are used for security as described later in the *USERFILE* section.

There are several source modifications that may be required before the system programs are compiled. These relate to the directories, local system name, and attributes of the local environment.

The *uucp* system uses several directories:

sources	(/usr/src/cmd/uucp) – This directory contains the *uucp* system source files.
program	(/usr/lib/uucp) – This is the directory used for some of the executable system programs and the system files. Some of the programs reside in */usr/bin.*
spool	(/usr/spool/uucp) – This is the *uucp* system spool directory.
xqtdir	(/usr/lib/uucp/.XQTDIR) – This directory is used during execution of the *uux* scripts.

The names in parentheses above are the default values for the directories. The italicized names *sources, program, xqtdir,* and *spool* are used in the following text to represent the appropriate directory names.

There are two files that may require modification, namely the *Makefile* file and the *uucp.h* file. In addition, the *uuxqt.c* program may be modified as indicated in the section entitled *Security,* above. The following sections describe the modifications.

uucp.h Modification

Several manifests in *uucp.h* may need modification for the local system environment:

UNAME	should be defined if the "uname" function is available.
ACULAST	is the character required by the ACU as the last character. For most systems, it is a "–". This is only relevant if you are using a DN11 autodialler.
DIALOUT	should be defined if the C library routine "dialout" is available.
UUDIR	should be defined if the *uucp* subdirectory *syskludge* is being used.
UUNAME	should be defined if the system name should be read from the SYSTEMNAME file.
UUSTAT	should be defined if you need the *uustat* program.
UUSUB	should be defined if you need the *uusub* program.

Makefile Modification

There are several *make* variable definitions that may need modification:

INSDIR	is the *program* directory (for example, INSDIR=/usr/lib/uucp). This parameter is used if "make cp" or "make install" is used.
PUBDIR	is a public directory for remote access. This is also the login directory for remote *uucp* users. It should be the same as that defined in *uucp.h*.
SPOOL	is the uucp spool directory. This should be the same as that defined in *uucp.h*.
XQTDIR	is the directory for uuxqt to use for command execution. It is also defined in *uucp.h*.
OWNER	is the administrative login for *uucp*.
LIBS	should include *syskludge/syskludge.a if the syskludge library is used. UUDIR should be defined in uucp.h.*
CFLAGS	add -DVMUNIX if on a VMUNIX system.

Compiling The System

The command:

```
make install
```

makes the required directories, compiles all programs, sets the proper file modes, and copies the programs to the proper directories. This command should be run as *root*. The command:

```
make
```

compiles the entire system.

The programs *uucp, uux,* and *uulog* should be put in */usr/bin*. The programs *uuxqt, uucico,* and *uuclean* should be put in the *program* directory.

Files Required For Execution

Six files are required for execution. They should reside in the *program* directory. The field separator for all files is a space. The required files are summarized here, and the following subsections describe them in detail.

L-devices	Contains call unit information.
L-dialcodes	Contains dialcode abbreviations.
USERFILE	Contains user accessibility and constraint information.
L.sys	Contains information about the systems which local *uucp* programs can call.
L.cmds	Contains commands which *uuxqt* is allowed to execute.
SYSTEMNAME	Contains the name of the system.

L-devices – Call Unit Information File

L-devices contains call-unit device and hardwired connection information. The special device files are assumed to be in the */dev* directory. The format for each entry in the *L-devices* file is:

```
type  line  call-unit  speed
```

type	is a device type such as ACU or DIR. The currently supported device types are described later. The field can also be used to specify particular ACUs for some calls by using a suffix on the ACU field (ACUDF03wats, for instance). This name should be used in *L.sys*.

line	is the device for the line (for example, cul0 if using a DN11, otherwise it is the same as the call-unit field).
call-unit	is the automatic call unit associated with *line* (for example, cua0). Hardwired lines have a number "0" in this field.
speed	is the line speed.

For example, an entry in the *L-devices* file like this:

```
ACU  cul0  cua0  300
```

would be set up for a system that has a DN11 device "/dev/cul0" wired to a call-unit "/dev/cua0" for use at 300 baud. An entry like:

```
ACUDF03  cua0  cua0  1200
```

would be set up for a system that has a DF03 device "/dev/cua0" for use at 1200 baud.

L-dialcodes – Dial Code Abbreviations File

L-dialcodes contains the dialcode abbreviations used in the *L.sys* file (for example, py, mh, boston). The entry format is:

```
abb  dial-seq
```

abb	is the abbreviation,
dial-seq	is the dial sequence to call that location.

For example, a line in the *L-dialcodes* file that looks like:

```
py  165–
```

would be set up so that entry py7777 would send 165–7777 to the dial-unit.

USERFILE

USERFILE contains user accessibility information. It specifies four types of constraint:

[1] which files can be accessed by a normal user of the local machine,

[2] which files can be accessed from a remote computer,

[3] which login name is used by a particular remote computer,

[4] whether a remote computer should be called back in order to confirm its identity.

Each line in *USERFILE* has the format:

```
login,sys  [ c ]  path-name  [ path-name ] ...
```

login	is the login name for a user or the remote computer,
sys	is the system name for a remote computer,
c	is the optional *call-back required* flag,
path-name	is a path-name prefix that is acceptable for *sys*.

The constraints are implemented as follows.

[1] When the program is obeying a command stored on the local machine (*MASTER* mode) the path-names allowed are those given on the first line in the *USERFILE* that has the login name of the user who entered the command. If no such line is found, the first line with a *null* login name is used.

[2] When the program is responding to a command from a remote machine (*SLAVE* mode) the path-names allowed are those given on the first line in the file that has the system name that matches the remote machine. If no such line is found, the first one with a *null* system name is used.

[3] When a remote computer logs in, the login name that it uses *must* appear in the *USERFILE.* There may be several lines with the same login name but one of them must either have the name of the remote system or must contain a *null* system name.

[4] If the line matched in ([3]) contains a "c", the remote machine is called back before any transactions take place.

Examples

The line:

```
u,m  /usr/xyz
```

allows machine *m* to login with name *u* and request the transfer of files whose names start with */usr/xyz.*

The line:

```
dan,  /usr/dan
```

allows the ordinary user *dan* to issue commands for files whose name starts with */usr/dan.* (Note that this type of restriction is seldom used.)

The lines:

```
u,m /usr/xyz  /usr/spool
u,  /usr/spool
```

allows any remote machine to login with name *u.* If its system name is not *m,* it can only ask to transfer files whose names start with */usr/spool.* If it is system *m,* it can send files from path */usr/xyz* as well as */usr/spool.*

The lines:

```
root, /
,  /usr
```

allow any user to transfer files beginning with */usr,* but the user with login *root* can transfer any file. Note that any file that is to be transferred must be readable by anybody. The *USERFILE* is normally set up as follows:

```
,myname /
, /usr/spool/uucppublic
```

where *myname* is the name of the current system. These lines allow any user on the current machine to access any file (subject to the normal permissions on the file) for *uucp* transfer, whereas users on other machines can only access files in */usr/spool/uucppublic.*

L.sys

Each entry in *L.sys* represents one system that can be called by the local *uucp* programs. More than one line may be present for a particular system. In this case, the additional lines represent alternative communication paths that are tried in sequential order. The fields are described below.

system name
    The name of the remote system.

time — This is a string that indicates the days-of-week and times-of-day when the system should be called (for example, MoTuTh0800–1730).

The day portion may be a list containing some of

Su Mo Tu We Th Fr Sa

or it may be *Wk* for any week-day or *Any* for any day.

The time should be a range of times (for example, 0800–1230). If no time portion is specified, any time of day is assumed to be okay for the call. Note that a time range that spans 0000 is permitted, for example, 0800-0600 means all times are ok other than times between 6 and 8 am.

A time specification of "None" is often used for a passive system, that is, one which cannot call the specified system. In this case the following fields may be omitted. Note that the string "None" has no special significance, but is merely a string that is not a correct time specification.

Several day-time specifications may be present, separated by a | character.

An optional subfield is available to indicate the minimum time (minutes) before a retry following a failed attempt. The subfield separator is a ",". For example, Any,9 means call any time but don't allow another call until at least 9 minutes after a failure has occurred.

device — This fields either starts with *ACU*, or is the hardwired device to be used for the call. For the hardwired case, the last part of the special file name is used (tty0, for instance).

class — This is usually the line speed for the call (for example, 300). The exception is when the C library routine "dialout" is available in which case this is the dialout class.

phone — The phone number is made up of an optional alphabetic abbreviation and a numeric part. The abbreviation should be one that appears in the *L-dialcodes* file (for example, mh5900, boston995–9980). For the hardwired devices, this field contains the same string as used for the *device* field.

login — The login information is given as a series of fields and subfields in the format

```
[ expect  send ] ...
```

where *expect* is the string expected to be read and *send* is the string to be sent when the *expect* string is received.

The expect field may be made up of subfields of the form

```
expect[–send–expect] ...
```

where the *send* is sent if the prior *expect* is *not* successfully read and the *expect* following the *send* is the next expected string. For example:

```
login--login
```

expects to see the word *login*; if it gets it, the program proceeds to the next field; if it does not get *login*, it sends *null* followed by a new line, then expects *login* again.

There are two special names available to be sent during the login sequence. The string *EOT* sends an EOT character and the string *BREAK* tries to send a BREAK character. The *BREAK* character is simulated using line speed changes and null characters and may not work on all devices and/or systems. A number from 1 to 9 may follow the *BREAK*. For example, *BREAK1* sends 1 null character instead of the default of 3. Note that *BREAK1* usually works best for 300/1200 baud lines.

The following escape sequences are also recognized:

\r	send a carriage-return.
\n	send a newline (linefeed) character.
\d	delay for 1 second.
\c	suppress newline at end of *send* string.
\s	send a space.

A typical entry in the L.sys file would be:

```
        sys  Any  ACU  300  mh7654  login:-EOT-login:-BREAK-login:  uucp  ssword:  word
```

The expect algorithm matches all or part of the input string as illustrated in the password field above. Very complex expect-send sequences are often required if the called system is connected to a terminal concentrator or a front end.

L.cmds

L.cmds contains a list of commands which *uuxqt* is allowed to execute. The commands should be one per line. At a minimum, *L.cmds* should contain the command "rmail". Other commands often allowed are "rnews", "uusend", and "lpr".

Device Types

The currently supported device types are:

Type Code	Device
ACU	DEC DN11
ACUDN11	DEC DN11
ACUDF02	DEC DF02 (300 baud only)
ACUDF03	DEC DF03 (1200 baud only)
ACUVENTEL	Ventel MD212 Plus
DIR	Hardwired Line.

The DN11 is only available on DEC PDP-11 and VAX systems. The DEC DF02, DF03, and Ventel MD212 Plus can be connected to any system using a standard RS232 terminal interface. When connecting one of these devices to a terminal line, the device node (/dev/tty*x*) should be renamed (to /dev/cua0 for instance) to emphasize that the line is for dialout only and to prevent accidentally starting a login process for that line.

The device type specified in the *L-devices* file should be one of those listed above, optionally qualified further by additional characters after the device type. The device type specified in the *L.sys* file should be a prefix of one of the devices specified in the *L-devices* file. For example, assume you have two DF03's, one connected to a local telephone line and the other connected to a WATS line. The *L-devices* file could be set up as follows:

```
        ACUDF03local  cua0  cua0  1200
        ACUDF03wats  cua0  cua0  1200
```

To call a system using only the WATS line, specify ACUDF03wats in the device type field. Similarly, to call a system using the local telephone line, specify ACUDF03local. To call a system using either DF03, specify ACUDF03 in the *L.sys* file.

Note that the telephone numbers specified in the *L.sys* file will have a format dependent on the ACU device type. This is a deficiency which may be corrected in the future.

ADMINISTRATION

This section indicates some events and files that must be administered for the *uucp* system. Some administration can be accomplished by *shell files* initiated by *crontab* entries. Others may require manual intervention. Some sample *shell files* are given toward the end of this section.

SQFILE – Sequence Check File

SQFILE is set up in the *program* directory and contains an entry for each remote system with which you agree to perform conversation sequence checks. The initial entry is just the system name of the remote system. The first conversation adds the conversation count and the date/time of the most resent conversation. These items are updated with each conversation. If a sequence check fails, the entry will have to be adjusted manually. Note that this feature is rarely used.

TM – Temporary Data Files

These files are created in the *spool* directory while a file is being copied from a remote machine. Their names have the form

```
TM.pid.ddd
```

where *pid* is a process-id and *ddd* is a sequential three digit number starting at zero. After the entire file is received, the *TM* file is moved/copied to the requested destination. If processing is abnormally terminated the file remains in the spool directory. The leftover files should be periodically removed; the *uuclean* program is useful in this regard. The command

```
program/uuclean  -pTM
```

removes all *TM* files older than three days.

LOG – Log Entry Files

During execution, log information is appended to the *LOGFILE.* If the *LOGFILE* is locked by another process, the log information is placed in individual log files with a with a *LOG* prefix. These individual files should be combined into the *LOGFILE* by using the *uulog* program. *Uulog* appends the contents of the individual log files onto the end of the *LOGFILE.* The command:

```
uulog
```

accomplishes the merge. Options are available to print some or all the log entries after the files are merged. The *LOGFILE* should be removed periodically.

The *LOG* files are created initially with mode 0222. If the program that creates the file terminates normally, it changes the mode to 0666. Aborted runs may leave the files with mode 0222 and the *uulog* program will not read or remove them. To remove them, either use *rm,* *uuclean,* or change the mode to 0666 and let *uulog* merge them into the *LOGFILE.*

STST – System Status Files

These files are created in the spool directory by the *uucico* program. They contain information such as login, dialup or sequence check failures or will contain a *TALKING* status when two machines are conversing. The form of the file name is

```
STST.sys
```

where *sys* is the remote system name, truncated to seven characters.

For ordinary failures, such as dialup or login, the file prevents repeated tries for about 55 minutes. This is the default time; it can be changed on an individual system basis by a subfield of the time field in the *L.sys* file. For sequence check failures, the file must be removed before any future attempts to converse with that remote system. Retries are accomplished by starting *uucico* from *crontab*, usually hourly.

LCK – Lock Files

Lock files are created for each device in use (e.g., automatic calling unit) and each system conversing. This prevents duplicate conversations and multiple attempts to use the same device. The form of the lock file name is:

LCK..str

where *str* is either a device or system name. The files may be left in the spool directory if runs abort (usually only on system crashes). They are ignored (reused) after 1.5 hours. When runs abort and calls are desired before the time limit, the lock files should be removed.

Shell Files

The *uucp* program spools work and attempts to start the *uucico* program, but *uucico* will not always be able to execute the request immediately. Therefore, the *uucico* program should be periodically started. The command to start *uucico* can be put in a "shell" file with a command to merge *LOG.* files and started by a crontab entry on an hourly basis. The file could contain the commands:

```
/usr/bin/uulog
program/uucico  -r1  -sinter
program/uucico  -r1
```

The "-r1" option is required to start the *uucico* program in *MASTER* mode. The "-s" option can be used for polling as illustrated in the second line where machine *inter* is being polled. The third line processes all other spooled work.

Another shell file may be set up on a daily basis to remove *TM, ST* and *LCK* files and *C.* or *D.* files for work that can not be accomplished for reasons like bad phone number, login changes, and so on. A shell file containing commands like:

```
program/uuclean  -pTM -pC. -pD.
program/uuclean  -pST -pLCK -n12
```

can be used. Note that the "-n12" option causes the *ST* and *LCK* files older than 12 hours to be deleted. The absence of the "-n" option uses a three day time limit.

A daily or weekly shell should also be created to remove or save old *LOGFILE*'s. A shell like:

```
cp spool/LOGFILE   spool/o.LOGFILE
rm spool/LOGFILE
```

can be used.

The shell files in *program*/uucp.* do a more extensive job than that described here. They should be started by entries in *crontab*. Read the shell files for more information.

Login Entry

Two or more logins should be set up for *uucp*. One should be an administrative login: the owner of all the uucp programs, directories and files. All others are used by remote systems to access the uucp system. Each of the */etc/passwd* entries for the *access* logins should have *program*/uucico as the shell to be executed. The login directory should be the public directory (usually */usr/spool/uucppublic*) for both the administrative login and the access logins. The various *access* login names are used in the *USERFILE* to restrict file access.

File Modes

The programs *uucp, uux, uucico, uulog, uuclean,* and *uuxqt* should be owned by the *uucp* administrative login with the "setuid" bit set and only execute permissions (mode 04111). The *L.sys*, *SQFILE,* and the *USERFILE,* which are put in the *program* directory should be owned by the *uucp* administrative login and set with mode 0400. The mode of *spool* should be 0755. The mode of *xqtdir* should be 0777. The *L-dialcodes* and the *L-devices* files should have mode 0444.

15

Line Printer Spooler Manual

Revised July 27, 1983

Ralph Campbell

Computer Systems Research Group
Computer Science Division
Department of Electrical Engineering and Computer Science
University of California, Berkeley
Berkeley, CA 94720
(415) 642-7780

ABSTRACT

This document describes the structure and installation procedure for the line printer spooling system implemented in this release of the Sun UNIX* operating system.

1. Overview

The line printer system supports:

- multiple printers,
- multiple spooling queues,
- both local and remote printers, and
- printers attached via serial lines which require line initialization such as the baud rate.

Raster output devices such as a Varian or Versatec, and laser printers such as an Imagen, are also supported by the line printer system.

The line printer system consists mainly of the following files and commands:

/etc/printcap	printer configuration and capability data base
/usr/lib/lpd	line printer daemon, does all the real work
/usr/ucb/lpr	program to enter a job in a printer queue
/usr/ucb/lpq	spooling queue examination program
/usr/ucb/lprm	program to delete jobs from a queue
/etc/lpc	program to administer printers and spooling queues
/dev/printer	socket on which lpd listens

The file /etc/printcap is a master data base describing line printers directly attached to a machine and, also, printers accessible across a network. The manual page entry *printcap*(5) provides the ultimate definition of the format of this data base, as well as indicating default values for important items such as the directory in which spooling is performed. This document highlights the important information which may be placed in *printcap*.

* UNIX is a trademark of Bell Labs

2. Commands

2.1. lpd – line printer dameon

The program *lpd*(8), usually invoked at boot time from the */etc/rc* file, acts as a master server for coordinating and controlling the spooling queues configured in the printcap file. When *lpd* is started it makes a single pass through the *printcap* database restarting any printers which have jobs. In normal operation *lpd* listens for service requests on multiple sockets, one in the UNIX domain (named "/dev/printer") for local requests, and one in the Internet domain (under the "printer" service specification) for requests for printer access from off machine; see *socket*(2) and *services*(5) for more information on sockets and service specifications, respectively. *Lpd* spawns a copy of itself to process the request; the master daemon continues to listen for new requests.

Clients communicate with *lpd* using a simple transaction oriented protocol. Authentication of remote clients is done based on the "privilege port" scheme employed by *rshd*(8C) and *rcmd*(3X). The following table shows the requests understood by *lpd*. In each request the first byte indicates the "meaning" of the request, followed by the name of the printer to which it should be applied. Additional qualifiers may follow, depending on the request.

Request	Interpretation
^Aprinter\n	check the queue for jobs and print any found
^Bprinter\n	receive and queue a job from another machine
^Cprinter [users ...] [jobs ...]\n	return short list of current queue state
^Dprinter [users ...] [jobs ...]\n	return long list of current queue state
^Eprinter person [users ...] [jobs ...]\n	remove jobs from a queue

The *lpr*(1) command is used by users to enter a print job in a local queue and to notify the local *lpd* that there are new jobs in the spooling area. *Lpd* either schedules the job to be printed locally, or in the case of remote printing, attempts to forward the job to the appropriate machine. If the printer cannot be opened or the destination machine is unreachable, the job will remain queued until it is possible to complete the work.

2.2. lpq – show line printer queue

The *lpq*(1) program works recursively backwards displaying the queue of the machine with the printer and then the queue(s) of the machine(s) that lead to it. *Lpq* has two forms of output: in the default, short, format it gives a single line of output per queued job; in the long format it shows the list of files, and their sizes, which comprise a job.

2.3. lprm – remove jobs from a queue

The *lprm*(1) command deletes jobs from a spooling queue. If necessary, *lprm* will first kill off a running daemon which is servicing the queue, restarting it after the required files are removed. When removing jobs destined for a remote printer, *lprm* acts similarly to *lpq* except it first checks locally for jobs to remove and then tries to remove files in queues off-machine.

2.4. lpc – line printer control program

The *lpc*(8) program is used by the system administrator to control the operation of the line printer system. For each line printer configured in */etc/printcap*, *lpc* may be used to:

- disable or enable a printer,
- disable or enable a printer's spooling queue,
- rearrange the order of jobs in a spooling queue,

- find the status of printers, and their associated spooling queues and printer dameons.

3. Access control

The printer system maintains protected spooling areas so that users cannot circumvent printer accounting or remove files other than their own. The strategy used to maintain protected spooling areas is as follows:

- The spooling area is writable only by a *daemon* user and *spooling* group.
- The *lpr* program runs setuid *root* and setgid *spooling*. The *root* access is used to read any file required, verifying accessibility with an *access*(2) call. The group ID is used in setting up proper ownership of files in the spooling area for *lprm*. Data files are owned by user, group spooling, mode 660.
- Control files in a spooling area are made with *daemon* ownership and group ownership *spooling*. Their mode is 0660. This insures control files are not modified by a user and that no user can remove files except through *lprm*.
- The spooling programs, *lpd, lpq,* and *lprm* run setuid *root* and setgid *spooling* to access spool files and printers.
- The printer server, *lpd*, uses the same verification procedures as *rshd*(8C) in authenticating remote clients. The host on which a client resides must be present in the file /etc/hosts.equiv, used to create clusters of machines under a single administration.

In practice, none of *lpd, lpq,* or *lprm* would have to run as user *root* if remote spooling were not supported. In previous incarnations of the printer system *lpd* ran setuid *daemon*, setgid *spooling*, and *lpq* and *lprm* ran setgid *spooling*.

4. Setting up

The Sun system release comes with the necessary programs installed and with the default line printer queue created. The real work in setting up is to create the *printcap* file and any printer filters for printers not supported in the distribution system.

4.1. Creating a printcap file

The *printcap* database contains one or more entries per printer. A printer should have a separate spooling directory; otherwise, jobs will be printed on different printers depending on which printer daemon starts first. This section describes how to create entries for printers which do not conform to the default printer description.

4.1.1. Printers on serial lines

When a printer is connected via a serial communication line it must have the proper baud rate and terminal modes set. The following example is for a DecWriter III printer connected locally via a 1200 baud serial line.

```
lp|LA-180 DecWriter III:\
        :lp=/dev/lp:br#1200:fs#06320:\
        :tr=\f:of=/usr/lib/lpf:lf=/usr/adm/lpd-errs:
```

The **lp** entry specifies the file name to open for output. In this case it could be left out since "/dev/lp" is the default. The **br** entry sets the baud rate for the tty line and the **fs** entry sets CRMOD, no parity, and XTABS (see *tty*(4)). The **tr** entry indicates a form-feed should be printed when the queue empties so the paper can be torn off without turning the printer off-line and pressing form feed. The **of** entry specifies the filter program *lpf* should be used for printing the files; more will be said about filters later. The last entry causes errors to be written to the file "/usr/adm/lpd-errs" instead of the console.

4.1.2. Remote printers

Printers which reside on remote hosts should have an empty **lp** entry. For example, the following printcap entry would send output to the printer named "lp" on the machine "ucbvax".

```
lp|default line printer:\
        :lp=:rm=ucbvax:rp=lp:sd=/usr/spool/vaxlpd:
```

The **rm** entry is the name of the remote machine to connect to; this name must appear in the /etc/hosts database, see *hosts*(5). The **rp** capability indicates the name of the printer on the remote machine is "lp"; in this case it could be left out since this is the default value. The **sd** entry specifies "/usr/spool/vaxlpd" as the spooling directory instead of the default value of "/usr/spool/lpd".

4.2. Output filters

Filters are used to handle device dependencies and to perform accounting functions. The output filter **of** is used to filter text data to the printer device when accounting is not used or when all text data must be passed through a filter. It is not intended to perform accounting since it is started only once, all text files are filtered through it, and no provision is made for passing owners' login name, identifying the begining and ending of jobs, etc. The other filters (if specified) are started for each file printed and perform accounting if there is an **af** entry. If entries for both **of** and one of the other filters are specified, the output filter is used only to print the banner page; it is then stopped to allow other filters access to the printer. An example of a printer which requires output filters is the Benson-Varian.

```
va|varian|Benson-Varian:\
        :lp=/dev/va0:sd=/usr/spool/vad:of=/usr/lib/vpf:\
        :tf=/usr/lib/rvcat:mx#2000:pl#58:tr=\f:
```

The **tf** entry specifies "/usr/lib/rvcat" as the filter to be used in printing *troff*(1) output. This filter is needed to set the device into print mode for text, and plot mode for printing *troff* files and raster images (see *vp* (4S)). Note that the page length is set to 58 lines by the **pl** entry for 8.5" by 11" fan-fold paper. To enable accounting, the varian entry would be augmented with an **af** filter as shown below.

```
va|varian|Benson-Varian:\
        :lp=/dev/va0:sd=/usr/spool/vad:of=/usr/lib/vpf:\
        :if=/usr/lib/vpf:tf=/usr/lib/rvcat:af=/usr/adm/vaacct:\
        :mx#2000:pl#58:tr=\f:
```

5. Output filter specifications

For most devices or accounting methods, it is probably necessary to create a new filter.

Filters are spawned by *lpd* with their standard input the data to be printed, and standard output the printer. The standard error is attached to the **lf** file for logging errors. A filter must return a 0 exit code if there were no errors, 1 if the job should be reprinted, and 2 if the job should be thrown away. When *lprm* sends a kill signal to the *lpd* process controlling printing, it sends a SIGTERM signal to all filters and descendents of filters. This signal can be trapped by filters which need to perform cleanup operations such as deleting temporary files.

Arguments passed to a filter depend on its type. The **of** filter is called with the following arguments.

ofiler **–wwidth –llength**

The *width* and *length* values come from the **pw** and **pl** entries in the printcap database. The **if** filter is passed the following parameters.

filter **[–c] –wwidth –llength –iindent –n login –h host accounting_file**

The **–c** flag is optional, and only supplied when control characters are to be passed uninterpreted to the printer (when the **–l** option of *lpr* is used to print the file). The **–w** and **–l** parameters are the same as for the **of** filter. The **–n** and **–h** parameters specify the login name and host name of the job owner. The last argument is the name of the accounting file from *printcap*.

All other filters are called with the following arguments:

filter **–xwidth –ylength –n login –h host accounting_file**

The **–x** and **–y** options specify the horizontal and vertical page size in pixels (from the **px** and **py** entries in the printcap file). The rest of the arguments are the same as for the **if** filter.

6. Line printer Administration

The *lpc* program provides local control over line printer activity. The major commands and their intended use will be described. The command format and remaining commands are described in *lpc*(8).

abort and **start**

Abort terminates an active spooling daemon on the local host immediately and then disables printing (preventing new daemons from being started by *lpr*). This is normally used to force a hung line printer daemon to restart (i.e., *lpq* reports that there is a daemon present but nothing is happening). It does not remove any jobs from the queue (use the *lprm* command instead). *Start* enables printing and requests *lpd* to start printing jobs.

enable and **disable**

Enable and *disable* allow spooling in the local queue to be turned on/off. This will allow/prevent *lpr* from putting new jobs in the spool queue. It is frequently convenient to turn spooling off while testing new line printer filters since the *root* user can still use *lpr* to put jobs in the queue but no one else can. The other main use is to prevent users from putting jobs in the queue when the printer is expected to be unavailable for a long time.

restart

Restart allows ordinary users to restart printer daemons when *lpq* reports that there is no daemon present.

stop

Stop is used to halt a spooling daemon after the current job completes; this also disables printing. This is a clean way to shutdown a printer in order to perform maintenence, etc. Note that users can still enter jobs in a spool queue while a printer is *stopped*.

topq

Topq places jobs at the top of a printer queue. This can be used to reorder high priority jobs since *lpr* only provides first-come-first-serve ordering of jobs.

7. Troubleshooting

There are a number of messages which may be generated by the the line printer system. This section categorizes the most common and explains the cause for their generation. Where the message indicates a failure, directions are given to remedy the problem.

In the examples below, the name *printer* is the name of the printer. This would be one of the names from the *printcap* database.

7.1. LPR

lpr: ***printer*** **: unknown printer**

The *printer* was not found in the *printcap* database. Usually this is a typing mistake; however, it may indicate a missing or incorrect entry in the /etc/printcap file.

lpr: ***printer*** **: jobs queued, but cannot start daemon.**

The connection to *lpd* on the local machine failed. This usually means the printer server started at boot time has died or is hung. Check the local socket /dev/printer to be sure it still exists (if it does not exist, there is no *lpd* process running). Use

```
% ps ax | fgrep lpd
```

to get a list of process identifiers of running lpd's. The *lpd* to kill is the one which is not listed in any of the "lock" files (the lock file is contained in the spool directory of each printer). Kill the master daemon using the following command.

```
% kill pid
```

Then remove /dev/printer and restart the daemon (and printer) with the following commands.

```
% rm /dev/printer
% /usr/lib/lpd
```

Another possibility is that the *lpr* program is not setuid *root*, setgid *spooling*. This can be checked with

```
% ls -lg /usr/ucb/lpr
```

lpr: ***printer*** **: printer queue is disabled**

This means the queue was turned off with

```
% lpc disable printer
```

to prevent *lpr* from putting files in the queue. This is normally done by the system manager when a printer is going to be down for a long time. The printer can be turned back on by a super-user with *lpc*.

7.2. LPQ

waiting for ***printer*** **to become ready (offline ?)**

The printer device could not be opened by the daemon. This can happen for a number of reasons, the most common being that the printer is turned off-line. This message can also be generated if the printer is out of paper, the paper is jammed, etc. The actual reason is dependent on the meaning of error codes returned by system device driver. Not all printers supply sufficient information to distinguish when a printer is off-line or having trouble (e.g. a printer connected through a serial line). Another possible cause of this message is some other process, such as an output filter, has an exclusive open on the device. Your only recourse here is to kill off the offending program(s) and restart the printer with *lpc*.

printer **is ready and printing**

The *lpq* program checks to see if a daemon process exists for *printer* and prints the file *status*. If the daemon is hung, a super user can use *lpc* to abort the current daemon and start a new one.

waiting for *host* to come up

This indicates there is a daemon trying to connect to the remote machine named *host* in order to send the files in the local queue. If the remote machine is up, *lpd* on the remote machine is probably dead or hung and should be restarted as mentioned for *lpr*.

sending to *host*

The files should be in the process of being transferred to the remote *host*. If not, the local daemon should be aborted and started with *lpc*.

Warning: *printer* is down

The printer has been marked as being unavailable with *lpc*.

Warning: no daemon present

The *lpd* process overseeing the spooling queue, as indicated in the "lock" file in that directory, does not exist. This normally occurs only when the daemon has unexpectedly died. The error log file for the printer should be checked for a diagnostic from the deceased process. To restart an *lpd*, use

```
% lpc restart printer
```

7.3. LPRM

lprm: *printer*: cannot restart printer daemon

This case is the same as when *lpr* prints that the daemon cannot be started.

7.4. LPD

The *lpd* program can write many different messages to the error log file (the file specified in the **lf** entry in *printcap*). Most of these messages are about files which can not be opened and usually indicate the *printcap* file or the protection modes of the files are not correct. Files may also be inaccessible if people manually manipulate the line printer system (i.e. they bypass the *lpr* program).

In addition to messages generated by *lpd*, any of the filters that *lpd* spawns may also log messages to this file.

7.5. LPC

could't start printer

This case is the same as when *lpr* reports that the daemon cannot be started.

cannot examine spool directory

Error messages beginning with "cannot ..." are usually due to incorrect ownership and/or protection mode of the lock file, spooling directory or the *lpc* program.