Installing and Operating 4.4BSD UNIX
May 14, 1994

Marshall Kirk McKusick

Keith Bostic

Michael J. Karels

Samuel J. Leffler

Computer Systems Research Group
Department of Electrical Engineering and Computer Science
University of California, Berkeley
Berkeley, California 94720
(415) 642-7780

Mike Hibler

Center for Software Science
Department of Computer Science
University of Utah
Salt Lake City, Utah 84112
(801) 581-5017

ABSTRACT

This document contains instructions for the installation and operation of the 4.4BSD release of UNIX as distributed by The University of California at Berkeley.

It discusses procedures for installing UNIX on a new machine, and for upgrading an existing 4.3BSD UNIX system to the new release. An explanation of how to lay out filesystems on available disks and the space requirements for various parts of the system are given. A brief overview of the major changes to the system between 4.3BSD and 4.4BSD are outlined. An explanation of how to set up terminal lines and user accounts, and how to do system-specific tailoring is provided. A description of how to install and configure the 4.4BSD networking facilities is included. Finally, the document details system operation procedures: shutdown and startup, filesystem backup procedures, resource control, performance monitoring, and procedures for recompiling and reinstalling system software.

1. Introduction

This document explains how to install the 4.4BSD Berkeley version of UNIX on your system. The filesystem format is compatible with 4.3BSD and it will only be necessary for you to do a full bootstrap procedure if you are installing the release on a new machine. The object file formats are completely different from the System V release, so the most straightforward procedure for upgrading a System V system is to do a full bootstrap.

The full bootstrap procedure is outlined in section 2; the process starts with copying a filesystem image onto a new disk. This filesystem is then booted and used to extract the remainder of the system binaries and sources from the archives on the tape(s).

The technique for upgrading a 4.3BSD system is described in section 3 of this document. The upgrade procedure involves extracting system binaries onto new root and `/usr` filesystems and merging local configuration files into the new system. User filesystems may be upgraded in place. Most 4.3BSD binaries may be used with 4.4BSD in the course of the conversion. It is desirable to recompile local sources after the conversion, as the new compiler (GCC) provides superior code optimization. Consult section 3.5 for a description of some of the differences between 4.3BSD and 4.4BSD.

1.1. Distribution format

The distribution comes in two formats:

(3) 6250bpi 2400’ 9-track magnetic tapes, or
(1) 8mm Exabyte tape

If you have the facilities, we **strongly** recommend copying the magnetic tape(s) in the distribution kit to guard against disaster. The tapes contain 10240-byte records. There are interspersed tape marks; end-of-tape is signaled by a double end-of-file. The first file on the tape is architecture dependent. Additional files on the tape(s) contain tape archive images of the system binaries and sources (see *tar*(1)[1]). See the tape label for a description of the contents and format of each individual tape.

1.2. UNIX device naming

Device names have a different syntax depending on whether you are talking to the standalone system or a running UNIX kernel. The standalone system syntax is currently architecture dependent and is described in the various architecture specific sections as applicable. When not running standalone, devices are available via files in the `/dev/` directory. The file name typically encodes the device type, its logical unit and a partition within that unit. For example, `/dev/sd2b` refers to the second partition (‘‘b’’) of SCSI (‘‘sd’’) drive number ‘‘2’’, while `/dev/rmt0` refers to the raw (‘‘r’’) interface of 9-track tape (‘‘mt’’) unit ‘‘0’’.

The mapping of physical addressing information (e.g. controller, target) to a logical unit number is dependent on the system configuration. In all simple cases, where only a single controller is present, a drive with physical unit number 0 (e.g., as determined by its unit specification, either unit plug or other selection mechanism) will be called unit 0 in its UNIX file name. This is not, however, strictly necessary, since the system has a level of indirection in this naming. If there are multiple controllers, the disk unit numbers will normally be counted sequentially across controllers. This can be taken advantage of to make the system less dependent on the interconnect topology, and to make reconfiguration after hardware failure easier.

Each UNIX physical disk is divided into at most 8 logical disk partitions, each of which may occupy any consecutive cylinder range on the physical device. The cylinders occupied by the 8 partitions for each drive type are specified initially in the disk description file `/etc/disktab` (c.f. *disktab*(5)). The partition information and description of the drive geometry are written in one of the first sectors of each disk with the *disklabel*(8) program. Each partition may be used for either a raw data area such as a paging area or to store a UNIX filesystem. It is conventional for the first partition on a disk to be used to store a root

[1] References of the form *X*(Y) mean the entry named *X* in section Y of the ‘‘UNIX Programmer’s Manual’’.

filesystem, from which UNIX may be bootstrapped. The second partition is traditionally used as a paging area, and the rest of the disk is divided into spaces for additional ‘‘mounted filesystems’’ by use of one or more additional partitions.

1.3. UNIX devices: block and raw

UNIX makes a distinction between ‘‘block’’ and ‘‘raw’’ (character) devices. Each disk has a block device interface where the system makes the device byte addressable and you can write a single byte in the middle of the disk. The system will read out the data from the disk sector, insert the byte you gave it and put the modified data back. The disks with the names `/dev/xx0[a-h]`, etc., are block devices. There are also raw devices available. These have names like `/dev/rxx0[a-h]`, the ‘‘r’’ here standing for ‘‘raw’’. Raw devices bypass the buffer cache and use DMA directly to/from the program’s I/O buffers; they are normally restricted to full-sector transfers. In the bootstrap procedures we will often suggest using the raw devices, because these tend to work faster. Raw devices are used when making new filesystems, when checking unmounted filesystems, or for copying quiescent filesystems. The block devices are used to mount filesystems.

You should be aware that it is sometimes important whether to use the character device (for efficiency) or not (because it would not work, e.g. to write a single byte in the middle of a sector). Do not change the instructions by using the wrong type of device indiscriminately.

2. Bootstrap procedure

This section explains the bootstrap procedure that can be used to get the kernel supplied with this distribution running on your machine. If you are not currently running 4.3BSD you will have to do a full bootstrap. Section 3 describes how to upgrade a 4.3BSD system. An understanding of the operations used in a full bootstrap is helpful in doing an upgrade as well. In either case, it is highly desirable to read and understand the remainder of this document before proceeding.

The distribution supports a somewhat wider set of machines than those for which we have built binaries. The architectures that are supported only in source form include:

- Intel 386/486-based machines (ISA/AT or EISA bus only)
- Sony News MIPS-based workstations
- Omron Luna 68000-based workstations

If you wish to run one of these architectures, you will have to build a cross compilation environment. Note that the distribution does **not** include the machine support for the Tahoe and VAX architectures found in previous BSD distributions. Our primary development environment is the HP9000/300 series machines. The other architectures are developed and supported by people outside the university. Consequently, we are not able to directly test or maintain these other architectures, so cannot comment on their robustness, reliability, or completeness.

2.1. Bootstrapping from the tape

The set of files on the distribution tape are as follows:

1) A *dd*(1) (HP300), *tar*(1) (DECstation), or *dump*(8) (SPARC) image of the root filesystem
2) A *tar* image of the `/var` filesystem
3) A *tar* image of the `/usr` filesystem
4) A *tar* image of `/usr/src/sys`
5) A *tar* image of `/usr/src` except sys and contrib
6) A *tar* image of `/usr/src/contrib`
7) (8mm Exabyte tape distributions only) A *tar* image of `/usr/src/X11R5`

The tape bootstrap procedure used to create a working system involves the following major steps:

1) Transfer a bootable root filesystem from the tape to a disk and get it booted and running.

2) Build and restore the `/var` and `/usr` filesystems from tape with *tar*(1).

3) Extract the system and utility source files as desired.

The following sections describe the above steps in detail. The details of the first step vary between architectures. The specific steps for the HP300, SPARC, and DECstation are given in the next three sections respectively. You should follow the instructions for your particular architecture. In all sections, commands you are expected to type are shown in italics, while that information printed by the system is shown emboldened.

2.2. Booting the HP300

2.2.1. Supported hardware

The hardware supported by 4.4BSD for the HP300/400 is as follows:

CPU's	68020 based (318, 319, 320, 330 and 350), 68030 based (340, 345, 360, 370, 375, 400) and 68040 based (380, 425, 433).
DISK's	HP-IB/CS80 (7912, 7914, 7933, 7936, 7945, 7957, 7958, 7959, 2200, 2203) and SCSI-I (including magneto-optical).
TAPE's	Low-density CS80 cartridge (7914, 7946, 9144), high-density CS80 cartridge (9145), HP SCSI DAT and SCSI Exabyte.
RS232	98644 built-in single-port, 98642 4-port and 98638 8-port interfaces.
NETWORK	98643 internal and external LAN cards.
GRAPHICS	Terminal emulation and raw frame buffer support for 98544 / 98545 / 98547 (Topcat color & monochrome), 98548 / 98549 / 98550 (Catseye color & monochrome), 98700 / 98710 (Gatorbox), 98720 / 98721 (Renaissance), 98730 / 98731 (DaVinci) and A1096A (Hyperion monochrome).
INPUT	General interface supporting all HIL devices. (e.g. keyboard, 2 and 3 button mice, ID module, ...)
MISC	Battery-backed real time clock, builtin and 98625A/B HP-IB interfaces, builtin and 98658A SCSI interfaces, serial printers and plotters on HP-IB, and SCSI autochanger device.

Major items that are not supported include the 310 and 332 CPU's, 400 series machines configured for Domain/OS, EISA and VME bus adaptors, audio, the centronics port, 1/2" tape drives (7980), CD-ROM, and the PVRX/TVRX 3D graphics displays.

2.2.2. Standalone device file naming

The standalone system device name syntax on the HP300 is of the form:

xx(a,c,u,p)

where *xx* is the device type, *a* specifies the adaptor to use, *c* the controller, *u* the unit, and *p* a partition. The *device type* differentiates the various disks and tapes and is one of: ‘‘rd’’ for HP-IB CS80 disks, ‘‘ct’’ for HP-IB CS80 cartridge tapes, or ‘‘sd’’ for SCSI-I disks (SCSI-I tapes are currently not supported). The *adaptor* field is a logical HP-IB or SCSI bus adaptor card number. This will typically be 0 for SCSI disks, 0 for devices on the ‘‘slow’’ HP-IB interface (usually tapes) and 1 for devices on the ‘‘fast’’ HP-IB interface (usually disks). To get a complete mapping of physical (select-code) to logical card numbers just type a ˆC at the standalone prompt. The *controller* field is the disk or tape's target number on the HP-IB or SCSI bus. For SCSI the range is 0 to 6 (7 is the adaptor address) and for HP-IB the range is 0 to 7. The *unit* field is unused and should be 0. The *partition* field is interpreted differently for tapes and disks: for disks it is a disk partition (in the range 0-7), and for tapes it is a file number offset on the tape. Thus, partition 2 of a SCSI disk drive at target 3 on SCSI bus 1 would be ‘‘sd(1,3,0,2)’’. If you have only one of any type bus adaptor, you may omit the adaptor and controller numbers; e.g. ‘‘sd(0,2)’’ could be used instead of ‘‘sd(0,0,0,2)’’. The following examples always use the full syntax for clarity.

2.2.3. The procedure

The basic steps involved in bringing up the HP300 are as follows:

1) Obtain a second disk and format it, if necessary.
2) Copy a root filesystem from the tape onto the beginning of the disk.
3) Boot the UNIX system on the new disk.
4) (Optional) Build a root filesystem optimized for your disk.
5) Label the disks with the *disklabel*(8) program.

2.2.3.1. Step 1: selecting and formatting a disk

For your first system you will have to obtain a formatted disk of a type given in the ‘‘supported hardware’’ list above. If you want to load an entire binary system (i.e., everything except `/usr/src`), on the single disk you will need a minimum of 290MB, ruling out anything smaller than a 7959B/S disk. The disklabel included in the bootstrap root image is laid out to accommodate this scenario. Note that an HP SCSI magneto-optical disk will work fine for this case. 4.4BSD will boot and run (albeit slowly) using one. If you want to load source on a single disk system, you will need at least 640MB (at least a 2213A SCSI or 2203A HP-IB disk). A disk as small as the 7945A (54MB) can be used for the bootstrap procedure but will hold only the root and primary swap partitions. If you plan to use multiple disks, refer to section 2.5 for suggestions on partitioning.

After selecting a disk, you may need to format it. Since most HP disk drives come pre-formatted (except optical media) you probably will not, but if necessary, you can format a disk under HP-UX using the *mediainit*(1m) program. Once you have 4.4BSD up and running on one machine you can use the *scsiformat*(8) program to format additional SCSI disks. Any additional HP-IB disks will have to be formatted using HP-UX.

2.2.3.2. Step 2: copying the root filesystem from tape to disk

Once you have a formatted second disk you can use the *dd*(1) command under HP-UX to copy the root filesystem image from the tape to the beginning of the second disk. For HP’s, the root filesystem image is the first file on the tape. It includes a disklabel and bootblock along with the root filesystem. An example command to copy the image from tape to the beginning of a disk is:

```
dd if=/dev/rmt/0m of=/dev/rdsk/1s0 bs=20b
```

The actual special file syntax may vary depending on unit numbers and the version of HP-UX that is running. Consult the HP-UX *mt*(7) and *disk*(7) man pages for details.

Note that if you have a SCSI disk, you don’t necessarily have to use HP-UX (or an HP) to create the boot disk. Any machine and operating system that will allow you to copy the raw disk image out to block 0 of the disk will do.

If you have only a single machine with a single disk, you may still be able to install and boot 4.4BSD if you have an HP-IB cartridge tape drive. If so, you can use a more difficult approach of booting a standalone copy program from the tape, and using that to copy the root filesystem image from the tape to the disk. To do this, you need to extract the first file of the distribution tape (the root image), copy it over to a machine with a cartridge drive and then copy the image onto tape. For example:

```
dd if=/dev/rst0 of=bootimage bs=20b
rcp bootimage foo:/tmp/bootimage
<login to foo>
dd if=/tmp/bootimage of=/dev/rct/0m bs=20b
```

Once this tape is created you can boot and run the standalone tape copy program from it. The copy program is loaded just as any other program would be loaded by the bootrom in ‘‘attended’’ mode: reset the CPU, hold down the space bar until the word ‘‘Keyboard’’ appears in the installed interface list, and enter the menu selection for SYS_TCOPY. Once loaded and running:

From: *^C*	(control-C to see logical adaptor assignments)
hpib0 at sc7	
scsi0 at sc14	
From: *ct(0,7,0,0)*	(HP-IB tape, target 7, first tape file)
To: *sd(0,0,0,2)*	(SCSI disk, target 0, third partition)
Copy completed: 1728 records copied	

This copy will likely take 30 minutes or more.

2.2.3.3. Step 3: booting the root filesystem

You now have a bootable root filesystem on the disk. If you were previously running with two disks, it would be best if you shut down the machine and turn off power on the HP-UX drive. It will be less confusing and it will eliminate any chance of accidentally destroying the HP-UX disk. If you used a cartridge tape for booting you should also unload the tape at this point. Whether you booted from tape or copied from disk you should now reboot the machine and do another attended boot (see previous section), this time with SYS_TBOOT. Once loaded and running the boot program will display the CPU type and prompt for a kernel file to boot:

HP433 CPU
Boot
: */vmunix*

After providing the kernel name, the machine will boot 4.4BSD with output that looks about like this:

```
597480+34120+139288 start 0xfe8019ec
Copyright (c) 1982, 1986, 1989, 1991, 1993
        The Regents of the University of California.
Copyright (c) 1992 Hewlett-Packard Company
Copyright (c) 1992 Motorola Inc.
All rights reserved.

4.4BSD UNIX #1: Tue Jul 20 11:40:36 PDT 1993
   mckusick@vangogh.CS.Berkeley.EDU:/usr/obj/sys/compile/GENERIC.hp300
HP9000/433 (33MHz MC68040 CPU+MMU+FPU, 4k on-chip physical I/D caches)
real mem = xxx
avail mem = ###
using ### buffers containing ### bytes of memory
(... information about available devices ...)
root device?
```

The first three numbers are printed out by the bootstrap program and are the sizes of different parts of the system (text, initialized and uninitialized data). The system also allocates several system data structures after it starts running. The sizes of these structures are based on the amount of available memory and the maximum count of active users expected, as declared in a system configuration description. This will be discussed later.

UNIX itself then runs for the first time and begins by printing out a banner identifying the release and version of the system that is in use and the date that it was compiled.

Next the *mem* messages give the amount of real (physical) memory and the memory available to user programs in bytes. For example, if your machine has 16Mb bytes of memory, then **xxx** will be 16777216.

The messages that come out next show what devices were found on the current processor. These messages are described in *autoconf*(4). The distributed system may not have found all the communications devices you have or all the mass storage peripherals you have, especially if you have more than two of anything. You will correct this when you create a description of your machine from which to configure a site-dependent version of UNIX. The messages printed at boot here contain much of the information that will be used in creating the configuration. In a correctly configured system most of the information present in the configuration description is printed out at boot time as the system verifies that each device is present.

The ‘‘root device?’’ prompt was printed by the system to ask you for the name of the root filesystem to use. This happens because the distribution system is a *generic* system, i.e., it can be bootstrapped on a cpu with its root device and paging area on any available disk drive. You will most likely respond to the root device question with ‘‘sd0’’ if you are booting from a SCSI disk, or with ‘‘rd0’’ if you are booting from an HP-IB disk. This response shows that the disk it is running on is drive 0 of type ‘‘sd’’ or ‘‘rd’’ respectively. If you have other disks attached to the system, it is possible that the drive you are using will not be configured as logical drive 0. Check the autoconfiguration messages printed out by the kernel to make sure. These messages will show the type of every logical drive and their associated controller and slave addresses. You will later build a system tailored to your configuration that will not prompt you for a root device when it is bootstrapped.

root device? *sd0*
WARNING: preposterous time in filesystem −− CHECK AND RESET THE DATE!
erase ˆ?, kill ˆU, intr ˆC
#

The ‘‘erase ...’’ message is part of the `/.profile` that was executed by the root shell when it started. This message tells you about the settings of the character erase, line erase, and interrupt characters.

UNIX is now running, and the *UNIX Programmer’s Manual* applies. The ‘‘#’’ is the prompt from the Bourne shell, and lets you know that you are the super-user, whose login name is ‘‘root’’.

At this point, the root filesystem is mounted read-only. Before continuing the installation, the filesystem needs to be ‘‘updated’’ to allow writing and device special files for the following steps need to be created. This is done as follows:

# *mount_mfs -s 1000 -T type /dev/null /tmp*	(create a writable filesystem)
(*type* is the disk type as determined from /etc/disktab)	
# *cd /tmp*	(connect to that directory)
# *../dev/MAKEDEV sd#*	(create special files for root disk)
(*sd* is the disk type, *#* is the unit number)	
(ignore warning from ‘‘sh’’)	
# *mount −uw /tmp/sd#a /*	(read-write mount root filesystem)
# *cd /dev*	(go to device directory)
# *./MAKEDEV sd#*	(create permanent special files for root disk)
(again, ignore warning from ‘‘sh’’)	

2.2.3.4. Step 4: (optional) restoring the root filesystem

The root filesystem that you are currently running on is complete, however it probably is not optimally laid out for the disk on which you are running. If you will be cloning copies of the system onto multiple disks for other machines, you are advised to connect one of these disks to this machine, and build and restore a properly laid out root filesystem onto it. If this is the only machine on which you will be running 4.4BSD or peak performance is not an issue, you can skip this step and proceed directly to step 5.

Connect a second disk to your machine. If you bootstrapped using the two disk method, you can overwrite your initial HP-UX disk, as it will no longer be needed (assuming you have no plans to run HP-UX again).

To really create the root filesystem on drive 1 you should first label the disk as described in step 5 below. Then run the following commands:

```
# cd /dev
# ./MAKEDEV sd1a
# newfs /dev/rsd1a
# mount /dev/sd1a /mnt
# cd /mnt
# dump 0f – /dev/rsd0a | restore xf –
```

(Note: restore will ask if you want to ‘‘set owner/mode for ’.’’’
to which you should reply ‘‘yes’’.)

When this completes, you should then shut down the system, and boot on the disk that you just created following the procedure in step (3) above.

2.2.3.5. Step 5: placing labels on the disks

For each disk on the HP300, 4.4BSD places information about the geometry of the drive and the partition layout at byte offset 1024. This information is written with *disklabel*(8).

The root image just loaded includes a ‘‘generic’’ label intended to allow easy installation of the root and `/usr` and may not be suitable for the actual disk on which it was installed. In particular, it may make your disk appear larger or smaller than its real size. In the former case, you lose some capacity. In the latter, some of the partitions may map non-existent sectors leading to errors if those partitions are used. It is also possible that the defined geometry will interact poorly with the filesystem code resulting in reduced performance. However, as long as you are willing to give up a little space, not use certain partitions or suffer minor performance degradation, you might want to avoid this step; especially if you do not know how to use *ed*(1).

If you choose to edit this label, you can fill in correct geometry information from `/etc/disktab`. You may also want to rework the ‘‘e’’ and ‘‘f’’ partitions used for loading `/usr` and `/var`. You should not attempt to, and *disklabel* will not let you, modify the ‘‘a’’, ‘‘b’’ and ‘‘d’’ partitions. To edit a label:

```
# EDITOR=ed
# export EDITOR
# disklabel -r -e /dev/rXX#d
```

where **XX** is the type and **#** is the logical drive number; e.g. `/dev/rsd0d` or `/dev/rrd0d`. Note the explicit use of the ‘‘d’’ partition. This partition includes the bootblock as does ‘‘c’’ and using it allows you to change the size of ‘‘c’’.

If you wish to label any additional disks, run the following command for each:

```
# disklabel -rw XX# type "optional_pack_name"
```

where **XX#** is the same as in the previous command and **type** is the HP300 disk device name as listed in `/etc/disktab`. The optional information may contain any descriptive name for the contents of a disk, and may be up to 16 characters long. This procedure will place the label on the disk using the information found in `/etc/disktab` for the disk type named. If you have changed the disk partition sizes, you may wish to add entries for the modified configuration in `/etc/disktab` before labeling the affected disks.

You have now completed the HP300 specific part of the installation. Now proceed to the generic part of the installation described starting in section 2.5 below. Note that where the disk name ‘‘sd’’ is used throughout section 2.5, you should substitute the name ‘‘rd’’ if you are running on an HP-IB disk. Also, if you are loading on a single disk with the default disklabel, `/var` should be restored to the ‘‘f’’ partition and `/usr` to the ‘‘e’’ partition.

2.3. Booting the SPARC

2.3.1. Supported hardware

The hardware supported by 4.4BSD for the SPARC is as follows:

CPU's	SPARCstation 1 series (1, 1+, SLC, IPC) and SPARCstation 2 series (2, IPX).
DISK's	SCSI.
TAPE's	none.
NETWORK	SPARCstation Lance (le).
GRAPHICS	bwtwo, cgthree, and the GX (cgsix).
INPUT	Keyboard and mouse.
MISC	Battery-backed real time clock, built-in serial devices, Sbus SCSI controller, and audio device.

Major items that are not supported include anything VME-based, the floppy disk, and SCSI tapes.

2.3.2. Limitations

There are several important limitations on the 4.4BSD distribution for the SPARC:

1) You **must** have SunOS 4.1.x or Solaris to bring up 4.4BSD. There is no SPARCstation bootstrap code in this distribution. The Sun-supplied boot loader will be used to boot 4.4BSD; you must copy this from your SunOS distribution. This imposes several restrictions on the system, as detailed below.

2) The 4.4BSD SPARC kernel does not remap SCSI IDs. A SCSI disk at target 0 will become ‘‘sd0’’, where in SunOS the same disk will normally be called ‘‘sd3’’. If your existing SunOS system is diskful, it will be least painful to have SunOS running on the disk on target 3 lun 0 and put 4.4BSD on the disk on target 0 lun 0. Both systems will then think they are running on ‘‘sd0’’, and you can boot either system as needed simply by changing the EEPROM's boot device.

3) There is no SCSI tape driver. You must have another system for tape reading and backups.

4) Although the 4.4BSD SPARC kernel will handle existing SunOS shared libraries, it does not use or create them itself, and therefore requires much more disk space than SunOS does.

5) It is currently difficult (though not completely impossible) to run 4.4BSD diskless. These instructions assume you will have a local boot, swap, and root filesystem.

6) When using a serial port rather than a graphics display as the console, only port `ttya` can be used. Attempts to use port `ttyb` will fail when the kernel tries to print the boot up messages to the console.

2.3.3. The procedure

You must have a spare disk on which to place 4.4BSD. The steps involved in bootstrapping this tape are as follows:

1) Bring up SunOS (preferably SunOS 4.1.x or Solaris 1.x, although Solaris 2 may work — this is untested).

2) Attach auxiliary SCSI disk(s). Format and label using the SunOS formatting and labeling programs as needed. Note that the root filesystem currently requires at least 10 MB; 16 MB or more is recommended. The b partition will be used for swap; this should be at least 32 MB.

3) Use the SunOS *newfs* to build the root filesystem. You may also want to build other filesystems at the same time. (By default, the 4.4BSD *newfs* builds a filesystem that SunOS will not handle; if you plan to switch OSes back and forth you may want to sacrifice the performance gain from the new filesystem format for compatibility.) You can build an old-format filesystem on 4.4BSD by giving the −O option to *newfs*(8). *Fsck*(8) can convert old format filesystems to new format filesystems, but not vice versa, so you may want to initially build old format filesystems so that they can be mounted under SunOS, and then later convert them to new format filesystems when you are satisfied that 4.4BSD is running properly. In any case, **you must build an old-style root filesystem** so that the SunOS boot program will work.

4) Mount the new root, then copy the SunOS `/boot` into place and use the SunOS ‘‘installboot’’ program to enable disk-based booting. Note that the filesystem must be mounted when you do the ‘‘installboot’’:

```
# mount /dev/sd3a /mnt
# cp /boot /mnt/boot
# cd /usr/kvm/mdec
# installboot /mnt/boot bootsd /dev/rsd3a
```

The SunOS `/boot` will load 4.4BSD kernels; there is no SPARCstation bootstrap code on the distribution. Note that the SunOS `/boot` does not handle the new 4.4BSD filesystem format.

5) Restore the contents of the 4.4BSD root filesystem.

```
# cd /mnt
# rrestore xf tapehost:/dev/nrst0
```

6) Boot the supplied kernel:

```
# halt
ok boot sd(0,3)vmunix -s      [for old proms] OR
ok boot disk3 -s          [for new proms]
... [4.4BSD boot messages]
```

To install the remaining filesystems, use the procedure described starting in section 2.5. In these instructions, `/usr` should be loaded into the ‘‘e’’ partition and `/var` in the ‘‘f’’ partition.

After completing the filesystem installation you may want to set up 4.4BSD to reboot automatically:

```
# halt
ok setenv boot-from sd(0,3)vmunix [for old proms] OR
ok setenv boot-device disk3       [for new proms]
```

If you build backwards-compatible filesystems, either with the SunOS newfs or with the 4.4BSD ‘‘−O’’ option, you can mount these under SunOS. The SunOS fsck will, however, always think that these filesystems are corrupted, as there are several new (previously unused) superblock fields that are updated in 4.4BSD. Running ‘‘fsck −b32’’ and letting it ‘‘fix’’ the superblock will take care of this.

If you wish to run SunOS binaries that use SunOS shared libraries, you simply need to copy all the dynamic linker files from an existing SunOS system:

```
# rcp sunos-host:/etc/ld.so.cache /etc/
# rcp sunos-host:'/usr/lib/*.so*' /usr/lib/
```

The SunOS compiler and linker should be able to produce SunOS binaries under 4.4BSD, but this has not been tested. If you plan to try it you will need the appropriate .sa files as well.

2.4. Booting the DECstation

2.4.1. Supported hardware

The hardware supported by 4.4BSD for the DECstation is as follows:

CPU's	R2000 based (3100) and R3000 based (5000/200, 5000/20, 5000/25, 5000/1xx).
DISK's	SCSI-I (tested RZ23, RZ55, RZ57, Maxtor 8760S).
TAPE's	SCSI-I (tested DEC TK50, Archive DAT, Emulex MT02).
RS232	Internal DEC dc7085 and AMD 8530 based interfaces.
NETWORK	TURBOchannel PMAD-AA and internal LANCE based interfaces.
GRAPHICS	Terminal emulation and raw frame buffer support for 3100 (color & monochrome), TURBOchannel PMAG-AA, PMAG-BA, PMAG-DV.
INPUT	Standard DEC keyboard (LK201) and mouse.
MISC	Battery-backed real time clock, internal and TURBOchannel PMAZ-AA SCSI interfaces.

Major items that are not supported include the 5000/240 (there is code but not compiled in or tested), R4000 based machines, FDDI and audio interfaces. Diskless machines are not supported but booting kernels and bootstrapping over the network is supported on the 5000 series.

2.4.2. The procedure

The first file on the distribution tape is a tar file that contains four files. The first step requires a running UNIX (or ULTRIX) system that can be used to extract the tar archive from the first file on the tape. The command:

```
tar xf /dev/rmt0
```

will extract the following four files:

A) root.image: *dd* image of the root filesystem
B) vmunix.tape: *dd* image for creating boot tapes
C) vmunix.net: file for booting over the network
D) root.dump: *dump* image of the root filesystem

There are three basic ways a system can be bootstrapped corresponding to the first three files. You may want to read the section on bootstrapping the HP300 since many of the steps are similar. A spare, formatted SCSI disk is also useful.

2.4.2.1. Procedure A: copy root filesystem to disk

This procedure is similar to the HP300. If you have an extra disk, the easiest approach is to use *dd* (1) under ULTRIX to copy the root filesystem image to the beginning of the spare disk. The root filesystem image includes a disklabel and bootblock along with the root filesystem. An example command to copy the image to the beginning of a disk is:

```
dd if=root.image of=/dev/rz1c bs=20b
```

The actual special file syntax will vary depending on unit numbers and the version of ULTRIX that is running. This system is now ready to boot. You can boot the kernel with one of the following PROM commands. If you are booting on a 3100, the disk must be SCSI id zero because of a bug.

```
DEC 3100:    boot −f rz(0,0,0)vmunix
DEC 5000:    boot 5/rz0/vmunix
```

You can then proceed to section 2.5 to create reasonable disk partitions for your machine and then install the rest of the system.

2.4.2.2. Procedure B: bootstrap from tape

If you have only a single machine with a single disk, you need to use the more difficult approach of booting a kernel and mini-root from tape or the network, and using it to restore the root filesystem.

First, you will need to create a boot tape. This can be done using *dd* as in the following example.

```
dd if=vmunix.tape of=/dev/nrmt0 bs=1b
dd if=root.dump of=/dev/nrmt0 bs=20b
```

The actual special file syntax for the tape drive will vary depending on unit numbers, tape device and the version of ULTRIX that is running.

The first file on the boot tape contains a boot header, kernel, and mini-root filesystem that the PROM can copy into memory. Installing from tape has only been tested on a 3100 and a 5000/200 using a TK50 tape drive. Here are two example PROM commands to boot from tape.

```
DEC 3100:    boot -f tz(0,5,0) m     # 5 is the SCSI id of the TK50
DEC 5000:    boot 5/tz6 m            # 6 is the SCSI id of the TK50
```

The ‘m’ argument tells the kernel to look for a root filesystem in memory. Next you should proceed to section 2.4.3 to build a disk-based root filesystem.

2.4.2.3. Procedure C: bootstrap over the network

You will need a host machine that is running the *bootp* server with the `vmunix.net` file installed in the default directory defined by the configuration file for *bootp*. Here are two example PROM commands to boot across the net:

```
DEC 3100:    boot -f tftp()vmunix.net m
DEC 5000:    boot 6/tftp/vmunix.net m
```

This command should load the kernel and mini-root into memory and run the same as the tape install (procedure B). The rest of the steps are the same except you will need to start the network (if you are unsure how to fill in the <name> fields below, see sections 4.4 and 5). Execute the following to start the networking:

```
# mount -uw /
# echo 127.0.0.1 localhost >> /etc/hosts
# echo <your.host.inet.number> myname.my.domain myname >> /etc/hosts
# echo <friend.host.inet.number> myfriend.my.domain myfriend >> /etc/hosts
# ifconfig le0 inet myname
```

Next you should proceed to section 2.4.3 to build a disk-based root filesystem.

2.4.3. Label disk and create the root filesystem

There are five steps to create a disk-based root filesystem.

1) Label the disk.

```
# disklabel -W /dev/rrz?c        # This enables writing the label
# disklabel -w -r -B /dev/rrz?c $DISKTYPE
# newfs /dev/rrz?a
...
# fsck /dev/rrz?a
...
```

Supported disk types are listed in `/etc/disktab`.

2) Restore the root filesystem.

```
# mount -uw /
# mount /dev/rz?a /a
# cd /a
```

If you are restoring locally (procedure B), run:

```
# mt −f /dev/nrmt0 rew
# restore −xsf 2 /dev/rmt0
```

If you are restoring across the net (procedure c), run:

```
# rrestore xf myfriend:/path/to/root.dump
```

When the restore finishes, clean up with:

```
# cd /
# sync
# umount /a
# fsck /dev/rz?a
```

3) Reset the system and initialize the PROM monitor to boot automatically.

```
DEC 3100:    setenv bootpath boot −f rz(0,?,0)vmunix
DEC 5000:    setenv bootpath 5/rz?/vmunix -a
```

4) After booting UNIX, you will need to create `/dev/mouse` to run X windows as in the following example.

```
rm /dev/mouse
ln /dev/xx /dev/mouse
```

The ’xx’ should be one of the following:

pm0	raw interface to PMAX graphics devices
cfb0	raw interface to TURBOchannel PMAG-BA color frame buffer
xcfb0	raw interface to maxine graphics devices
mfb0	raw interface to mono graphics devices

You can then proceed to section 2.5 to install the rest of the system. Note that where the disk name ‘‘sd’’ is used throughout section 2.5, you should substitute the name ‘‘rz’’.

2.5. Disk configuration

All architectures now have a root filesystem up and running and proceed from this point to layout filesystems to make use of the available space and to balance disk load for better system performance.

2.5.1. Disk naming and divisions

Each physical disk drive can be divided into up to 8 partitions; UNIX typically uses only 3 or 4 partitions. For instance, the first partition, sd0a, is used for a root filesystem, a backup thereof, or a small filesystem like, `/var/tmp`; the second partition, sd0b, is used for paging and swapping; and a third partition, typically sd0e, holds a user filesystem.

The space available on a disk varies per device. Each disk typically has a paging area of 30 to 100 megabytes and a root filesystem of about 17 megabytes. The distributed system binaries occupy about 150 (180 with X11R5) megabytes while the major sources occupy another 250 (340 with X11R5) megabytes. The `/var` filesystem as delivered on the tape is only 2Mb, however it should have at least 50Mb allocated to it just for normal system activity. Usually it is allocated the last partition on the disk so that it can provide as much space as possible to the `/var/users` filesystem. See section 2.5.4 for further details on disk layouts.

Be aware that the disks have their sizes measured in disk sectors (usually 512 bytes), while the UNIX filesystem blocks are variable sized. If BLOCKSIZE=1k is set in the user’s environment, all user programs report disk space in kilobytes, otherwise, disk sizes are always reported in units of 512-byte sectors[2]. The

[2] You can thank System V intransigence and POSIX duplicity for requiring that 512-byte blocks be the units that programs report.

`/etc/disktab` file used in labelling disks and making filesystems specifies disk partition sizes in sectors.

2.5.2. Layout considerations

There are several considerations in deciding how to adjust the arrangement of things on your disks. The most important is making sure that there is adequate space for what is required; secondarily, throughput should be maximized. Paging space is an important parameter. The system, as distributed, sizes the configured paging areas each time the system is booted. Further, multiple paging areas of different sizes may be interleaved.

Many common system programs (C, the editor, the assembler etc.) create intermediate files in the `/tmp` directory, so the filesystem where this is stored also should be made large enough to accommodate most high-water marks. Typically, `/tmp` is constructed from a memory-based filesystem (see *mount_mfs*(8)). Programs that want their temporary files to persist across system reboots (such as editors) should use `/var/tmp`. If you plan to use a disk-based `/tmp` filesystem to avoid loss across system reboots, it makes sense to mount this in a ‘‘root’’ (i.e. first partition) filesystem on another disk. All the programs that create files in `/tmp` take care to delete them, but are not immune to rare events and can leave dregs. The directory should be examined every so often and the old files deleted.

The efficiency with which UNIX is able to use the CPU is often strongly affected by the configuration of disk controllers; it is critical for good performance to balance disk load. There are at least five components of the disk load that you can divide between the available disks:

1) The root filesystem.
2) The `/var` and `/var/tmp` filesystems.
3) The `/usr` filesystem.
4) The user filesystems.
5) The paging activity.

The following possibilities are ones we have used at times when we had 2, 3 and 4 disks:

	disks		
what	2	3	4
root	0	0	0
var	1	2	3
usr	1	1	1
paging	0+1	0+2	0+2+3
users	0	0+2	0+2
archive	x	x	3

The most important things to consider are to even out the disk load as much as possible, and to do this by decoupling filesystems (on separate arms) between which heavy copying occurs. Note that a long term average balanced load is not important; it is much more important to have an instantaneously balanced load when the system is busy.

Intelligent experimentation with a few filesystem arrangements can pay off in much improved performance. It is particularly easy to move the root, the `/var` and `/var/tmp` filesystems and the paging areas. Place the user files and the `/usr` directory as space needs dictate and experiment with the other, more easily moved filesystems.

2.5.3. Filesystem parameters

Each filesystem is parameterized according to its block size, fragment size, and the disk geometry characteristics of the medium on which it resides. Inaccurate specification of the disk characteristics or haphazard choice of the filesystem parameters can result in substantial throughput degradation or significant waste of disk space. As distributed, filesystems are configured according to the following table.

Filesystem	Block size	Fragment size
root	8 kbytes	1 kbytes
usr	8 kbytes	1 kbytes
users	4 kbytes	512 bytes

The root filesystem block size is made large to optimize bandwidth to the associated disk. The large block size is important as many of the most heavily used programs are demand paged out of the `/bin` directory. The fragment size of 1 kbyte is a ‘‘nominal’’ value to use with a filesystem. With a 1 kbyte fragment size disk space utilization is about the same as with the earlier versions of the filesystem.

The filesystems for users have a 4 kbyte block size with 512 byte fragment size. These parameters have been selected based on observations of the performance of our user filesystems. The 4 kbyte block size provides adequate bandwidth while the 512 byte fragment size provides acceptable space compaction and disk fragmentation.

Other parameters may be chosen in constructing filesystems, but the factors involved in choosing a block size and fragment size are many and interact in complex ways. Larger block sizes result in better throughput to large files in the filesystem as larger I/O requests will then be done by the system. However, consideration must be given to the average file sizes found in the filesystem and the performance of the internal system buffer cache. The system currently provides space in the inode for 12 direct block pointers, 1 single indirect block pointer, 1 double indirect block pointer, and 1 triple indirect block pointer. If a file uses only direct blocks, access time to it will be optimized by maximizing the block size. If a file spills over into an indirect block, increasing the block size of the filesystem may decrease the amount of space used by eliminating the need to allocate an indirect block. However, if the block size is increased and an indirect block is still required, then more disk space will be used by the file because indirect blocks are allocated according to the block size of the filesystem.

In selecting a fragment size for a filesystem, at least two considerations should be given. The major performance tradeoffs observed are between an 8 kbyte block filesystem and a 4 kbyte block filesystem. Because of implementation constraints, the block size versus fragment size ratio can not be greater than 8. This means that an 8 kbyte filesystem will always have a fragment size of at least 1 kbytes. If a filesystem is created with a 4 kbyte block size and a 1 kbyte fragment size, then upgraded to an 8 kbyte block size and 1 kbyte fragment size, identical space compaction will be observed. However, if a filesystem has a 4 kbyte block size and 512 byte fragment size, converting it to an 8K/1K filesystem will result in 4-8% more space being used. This implies that 4 kbyte block filesystems that might be upgraded to 8 kbyte blocks for higher performance should use fragment sizes of at least 1 kbytes to minimize the amount of work required in conversion.

A second, more important, consideration when selecting the fragment size for a filesystem is the level of fragmentation on the disk. With an 8:1 fragment to block ratio, storage fragmentation occurs much sooner, particularly with a busy filesystem running near full capacity. By comparison, the level of fragmentation in a 4:1 fragment to block ratio filesystem is one tenth as severe. This means that on filesystems where many files are created and deleted, the 512 byte fragment size is more likely to result in apparent space exhaustion because of fragmentation. That is, when the filesystem is nearly full, file expansion that requires locating a contiguous area of disk space is more likely to fail on a 512 byte filesystem than on a 1 kbyte filesystem. To minimize fragmentation problems of this sort, a parameter in the super block specifies a minimum acceptable free space threshold. When normal users (i.e. anyone but the super-user) attempt to allocate disk space and the free space threshold is exceeded, the user is returned an error as if the filesystem were really full. This parameter is nominally set to 5%; it may be changed by supplying a parameter to *newfs*(8), or by updating the super block of an existing filesystem using *tunefs*(8).

Finally, a third, less common consideration is the attributes of the disk itself. The fragment size should not be smaller than the physical sector size of the disk. As an example, the HP magneto-optical disks have 1024 byte physical sectors. Using a 512 byte fragment size on such disks will work but is extremely inefficient.

Note that the above discussion considers block sizes of up to only 8k. As of the 4.4 release, the maximum block size has been increased to 64k. This allows an entirely new set of block/fragment combinations for which there is little experience to date. In general though, unless a filesystem is to be used for a special purpose application (for example, storing image processing data), we recommend using the values supplied above. Remember that the current implementation limits the block size to at most 64 kbytes and the ratio of block size versus fragment size must be 1, 2, 4, or 8.

The disk geometry information used by the filesystem affects the block layout policies employed. The file `/etc/disktab`, as supplied, contains the data for most all drives supported by the system. Before constructing a filesystem with *newfs*(8) you should label the disk (if it has not yet been labeled, and the driver supports labels). If labels cannot be used, you must instead specify the type of disk on which the filesystem resides; *newfs* then reads `/etc/disktab` instead of the pack label. This file also contains the default filesystem partition sizes, and default block and fragment sizes. To override any of the default values you can modify the file, edit the disk label, or use an option to *newfs*.

2.5.4. Implementing a layout

To put a chosen disk layout into effect, you should use the *newfs*(8) command to create each new filesystem. Each filesystem must also be added to the file `/etc/fstab` so that it will be checked and mounted when the system is bootstrapped.

First we will consider a system with a single disk. There is little real choice on how to do the layout; the root filesystem goes in the ‘‘a’’ partition, `/usr` goes in the ‘‘e’’ partition, and `/var` fills out the remainder of the disk in the ‘‘f’’ partition. This is the organization used if you loaded the disk-image root filesystem. With the addition of a memory-based `/tmp` filesystem, its fstab entry would be as follows:

/dev/sd0a	/	ufs	rw	1	1
/dev/sd0b	none	swap	sw	0	0
/dev/sd0b	/tmp	mfs	rw,-s=14000,-b=8192,-f=1024,-T=sd660	0	0
/dev/sd0e	/usr	ufs	ro	1	2
/dev/sd0f	/var	ufs	rw	1	2

If we had a second disk, we would split the load between the drives. On the second disk, we place the `/usr` and `/var` filesystems in their usual sd1e and sd1f partitions respectively. The sd1b partition would be used as a second paging area, and the sd1a partition left as a spare root filesystem (alternatively sd1a could be used for `/var/tmp`). The first disk still holds the the root filesystem in sd0a, and the primary swap area in sd0b. The sd0e partition is used to hold home directories in `/var/users`. The sd0f partition can be used for `/usr/src` or alternately the sd0e partition can be extended to cover the rest of the disk with *disklabel*(8). As before, the `/tmp` directory is a memory-based filesystem. Note that to interleave the paging between the two disks you must build a system configuration that specifies:

config vmunix root on sd0 swap on sd0 and sd1

The `/etc/fstab` file would then contain

/dev/sd0a	/	ufs	rw	1	1
/dev/sd0b	none	swap	sw	0	0
/dev/sd1b	none	swap	sw	0	0
/dev/sd0b	/tmp	mfs	rw,-s=14000,-b=8192,-f=1024,-T=sd660	0	0
/dev/sd1e	/usr	ufs	ro	1	2
/dev/sd0f	/usr/src	ufs	rw	1	2
/dev/sd1f	/var	ufs	rw	1	2
/dev/sd0e	/var/users	ufs	rw	1	2

To make the `/var` filesystem we would do:

cd /dev
MAKEDEV sd1
disklabel -wr sd1 "disk type" "disk name"
newfs sd1f
(information about filesystem prints out)
mkdir /var
mount /dev/sd1f /var

2.6. Installing the rest of the system

At this point you should have your disks partitioned. The next step is to extract the rest of the data from the tape. At a minimum you need to set up the `/var` and `/usr` filesystems. You may also want to extract some or all the program sources. Since not all architectures support tape drives or don't support the correct ones, you may need to extract the files indirectly using *rsh*(1). For example, for a directly connected tape drive you might do:

mt -f /dev/nrmt0 fsf
tar xbpf 20 /dev/nrmt0

The equivalent indirect procedure (where the tape drive is on machine ‘‘foo’’) is:

rsh foo mt -f /dev/nrmt0 fsf
rsh foo dd if=/dev/nrmt0 bs=20b | tar xbpf 20 -

Obviously, the target machine must be connected to the local network for this to work. To do this:

echo 127.0.0.1 localhost >> /etc/hosts
echo your.host.inet.number myname.my.domain myname >> */etc/hosts*
echo friend.host.inet.number myfriend.my.domain myfriend >> */etc/hosts*
ifconfig le0 inet myname

where the ‘‘host.inet.number’’ fields are the IP addresses for your host and the host with the tape drive and the ‘‘my.domain’’ fields are the names of your machine and the tape-hosting machine. See sections 4.4 and 5 for more information on setting up the network.

Assuming a directly connected tape drive, here is how to extract and install `/var` and `/usr`:

# *mount –uw /dev/sd#a /*	(read-write mount root filesystem)
# *date yymmddhhmm*	(set date, see *date*(1))
....	
# *passwd -l root*	(set password for super-user)
New password:	(password will not echo)
Retype new password:	
# *passwd -l toor*	(set password for super-user)
New password:	(password will not echo)
Retype new password:	
# *hostname mysitename*	(set your hostname)
# *newfs rsd#p*	(create empty user filesystem)
(*sd* is the disk type, *#* is the unit number, *p* is the partition; this takes a few minutes)	
# *mount /dev/sd#p /var*	(mount the var filesystem)
# *cd /var*	(make /var the current directory)
# *mt -f /dev/nrmt0 fsf*	(space to end of previous tape file)
# *tar xbpf 20 /dev/nrmt0*	(extract all of var)
(this takes a few minutes)	
# *newfs rsd#p*	(create empty user filesystem)
(as before *sd* is the disk type, *#* is the unit number, *p* is the partition)	

# *mount /dev/sd#p /mnt*	(mount the new /usr in temporary location)
# *cd /mnt*	(make /mnt the current directory)
# *mt -f /dev/nrmt0 fsf*	(space to end of previous tape file)
# *tar xbpf 20 /dev/nrmt0*	(extract all of usr except usr/src)
(this takes about 15-20 minutes)	
# *cd /*	(make / the current directory)
# *umount /mnt*	(unmount from temporary mount point)
# *rm -r /usr/**	(remove excess bootstrap binaries)
# *mount /dev/sd#p /usr*	(remount /usr)

If no disk label has been installed on the disk, the *newfs* command will require a third argument to specify the disk type, using one of the names in `/etc/disktab`. If the tape had been rewound or positioned incorrectly before the *tar*, to extract `/var` it may be repositioned by the following commands.

mt -f /dev/nrmt0 rew
mt -f /dev/nrmt0 fsf 1

The data on the second and third tape files has now been extracted. If you are using 6250bpi tapes, the first reel of the distribution is no longer needed; you should now mount the second reel instead. The installation procedure continues from this point on the 8mm tape. The next step is to extract the sources. As previously noted, `/usr/src` requires about 250-340Mb of space. Ideally sources should be in a separate filesystem; if you plan to put them into your `/usr` filesystem, it will need at least 500Mb of space. Assuming that you will be using a separate filesystem on sd0f for `/usr/src`, you will start by creating and mounting it:

newfs sd0f
(information about filesystem prints out)
mkdir /usr/src
mount /dev/sd0f /usr/src

First you will extract the kernel source:

# *cd /usr/src*	
# *mt -f /dev/nrmt0 fsf*	(space to end of previous tape file)
(this should only be done on Exabyte distributions)	
# *tar xpbf 20 /dev/nrmt0*	(extract the kernel sources)
(this takes about 15-30 minutes)	

The next tar file contains the sources for the utilities. It is extracted as follows:

# *cd /usr/src*	
# *mt -f /dev/nrmt0 fsf*	(space to end of previous tape file)
# *tar xpbf 20 /dev/rmt12*	(extract the utility source)
(this takes about 30-60 minutes)	

If you are using 6250bpi tapes, the second reel of the distribution is no longer needed; you should now mount the third reel instead. The installation procedure continues from this point on the 8mm tape.

The next tar file contains the sources for the contributed software. It is extracted as follows:

# *cd /usr/src*	
# *mt -f /dev/nrmt0 fsf*	(space to end of previous tape file)
(this should only be done on Exabyte distributions)	
# *tar xpbf 20 /dev/rmt12*	(extract the contributed software source)
(this takes about 30-60 minutes)	

If you received a distribution on 8mm Exabyte tape, there is one additional tape file on the distribution tape that has not been installed to this point; it contains the sources for X11R5 in *tar*(1) format. As distributed, X11R5 should be placed in `/usr/src/X11R5`.

# *cd /usr/src*	
# *mt -f /dev/nrmt0 fsf*	(space to end of previous tape file)
# *tar xpbf 20 /dev/nrmt0*	(extract the X11R5 source)
(this takes about 30-60 minutes)	

Many of the X11 utilities search using the path `/usr/X11`, so be sure that you have a symbolic link that points at the location of your X11 binaries (here, X11R5).

Having now completed the extraction of the sources, you may want to verify that your `/usr/src` filesystem is consistent. To do so, you must unmount it, and run *fsck*(8); assuming that you used sd0f you would proceed as follows:

# *cd /*	(change directory, back to the root)
# *umount /usr/src*	(unmount /usr/src)
# *fsck /dev/rsd0f*	

The output from *fsck* should look something like:

**** /dev/rsd0f**
**** Last Mounted on /usr/src**
**** Phase 1 - Check Blocks and Sizes**
**** Phase 2 - Check Pathnames**
**** Phase 3 - Check Connectivity**
**** Phase 4 - Check Reference Counts**
**** Phase 5 - Check Cyl groups**
23000 files, 261000 used, 39000 free (2200 frags, 4600 blocks)

If there are inconsistencies in the filesystem, you may be prompted to apply corrective action; see the *fsck*(8) or *Fsck – The UNIX File System Check Program* (SMM:3) for more details.

To use the `/usr/src` filesystem, you should now remount it with:

mount /dev/sd0f /usr/src

or if you have made an entry for it in `/etc/fstab` you can remount it with:

mount /usr/src

2.7. Additional conversion information

After setting up the new 4.4BSD filesystems, you may restore the user files that were saved on tape before beginning the conversion. Note that the 4.4BSD *restore* program does its work on a mounted filesystem using normal system operations. This means that filesystem dumps may be restored even if the characteristics of the filesystem changed. To restore a dump tape for, say, the `/a` filesystem something like the following would be used:

mkdir /a
newfs sd#p
mount /dev/sd#p /a
cd /a
restore x

If *tar* images were written instead of doing a dump, you should be sure to use its '−p' option when reading the files back. No matter how you restore a filesystem, be sure to unmount it and and check its integrity with *fsck*(8) when the job is complete.

3. Upgrading a 4.3BSD system

This section describes the procedure for upgrading a 4.3BSD system to 4.4BSD. This procedure may vary according to the version of the system running before conversion. If you are converting from a System V system, some of this section will still apply (in particular, the filesystem conversion). However, many of the system configuration files are different, and the executable file formats are completely incompatible.

In particular be wary when using this information to upgrade a 4.3BSD HP300 system. There are at least four different versions of ''4.3BSD'' out there:

1) HPBSD 1.x from Utah.
This was the original version of 4.3BSD for HP300s from which the other variants (and 4.4BSD) are derived. It is largely a 4.3BSD system with Sun's NFS 3.0 filesystem code and some 4.3BSD-Tahoe features (e.g. networking code). Since the filesystem code is 4.2/4.3 vintage and the filesystem hierarchy is largely 4.3BSD, most of this section should apply.

2) MORE/bsd from Mt. Xinu.
This is a 4.3BSD-Tahoe vintage system with Sun's NFS 4.0 filesystem code upgraded with Tahoe UFS features. The instructions for 4.3BSD-Tahoe should largely apply.

3) 4.3BSD-Reno from CSRG.
At least one site bootstrapped HP300 support from the Reno distribution. The Reno filesystem code was somewhere between 4.3BSD and 4.4BSD: the VFS switch had been added but many of the UFS features (e.g. ''inline'' symlinks) were missing. The filesystem hierarchy reorganization first appeared in this release. Be extremely careful following these instructions if you are upgrading from the Reno distribution.

4) HPBSD 2.0 from Utah.
As if things were not bad enough already, this release has the 4.4BSD filesystem and networking code as well as some utilities, but still has a 4.3BSD hierarchy. No filesystem conversions are necessary for this upgrade, but files will still need to be moved around.

3.1. Installation overview

If you are running 4.3BSD, upgrading your system involves replacing your kernel and system utilities. In general, there are three possible ways to install a new BSD distribution: (1) boot directly from the distribution tape, use it to load new binaries onto empty disks, and then merge or restore any existing configuration files and filesystems; (2) use an existing 4.3BSD or later system to extract the root and `/usr` filesystems from the distribution tape, boot from the new system, then merge or restore existing configuration files and filesystems; or (3) extract the sources from the distribution tape onto an existing system, and use that system to cross-compile and install 4.4BSD. For this release, the second alternative is strongly advised, with the third alternative reserved as a last resort. In general, older binaries will continue to run under 4.4BSD, but there are many exceptions that are on the critical path for getting the system running. Ideally, the new system binaries (root and `/usr` filesystems) should be installed on spare disk partitions, then site-specific files should be merged into them. Once the new system is up and fully merged, the previous root and `/usr` filesystems can be reused. Other existing filesystems can be retained and used, except that (as usual) the new *fsck* should be run before they are mounted.

It is **STRONGLY** advised that you make full dumps of each filesystem before beginning, especially any that you intend to modify in place during the merge. It is also desirable to run filesystem checks of all filesystems to be converted to 4.4BSD before shutting down. This is an excellent time to review your disk configuration for possible tuning of the layout. Most systems will need to provide a new filesystem for system use mounted on `/var` (see below). However, the `/tmp` filesystem can be an MFS virtual-memory-resident filesystem, potentially freeing an existing disk partition. (Additional swap space may be desirable as a consequence.) See *mount_mfs*(8).

The recommended installation procedure includes the following steps. The order of these steps will probably vary according to local needs.

- Extract root and `/usr` filesystems from the distribution tapes.
- Extract kernel and/or user-level sources from the distribution tape if space permits. This can serve as the backup documentation as needed.
- Configure and boot a kernel for the local system. This can be delayed if the generic kernel from the distribution supports enough hardware to proceed.
- Build a skeletal `/var` filesystem (see *mtree*(8)).
- Merge site-dependent configuration files from `/etc` and `/usr/lib` into the new `/etc` directory. Note that many file formats and contents have changed; see section 3.4 of this document.
- Copy or merge files from `/usr/adm`, `/usr/spool`, `/usr/preserve`, `/usr/lib`, and other locations into `/var`.
- Merge local macros, dictionaries, etc. into `/usr/share`.
- Merge and update local software to reflect the system changes.
- Take off the rest of the morning, you've earned it!

Section 3.2 lists the files to be saved as part of the conversion process. Section 3.3 describes the bootstrap process. Section 3.4 discusses the merger of the saved files back into the new system. Section 3.5 gives an overview of the major bug fixes and changes between 4.3BSD and 4.4BSD. Section 3.6 provides general hints on possible problems to be aware of when converting from 4.3BSD to 4.4BSD.

3.2. Files to save

The following list enumerates the standard set of files you will want to save and suggests directories in which site-specific files should be present. This list will likely be augmented with non-standard files you have added to your system. If you do not have enough space to create parallel filesystems, you should create a *tar* image of the following files before the new filesystems are created. The rest of this subsection describes where theses files have moved and how they have changed.

`/.cshrc`	†	root csh startup script (moves to `/root/.cshrc`)
`/.login`	†	root csh login script (moves to `/root/.login`)
`/.profile`	†	root sh startup script (moves to `/root/.profile`)
`/.rhosts`	†	for trusted machines and users (moves to `/root/.rhosts`)
`/etc/disktab`	‡	in case you changed disk partition sizes
`/etc/fstab`	*	disk configuration data
`/etc/ftpusers`	†	for local additions
`/etc/gettytab`	‡	getty database
`/etc/group`	*	group data base
`/etc/hosts`	†	for local host information
`/etc/hosts.equiv`	†	for local host equivalence information
`/etc/hosts.lpd`	†	printer access file
`/etc/inetd.conf`	*	Internet services configuration data
`/etc/named*`	†	named configuration files
`/etc/netstart`	†	network initialization
`/etc/networks`	†	for local network information
`/etc/passwd`	*	user data base
`/etc/printcap`	*	line printer database
`/etc/protocols`	‡	in case you added any local protocols
`/etc/rc`	*	for any local additions
`/etc/rc.local`	*	site specific system startup commands
`/etc/remote`	†	auto-dialer configuration
`/etc/services`	‡	for local additions
`/etc/shells`	‡	list of valid shells
`/etc/syslog.conf`	*	system logger configuration
`/etc/securettys`	*	merged into ttys
`/etc/ttys`	*	terminal line configuration data

/etc/ttytype	*	merged into ttys
/etc/termcap	‡	for any local entries that may have been added
/lib	‡	for any locally developed language processors
/usr/dict/*	‡	for local additions to words and papers
/usr/include/*	‡	for local additions
/usr/lib/aliases	*	mail forwarding data base (moves to /etc/aliases)
/usr/lib/crontab	*	cron daemon data base (moves to /etc/crontab)
/usr/lib/crontab.local	*	local cron daemon data base (moves to /etc/crontab.local)
/usr/lib/lib*.a	†	for local libraries
/usr/lib/mail.rc	†	system-wide mail(1) initialization (moves to /etc/mail.rc)
/usr/lib/sendmail.cf	*	sendmail configuration (moves to /etc/sendmail.cf)
/usr/lib/tmac/*	‡	for locally developed troff/nroff macros (moves to /usr/share/tmac/*)
/usr/lib/uucp/*	†	for local uucp configuration files
/usr/man/manl	*	for manual pages for locally developed programs (moves to /usr/local/man)
/usr/spool/*	†	for current mail, news, uucp files, etc. (moves to /var/spool)
/usr/src/local	†	for source for locally developed programs
/sys/conf/HOST	†	configuration file for your machine (moves to /sys/<arch>/conf)
/sys/conf/files.HOST	†	list of special files in your kernel (moves to /sys/<arch>/conf)
/*/quotas	*	filesystem quota files (moves to /*/quotas.user)

† Files that can be used from 4.3BSD without change.
‡ Files that need local changes merged into 4.4BSD files.
* Files that require special work to merge and are discussed in section 3.4.

3.3. Installing 4.4BSD

The next step is to build a working 4.4BSD system. This can be done by following the steps in section 2 of this document for extracting the root and /usr filesystems from the distribution tape onto unused disk partitions. For the SPARC, the root filesystem dump on the tape could also be extracted directly. For the HP300 and DECstation, the raw disk image can be copied into an unused partition and this partition can then be dumped to create an image that can be restored. The exact procedure chosen will depend on the disk configuration and the number of suitable disk partitions that may be used. It is also desirable to run filesystem checks of all filesystems to be converted to 4.4BSD before shutting down. In any case, this is an excellent time to review your disk configuration for possible tuning of the layout. Section 2.5 and *config*(8) are required reading.

The filesystem in 4.4BSD has been reorganized in an effort to meet several goals:

1) The root filesystem should be small.
2) There should be a per-architecture centrally-shareable read-only /usr filesystem.
3) Variable per-machine directories should be concentrated below a single mount point named /var.
4) Site-wide machine independent shareable text files should be separated from architecture specific binary files and should be concentrated below a single mount point named /usr/share.

These goals are realized with the following general layouts. The reorganized root filesystem has the following directories:

/etc	(config files)
/bin	(user binaries needed when single-user)
/sbin	(root binaries needed when single-user)
/local	(locally added binaries used only by this machine)
/tmp	(mount point for memory based filesystem)
/dev	(local devices)
/home	(mount point for AMD)
/var	(mount point for per-machine variable directories)
/usr	(mount point for multiuser binaries and files)

The reorganized `/usr` filesystem has the following directories:

`/usr/bin`	(user binaries)
`/usr/contrib`	(software contributed to 4.4BSD)
`/usr/games`	(binaries for games, score files in `/var`)
`/usr/include`	(standard include files)
`/usr/lib`	(lib*.a from old `/usr/lib`)
`/usr/libdata`	(databases from old `/usr/lib`)
`/usr/libexec`	(executables from old `/usr/lib`)
`/usr/local`	(locally added binaries used site-wide)
`/usr/old`	(deprecated binaries)
`/usr/sbin`	(root binaries)
`/usr/share`	(mount point for site-wide shared text)
`/usr/src`	(mount point for sources)

The reorganized `/usr/share` filesystem has the following directories:

`/usr/share/calendar`	(various useful calendar files)
`/usr/share/dict`	(dictionaries)
`/usr/share/doc`	(4.4BSD manual sources)
`/usr/share/games`	(games text files)
`/usr/share/groff_font`	(groff font information)
`/usr/share/man`	(typeset manual pages)
`/usr/share/misc`	(dumping ground for random text files)
`/usr/share/mk`	(templates for 4.4BSD makefiles)
`/usr/share/skel`	(template user home directory files)
`/usr/share/tmac`	(various groff macro packages)
`/usr/share/zoneinfo`	(information on time zones)

The reorganized `/var` filesystem has the following directories:

`/var/account`	(accounting files, formerly `/usr/adm`)
`/var/at`	(*at*(1) spooling area)
`/var/backups`	(backups of system files)
`/var/crash`	(crash dumps)
`/var/db`	(system-wide databases, e.g. tags)
`/var/games`	(score files)
`/var/log`	(log files)
`/var/mail`	(users mail)
`/var/obj`	(hierarchy to build `/usr/src`)
`/var/preserve`	(preserve area for vi)
`/var/quotas`	(directory to store quota files)
`/var/run`	(directory to store *.pid files)
`/var/rwho`	(rwho databases)
`/var/spool/ftp`	(home directory for anonymous ftp)
`/var/spool/mqueue`	(sendmail spooling directory)
`/var/spool/news`	(news spooling area)
`/var/spool/output`	(printer spooling area)
`/var/spool/uucp`	(uucp spooling area)
`/var/tmp`	(disk-based temporary directory)
`/var/users`	(root of per-machine user home directories)

The 4.4BSD bootstrap routines pass the identity of the boot device through to the kernel. The kernel then uses that device as its root filesystem. Thus, for example, if you boot from `/dev/sd1a`, the kernel will use `sd1a` as its root filesystem. If `/dev/sd1b` is configured as a swap partition, it will be used as the initial swap area, otherwise the normal primary swap area (`/dev/sd0b`) will be used. The 4.4BSD bootstrap is backward compatible with 4.3BSD, so you can replace your old bootstrap if you use it to boot

your first 4.4BSD kernel. However, the 4.3BSD bootstrap cannot access 4.4BSD filesystems, so if you plan to convert your filesystems to 4.4BSD, you must install a new bootstrap *before* doing the conversion. Note that SPARC users cannot build a 4.4BSD compatible version of the bootstrap, so must *not* convert their root filesystem to the new 4.4BSD format.

Once you have extracted the 4.4BSD system and booted from it, you will have to build a kernel customized for your configuration. If you have any local device drivers, they will have to be incorporated into the new kernel. See section 4.1.3 and ‘‘Building 4.3BSD UNIX Systems with Config’’ (SMM:2).

If converting from 4.3BSD, your old filesystems should be converted. If you’ve modified the partition sizes from the original 4.3BSD ones, and are not already using the 4.4BSD disk labels, you will have to modify the default disk partition tables in the kernel. Make the necessary table changes and boot your custom kernel **BEFORE** trying to access any of your old filesystems! After doing this, if necessary, the remaining filesystems may be converted in place by running the 4.4BSD version of *fsck*(8) on each filesystem and allowing it to make the necessary corrections. The new version of *fsck* is more strict about the size of directories than the version supplied with 4.3BSD. Thus the first time that it is run on a 4.3BSD filesystem, it will produce messages of the form:

DIRECTORY ...: LENGTH xx **NOT MULTIPLE OF 512 (ADJUSTED)**

Length ‘‘xx’’ will be the size of the directory; it will be expanded to the next multiple of 512 bytes. The new *fsck* will also set default *interleave* and *npsect* (number of physical sectors per track) values on older filesystems, in which these fields were unused spares; this correction will produce messages of the form:

IMPOSSIBLE INTERLEAVE=0 IN SUPERBLOCK (SET TO DEFAULT)[3]
IMPOSSIBLE NPSECT=0 IN SUPERBLOCK (SET TO DEFAULT)

Filesystems that have had their interleave and npsect values set will be diagnosed by the old *fsck* as having a bad superblock; the old *fsck* will run only if given an alternate superblock (*fsck −b32*), in which case it will re-zero these fields. The 4.4BSD kernel will internally set these fields to their defaults if fsck has not done so; again, the *−b32* option may be necessary for running the old *fsck*.

In addition, 4.4BSD removes several limits on filesystem sizes that were present in 4.3BSD. The limited filesystems continue to work in 4.4BSD, but should be converted as soon as it is convenient by running *fsck* with the *−c 2* option. The sequence *fsck −p −c 2* will update them all, fix the interleave and npsect fields, fix any incorrect directory lengths, expand maximum uid’s and gid’s to 32-bits, place symbolic links less than 60 bytes into their inode, and fill in directory type fields all at once. The new filesystem formats are incompatible with older systems. If you wish to continue using these filesystems with the older systems you should make only the compatible changes using *fsck −c 1*.

3.4. Merging your files from 4.3BSD into 4.4BSD

When your system is booting reliably and you have the 4.4BSD root and `/usr` filesystems fully installed you will be ready to continue with the next step in the conversion process, merging your old files into the new system.

If you saved the files on a *tar* tape, extract them into a scratch directory, say `/usr/convert`:

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

The data files marked in the previous table with a dagger (†) may be used without change from the previous system. Those data files marked with a double dagger (‡) have syntax changes or substantial enhancements. You should start with the 4.4BSD version and carefully integrate any local changes into the new file. Usually these local changes can be incorporated without conflict into the new file; some exceptions are noted below. The files marked with an asterisk (*) require particular attention and are discussed

[3] The defaults are to set *interleave* to 1 and *npsect* to *nsect*. This is correct on most drives; it affects only performance (usually virtually unmeasurably).

below.

As described in section 3.3, the most immediately obvious change in 4.4BSD is the reorganization of the system filesystems. Users of certain recent vendor releases have seen this general organization, although 4.4BSD takes the reorganization a bit further. The directories most affected are `/etc`, that now contains only system configuration files; `/var`, a new filesystem containing per-system spool and log files; and `/usr/share,` that contains most of the text files shareable across architectures such as documentation and macros. System administration programs formerly in `/etc` are now found in `/sbin` and `/usr/sbin`. Various programs and data files formerly in `/usr/lib` are now found in `/usr/libexec` and `/usr/libdata`, respectively. Administrative files formerly in `/usr/adm` are in `/var/account` and, similarly, log files are now in `/var/log`. The directory `/usr/ucb` has been merged into `/usr/bin`, and the sources for programs in `/usr/bin` are in `/usr/src/usr.bin`. Other source directories parallel the destination directories; `/usr/src/etc` has been greatly expanded, and `/usr/src/share` is new. The source for the manual pages, in general, are with the source code for the applications they document. Manual pages not closely corresponding to an application program are found in `/usr/src/share/man`. The locations of all man pages is listed in `/usr/src/share/man/man0/man[1-8]`. The manual page *hier*(7) has been updated and made more detailed; it is included in the printed documentation. You should review it to familiarize yourself with the new layout.

A new utility, *mtree*(8), is provided to build and check filesystem hierarchies with the proper contents, owners and permissions. Scripts are provided in `/etc/mtree` (and `/usr/src/etc/mtree`) for the root, `/usr` and `/var` filesystems. Once a filesystem has been made for `/var`, *mtree* can be used to create a directory hierarchy there or you can simply use tar to extract the prototype from the second file of the distribution tape.

3.4.1. Changes in the `/etc` directory

The `/etc` directory now contains nearly all the host-specific configuration files. Note that some file formats have changed, and those configuration files containing pathnames are nearly all affected by the reorganization. See the examples provided in `/etc` (installed from `/usr/src/etc`) as a guide. The following table lists some of the local configuration files whose locations and/or contents have changed.

4.3BSD and Earlier	4.4BSD	Comments
`/etc/fstab`	`/etc/fstab`	new format; see below
`/etc/inetd.conf`	`/etc/inetd.conf`	pathnames of executables changed
`/etc/printcap`	`/etc/printcap`	pathnames changed
`/etc/syslog.conf`	`/etc/syslog.conf`	pathnames of log files changed
`/etc/ttys`	`/etc/ttys`	pathnames of executables changed
`/etc/passwd`	`/etc/master.passwd`	new format; see below
`/usr/lib/sendmail.cf`	`/etc/sendmail.cf`	changed pathnames
`/usr/lib/aliases`	`/etc/aliases`	may contain changed pathnames
`/etc/*.pid`	`/var/run/*.pid`	

New in 4.3BSD-Tahoe	4.4BSD	Comments
`/usr/games/dm.config`	`/etc/dm.conf`	configuration for games (see *dm*(8))
`/etc/zoneinfo/localtime`	`/etc/localtime`	timezone configuration
`/etc/zoneinfo`	`/usr/share/zoneinfo`	timezone configuration

New in 4.4BSD	Comments
`/etc/aliases.db`	database version of the aliases file
`/etc/amd-home`	location database of home directories
`/etc/amd-vol`	location database of exported filesystems
`/etc/changelist`	`/etc/security` files to back up
`/etc/csh.cshrc`	system-wide csh(1) initialization file
`/etc/csh.login`	system-wide csh(1) login file
`/etc/csh.logout`	system-wide csh(1) logout file
`/etc/disklabels`	directory for saving disklabels
`/etc/exports`	NFS list of export permissions
`/etc/ftpwelcome`	message displayed for ftp users; see ftpd(8)
`/etc/kerberosIV`	Kerberos directory; see below
`/etc/man.conf`	lists directories searched by *man* (1)
`/etc/mtree`	directory for local mtree files; see mtree(8)
`/etc/netgroup`	NFS group list used in `/etc/exports`
`/etc/pwd.db`	non-secure hashed user data base file
`/etc/spwd.db`	secure hashed user data base file
`/etc/security`	daily system security checker

System security changes require adding several new ‘‘well-known’’ groups to `/etc/group`. The groups that are needed by the system as distributed are:

name	number	purpose
wheel	0	users allowed superuser privilege
daemon	1	processes that need less than wheel privilege
kmem	2	read access to kernel memory
sys	3	access to kernel sources
tty	4	access to terminals
operator	5	read access to raw disks
bin	7	group for system binaries
news	8	group for news
wsrc	9	write access to sources
games	13	access to games
staff	20	system staff
guest	31	system guests
nobody	39	the least privileged group
utmp	45	access to utmp files
dialer	117	access to remote ports and dialers

Only users in the ‘‘wheel’’ group are permitted to *su* to ‘‘root’’. Most programs that manage directories in `/var/spool` now run set-group-id to ‘‘daemon’’ so that users cannot directly access the files in the spool directories. The special files that access kernel memory, `/dev/kmem` and `/dev/mem`, are made readable only by group ‘‘kmem’’. Standard system programs that require this access are made set-group-id to that group. The group ‘‘sys’’ is intended to control access to kernel sources, and other sources belong to group ‘‘wsrc.’’ Rather than make user terminals writable by all users, they are now placed in group ‘‘tty’’ and made only group writable. Programs that should legitimately have access to write on user terminals such as *talkd* and *write* now run set-group-id to ‘‘tty’’. The ‘‘operator’’ group controls access to disks. By default, disks are readable by group ‘‘operator’’,

so that programs such as *dump* can access the filesystem information without being set-user-id to ‘‘root’’. The *shutdown*(8) program is executable only by group operator and is setuid to root so that members of group operator may shut down the system without root access.

The ownership and modes of some directories have changed. The *at* programs now run set-user-id ‘‘root’’ instead of ‘‘daemon.’’ Also, the uucp directory no longer needs to be publicly writable, as *tip* reverts to privileged status to remove its lock files. After copying your version of `/var/spool`, you should do:

> *# chown −R root /var/spool/at*
> *# chown −R uucp.daemon /var/spool/uucp*
> *# chmod −R o−w /var/spool/uucp*

The format of the cron table, `/etc/crontab`, has been changed to specify the user-id that should be used to run a process. The userid ‘‘nobody’’ is frequently useful for non-privileged programs. Local changes are now put in a separate file, `/etc/crontab.local`.

Some of the commands previously in `/etc/rc.local` have been moved to `/etc/rc`; several new functions are now handled by `/etc/rc`, `/etc/netstart` and `/etc/rc.local`. You should look closely at the prototype version of these files and read the manual pages for the commands contained in it before trying to merge your local copy. Note in particular that *ifconfig* has had many changes, and that host names are now fully specified as domain-style names (e.g., vangogh.CS.Berkeley.EDU) for the benefit of the name server.

Some of the commands previously in `/etc/daily` have been moved to `/etc/security`, and several new functions have been added to `/etc/security` to do nightly security checks on the system. The script `/etc/daily` runs `/etc/security` each night, and mails the output to the super-user. Some of the checks done by `/etc/security` are:

- Syntax errors in the password and group files.
- Duplicate user and group names and id’s.
- Dangerous search paths and umask values for the superuser.
- Dangerous values in various initialization files.
- Dangerous .rhosts files.
- Dangerous directory and file ownership or permissions.
- Globally exported filesystems.
- Dangerous owners or permissions for special devices.

In addition, it reports any changes to setuid and setgid files, special devices, or the files in `/etc/changelist` since the last run of `/etc/security`. Backup copies of the files are saved in `/var/backups`. Finally, the system binaries are checksummed and their permissions validated against the *mtree*(8) specifications in `/etc/mtree`.

The C-library and system binaries on the distribution tape are compiled with new versions of *gethostbyname* and *gethostbyaddr* that use the name server, *named*(8). If you have only a small network and are not connected to a large network, you can use the distributed library routines without any problems; they use a linear scan of the host table `/etc/hosts` if the name server is not running. If you are on the Internet or have a large local network, it is recommend that you set up and use the name server. For instructions on how to set up the necessary configuration files, refer to ‘‘Name Server Operations Guide for BIND’’ (SMM:10). Several programs rely on the host name returned by *gethostname* to determine the local domain name.

If you are using the name server, your *sendmail* configuration file will need some updates to accommodate it. See the ‘‘Sendmail Installation and Operation Guide’’ (SMM:8) and the sample *sendmail* configuration files in `/usr/src/usr.sbin/sendmail/cf`. The aliases file, `/etc/aliases` has also been changed to add certain well-known addresses.

3.4.2. Shadow password files

The password file format adds change and expiration fields and its location has changed to protect the encrypted passwords stored there. The actual password file is now stored in `/etc/master.passwd`. The hashed dbm password files do not contain encrypted passwords, but contain the file offset to the entry with the password in `/etc/master.passwd` (that is readable only by root). Thus, the *getpwnam*() and *getpwuid*() functions will no longer return an encrypted password string to non-root callers. An old-style passwd file is created in `/etc/passwd` by the *vipw*(8) and *pwd_mkdb*(8) programs. See also *passwd*(5).

Several new users have also been added to the group of ‘‘well-known’’ users in `/etc/passwd`. The current list is:

name	number
root	0
daemon	1
operator	2
bin	3
games	7
uucp	66
nobody	32767

The ‘‘daemon’’ user is used for daemon processes that do not need root privileges. The ‘‘operator’’ user-id is used as an account for dumpers so that they can log in without having the root password. By placing them in the ‘‘operator’’ group, they can get read access to the disks. The ‘‘uucp’’ login has existed long before 4.4BSD, and is noted here just to provide a common user-id. The password entry ‘‘nobody’’ has been added to specify the user with least privilege. The ‘‘games’’ user is a pseudo-user that controls access to game programs.

After installing your updated password file, you must run *pwd_mkdb*(8) to create the password database. Note that *pwd_mkdb*(8) is run whenever *vipw*(8) is run.

3.4.3. The `/var` filesystem

The spooling directories saved on tape may be restored in their eventual resting places without too much concern. Be sure to use the ‘−p’ option to *tar*(1) so that files are recreated with the same file modes. The following commands provide a guide for copying spool and log files from an existing system into a new `/var` filesystem. At least the following directories should already exist on `/var`: `output`, `log`, `backups` and `db`.

```
SRC=/oldroot/usr

cd $SRC; tar cf - msgs preserve | (cd /var && tar xpf -)

# copy $SRC/spool to /var
cd $SRC/spool
tar cf - at mail rwho | (cd /var && tar xpf -)
tar cf - ftp mqueue news secretmail uucp uucppublic | \
    (cd /var/spool && tar xpf -)

# everything else in spool is probably a printer area
mkdir .save
mv at ftp mail mqueue rwho secretmail uucp uucppublic .save
tar cf - * | (cd /var/spool/output && tar xpf -)
mv .save/* .
rmdir .save
```

```
cd /var/spool/mqueue
mv syslog.7 /var/log/maillog.7
mv syslog.6 /var/log/maillog.6
mv syslog.5 /var/log/maillog.5
mv syslog.4 /var/log/maillog.4
mv syslog.3 /var/log/maillog.3
mv syslog.2 /var/log/maillog.2
mv syslog.1 /var/log/maillog.1
mv syslog.0 /var/log/maillog.0
mv syslog /var/log/maillog

# move $SRC/adm to /var
cd $SRC/adm
tar cf - . | (cd /var/account && tar  xpf -)
cd /var/account
rm -f msgbuf
mv messages messages.[0-9] ../log
mv wtmp wtmp.[0-9] ../log
mv lastlog ../log
```

3.5. Bug fixes and changes between 4.3BSD and 4.4BSD

The major new facilities available in the 4.4BSD release are a new virtual memory system, the addition of ISO/OSI networking support, a new virtual filesystem interface supporting filesystem stacking, a freely redistributable implementation of NFS, a log-structured filesystem, enhancement of the local filesystems to support files and filesystems that are up to 2ˆ63 bytes in size, enhanced security and system management support, and the conversion to and addition of the IEEE Std1003.1 (‘‘POSIX’’) facilities and many of the IEEE Std1003.2 facilities. In addition, many new utilities and additions to the C library are present as well. The kernel sources have been reorganized to collect all machine-dependent files for each architecture under one directory, and most of the machine-independent code is now free of code conditional on specific machines. The user structure and process structure have been reorganized to eliminate the statically-mapped user structure and to make most of the process resources shareable by multiple processes. The system and include files have been converted to be compatible with ANSI C, including function prototypes for most of the exported functions. There are numerous other changes throughout the system.

3.5.1. Changes to the kernel

This release includes several important structural kernel changes. The kernel uses a new internal system call convention; the use of global (‘‘u-dot’’) variables for parameters and error returns has been eliminated, and interrupted system calls no longer abort using non-local goto’s (longjmp’s). A new sleep interface separates signal handling from scheduling priority, returning characteristic errors to abort or restart the current system call. This sleep call also passes a string describing the process state, that is used by the ps(1) program. The old sleep interface can be used only for non-interruptible sleeps. The sleep interface (*tsleep*) can be used at any priority, but is only interruptible if the PCATCH flag is set. When interrupted, *tsleep* returns EINTR or ERESTART.

Many data structures that were previously statically allocated are now allocated dynamically. These structures include mount entries, file entries, user open file descriptors, the process entries, the vnode table, the name cache, and the quota structures.

To protect against indiscriminate reading or writing of kernel memory, all writing and most reading of kernel data structures must be done using a new ‘‘sysctl’’ interface. The information to be accessed is described through an extensible ‘‘Management Information Base’’ (MIB) style name, described as a dotted set of components. A new utility, *sysctl*(8), retrieves kernel state and allows processes with appropriate privilege to set kernel state.

3.5.2. Security

The kernel runs with four different levels of security. Any superuser process can raise the security level, but only *init*()(8) can lower it. Security levels are defined as follows:

−1 Permanently insecure mode − always run system in level 0 mode.

0 Insecure mode − immutable and append-only flags may be turned off. All devices may be read or written subject to their permissions.

1 Secure mode − immutable and append-only flags may not be cleared; disks for mounted filesystems, `/dev/mem`, and `/dev/kmem` are read-only.

2 Highly secure mode − same as secure mode, plus disks are always read-only whether mounted or not. This level precludes tampering with filesystems by unmounting them, but also inhibits running *newfs*(8) while the system is multi-user. See *chflags*(1) and the −**o** option to *ls*(1) for information on setting and displaying the immutable and append-only flags.

Normally, the system runs in level 0 mode while single user and in level 1 mode while multiuser. If the level 2 mode is desired while running multiuser, it can be set in the startup script `/etc/rc` using *sysctl*(1). If it is desired to run the system in level 0 mode while multiuser, the administrator must build a kernel with the variable `securelevel` in the kernel source file `/sys/kern/kern_sysctl.c` initialized to −1.

3.5.2.1. Virtual memory changes

The new virtual memory implementation is derived from the Mach operating system developed at Carnegie-Mellon, and was ported to the BSD kernel at the University of Utah. It is based on the 2.0 release of Mach (with some bug fixes from the 2.5 and 3.0 releases) and retains many of its essential features such as the separation of the machine dependent and independent layers (the ‘‘pmap’’ interface), efficient memory utilization using copy-on-write and other lazy-evaluation techniques, and support for large, sparse address spaces. It does not include the ‘‘external pager’’ interface instead using a primitive internal pager interface. The Mach virtual memory system call interface has been replaced with the ‘‘mmap’’-based interface described in the ‘‘Berkeley Software Architecture Manual’’ (see UNIX Programmer’s Manual, Supplementary Documents, PSD:5). The interface is similar to the interfaces shipped by several commercial vendors such as Sun, USL, and Convex Computer Corp. The integration of the new virtual memory is functionally complete, but still has serious performance problems under heavy memory load. The internal kernel interfaces have not yet been completed and the memory pool and buffer cache have not been merged. Some additional caveats:

- Since the code is based on the 2.0 release of Mach, bugs and misfeatures of the BSD version should not be considered short-comings of the current Mach virtual memory system.
- Because of the disjoint virtual memory (page) and IO (buffer) caches, it is possible to see inconsistencies if using both the mmap and read/write interfaces on the same file simultaneously.
- Swap space is allocated on-demand rather than up front and no allocation checks are performed so it is possible to over-commit memory and eventually deadlock.
- The semantics of the *vfork*(2) system call are slightly different. The synchronization between parent and child is preserved, but the memory sharing aspect is not. In practice this has been enough for backward compatibility, but newer code should just use *fork*(2).

3.5.2.2. Networking additions and changes

The ISO/OSI Networking consists of a kernel implementation of transport class 4 (TP-4), connectionless networking protocol (CLNP), and 802.3-based link-level support (hardware-compatible with Ethernet[4]). We also include support for ISO Connection-Oriented Network Service, X.25, TP-0. The session and presentation layers are provided outside the kernel using the ISO Development Environment by Marshall Rose, that is available via anonymous FTP (but is not included on the distribution tape). Included

[4] Ethernet is a trademark of the Xerox Corporation.

in this development environment are file transfer and management (FTAM), virtual terminals (VT), a directory services implementation (X.500), and miscellaneous other utilities.

Kernel support for the ISO OSI protocols is enabled with the ISO option in the kernel configuration file. The *iso*(4) manual page describes the protocols and addressing; see also *clnp*(4), *tp*(4) and *cltp*(4). The OSI equivalent to ARP is ESIS (End System to Intermediate System Routing Protocol); running this protocol is mandatory, however one can manually add translations for machines that do not participate by use of the *route*(8) command. Additional information is provided in the manual page describing *esis*(4).

The command *route*(8) has a new syntax and several new capabilities: it can install routes with a specified destination and mask, and can change route characteristics such as hop count, packet size and window size.

Several important enhancements have been added to the TCP/IP protocols including TCP header prediction and serial line IP (SLIP) with header compression. The routing implementation has been completely rewritten to use a hierarchical routing tree with a mask per route to support the arbitrary levels of routing found in the ISO protocols. The routing table also stores and caches route characteristics to speed the adaptation of the throughput and congestion avoidance algorithms.

The format of the *sockaddr* structure (the structure used to describe a generic network address with an address family and family-specific data) has changed from previous releases, as have the address family-specific versions of this structure. The *sa_family* family field has been split into a length, `sa_len`, and a family, `sa_family`. System calls that pass a *sockaddr* structure into the kernel (e.g. *sendto*() and *connect*()) have a separate parameter that specifies the *sockaddr* length, and thus it is not necessary to fill in the *sa_len* field for those system calls. System calls that pass a *sockaddr* structure back from the kernel (e.g. *recvfrom*() and *accept*()) receive a completely filled-in *sockaddr* structure, thus the length field is valid. Because this would not work for old binaries, the new library uses a different system call number. Thus, most networking programs compiled under 4.4BSD are incompatible with older systems.

Although this change is mostly source and binary compatible with old programs, there are three exceptions. Programs with statically initialized *sockaddr* structures (usually the Internet form, a *sockaddr_in*) are not compatible. Generally, such programs should be changed to fill in the structure at run time, as C allows no way to initialize a structure without assuming the order and number of fields. Also, programs with use structures to describe a network packet format that contain embedded *sockaddr* structures also require change; a definition of an *osockaddr* structure is provided for this purpose. Finally, programs that use the SIOCGIFCONF ioctl to get a complete list of interface addresses need to check the *sa_len* field when iterating through the array of addresses returned, as not all the structures returned have the same length (this variance in length is nearly guaranteed by the presence of link-layer address structures).

3.5.2.3. Additions and changes to filesystems

The 4.4BSD distribution contains most of the interfaces specified in the IEEE Std1003.1 system interface standard. Filesystem additions include IEEE Std1003.1 FIFOs, byte-range file locking, and saved user and group identifiers.

A new virtual filesystem interface has been added to the kernel to support multiple filesystems. In comparison with other interfaces, the Berkeley interface has been structured for more efficient support of filesystems that maintain state (such as the local filesystem). The interface has been extended with support for stackable filesystems done at UCLA. These extensions allow for filesystems to be layered on top of each other and allow new vnode operations to be added without requiring changes to existing filesystem implementations. For example, the umap filesystem (see *mount_umap*(8)) is used to mount a sub-tree of an existing filesystem that uses a different set of uids and gids than the local system. Such a filesystem could be mounted from a remote site via NFS or it could be a filesystem on removable media brought from some foreign location that uses a different password file.

Other new filesystems that may be stacked include the loopback filesystem *mount_lofs*(8), the kernel filesystem *mount_kernfs*(8), and the portal filesystem *mount_portal*(8).

The buffer cache in the kernel is now organized as a file block cache rather than a device block cache. As a consequence, cached blocks from a file and from the corresponding block device would no longer be kept consistent. The block device thus has little remaining value. Three changes have been made for these reasons:

1) block devices may not be opened while they are mounted, and may not be mounted while open, so that the two versions of cached file blocks cannot be created,
2) filesystem checks of the root now use the raw device to access the root filesystem, and
3) the root filesystem is initially mounted read-only so that nothing can be written back to disk during or after change to the raw filesystem by *fsck*.

The root filesystem may be made writable while in single-user mode with the command:

```
mount −uw /
```

The mount command has an option to update the flags on a mounted filesystem, including the ability to upgrade a filesystem from read-only to read-write or downgrade it from read-write to read-only.

In addition to the local ‘‘fast filesystem’’, we have added an implementation of the network filesystem (NFS) that fully interoperates with the NFS shipped by Sun and its licensees. Because our NFS implementation was implemented by Rick Macklem of the University of Guelph using only the publicly available NFS specification, it does not require a license from Sun to use in source or binary form. By default it runs over UDP to be compatible with Sun’s implementation. However, it can be configured on a per-mount basis to run over TCP. Using TCP allows it to be used quickly and efficiently through gateways and over long-haul networks. Using an extended protocol, it supports Leases to allow a limited callback mechanism that greatly reduces the network traffic necessary to maintain cache consistency between the server and its clients. Its use will be familiar to users of other implementations of NFS. See the manual pages *mount*(8), *mountd*(8), *fstab*(5), *exports*(5), *netgroup*(5), *nfsd*(8), *nfsiod*(8), and *nfssvc*(8). and the document ‘‘The 4.4BSD NFS Implementation’’ (SMM:6) for further information. The format of `/etc/fstab` has changed from previous BSD releases to a blank-separated format to allow colons in pathnames.

A new local filesystem, the log-structured filesystem (LFS), has been added to the system. It provides near disk-speed output and fast crash recovery. This work is based, in part, on the LFS filesystem created for the Sprite operating system at Berkeley. While the kernel implementation is almost complete, only some of the utilities to support the filesystem have been written, so we do not recommend it for production use. See *newlfs*(8), *mount_lfs*(8) and *lfs_cleanerd*(8) for more information. For a in-depth description of the implementation and performance characteristics of log-structured filesystems in general, and this one in particular, see Dr. Margo Seltzer’s doctoral thesis, available from the University of California Computer Science Department.

We have also added a memory-based filesystem that runs in pageable memory, allowing large temporary filesystems without requiring dedicated physical memory.

The local ‘‘fast filesystem’’ has been enhanced to do clustering that allows large pieces of files to be allocated contiguously resulting in near doubling of filesystem throughput. The filesystem interface has been extended to allow files and filesystems to grow to 2ˆ63 bytes in size. The quota system has been rewritten to support both user and group quotas (simultaneously if desired). Quota expiration is based on time rather than the previous metric of number of logins over quota. This change makes quotas more useful on fileservers onto which users seldom login.

The system security has been greatly enhanced by the addition of additional file flags that permit a file to be marked as immutable or append only. Once set, these flags can only be cleared by the super-user when the system is running in insecure mode (normally, single-user). In addition to the immutable and append-only flags, the filesystem supports a new user-settable flag ‘‘nodump’’. (File flags are set using the *chflags*(1) utility.) When set on a file, *dump*(8) will omit the file from incremental backups but retain them on full backups. See the ‘‘-h’’ flag to *dump*(8) for details on how to change this default. The ‘‘nodump’’ flag is usually set on core dumps, system crash dumps, and object files generated by the compiler. Note that the flag is not preserved when files are copied so that installing an object file will cause it to be preserved.

The filesystem format used in 4.4BSD has several additions. Directory entries have an additional field, `d_type`, that identifies the type of the entry (normally found in the `st_mode` field of the `stat` structure). This field is particularly useful for identifying directories without the need to use *stat*(2).

Short (less than sixty byte) symbolic links are now stored in the inode itself rather than in a separate data block. This saves disk space and makes access of symbolic links faster. Short symbolic links are not given a special type, so a user-level application is unaware of their special treatment. Unlike pre-4.4BSD systems, symbolic links do not have an owner, group, access mode, times, etc. Instead, these attributes are taken from the directory that contains the link. The only attributes returned from an *lstat*(2) that refer to the symbolic link itself are the file type (S_IFLNK), size, blocks, and link count (always 1).

An implementation of an auto-mounter daemon, *amd*, was contributed by Jan-Simon Pendry of the Imperial College of Science, Technology & Medicine. See the document ''AMD – The 4.4BSD Automounter'' (SMM:13) for further information.

The directory `/dev/fd` contains special files `0` through `63` that, when opened, duplicate the corresponding file descriptor. The names `/dev/stdin`, `/dev/stdout` and `/dev/stderr` refer to file descriptors 0, 1 and 2. See *fd*(4) and *mount_fdesc*(8) for more information.

3.5.2.4. POSIX terminal driver changes

The 4.4BSD system uses the IEEE P1003.1 (POSIX.1) terminal interface rather than the previous BSD terminal interface. The terminal driver is similar to the System V terminal driver with the addition of the necessary extensions to get the functionality previously available in the 4.3BSD terminal driver. Both the old *ioctl* calls and old options to *stty*(1) are emulated. This emulation is expected to be unavailable in many vendors releases, so conversion to the new interface is encouraged.

4.4BSD also adds the IEEE Std1003.1 job control interface, that is similar to the 4.3BSD job control interface, but adds a security model that was missing in the 4.3BSD job control implementation. A new system call, *setsid*(), creates a job-control session consisting of a single process group with one member, the caller, that becomes a session leader. Only a session leader may acquire a controlling terminal. This is done explicitly via a TIOCSCTTY *ioctl*() call, not implicitly by an *open*() call. The call fails if the terminal is in use. Programs that allocate controlling terminals (or pseudo-terminals) require change to work in this environment. The versions of *xterm* provided in the X11R5 release includes the necessary changes. New library routines are available for allocating and initializing pseudo-terminals and other terminals as controlling terminal; see `/usr/src/lib/libutil/pty.c` and `/usr/src/lib/libutil/login_tty.c`.

The POSIX job control model formalizes the previous conventions used in setting up a process group. Unfortunately, this requires that changes be made in a defined order and with some synchronization that were not necessary in the past. Older job control shells (csh, ksh) will generally not operate correctly with the new system.

Most of the other kernel interfaces have been changed to correspond with the POSIX.1 interface, although that work is not complete. See the relevant manual pages and the IEEE POSIX standard.

3.5.2.5. Native operating system compatibility

Both the HP300 and SPARC ports feature the ability to run binaries built for the native operating system (HP-UX or SunOS) by emulating their system calls. Building an HP300 kernel with the HPUXCOMPAT and COMPAT_OHPUX options or a SPARC kernel with the COMPAT_SUNOS option will enable this feature (on by default in the generic kernel provided in the root filesystem image). Though this native operating system compatibility was provided by the developers as needed for their purposes and is by no means complete, it is complete enough to run several non-trivial applications including those that require HP-UX or SunOS shared libraries. For example, the vendor supplied X11 server and windowing environment can be used on both the HP300 and SPARC.

It is important to remember that merely copying over a native binary and executing it (or executing it directly across NFS) does not imply that it will run. All but the most trivial of applications are likely to require access to auxiliary files that do not exist under 4.4BSD (e.g. `/etc/ld.so.cache`) or have a slightly different format (e.g. `/etc/passwd`). However, by using system call tracing and through creative use of symlinks, many problems can be tracked down and corrected.

The DECstation port also has code for ULTRIX emulation (kernel option ULTRIXCOMPAT, not compiled into the generic kernel) but it was used primarily for initially bootstrapping the port and has not been used since. Hence, some work may be required to make it generally useful.

3.5.3. Changes to the utilities

We have been tracking the IEEE Std1003.2 shell and utility work and have included prototypes of many of the proposed utilities based on draft 12 of the POSIX.2 Shell and Utilities document. Because most of the traditional utilities have been replaced with implementations conformant to the POSIX standards, you should realize that the utility software may not be as stable, reliable or well documented as in traditional Berkeley releases. In particular, almost the entire manual suite has been rewritten to reflect the POSIX defined interfaces, and in some instances it does not correctly reflect the current state of the software. It is also worth noting that, in rewriting this software, we have generally been rewarded with significant performance improvements. Most of the libraries and header files have been converted to be compliant with ANSI C. The shipped compiler (gcc) is a superset of ANSI C, but supports traditional C as a command-line option. The system libraries and utilities all compile with either ANSI or traditional C.

3.5.3.1. Make and Makefiles

This release uses a completely new version of the *make* program derived from the *pmake* program developed by the Sprite project at Berkeley. It supports existing makefiles, although certain incorrect makefiles may fail. The makefiles for the 4.4BSD sources make extensive use of the new facilities, especially conditionals and file inclusion, and are thus completely incompatible with older versions of *make* (but nearly all the makefiles are now trivial!). The standard include files for *make* are in `/usr/share/mk`. There is a `bsd.README` file in `/usr/src/share/mk`.

Another global change supported by the new *make* is designed to allow multiple architectures to share a copy of the sources. If a subdirectory named `obj` is present in the current directory, *make* descends into that directory and creates all object and other files there. We use this by building a directory hierarchy in `/var/obj` that parallels `/usr/src`. We then create the `obj` subdirectories in `/usr/src` as symbolic links to the corresponding directories in `/var/obj`. (This step is automated. The command ‘‘make obj’’ in `/usr/src` builds both the local symlink and the shadow directory, using `/usr/obj`, that may be a symbolic link, as the root of the shadow tree. The use of `/usr/obj` is for historic reasons only, and the system make configuration files in `/usr/share/mk` can trivially be modified to use `/var/obj` instead.) We have one `/var/obj` hierarchy on the local system, and another on each system that shares the source filesystem. All the sources in `/usr/src` except for `/usr/src/contrib` and portions of `/usr/src/old` have been converted to use the new make and `obj` subdirectories; this change allows compilation for multiple architectures from the same source tree (that may be mounted read-only).

3.5.3.2. Kerberos

The Kerberos authentication server from MIT (version 4) is included in this release. See *kerberos*(1) for a general, if MIT-specific, introduction. If it is configured, *login*(1), *passwd*(1), *rlogin*(1) and *rsh*(1) will all begin to use it automatically. The file `/etc/kerberosIV/README` describes the configuration. Each system needs the file `/etc/kerberosIV/krb.conf` to set its realm and local servers, and a private key stored in `/etc/kerberosIV/srvtab` (see *ext_srvtab*(8)). The Kerberos server should be set up on a single, physically secure, server machine. Users and hosts may be added to the server database manually with *kdb_edit*(8), or users on authorized hosts can add themselves and a Kerberos password after verification of their ‘‘local’’ (passwd-file) password using the *register*(1) program.

Note that by default the password-changing program *passwd*(1) changes the Kerberos password, that must exist. The `−l` option to *passwd*(1) changes the ‘‘local’’ password if one exists.

Note that Version 5 of Kerberos will be released soon; Version 4 should probably be replaced at that time.

3.5.3.3. Timezone support

The timezone conversion code in the C library uses data files installed in `/usr/share/zoneinfo` to convert from ‘‘GMT’’ to various timezones. The data file for the default timezone for the system should be copied to `/etc/localtime`. Other timezones can be selected by setting the TZ environment variable.

The data files initially installed in `/usr/share/zoneinfo` include corrections for leap seconds since the beginning of 1970. Thus, they assume that the kernel will increment the time at a constant rate during a leap second; that is, time just keeps on ticking. The conversion routines will then name a leap second 23:59:60. For purists, this effectively means that the kernel maintains TAI (International Atomic Time) rather than UTC (Coordinated Universal Time, aka GMT).

For systems that run current NTP (Network Time Protocol) implementations or that wish to conform to the letter of the POSIX.1 law, it is possible to rebuild the timezone data files so that leap seconds are not counted. (NTP causes the time to jump over a leap second, and POSIX effectively requires the clock to be reset by hand when a leap second occurs. In this mode, the kernel effectively runs UTC rather than TAI.)

The data files without leap second information are constructed from the source directory, `/usr/src/share/zoneinfo`. Change the variable REDO in Makefile from ‘‘right’’ to ‘‘posix’’, and then do

```
make obj     (if necessary)
make
make install
```

You will then need to copy the correct default zone file to `/etc/localtime`, as the old one would still have used leap seconds, and because the Makefile installs a default `/etc/localtime` each time ‘‘make install’’ is done.

It is possible to install both sets of timezone data files. This results in subdirectories `/usr/share/zoneinfo/right` and `/usr/share/zoneinfo/posix`. Each contain a complete set of zone files. See `/usr/src/share/zoneinfo/Makefile` for details.

3.5.3.4. Additions and changes to the libraries

Notable additions to the libraries include functions to traverse a filesystem hierarchy, database interfaces to btree and hashing functions, a new, faster implementation of stdio and a radix and merge sort functions.

The *fts*(3) functions will do either physical or logical traversal of a file hierarchy as well as handle essentially infinite depth filesystems and filesystems with cycles. All the utilities in 4.4BSD which traverse file hierarchies have been converted to use *fts*(3). The conversion has always resulted in a significant performance gain, often of four or five to one in system time.

The *dbopen*(3) functions are intended to be a family of database access methods. Currently, they consist of *hash*(3), an extensible, dynamic hashing scheme, *btree*(3), a sorted, balanced tree structure (B+tree’s), and *recno*(3), a flat-file interface for fixed or variable length records referenced by logical record number. Each of the access methods stores associated key/data pairs and uses the same record oriented interface for access.

The *qsort*(3) function has been rewritten for additional performance. In addition, three new types of sorting functions, *heapsort*(3), *mergesort*(3) and *radixsort*(3) have been added to the system. The *mergesort* function is optimized for data with pre-existing order, in which case it usually significantly outperforms *qsort*. The *radixsort*(3) functions are variants of most-significant-byte radix sorting. They take time linear to the number of bytes to be sorted, usually significantly outperforming *qsort* on data that can be sorted in this fashion. An implementation of the POSIX 1003.2 standard *sort*(1), based on *radixsort*, is included in `/usr/src/contrib/sort`.

Some additional comments about the 4.4BSD C library:

- The floating point support in the C library has been replaced and is now accurate.

- The C functions specified by both ANSI C, POSIX 1003.1 and 1003.2 are now part of the C library. This includes support for file name matching, shell globbing and both basic and extended regular expressions.
- ANSI C multibyte and wide character support has been integrated. The rune functionality from the Bell Labs' Plan 9 system is provided as well.
- The *termcap*(3) functions have been generalized and replaced with a general purpose interface named *getcap*(3).
- The *stdio*(3) routines have been replaced, and are usually much faster. In addition, the *funopen*(3) interface permits applications to provide their own I/O stream function support.

The *curses*(3) library has been largely rewritten. Important additional features include support for scrolling and *termios*(3).

An application front-end editing library, named libedit, has been added to the system.

A superset implementation of the SunOS kernel memory interface library, libkvm, has been integrated into the system.

3.5.3.5. Additions and changes to other utilities

There are many new utilities, offering many new capabilities, in 4.4BSD. Skimming through the section 1 and section 8 manual pages is sure to be useful. The additions to the utility suite include greatly enhanced versions of programs that display system status information, implementations of various traditional tools described in the IEEE Std1003.2 standard, new tools not previous available on Berkeley UNIX systems, and many others. Also, with only a very few exceptions, all the utilities from 4.3BSD that included proprietary source code have been replaced, and their 4.4BSD counterparts are freely redistributable. Normally, this replacement resulted in significant performance improvements and the increase of the limits imposed on data by the utility as well.

A summary of specific additions and changes are as follows:

`amd`	An auto-mounter implementation.
`ar`	Replacement of the historic archive format with a new one.
`awk`	Replaced by gawk; see /usr/src/old/awk for the historic version.
`bdes`	Utility implementing DES modes of operation described in FIPS PUB 81.
`calendar`	Addition of an interface for system calendars.
`cap_mkdb`	Utility for building hashed versions of termcap style databases.
`cc`	Replacement of pcc with gcc suite.
`chflags`	A utility for setting the per-file user and system flags.
`chfn`	An editor based replacement for changing user information.
`chpass`	An editor based replacement for changing user information.
`chsh`	An editor based replacement for changing user information.
`cksum`	The POSIX 1003.2 checksum utility; compatible with sum.
`column`	A columnar text formatting utility.
`cp`	POSIX 1003.2 compatible, able to copy special files.
`csh`	Freely redistributable and 8-bit clean.
`date`	User specified formats added.
`dd`	New EBCDIC conversion tables, major performance improvements.
`dev_mkdb`	Hashed interface to devices.
`dm`	Dungeon master.
`find`	Several new options and primaries, major performance improvements.
`fstat`	Utility displaying information on files open on the system.
`ftpd`	Connection logging added.
`hexdump`	A binary dump utility, superseding od.
`id`	The POSIX 1003.2 user identification utility.
`inetd`	Tcpmux added.
`jot`	A text formatting utility.

`kdump`	A system-call tracing facility.
`ktrace`	A system-call tracing facility.
`kvm_mkdb`	Hashed interface to the kernel name list.
`lam`	A text formatting utility.
`lex`	A new, freely redistributable, significantly faster version.
`locate`	A database of the system files, by name, constructed weekly.
`logname`	The POSIX 1003.2 user identification utility.
`mail.local`	New local mail delivery agent, replacing mail.
`make`	Replaced with a new, more powerful make, supporting include files.
`man`	Added support for man page location configuration.
`mkdep`	A new utility for generating make dependency lists.
`mkfifo`	The POSIX 1003.2 FIFO creation utility.
`mtree`	A new utility for mapping file hierarchies to a file.
`nfsstat`	An NFS statistics utility.
`nvi`	A freely redistributable replacement for the ex/vi editors.
`pax`	The POSIX 1003.2 replacement for cpio and tar.
`printf`	The POSIX 1003.2 replacement for echo.
`roff`	Replaced by groff; see /usr/src/old/roff for the historic versions.
`rs`	New utility for text formatting.
`shar`	An archive building utility.
`sysctl`	MIB-style interface to system state.
`tcopy`	Fast tape-to-tape copying and verification.
`touch`	Time and file reference specifications.
`tput`	The POSIX 1003.2 terminal display utility.
`tr`	Addition of character classes.
`uname`	The POSIX 1003.2 system identification utility.
`vis`	A filter for converting and displaying non-printable characters.
`xargs`	The POSIX 1003.2 argument list constructor utility.
`yacc`	A new, freely redistributable, significantly faster version.

The new versions of *lex*(1) (‘‘flex’’) and *yacc*(1) (‘‘zoo’’) should be installed early on if attempting to cross-compile 4.4BSD on another system. Note that the new *lex* program is not completely backward compatible with historic versions of *lex*, although it is believed that all documented features are supported.

The *find* utility has two new options that are important to be aware of if you intend to use NFS. The ‘‘fstype’’ and ‘‘prune’’ options can be used together to prevent find from crossing NFS mount points. See `/etc/daily` for an example of their use.

3.6. Hints on converting from 4.3BSD to 4.4BSD

This section summarizes changes between 4.3BSD and 4.4BSD that are likely to cause difficulty in doing the conversion. It does not include changes in the network; see section 5 for information on setting up the network.

Since the stat st_size field is now 64-bits instead of 32, doing something like:

```
foo(st.st_size);
```

and then (improperly) defining foo with an ‘‘int’’ or ‘‘long’’ parameter:

```
foo(size)
	int size;
{
	...
}
```

will fail miserably (well, it might work on a little endian machine). This problem showed up in *emacs*(1) as well as several other programs. A related problem is improperly casting (or failing to cast) the second argument to *lseek*(2), *truncate*(2), or *ftruncate*(2) ala:

```
lseek(fd, (long)off, 0);
```

or

```
lseek(fd, 0, 0);
```

The best solution is to include `<unistd.h>` which has prototypes that catch these types of errors.

Determining the ‘‘namelen’’ parameter for a *connect*(2) call on a unix domain socket should use the ‘‘SUN_LEN’’ macro from `<sys/un.h>`. One old way that was used:

```
addrlen = strlen(unaddr.sun_path) + sizeof(unaddr.sun_family);
```

no longer works as there is an additional `sun_len` field.

The kernel’s limit on the number of open files has been increased from 20 to 64. It is now possible to change this limit almost arbitrarily. The standard I/O library autoconfigures to the kernel limit. Note that file (‘‘_iob’’) entries may be allocated by *malloc* from *fopen*; this allocation has been known to cause problems with programs that use their own memory allocators. Memory allocation does not occur until after 20 files have been opened by the standard I/O library.

Select can be used with more than 32 descriptors by using arrays of **int**s for the bit fields rather than single **int**s. Programs that used *getdtablesize* as their first argument to *select* will no longer work correctly. Usually the program can be modified to correctly specify the number of bits in an **int**. Alternatively the program can be modified to use an array of **int**s. There are a set of macros available in `<sys/types.h>` to simplify this. See *select*(2).

Old core files will not be intelligible by the current debuggers because of numerous changes to the user structure and because the kernel stack has been enlarged. The *a.out* header that was in the user structure is no longer present. Locally-written debuggers that try to check the magic number will need to be changed.

Files may not be deleted from directories having the ‘‘sticky’’ (ISVTX) bit set in their modes except by the owner of the file or of the directory, or by the superuser. This is primarily to protect users’ files in publicly-writable directories such as `/tmp` and `/var/tmp`. All publicly-writable directories should have their ‘‘sticky’’ bits set with ‘‘chmod +t.’’

The following two sections contain additional notes about changes in 4.4BSD that affect the installation of local files; be sure to read them as well.

4. System setup

This section describes procedures used to set up a 4.4BSD UNIX system. These procedures are used when a system is first installed or when the system configuration changes. Procedures for normal system operation are described in the next section.

4.1. Kernel configuration

This section briefly describes the layout of the kernel code and how files for devices are made. For a full discussion of configuring and building system images, consult the document ‘‘Building 4.3BSD UNIX Systems with Config’’ (SMM:2).

4.1.1. Kernel organization

As distributed, the kernel source is in a separate tar image. The source may be physically located anywhere within any filesystem so long as a symbolic link to the location is created for the file `/sys` (many files in `/usr/include` are normally symbolic links relative to `/sys`). In further discussions of the system source all path names will be given relative to `/sys`.

The kernel is made up of several large generic parts:

sys		main kernel header files
kern		kernel functions broken down as follows
	init	system startup, syscall dispatching, entry points

	kern	scheduling, descriptor handling and generic I/O
	sys	process management, signals
	tty	terminal handling and job control
	vfs	filesystem management
	uipc	interprocess communication (sockets)
	subr	miscellaneous support routines
vm		virtual memory management
ufs		local filesystems broken down as follows
	ufs	common local filesystem routines
	ffs	fast filesystem
	lfs	log-based filesystem
	mfs	memory based filesystem
nfs		Sun-compatible network filesystem
miscfs		miscellaneous filesystems broken down as follows
	deadfs	where rejected vnodes go to die
	fdesc	access to per-process file descriptors
	fifofs	IEEE Std1003.1 FIFOs
	kernfs	filesystem access to kernel data structures
	lofs	loopback filesystem
	nullfs	another loopback filesystem
	portal	associate processes with filesystem locations
	specfs	device special files
	umapfs	provide alternate uid/gid mappings
dev		generic device drivers (SCSI, vnode, concatenated disk)

The networking code is organized by protocol

net	routing and generic interface drivers
netinet	Internet protocols (TCP, UDP, IP, etc)
netiso	ISO protocols (TP-4, CLNP, CLTP, etc)
netns	Xerox network systems protocols (IDP, SPP, etc)
netx25	CCITT X.25 protocols (X.25 Packet Level, HDLC/LAPB)

A separate subdirectory is provided for each machine architecture

hp300	HP 9000/300 series of Motorola 68000-based machines
hp	code common to both HP 68k and (non-existent) PA-RISC ports
i386	Intel 386/486-based PC machines
luna68k	Omron 68000-based workstations
news3400	Sony News MIPS-based workstations
pmax	Digital 3100/5000 MIPS-based workstations
sparc	Sun Microsystems SPARCstation 1, 1+, and 2
tahoe	(deprecated) CCI Power 6-series machines
vax	(deprecated) Digital VAX machines

Each machine directory is subdivided by function; for example the hp300 directory contains

include	exported machine-dependent header files
hp300	machine-dependent support code and private header files
dev	device drivers
conf	configuration files
stand	machine-dependent standalone code

Other kernel related directories

compile	area to compile kernels
conf	machine-independent configuration files
stand	machine-independent standalone code

4.1.2. Devices and device drivers

Devices supported by UNIX are implemented in the kernel by drivers whose source is kept in `/sys/<architecture>/dev`. These drivers are loaded into the system when included in a cpu specific configuration file kept in the conf directory. Devices are accessed through special files in the filesystem, made by the *mknod*(8) program and normally kept in the `/dev` directory. For all the devices supported by the distribution system, the files in `/dev` are created by the `/dev/MAKEDEV` shell script.

Determine the set of devices that you have and create a new `/dev` directory by running the MAKEDEV script. First create a new directory `/newdev`, copy MAKEDEV into it, edit the file MAKEDEV.local to provide an entry for local needs, and run it to generate a `/newdev`directory. For instance,

```
# cd /
# mkdir newdev
# cp dev/MAKEDEV newdev/MAKEDEV
# cd newdev
# MAKEDEV sd0 pt0 std LOCAL
```

Note the ‘‘std’’ argument causes standard devices such as `/dev/console`, the machine console, to be created.

You can then do

```
# cd /
# mv dev olddev ; mv newdev dev
# sync
```

to install the new device directory.

4.1.3. Building new system images

The kernel configuration of each UNIX system is described by a single configuration file, stored in the `/sys/<architecture>/conf` directory. To learn about the format of this file and the procedure used to build system images, start by reading ‘‘Building 4.3BSD UNIX Systems with Config’’ (SMM:2), look at the manual pages in section 4 of the UNIX manual for the devices you have, and look at the sample configuration files in the `/sys/<architecture>/conf` directory.

The configured system image `vmunix` should be copied to the root, and then booted to try it out. It is best to name it `/newvmunix` so as not to destroy the working system until you are sure it does work:

```
# cp vmunix /newvmunix
# sync
```

It is also a good idea to keep the previous system around under some other name. In particular, we recommend that you save the generic distribution version of the system permanently as `/genvmunix` for use in emergencies. To boot the new version of the system you should follow the bootstrap procedures outlined in section 6.1. After having booted and tested the new system, it should be installed as `/vmunix` before going into multiuser operation. A systematic scheme for numbering and saving old versions of the system may be useful.

4.2. Configuring terminals

If UNIX is to support simultaneous access from directly-connected terminals other than the console, the file `/etc/ttys` (see *ttys*(5)) must be edited.

To add a new terminal device, be sure the device is configured into the system and that the special files for the device have been made by `/dev/MAKEDEV`. Then, enable the appropriate lines of `/etc/ttys` by setting the ‘‘status’’ field to **on** (or add new lines). Note that lines in `/etc/ttys` are one-for-one with entries in the file of current users (see `/var/run/utmp`), and therefore it is best to make changes while running in single-user mode and to add all the entries for a new device at once.

Each line in the `/etc/ttys` file is broken into four tab separated fields (comments are shown by a '#' character and extend to the end of the line). For each terminal line the four fields are: the device (without a leading `/dev`), the program `/sbin/init` should startup to service the line (or **none** if the line is to be left alone), the terminal type (found in `/usr/share/misc/termcap`), and optional status information describing if the terminal is enabled or not and if it is ''secure'' (i.e. the super user should be allowed to login on the line). If the console is marked as ''insecure'', then the root password is required to bring the machine up single-user. All fields are character strings with entries requiring embedded white space enclosed in double quotes. Thus a newly added terminal `/dev/tty00` could be added as

```
tty00  "/usr/libexec/getty std.9600"    vt100 on secure    # mike's office
```

The std.9600 parameter provided to `/usr/libexec/getty` is used in searching the file `/etc/gettytab`; it specifies a terminal's characteristics (such as baud rate). To make custom terminal types, consult *gettytab*(5) before modifying `/etc/gettytab`.

Dialup terminals should be wired so that carrier is asserted only when the phone line is dialed up. For non-dialup terminals, from which modem control is not available, you must wire back the signals so that the carrier appears to always be present. For further details, find your terminal driver in section 4 of the manual.

For network terminals (i.e. pseudo terminals), no program should be started up on the lines. Thus, the normal entry in `/etc/ttys` would look like

```
ttyp0  none  network
```

(Note, the fourth field is not needed here.)

When the system is running multi-user, all terminals that are listed in `/etc/ttys` as **on** have their line enabled. If, during normal operations, you wish to disable a terminal line, you can edit the file `/etc/ttys` to change the terminal's status to **off** and then send a hangup signal to the *init* process, by doing

```
# kill −s HUP 1
```

Terminals can similarly be enabled by changing the status field from **off** to **on** and sending a hangup signal to *init*.

Note that if a special file is inaccessible when *init* tries to create a process for it, *init* will log a message to the system error logging process (see *syslogd*(8)) and try to reopen the terminal every minute, reprinting the warning message every 10 minutes. Messages of this sort are normally printed on the console, though other actions may occur depending on the configuration information found in `/etc/syslog.conf`.

Finally note that you should change the names of any dialup terminals to ttyd? where ? is in [0-9a-zA-Z], as some programs use this property of the names to determine if a terminal is a dialup. Shell commands to do this should be put in the `/dev/MAKEDEV.local` script.

While it is possible to use truly arbitrary strings for terminal names, the accounting and noticeably the *ps*(1) command make good use of the convention that tty names (by default, and also after dialups are named as suggested above) are distinct in the last 2 characters. Change this and you may be sorry later, as the heuristic *ps*(1) uses based on these conventions will then break down and *ps* will run MUCH slower.

4.3. Adding users

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.

Several guest accounts have been provided on the distribution system; these accounts are for people at Berkeley, Bell Laboratories, and others who have done major work on UNIX in the past. You can delete these accounts, or leave them on the system if you expect that these people would have occasion to login as guests on your system.

4.4. Site tailoring

All programs that require the site's name, or some similar characteristic, obtain the information through system calls or from files located in `/etc`. Aside from parts of the system related to the network, to tailor the system to your site you must simply select a site name, then edit the file

/etc/netstart

The first lines in `/etc/netstart` use a variable to set the hostname,

hostname=*mysitename*
/bin/hostname $hostname

to define the value returned by the *gethostname*(2) system call. If you are running the name server, your site name should be your fully qualified domain name. Programs such as *getty*(8), *mail*(1), *wall*(1), and *uucp*(1) use this system call so that the binary images are site independent.

You will also need to edit `/etc/netstart` to do the network interface initialization using *ifconfig*(8). If you are not sure how to do this, see sections 5.1, 5.2, and 5.3. If you are not running a routing daemon and have more than one Ethernet in your environment you will need to set up a default route; see section 5.4 for details. Before bringing your system up multiuser, you should ensure that the networking is properly configured. The network is started by running `/etc/netstart`. Once started, you should test connectivity using *ping*(8). You should first test connectivity to yourself, then another host on your Ethernet, and finally a host on another Ethernet. The *netstat*(8) program can be used to inspect and debug your routes; see section 5.4.

4.5. Setting up the line printer system

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

/usr/bin/lpq	spooling queue examination program
/usr/bin/lprm	program to delete jobs from a queue
/usr/bin/lpr	program to enter a job in a printer queue
/etc/printcap	printer configuration and capability database
/usr/sbin/lpd	line printer daemon, scans spooling queues
/usr/sbin/lpc	line printer control program
/etc/hosts.lpd	list of host allowed to use the printers

The file `/etc/printcap` is a master database describing line printers directly attached to a machine and, also, printers accessible across a network. The manual page *printcap*(5) describes the format of this database and also shows 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 also printers attached via serial lines that 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.

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 started with *lpr*, the job is queued locally and a daemon process 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 and the accompanying document, ‘‘4.3BSD Line Printer Spooler Manual’’ (SMM:7). A call to the *lpd* program should be present in `/etc/rc`.

4.6. Setting up the mail system

The mail system consists of the following commands:

/usr/bin/mail	UCB mail program, described in *mail*(1)
/usr/sbin/sendmail	mail routing program
/var/spool/mail	mail spooling directory
/var/spool/secretmail	secure mail directory
/usr/bin/xsend	secure mail sender
/usr/bin/xget	secure mail receiver
/etc/aliases	mail forwarding information
/usr/bin/newaliases	command to rebuild binary forwarding database
/usr/bin/biff	mail notification enabler
/usr/libexec/comsat	mail notification daemon

Mail is normally sent and received using the *mail*(1) command (found in `/usr/bin/mail`), 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 `/usr/libexec/mail.local` that adds it to the mailboxes in the directory `/var/spool/mail/<username>`, using a locking protocol to avoid problems with simultaneous updates. After the mail is delivered, the local mail delivery daemon `/usr/libexec/comsat` is notified, which in turn notifies users who have issued a ‘‘*biff* y’’ command that mail has arrived.

Mail queued in the directory `/var/spool/mail` is normally readable only by the recipient. To send mail that is secure against perusal (except by a code-breaker) you should use the secret mail facility, which encrypts the mail.

To set up the mail facility you should read the instructions in the file READ_ME in the directory `/usr/src/usr.sbin/sendmail` and then adjust the necessary configuration files. You should also set up the file `/etc/aliases` for your installation, creating mail groups as appropriate. For more informations see ‘‘Sendmail Installation and Operation Guide’’ (SMM:8) and ‘‘Sendmail – An Internetwork Mail Router’’ (SMM:9).

4.6.1. Setting up a UUCP connection

The version of *uucp* included in 4.4BSD has the following features:

- support for many auto call units and dialers in addition to the DEC DN11,
- breakup of the spooling area into multiple subdirectories,
- addition of an `L.cmds` file to control the set of commands that may be executed by a remote site,
- enhanced ‘‘expect-send’’ sequence capabilities when logging in to a remote site,
- new commands to be used in polling sites and obtaining snap shots of *uucp* activity,
- additional protocols for different communication media.

This section gives a brief overview of *uucp* and points out the most important steps in its installation.

To connect two UNIX machines with a *uucp* network link using modems, one site must have an automatic call unit and the other must have a dialup port. It is better if both sites have both.

You should first read the paper in the UNIX System Manager’s Manual: ‘‘Uucp Implementation Description’’ (SMM:14). It describes in detail the file formats and conventions, and will give you a little context. In addition, the document ‘‘setup.tblms’’, located in the directory `/usr/src/usr.bin/uucp/UUAIDS`, may be of use in tailoring the software to your needs.

The *uucp* support is located in three major directories: `/usr/bin`, `/usr/lib/uucp`, and `/var/spool/uucp`. User commands are kept in `/usr/bin`, operational commands in

`/usr/lib/uucp`, and `/var/spool/uucp` is used as a 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
/usr/bin/uusnap	gives a snap-shot of *uucp* activity
/usr/bin/uupoll	polls remote system until an answer is received
/usr/bin/uuname	prints a list of known uucp hosts
/usr/bin/uuq	gives information about the queue

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

/usr/lib/uucp/L-devices	list of dialers and hard-wired lines
/usr/lib/uucp/L-dialcodes	dialcode abbreviations
/usr/lib/uucp/L.aliases	hostname aliases
/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/uucico	*uucp* protocol daemon
/usr/lib/uucp/uuclean	cleans up garbage files in spool area
/usr/lib/uucp/uuxqt	*uucp* remote execution server

while the spooling area contains the following important files and directories:

/var/spool/uucp/C.	directory for command, ‘‘C.’’ files
/var/spool/uucp/D.	directory for data, ‘‘D.’’, files
/var/spool/uucp/X.	directory for command execution, ‘‘X.’’, files
/var/spool/uucp/D.*machine*	directory for local ‘‘D.’’ files
/var/spool/uucp/D.*machine*X	directory for local ‘‘X.’’ files
/var/spool/uucp/TM.	directory for temporary, ‘‘TM.’’, files
/var/spool/uucp/LOGFILE	log file of *uucp* activity
/var/spool/uucp/SYSLOG	log file of *uucp* file transfers

To install *uucp* on your system, start by selecting a site name (shorter than 14 characters). A *uucp* account must be created in the password file and a password set up. Then, create the appropriate spooling directories with mode 755 and owned by user *uucp*, group *daemon*.

If you have an auto-call unit, the L.sys, L-dialcodes, and L-devices files should be created. The L.sys file should contain the phone numbers and login sequences required to establish a connection with a *uucp* daemon on another machine. For example, our L.sys file looks something like:

```
adiron Any ACU 1200 out0123456789- ogin-EOT-ogin uucp
cbosg Never Slave 300
cbosgd Never Slave 300
chico Never Slave 1200 out2010123456
```

The first field is the name of a site, the second shows when the machine may be called, the third field specifies how the host is connected (through an ACU, a hard-wired line, etc.), then comes the phone number to use in connecting through an auto-call unit, and finally a login sequence. The phone number may contain common abbreviations that are defined in the L-dialcodes file. The device specification should refer to devices specified in the L-devices file. Listing only ACU causes the *uucp* daemon, *uucico*, to search for any available auto-call unit in L-devices. Our L-dialcodes file is of the form:

```
ucb 2
out 9%
```

while our L-devices file is:

```
ACU cul0 unused 1200 ventel
```

Refer to the README file in the *uucp* source directory for more information about installation.

As *uucp* operates it creates (and removes) many small files in the directories underneath `/var/spool/uucp`. Sometimes files are left undeleted; these are most easily purged with the *uuclean* program. The log files can grow without bound unless trimmed back; *uulog* maintains these files. Many useful aids in maintaining your *uucp* installation are included in a subdirectory UUAIDS beneath `/usr/src/usr.bin/uucp`. Peruse this directory and read the ‘‘setup’’ instructions also located there.

5. Network setup

4.4BSD provides support for the standard Internet protocols IP, ICMP, TCP, and UDP. These protocols may be used on top of a variety of hardware devices ranging from serial lines to local area network controllers for the Ethernet. Network services are split between the kernel (communication protocols) and user programs (user services such as TELNET and FTP). This section describes how to configure your system to use the Internet networking support. 4.4BSD also supports the Xerox Network Systems (NS) protocols. IDP and SPP are implemented in the kernel, and other protocols such as Courier run at the user level. 4.4BSD provides some support for the ISO OSI protocols CLNP TP4, and ESIS. User level process complete the application protocols such as X.400 and X.500.

5.1. System configuration

To configure the kernel to include the Internet communication protocols, define the INET option. Xerox NS support is enabled with the NS option. ISO OSI support is enabled with the ISO option. In either case, include the pseudo-devices ‘‘pty’’, and ‘‘loop’’ in your machine’s configuration file. The ‘‘pty’’ pseudo-device forces the pseudo terminal device driver to be configured into the system, see *pty*(4), while the ‘‘loop’’ pseudo-device forces inclusion of the software loopback interface driver. The loop driver is used in network testing and also by the error logging system.

If you are planning to use the Internet network facilities on a 10Mb/s Ethernet, the pseudo-device ‘‘ether’’ should also be included in the configuration; this forces inclusion of the Address Resolution Protocol module used in mapping between 48-bit Ethernet and 32-bit Internet addresses.

Before configuring the appropriate networking hardware, you should consult the manual pages in section 4 of the Programmer’s Manual selecting the appropriate interfaces for your architecture.

All network interface drivers including the loopback interface, require that their host address(es) be defined at boot time. This is done with *ifconfig*(8) commands included in the `/etc/netstart` file. Interfaces that are able to dynamically deduce the host part of an address may check that the host part of the address is correct. The manual page for each network interface describes the method used to establish a host’s address. *Ifconfig*(8) can also be used to set options for the interface at boot time. Options are set independently for each interface, and apply to all packets sent using that interface. Alternatively, translations for such hosts may be set in advance or ‘‘published’’ by a 4.4BSD host by use of the *arp*(8) command. Note that the use of trailer link-level is now negotiated between 4.4BSD hosts using ARP, and it is thus no longer necessary to disable the use of trailers with *ifconfig*.

The OSI equivalent to ARP is ESIS (End System to Intermediate System Routing Protocol); running this protocol is mandatory, however one can manually add translations for machines that do not participate by use of the *route*(8) command. Additional information is provided in the manual page describing *ESIS*(4).

To use the pseudo terminals just configured, device entries must be created in the `/dev` directory. To create 32 pseudo terminals (plenty, unless you have a heavy network load) execute the following commands.

 # cd /dev
 # MAKEDEV pty0 pty1

More pseudo terminals may be made by specifying `pty2`, `pty3`, etc. The kernel normally includes support for 32 pseudo terminals unless the configuration file specifies a different number. Each pseudo terminal really consists of two files in `/dev`: a master and a slave. The master pseudo terminal file is named `/dev/ptyp?`, while the slave side is `/dev/ttyp?`. Pseudo terminals are also used by several programs not related to the network. In addition to creating the pseudo terminals, be sure to install them in the `/etc/ttys` file (with a 'none' in the second column so no *getty* is started).

5.2. Local subnets

In 4.4BSD the Internet support includes the notion of ''subnets''. This is a mechanism by which multiple local networks may appears as a single Internet network to off-site hosts. Subnetworks are useful because they allow a site to hide their local topology, requiring only a single route in external gateways; it also means that local network numbers may be locally administered. The standard describing this change in Internet addressing is RFC-950.

To set up local subnets one must first decide how the available address space (the Internet ''host part'' of the 32-bit address) is to be partitioned. Sites with a class A network number have a 24-bit host address space with which to work, sites with a class B network number have a 16-bit host address space, while sites with a class C network number have an 8-bit host address space[5]. To define local subnets you must steal some bits from the local host address space for use in extending the network portion of the Internet address. This reinterpretation of Internet addresses is done only for local networks; i.e. it is not visible to hosts off-site. For example, if your site has a class B network number, hosts on this network have an Internet address that contains the network number, 16 bits, and the host number, another 16 bits. To define 254 local subnets, each possessing at most 255 hosts, 8 bits may be taken from the local part. (The use of subnets 0 and all-1's, 255 in this example, is discouraged to avoid confusion about broadcast addresses.) These new network numbers are then constructed by concatenating the original 16-bit network number with the extra 8 bits containing the local subnet number.

The existence of local subnets is communicated to the system at the time a network interface is configured with the *netmask* option to the *ifconfig* program. A ''network mask'' is specified to define the portion of the Internet address that is to be considered the network part for that network. This mask normally contains the bits corresponding to the standard network part as well as the portion of the local part that has been assigned to subnets. If no mask is specified when the address is set, it will be set according to the class of the network. For example, at Berkeley (class B network 128.32) 8 bits of the local part have been reserved for defining subnets; consequently the `/etc/netstart` file contains lines of the form

```
/sbin/ifconfig le0 netmask 0xffffff00 128.32.1.7
```

This specifies that for interface ''le0'', the upper 24 bits of the Internet address should be used in calculating network numbers (netmask 0xffffff00), and the interface's Internet address is ''128.32.1.7'' (host 7 on network 128.32.1). Hosts *m* on sub-network *n* of this network would then have addresses of the form ''128.32.*n.m*''; for example, host 99 on network 129 would have an address ''128.32.129.99''. For hosts with multiple interfaces, the network mask should be set for each interface, although in practice only the mask of the first interface on each network is really used.

5.3. Internet broadcast addresses

The address defined as the broadcast address for Internet networks according to RFC-919 is the address with a host part of all 1's. The address used by 4.2BSD was the address with a host part of 0. 4.4BSD uses the standard broadcast address (all 1's) by default, but allows the broadcast address to be set (with *ifconfig*) for each interface. This allows networks consisting of both 4.2BSD, 4.3BSD and 4.4BSD hosts to coexist while the upgrade process proceeds. In the presence of subnets, the broadcast address uses

[5] If you are unfamiliar with the Internet addressing structure, consult ''Address Mappings'', Internet RFC-796, J. Postel; available from the Internet Network Information Center at SRI.

the subnet field as for normal host addresses, with the remaining host part set to 1's (or 0's, on a network that has not yet been converted). 4.4BSD hosts recognize and accept packets sent to the logical-network broadcast address as well as those sent to the subnet broadcast address, and when using an all-1's broadcast, also recognize and receive packets sent to host 0 as a broadcast.

5.4. Routing

If your environment allows access to networks not directly attached to your host you will need to set up routing information to allow packets to be properly routed. Two schemes are supported by the system. The first scheme employs a routing table management daemon. Optimally, you should use the routing daemon *gated* available from Cornell university. We use it on our systems and it works well, especially for multi-homed hosts using Serial Line IP (SLIP). Unfortunately, we were not able to obtain permission to include it on 4.4BSD.

If you do not wish to or cannot obtain *gated*, the distribution does include *routed*(8) to maintain the system routing tables. The routing daemon uses a variant of the Xerox Routing Information Protocol to maintain up to date routing tables in a cluster of local area networks. By using the `/etc/gateways` file, the routing daemon can also be used to initialize static routes to distant networks (see the next section for further discussion). When the routing daemon is started up (usually from `/etc/rc`) it reads `/etc/gateways` if it exists and installs those routes defined there, then broadcasts on each local network to which the host is attached to find other instances of the routing daemon. If any responses are received, the routing daemons cooperate in maintaining a globally consistent view of routing in the local environment. This view can be extended to include remote sites also running the routing daemon by setting up suitable entries in `/etc/gateways`; consult *routed*(8) for a more thorough discussion.

The second approach is to define a default or wildcard route to a smart gateway and depend on the gateway to provide ICMP routing redirect information to dynamically create a routing data base. This is done by adding an entry of the form

`/sbin/route add default` *smart-gateway* `1`

to `/etc/netstart`; see *route*(8) for more information. The default route will be used by the system as a ''last resort'' in routing packets to their destination. Assuming the gateway to which packets are directed is able to generate the proper routing redirect messages, the system will then add routing table entries based on the information supplied. This approach has certain advantages over the routing daemon, but is unsuitable in an environment where there are only bridges (i.e. pseudo gateways that, for instance, do not generate routing redirect messages). Further, if the smart gateway goes down there is no alternative, save manual alteration of the routing table entry, to maintaining service.

The system always listens, and processes, routing redirect information, so it is possible to combine both of the above facilities. For example, the routing table management process might be used to maintain up to date information about routes to geographically local networks, while employing the wildcard routing techniques for ''distant'' networks. The *netstat*(1) program may be used to display routing table contents as well as various routing oriented statistics. For example,

\# *netstat −r*

will display the contents of the routing tables, while

\# *netstat −r −s*

will show the number of routing table entries dynamically created as a result of routing redirect messages, etc.

5.5. Use of 4.4BSD machines as gateways

Several changes have been made in 4.4BSD in the area of gateway support (or packet forwarding, if one prefers). A new configuration option, GATEWAY, is used when configuring a machine to be used as a gateway. This option increases the size of the routing hash tables in the kernel. Unless configured with that option, hosts with only a single non-loopback interface never attempt to forward packets or to respond with ICMP error messages to misdirected packets. This change reduces the problems that may occur when different hosts on a network disagree on the network number or broadcast address. Another change is that 4.4BSD machines that forward packets back through the same interface on which they arrived will send ICMP redirects to the source host if it is on the same network. This improves the interaction of 4.4BSD gateways with hosts that configure their routes via default gateways and redirects. The generation of redirects may be disabled with the configuration option IPSENDREDIRECTS=0 or while the system is running by using the command:

```
sysctl -w net.inet.ip.redirect=0
```

in environments where it may cause difficulties.

5.6. Network databases

Several data files are used by the network library routines and server programs. Most of these files are host independent and updated only rarely.

File	Manual reference	Use
`/etc/hosts`	*hosts* (5)	local host names
`/etc/networks`	*networks* (5)	network names
`/etc/services`	*services* (5)	list of known services
`/etc/protocols`	*protocols* (5)	protocol names
`/etc/hosts.equiv`	*rshd* (8)	list of ‘‘trusted’’ hosts
`/etc/netstart`	*rc* (8)	command script for initializing network
`/etc/rc`	*rc* (8)	command script for starting standard servers
`/etc/rc.local`	*rc* (8)	command script for starting local servers
`/etc/ftpusers`	*ftpd* (8)	list of ‘‘unwelcome’’ ftp users
`/etc/hosts.lpd`	*lpd* (8)	list of hosts allowed to access printers
`/etc/inetd.conf`	*inetd* (8)	list of servers started by *inetd*

The files distributed are set up for Internet hosts. Local networks and hosts should be added to describe the local configuration; the Berkeley entries may serve as examples (see also the section on `/etc/hosts`). Network numbers will have to be chosen for each Ethernet. For sites connected to the Internet, the normal channels should be used for allocation of network numbers (contact hostmaster@SRI-NIC.ARPA). For other sites, these could be chosen more or less arbitrarily, but it is generally better to request official numbers to avoid conversion if a connection to the Internet (or others on the Internet) is ever established.

5.6.1. Network servers

Most network servers are automatically started up at boot time by the command file `/etc/rc` or by the Internet daemon (see below). These include the following:

Program	Server	Started by
`/usr/sbin/syslogd`	error logging server	`/etc/rc`
`/usr/sbin/named`	Internet name server	`/etc/rc`
`/sbin/routed`	routing table management daemon	`/etc/rc`
`/usr/sbin/rwhod`	system status daemon	`/etc/rc`
`/usr/sbin/timed`	time synchronization daemon	`/etc/rc`
`/usr/sbin/sendmail`	SMTP server	`/etc/rc`
`/usr/libexec/rshd`	shell server	inetd
`/usr/libexec/rexecd`	exec server	inetd
`/usr/libexec/rlogind`	login server	inetd
`/usr/libexec/telnetd`	TELNET server	inetd
`/usr/libexec/ftpd`	FTP server	inetd
`/usr/libexec/fingerd`	Finger server	inetd
`/usr/libexec/tftpd`	TFTP server	inetd

Consult the manual pages and accompanying documentation (particularly for named and sendmail) for details about their operation.

The use of *routed* and *rwhod* is controlled by shell variables set in `/etc/netstart`. By default, *routed* is used, but *rwhod* is not; they are enabled by setting the variables *routedflags* and *rwhod* to strings other than ‘‘NO.’’ The value of *routedflags* provides host-specific options to *routed*. For example,

```
routedflags=-q
rwhod=NO
```

would run *routed -q* and would not run *rwhod*.

To have other network servers started as well, commands of the following sort should be placed in the site-dependent file `/etc/rc.local`.

```
if [ -f /usr/sbin/timed ]; then
    /usr/sbin/timed & echo -n ' timed'            >/dev/console
fi
```

5.6.2. Internet daemon

In 4.4BSD most of the servers for user-visible services are started up by a ‘‘super server’’, the Internet daemon. The Internet daemon, `/usr/sbin/inetd`, acts as a master server for programs specified in its configuration file, `/etc/inetd.conf`, listening for service requests for these servers, and starting up the appropriate program whenever a request is received. The configuration file contains lines containing a service name (as found in `/etc/services`), the type of socket the server expects (e.g. stream or dgram), the protocol to be used with the socket (as found in `/etc/protocols`), whether to wait for each server to complete before starting up another, the user name by which the server should run, the server program’s name, and at most five arguments to pass to the server program. Some trivial services are implemented internally in *inetd*, and their servers are listed as ‘‘internal.’’ For example, an entry for the file transfer protocol server would appear as

```
ftp stream   tcp nowait   root/usr/libexec/ftpd    ftpd
```

Consult *inetd*(8) for more detail on the format of the configuration file and the operation of the Internet daemon.

5.6.3. The `/etc/hosts.equiv` file

The remote login and shell servers use an authentication scheme based on trusted hosts. The `hosts.equiv` file contains a list of hosts that are considered trusted and, under a single administrative control. When a user contacts a remote login or shell server requesting service, the client process passes the user’s name and the official name of the host on which the client is located. In the simple case, if the host’s name is located in `hosts.equiv` and the user has an account on the server’s machine, then service is rendered (i.e. the user is allowed to log in, or the command is executed). Users may expand this ‘‘equivalence’’ of machines by installing a `.rhosts` file in their login directory. The root login is handled specially, bypassing the `hosts.equiv` file, and using only the `/.rhosts` file.

Thus, to create a class of equivalent machines, the `hosts.equiv` file should contain the *official* names for those machines. If you are running the name server, you may omit the domain part of the host name for machines in your local domain. For example, four machines on our local network are considered trusted, so the `hosts.equiv` file is of the form:

```
vangogh.CS.Berkeley.EDU
picasso.CS.Berkeley.EDU
okeeffe.CS.Berkeley.EDU
```

5.6.4. The `/etc/ftpusers` file

The FTP server included in the system provides support for an anonymous FTP account. Because of the inherent security problems with such a facility you should read this section carefully if you consider providing such a service.

An anonymous account is enabled by creating a user *ftp*. When a client uses the anonymous account a *chroot*(2) system call is performed by the server to restrict the client from moving outside that part of the filesystem where the user ftp home directory is located. Because a *chroot* call is used, certain programs and files used by the server process must be placed in the ftp home directory. Further, one must be sure that all directories and executable images are unwritable. The following directory setup is recommended. The use of the *awk* commands to copy the `/etc/passwd` and `/etc/group` files are **STRONGLY** recommended.

```
# cd ˜ftp
# chmod 555 .; chown ftp .; chgrp ftp .
# mkdir bin etc pub
# chown root bin etc
# chmod 555 bin etc
# chown ftp pub
# chmod 777 pub
# cd bin
# cp /bin/sh /bin/ls .
# chmod 111 sh ls
# cd ../etc
# awk -F: '{$2="*";print$1":"$2":"$3":"$4":"$5":"$6":"}' < /etc/passwd > passwd
# awk -F: '{$2="*";print$1":"$2":"}' < /etc/group > group
# chmod 444 passwd group
```

When local users wish to place files in the anonymous area, they must be placed in a subdirectory. In the setup here, the directory `˜ftp/pub` is used.

Aside from the problems of directory modes and such, the ftp server may provide a loophole for interlopers if certain user accounts are allowed. The file `/etc/ftpusers` is checked on each connection. If the requested user name is located in the file, the request for service is denied. This file normally has the following names on our systems.

```
uucp
root
```

Accounts without passwords need not be listed in this file as the ftp server will refuse service to these users. Accounts with nonstandard shells (any not listed in `/etc/shells`) will also be denied access via ftp.

6. System operation

This section describes procedures used to operate a 4.4BSD UNIX system. Procedures described here are used periodically, to reboot the system, analyze error messages from devices, do disk backups, monitor system performance, recompile system software and control local changes.

6.1. 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 ˆC (interrupt). This will leave the system in single-user mode, with only the console terminal active. (If the console has been marked ‘‘insecure’’ in `/etc/ttys` you must enter the root password to bring the machine to single-user mode.) It is also possible to allow the filesystem checks to complete and then to return to single-user mode by signaling *fsck*(8) with a QUIT signal (ˆ\).

To bring the system up to a multi-user configuration from the single-user status, all you have to do is hit ˆD on the console. The system will then execute `/etc/rc`, a multi-user restart script (and `/etc/rc.local`), and come up on the terminals listed as active in the file `/etc/ttys`. See *init*(8) and *ttys*(5) Note, however, that this does not cause a filesystem check to be done. Unless the system was taken down cleanly, you should run ‘‘fsck −p’’ or force a reboot with *reboot*(8) to have the disks checked.

To take the system down to a single user state you can use

kill 1

or use the *shutdown*(8) command (which is much more polite, if there are other users logged in) when you are running multi-user. Either command will kill all processes and give you a shell on the console, as if you had just booted. Filesystems remain mounted after the system is taken single-user. If you wish to come up multi-user again, you should do this by:

cd /
/sbin/umount -a
ˆD

Each system shutdown, crash, processor halt and reboot is recorded in the system log with its cause.

6.2. Device errors and diagnostics

When serious errors occur on peripherals or in the system, the system prints a warning diagnostic on the console. These messages are collected by the system error logging process *syslogd*(8) and written into a system error log file `/var/log/messages`. Less serious errors are sent directly to *syslogd*, which may log them on the console. The error priorities that are logged and the locations to which they are logged are controlled by `/etc/syslog.conf`. See *syslogd*(8) for further details.

Error messages printed by the devices in the system are described with the drivers for the devices in section 4 of the programmer's manual. If errors occur suggesting hardware problems, you should contact your hardware support group or field service. It is a good idea to examine the error log file regularly (e.g. with the command *tail −r /var/log/messages*).

6.3. Filesystem 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 filesystems should be checked for consistency by *fsck*(1). The procedures of *reboot*(8) should be used to get the system to a state where a filesystem check can be done manually or automatically.

Dumping of the filesystems should be done regularly, since once the system is going it is easy to become complacent. Complete and incremental dumps are easily done with *dump*(8). You should arrange to do a towers-of-hanoi dump sequence; we tune ours so that almost all files are dumped on two tapes and kept for at least a week in most every case. We take full dumps every month (and keep these indefinitely). Operators can execute ‘‘dump w’’ at login that will tell them what needs to be dumped (based on the `/etc/fstab` information). Be sure to create a group **operator** in the file `/etc/group` so that dump can notify logged-in operators when it needs help.

More precisely, we have three sets of dump tapes: 10 daily tapes, 5 weekly sets of 2 tapes, and fresh sets of three tapes monthly. We do daily dumps circularly on the daily tapes with sequence ‘3 2 5 4 7 6 9 8 9 9 9 ...’. Each weekly is a level 1 and the daily dump sequence level restarts after each weekly dump. Full dumps are level 0 and the daily sequence restarts after each full dump also.

Thus a typical dump sequence would be:

tape name	level number	date	opr	size
FULL	0	Nov 24, 1992	operator	137K
D1	3	Nov 28, 1992	operator	29K
D2	2	Nov 29, 1992	operator	34K
D3	5	Nov 30, 1992	operator	19K
D4	4	Dec 1, 1992	operator	22K
W1	1	Dec 2, 1992	operator	40K
D5	3	Dec 4, 1992	operator	15K
D6	2	Dec 5, 1992	operator	25K

D7	5	Dec 6, 1992	operator	15K
D8	4	Dec 7, 1992	operator	19K
W2	1	Dec 9, 1992	operator	118K
D9	3	Dec 11, 1992	operator	15K
D10	2	Dec 12, 1992	operator	26K
D1	5	Dec 15, 1992	operator	14K
W3	1	Dec 17, 1992	operator	71K
D2	3	Dec 18, 1992	operator	13K
FULL	0	Dec 22, 1992	operator	135K

We do weekly dumps often enough that daily dumps always fit on one tape.

Dumping of 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. Finally if there are enough drives entire disks can be copied with *dd*(1) using the raw special files and an appropriate blocking factor; the number of sectors per track is usually a good value to use, consult `/etc/disktab`.

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

Exhaustion of user-file space is certain to occur now and then; disk quotas may be imposed, or if you prefer a less fascist approach, try using the programs *du*(1), *df*(1), and *quot*(8), combined with threatening messages of the day, and personal letters.

6.4. Moving filesystem data

If you have the resources, the best way to move a filesystem is to dump it to a spare disk partition, or magtape, using *dump*(8), use *newfs*(8) to create the new filesystem, and restore the filesystem using *restore*(8). Filesystems may also be moved by piping the output of *dump* to *restore*. The *restore* program uses an ''in-place'' algorithm that allows filesystem dumps to be restored without concern for the original size of the filesystem. Further, portions of a filesystem may be selectively restored using a method similar to the tape archive program.

If you have to merge a filesystem into another, existing one, the best bet is to use *tar*(1). If you must shrink a filesystem, the best bet is to dump the original and restore it onto the new filesystem. If you are playing with the root filesystem and only have one drive, the procedure is more complicated. If the only drive is a Winchester disk, this procedure may not be used without overwriting the existing root or another partition. What you do is the following:

1. GET A SECOND PACK, OR USE ANOTHER DISK DRIVE!!!!
2. Dump the root filesystem to tape using *dump*(8).
3. Bring the system down.
4. Mount the new pack in the correct disk drive, if using removable media.
5. Load the distribution tape and install the new root filesystem as you did when first installing the system. Boot normally using the newly created disk filesystem.

Note that if you change the disk partition tables or add new disk drivers they should also be added to the standalone system in `/sys/<architecture>/stand`, and the default disk partition tables in `/etc/disktab` should be modified.

6.5. Monitoring system performance

The *systat* program provided with the system is designed to be an aid to monitoring systemwide activity. The default ''pigs'' mode shows a dynamic ''ps''. By running in the ''vmstat'' mode when the system is active you can judge the system activity in several dimensions: job distribution, virtual memory load, paging and swapping activity, device interrupts, and 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 20-30 tps in practice), and the user cpu utilization (us) should be high (above 50%).

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 demon 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 in the ‘‘vmstat’’ mode when the system is busy, 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) and per-device interrupt counts, or system call activity (sy). Cumulatively on one of our large machines we average about 60-200 context switches and interrupts per second and about 50-500 system calls per second.

If the system is heavily loaded, or if you have little memory for your load (2M 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 swap out. If you expect to be in a memory-poor environment for an extended period you might consider administratively limiting system load.

6.6. Recompiling and reinstalling system software

It is easy to regenerate either the entire system or a single utility, and it is a good idea to try rebuilding pieces of the system to build confidence in the procedures.

In general, there are six well-known targets supported by all the makefiles on the system:

all — This entry is the default target, the same as if no target is specified. This target builds the kernel, binary or library, as well as its associated manual pages. This target **does not** build the dependency files. Some of the utilities require that a *make depend* be done before a *make all* can succeed.

depend — Build the include file dependency file, ‘‘.depend’’, which is read by *make*. See *mkdep*(1) for further details.

install — Install the kernel, binary or library, as well as its associated manual pages. See *install*(1) for further details.

clean — Remove the kernel, binary or library, as well as any object files created when building it.

cleandir — The same as clean, except that the dependency files and formatted manual pages are removed as well.

obj — Build a shadow directory structure in the area referenced by `/usr/obj` and create a symbolic link in the current source directory to referenced it, named ‘‘obj’’. Once this shadow structure has been created, all the files created by *make* will live in the shadow structure, and `/usr/src` may be mounted read-only by multiple machines. Doing a *make obj* in `/usr/src` will build the shadow directory structure for everything on the system except for the contributed, old, and kernel software.

The system consists of three major parts: the kernel itself, found in `/usr/src/sys`, the libraries , found in `/usr/src/lib`, and the user programs (the rest of `/usr/src`).

Deprecated software, found in `/usr/src/old`, often has old style makefiles; some of it does not compile under 4.4BSD at all.

Contributed software, found in `/usr/src/contrib`, usually does not support the ‘‘cleandir’’, ‘‘depend’’, or ‘‘obj’’ targets.

The kernel does not support the ‘‘obj’’ shadow structure. All kernels are compiled in subdirectories of `/usr/src/sys/compile` which is usually abbreviated as `/sys/compile`. If you want to mount your source tree read-only, `/usr/src/sys/compile` will have to be on a separate filesystem from `/usr/src`. Separation from `/usr/src` can be done by making `/usr/src/sys/compile` a symbolic link that references `/usr/obj/sys/compile`. If it is a symbolic link, the *S* variable in the kernel Makefile must be changed from `../..` to the absolute pathname needed to locate the kernel sources, usually `/usr/src/sys`. The symbolic link created by *config*(8) for `machine` must also be manually changed to an absolute pathname. Finally, the `/usr/src/sys/libkern/obj` directory must be located in `/usr/obj/sys/libkern`.

Each of the standard utilities and libraries may be built and installed by changing directories into the correct location and doing:

> # *make*
> # *make install*

Note, if system include files have changed between compiles, *make* will not do the correct dependency checks if the dependency files have not been built using the ‘‘depend’’ target.

The entire library and utility suite for the system may be recompiled from scratch by changing directory to `/usr/src` and doing:

> # *make build*

This target installs the system include files, cleans the source tree, builds and installs the libraries, and builds and installs the system utilities.

To recompile a specific program, first determine where the binary resides with the *whereis*(1) command, then change to the corresponding source directory and build it with the Makefile in the directory. For instance, to recompile ‘‘passwd’’, all one has to do is:

> # *whereis passwd*
> **/usr/bin/passwd**
> # *cd /usr/src/usr.bin/passwd*
> # *make*
> # *make install*

this will compile and install the *passwd* utility.

If you wish to recompile and install all programs into a particular target area you can override the default path prefix by doing:

> # *make*
> # *make DESTDIR*=pathname *install*

Similarly, the mode, owner, group, and other characteristics of the installed object can be modified by changing other default make variables. See *make*(1), `/usr/src/share/mk/bsd.README`, and the ‘‘.mk’’ scripts in the `/usr/share/mk` directory for more information.

If you modify the C library or system include files, to change a system call for example, and want to rebuild and install everything, you have to be a little careful. You must ensure that the include files are installed before anything is compiled, and that the libraries are installed before the remainder of the source, otherwise the loaded images will not contain the new routine from the library. If include files have been modified, the following commands should be done first:

> # *cd /usr/src/include*
> # *make install*

Then, if, for example, C library files have been modified, the following commands should be executed:

> # *cd /usr/src/lib/libc*
> # *make depend*
> # *make*

```
# make install
      # cd /usr/src
      # make depend
      # make
      # make install
```

Alternatively, the *make build* command described above will accomplish the same tasks. This takes several hours on a reasonably configured machine.

6.7. Making local modifications

The source for locally written commands is normally stored in `/usr/src/local`, and their binaries are kept in `/usr/local/bin`. This isolation of local binaries allows `/usr/bin`, and `/bin` to correspond to the distribution tape (and to the manuals that people can buy). People using local commands should be made aware that they are not in the base manual. Manual pages for local commands should be installed in `/usr/local/man/cat[1-8].` The *man*(1) command automatically finds manual pages placed in /usr/local/man/cat[1-8] to encourage this practice (see *man.conf*(5)).

6.8. Accounting

UNIX optionally records two kinds of accounting information: connect time accounting and process resource accounting. The connect time accounting information is stored in the file `/var/log/wtmp`, which is summarized by the program *ac*(8). The process time accounting information is stored in the file `/var/account/acct` after it is enabled by *accton*(8), and is analyzed and summarized by the program *sa*(8).

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

6.9. Resource control

Resource control in the current version of UNIX is more elaborate than in most UNIX systems. The disk quota facilities developed at the University of Melbourne have been incorporated in the system and allow control over the number of files and amount of disk space each user and/or group may use on each filesystem. In addition, the resources consumed by any single process can be limited by the mechanisms of *setrlimit*(2). As distributed, the latter mechanism is voluntary, though sites may choose to modify the login mechanism to impose limits not covered with disk quotas.

To use the disk quota facilities, the system must be configured with ‘‘options QUOTA’’. Filesystems may then be placed under the quota mechanism by creating a null file `quota.user` and/or `quota.group` at the root of the filesystem, running *quotacheck*(8), and modifying `/etc/fstab` to show that the filesystem is to run with disk quotas (options userquota and/or groupquota). The *quotaon*(8) program may then be run to enable quotas.

Individual quotas are applied by using the quota editor *edquota*(8). Users may view their quotas (but not those of other users) with the *quota*(1) program. The *repquota*(8) program may be used to summarize the quotas and current space usage on a particular filesystem or filesystems.

Quotas are enforced with *soft* and *hard* limits. When a user and/or group first reaches a soft limit on a resource, a message is generated on their terminal. If the user and/or group fails to lower the resource usage below the soft limit for longer than the time limit established for that filesystem (default seven days) the system then treats the soft limit as a *hard* limit and disallows any allocations until enough space is reclaimed to bring the user and/or group back below the soft limit. Hard limits are enforced strictly resulting in errors when a user and/or group tries to create or write a file. Each time a hard limit is exceeded the system will generate a message on the user’s terminal.

Consult the auxiliary document, ‘‘Disc Quotas in a UNIX Environment’’ (SMM:4) and the appropriate manual entries for more information.

6.10. 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 power cable can result in severely deteriorated service. The *netstat*(1) program may be of aid in tracking down hardware malfunctions. In particular, look at the −**i** and −**s** options in the manual page.

Should you believe a communication protocol problem exists, consult the protocol specifications and attempt to isolate the problem in a packet trace. The SO_DEBUG option may be supplied before establishing a connection on a socket, in which case the system will trace all traffic and internal actions (such as timers expiring) in a circular trace buffer. This buffer may then be printed out with the *trpt*(8) program. Most of the servers distributed with the system accept a −**d** option forcing all sockets to be created with debugging turned on. Consult the appropriate manual pages for more information.

6.11. Files that 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/disktab`	default disk partition sizes/labels
`/etc/printcap`	printer database
`/etc/gettytab`	terminal type definitions
`/etc/remote`	names and phone numbers of remote machines for *tip*(1)
`/etc/group`	group memberships
`/etc/motd`	message of the day
`/etc/master.passwd`	password file; each account has a line
`/etc/rc.local`	local system restart script; runs reboot; starts daemons
`/etc/inetd.conf`	local internet servers
`/etc/hosts`	local host name database
`/etc/networks`	network name database
`/etc/services`	network services database
`/etc/hosts.equiv`	hosts under same administrative control
`/etc/syslog.conf`	error log configuration for *syslogd*(8)
`/etc/ttys`	enables/disables ports
`/etc/crontab`	commands that are run periodically
`/etc/crontab.local`	local commands that are run periodically
`/etc/aliases`	mail forwarding and distribution groups
`/var/account/acct`	raw process account data
`/var/log/messages`	system error log
`/var/log/wtmp`	login session accounting

Table of Contents

1.	Introduction	4
1.1.	Distribution format	4
1.2.	UNIX device naming	4
1.3.	UNIX devices: block and raw	5
2.	Bootstrap procedure	5
2.1.	Bootstrapping from the tape	5
2.2.	Booting the HP300	6
2.2.1.	Supported hardware	6
2.2.2.	Standalone device file naming	6
2.2.3.	The procedure	7
2.2.3.1.	Step 1: selecting and formatting a disk	7
2.2.3.2.	Step 2: copying the root filesystem from tape to disk	7
2.2.3.3.	Step 3: booting the root filesystem	8
2.2.3.4.	Step 4: (optional) restoring the root filesystem	9
2.2.3.5.	Step 5: placing labels on the disks	10
2.3.	Booting the SPARC	10
2.3.1.	Supported hardware	10
2.3.2.	Limitations	11
2.3.3.	The procedure	11
2.4.	Booting the DECstation	12
2.4.1.	Supported hardware	12
2.4.2.	The procedure	13
2.4.2.1.	Procedure A: copy root filesystem to disk	13
2.4.2.2.	Procedure B: bootstrap from tape	13
2.4.2.3.	Procedure C: bootstrap over the network	14
2.4.3.	Label disk and create the root filesystem	14
2.5.	Disk configuration	15
2.5.1.	Disk naming and divisions	15
2.5.2.	Layout considerations	16
2.5.3.	Filesystem parameters	16
2.5.4.	Implementing a layout	18
2.6.	Installing the rest of the system	19
2.7.	Additional conversion information	21
3.	Upgrading a 4.3BSD system	22
3.1.	Installation overview	22
3.2.	Files to save	23
3.3.	Installing 4.4BSD	24
3.4.	Merging your files from 4.3BSD into 4.4BSD	26
3.4.1.	Changes in the `/etc` directory	27
3.4.2.	Shadow password files	29
3.4.3.	The `/var` filesystem	30
3.5.	Bug fixes and changes between 4.3BSD and 4.4BSD	31
3.5.1.	Changes to the kernel	31

3.5.2.	Security	32
3.5.2.1.	Virtual memory changes	32
3.5.2.2.	Networking additions and changes	32
3.5.2.3.	Additions and changes to filesystems	33
3.5.2.4.	POSIX terminal driver changes	35
3.5.2.5.	Native operating system compatibility	35
3.5.3.	Changes to the utilities	36
3.5.3.1.	Make and Makefiles	36
3.5.3.2.	Kerberos	36
3.5.3.3.	Timezone support	37
3.5.3.4.	Additions and changes to the libraries	37
3.5.3.5.	Additions and changes to other utilities	38
3.6.	Hints on converting from 4.3BSD to 4.4BSD	39
4.	System setup	40
4.1.	Kernel configuration	40
4.1.1.	Kernel organization	40
4.1.2.	Devices and device drivers	42
4.1.3.	Building new system images	42
4.2.	Configuring terminals	42
4.3.	Adding users	43
4.4.	Site tailoring	44
4.5.	Setting up the line printer system	44
4.6.	Setting up the mail system	45
4.6.1.	Setting up a UUCP connection	45
5.	Network setup	47
5.1.	System configuration	47
5.2.	Local subnets	48
5.3.	Internet broadcast addresses	48
5.4.	Routing	49
5.5.	Use of 4.4BSD machines as gateways	49
5.6.	Network databases	50
5.6.1.	Network servers	50
5.6.2.	Internet daemon	51
5.6.3.	The `/etc/hosts.equiv` file	51
5.6.4.	The `/etc/ftpusers` file	51
6.	System operation	52
6.1.	Bootstrap and shutdown procedures	52
6.2.	Device errors and diagnostics	53
6.3.	Filesystem checks, backups, and disaster recovery	53
6.4.	Moving filesystem data	54
6.5.	Monitoring system performance	54
6.6.	Recompiling and reinstalling system software	55
6.7.	Making local modifications	57
6.8.	Accounting	57
6.9.	Resource control	57
6.10.	Network troubleshooting	58
6.11.	Files that need periodic attention	58