Installing and Operating 4.3BSD-tahoe UNIX* on the VAX†
July 14, 1988

Michael J. Karels

Chris Torek

James M. Bloom

Marshall Kirk McKusick

Samuel J. Leffler

William N. Joy

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

ABSTRACT

This document contains instructions for the installation and operation of the 4.3BSD-tahoe release of the VAX UNIX system, as distributed by The University of California at Berkeley.

It discusses procedures for installing UNIX on a new VAX, and for upgrading an existing 4.2BSD or 4.3BSD VAX UNIX system to the new release. An explanation of how to lay out file systems on available disks, 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 networking facilities included with 4.3BSD-tahoe is included. Finally, the document details system operation procedures: shutdown and startup, hardware error reporting and diagnosis, file system backup procedures, resource control, performance monitoring, and procedures for recompiling and reinstalling system software.

*UNIX is a register trademark of AT&T in the USA and other countries.
† DEC, VAX, IDC, SBI, UNIBUS and MASSBUS are trademarks of Digital Equipment Corporation.

1. INTRODUCTION

This document explains how to install the 4.3BSD-tahoe release of the Berkeley version of UNIX for the VAX on your system. Because of the file system organization used in 4.3BSD-tahoe, if you are not currently running 4.2BSD or 4.3BSD you will have to do a full bootstrap from the distribution tape. The procedure for performing a full bootstrap is outlined in chapter 2. The process includes booting standalone utilities from tape to format a disk if necessary, then to copy a small root filesystem image onto a swap area. This filesystem is then booted and used to extract a dump of a standard root filesystem. Finally, that root filesystem is booted, and the remainder of the system binaries and sources are read from the archives on the tape(s).

The technique for upgrading a 4.2BSD or 4.3BSD system is described in chapter 3 of this document. As 4.3BSD-tahoe is upward-compatible with 4.2BSD, the upgrade procedure involves extracting a new set of system binaries onto new root and /usr filesystems. The sources are then extracted, and local configuration files are merged into the new system. 4.2BSD and 4.3BSD user filesystems may up upgraded in place, and 4.2BSD and 4.3BSD binaries may be used with 4.3BSD-tahoe in the course of the conversion. It is desirable to recompile most local software after the conversion, as there are many changes and performance improvements in the standard libraries.

1.1. Hardware supported

Note that some VAX models are identical to others in all respects except speed. The VAX 8650 will be hereafter referred to as a VAX 8600; likewise, the VAX 8250 will be referred to as a VAX 8200, the VAX-11/785 as an 11/780, and the 11/725 as an 11/730. These names are sometimes shortened to ‘‘8600,’’ ‘‘8200,’’ ‘‘780,’’ ‘‘750,’’ and ‘‘730,’’ and the MicroVAX II is sometimes called the ‘‘630.’’

This distribution can be booted on a VAX 8600, VAX 8200, VAX-11/780, VAX-11/750, VAX-11/730, or MicroVAX II cpu with at least 2 megabytes of memory, and any of the following disks:

DEC MASSBUS:	RM03, RM05, RM80, RP06, RP07
EMULEX MASSBUS:	AMPEX Capricorn, 9300, CDC 9766, 9775, FUJITSU 2351 Eagle, 2361†
DEC UNIBUS:	RK07, RL02, RAxx†, RC25
EMULEX SC-21V, SC-31 UNIBUS†:	AMPEX DM980, Capricorn, 9300, CDC 9762, 9766, FUJITSU 160M, 330M
EMULEX SC-31 UNIBUS†:	FUJITSU 2351 Eagle
DEC IDC:	R80, RL02
DEC BI:	RAxx†
DEC QBUS:	RD53, RD54, RAxx†

The tape drives supported by this distribution are:

DEC MASSBUS:	TE16, TU45, TU77, TU78
EMULEX MASSBUS:	TC-7000
DEC UNIBUS:	TS11, TU80, TU81†
EMULEX TC-11, AVIV UNIBUS:	KENNEDY 9300, STC, CIPHER
TU45 UNIBUS:	SI 9700
DEC QBUS:	TK50‡

† Other compatible UNIBUS controllers and drives may be easily usable with the system, but may require minor modifications to the system to allow bootstrapping. The EMULEX disk and SI tape controllers, and the drives shown here are known to work as bootstrap devices. RAxx includes the RA60, RA70, RA80, RA81, and RA82, as well as the RX50 floppy drives on the MicroVAX II. Other SMD and MSCP drives can be added with minor or no modifications.

† The TU81 support is untested but is identical to the TK50 code.

The tapes and disks may be on any available UNIBUS or MASSBUS adapter at any slot.

This distribution does not support the DEC CI780 or the HSC50 disk controller. As such, this distribution will not boot on the standard VAX 8600 cluster configurations. You will need to configure your system to use only UNIBUS, MASSBUS, and BI bus disk and tape devices. In addition, BI Ethernet, tape, and terminal controllers are unsupported. You cannot boot this distribution on a VAX 8200 without a UNIBUS.

1.2. Distribution format

The basic distribution contains the following items:

(3) 1600bpi 9-track 2400' magnetic tapes, or
(1) 6250bpi 9-track 2400' magnetic tape, and
(1) TU58 console cassette, and
(1) RX01 console floppy disk.

Installation on any machine requires a tape unit. Since certain standard VAX packages do not include a tape drive, this means one must either borrow one from another VAX system or one must be purchased separately. The console media distributed with the system are not suitable for use as the standard console media, as they do not contain microcode needed upon power-up; their intended use is only for installation.

If you have the facilities, it is a good idea to copy the magnetic tape(s) in the distribution kit to guard against disaster. The tapes contain some 512-byte records followed by many 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 contains preliminary bootstrapping programs. This is followed by a binary image of a 2 megabyte ''mini root'' file system. Following the mini root file is a full dump of the root file system (see *dump* (8)*). Additional files on the tape(s) contain tape archive images (see *tar* (1)). See Appendix A for a description of the contents and format of the tape(s). One file contains software contributed by the user community; refer to the accompanying documentation for a description of its contents and an explanation of how it should be installed.

1.3. VAX hardware terminology

This section gives a short discussion of VAX hardware terminology to help you get your bearings.

If you have MASSBUS disks and tapes it is necessary to know the MASSBUS that they are attached to, at least for the purposes of bootstrapping and system description. The MASSBUSes can have up to 8 devices attached to them. A disk counts as a device. A tape *formatter* counts as a device, and several tape drives may be attached to a formatter. If you have a separate MASSBUS adapter for a disk and one for a tape then it is conventional to put the disk as unit 0 on the MASSBUS with the lowest ''TR'' number, and the tape formatter as unit 0 on the next MASSBUS. On a 11/780 this would correspond to having the disk on ''mba0'' at ''tr8'' and the tape on ''mba1'' at ''tr9''. Here the MASSBUS adapter with the lowest TR number has been called ''mba0'' and the one with the next lowest number is called ''mba1''.

To find out the MASSBUS that your tape and disk are on you can examine the cabling and the unit numbers or your site maintenance guide. Do not be fooled into thinking that the number on the front of the tape drive is a device number; it is a *slave* number, one of several possible tapes on the single tape formatter. For bootstrapping, the slave number **must** be 0. The formatter unit number may be anything distinct from the other numbers on the same MASSBUS, but you must know what it is.

The MASSBUS devices are known by several different names by DEC software and by UNIX. At various times it is necessary to know both names. There is, of course, the name of the device like ''RM03'' or ''RM80''; these are easy to remember because they are printed on the front of the device. DEC also names devices based on the driver name in the system using a convention that reflects the interconnect topology of the machine. The first letter of such a name is a ''D'' for a disk, the second letter

‡ No TK50 media are included in the distribution, hence a machine with only a TK50 must already be running some version of UNIX that can be used to load the software over a network.
* References of the form X(Y) mean the subsection named X in section Y of the UNIX programmer's manual.

depends on the type of the drive, ‘‘DR’’ for RM03, RM05, and RM80’s, ‘‘DB’’ for RP06’s. The next letter is related to the interconnect; DEC calls the first MASSBUS or UNIBUS adapter ‘‘A’’, the second ‘‘B’’, etc. Thus, ‘‘DRA’’ is an RM drive on the first MASSBUS adapter. Finally, the name ends in a digit corresponding to the unit number for the device on the MASSBUS: e.g., ‘‘DRA0’’ is a disk at the first device slot on the first MASSBUS adapter and is an RM disk.

1.4. UNIX device naming

UNIX has a set of names for devices which are different from the DEC software names for the devices. The following table lists both the DEC and UNIX names for the supported devices:

Hardware	UNIX	DEC
RM disks	hp	DR
RP disks	hp	DB
MASSBUS TE/TU tapes	ht	MT
TU78 tape	mt	MF
RK disks	hk	DM
RL disks	rl	DL
TS tapes	ts	MS
UDA disks	ra	DU
RC25 disks	ra	DU
IDC disks	rb	DQ
UNIBUS SMD disks	up	
TM tapes	tm	
TMSCP tapes	tms	MU
UNIBUS TU tapes	ut	
BI KDB disks	kra	DU

Here UNIBUS SMD disks are disks on an RM-emulating controller on the UNIBUS, and TM tapes are tapes on a controller that emulates the DEC TM11. UNIBUS TU tapes are tapes on a UNIBUS controller that emulates the DEC TU45. IDC disks are disks on an 11/730 Integral Disk Controller. TS tapes are tapes on a controller compatible with the DEC TS11 (e.g. a TU80). TMSCP tapes include the TU81 and TK50.

The normal standalone system, used to bootstrap the full UNIX system, uses device names:

xx(a,c,d,p)

where *xx* is any of the UNIX device names in the table above. The parameters *a*, *c*, and *d* are the *adapter*, *controller*, and *drive* numbers respectively. The adapter is the index number of the MASSBUS or UNIBUS (with the first one found as number 0, and others numbered sequentially as found). The controller (or ‘‘device’’) number is the index number of the controller on that adapter. On the MASSBUS, the controller number is ignored for disk, and is used for the formatter number for tape. The drive number is the index of the disk or tape drive on that controller or formatter. The *p* value is interpreted differently for tapes and disks: for disks it is a disk *partition* (in the range 0-7); for tapes it is a file number on the tape. Here ‘‘file’’ means a tape file containing a single data stream terminated by a tape mark.* For example, partition 7 of drive 2 on an RA81 connected to the only UDA50 on UNIBUS 1 would be ‘‘ra(1,0,2,7)’’. Normally, the adapter and controller will both be 0; they may therefore be omitted from the device specification, and most of the examples in this document do so. When not running standalone, this partition would normally be available as ‘‘/dev/ra2g’’. Here the prefix ‘‘/dev’’ is the name of the directory where all ‘‘special files’’ normally live, the ‘‘ra’’ serves the obvious purpose, the ‘‘2’’ identifies this as a partition of ra drive number ‘‘2,’’ and the ‘‘g’’ identifies this as the seventh partition.

* Note that while a tape file consists of a single data stream, the distribution tape(s) have data structures in these files. Although the tape(s) contain only a few tape files, they comprise several thousand UNIX files.

On the VAX 8200, the adapter numbering is controlled by the ordering of the nodes on the BI; the BI is probed from low node numbers towards high. Hence if there are two KDB50 adapters, one at node 4, and one at node 7, the one at node 4 is kdb0, and the one at node 7 is kdb1. The numbering for UNIBUS adapters works similarly. Usually, the first UNIBUS on an 8200 is at node 0; you will need this node number to boot from tape. Other VAX models do not permit such chaotic ordering of adapters.

In all simple cases, where only a single controller is present, a drive with unit number 0 (in its unit plug on the front of the drive) 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 extremely easy.

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 the first sector of each disk with the *disklabel*(8) program; currently, this is possible on hp and ra disks, but not on the other types of disks on the VAX. Each partition may be used for either a raw data area such as a paging area or to store a UNIX file system. It is conventional for the first partition on a disk to be used to store a root file system, 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 file systems’’ by use of one or more additional partitions.

The third logical partition of each physical disk also has a conventional usage on the VAX: it allows access to the entire physical device, in many cases including bad sector forwarding information recorded at the end of the disk (one track plus 126 sectors). It is occasionally used to store a single large file system or to access the entire pack when making a copy of it on another. Care must be taken if using this partition not to overwrite the last few tracks and thereby clobber the bad sector information. Note that the sector containing the disk label is normally write-protected so that it is not accidentally overwritten. Pack-to-pack copies should normally skip the first 16 sectors of a pack, which contain the label and the initial bootstrap for some processors.

1.5. 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/xx0a’’, etc are block devices. There are also raw devices available. These have names like ‘‘/dev/rxx0a’’, 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 file systems, or when operating on a mounted filesystem such as the root.

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

2. BOOTSTRAP PROCEDURE

Note: The 4.3BSD-tahoe release contains only Tahoe filesystems and executable images. The procedures in this section cannot be used on the VAX with the 4.3BSD-tahoe distribution tape supplied by Berkeley. However, it is possible to make a boot tape that can be used in this way by extracting the sources in the distribution tape on a VAX, compiling, and making a tape using the procedures described in Appendix A. If you are not currently running 4.2BSD or 4.3BSD you will have to do a full bootstrap using a 4.3BSD tape; to install the 4.3BSD-tahoe release, the new sources must then be loaded and compiled. Chapter 3 describes how to upgrade an existing 4.2BSD or 4.3BSD system. programs. An understanding of the operations used in a full bootstrap is very helpful in performing an upgrade as well. In either case, it is highly desirable to read and understand the remainder of this document before proceeding.

2.1. Converting pre-4.2BSD Systems

The file system format was changed between 3BSD and 4.0BSD, and again between 4.1BSD and 4.2BSD. At a minimum you will have to dump any old file systems, and then restore them onto the 4.3BSD-tahoe file system. Sites running 3BSD or 32/V may be able to modify the *restore* program to understand the old 512 byte block file system, but this has never been tried. The dump format used in 4.0BSD and 4.1BSD is backward-compatible with that used in 4.3BSD-tahoe (which is unchanged from 4.2BSD). That is, the 4.3BSD-tahoe *restore* program understands how to read 4.0BSD and 4.1BSD dump tapes, although 4.3BSD-tahoe dump tapes cannot be restored under 4.0BSD or 4.1BSD. It is also desirable to make a convenient copy of system configuration files for use as guides when setting up the new system; the list of files to save from 4.2BSD systems in chapter 3 may be used as a guideline.

The first step is to dump your file systems with *dump* (8). For the utmost of safety this should be done to magtape. However, if you enjoy gambling with your life (or you have a VERY friendly user community) and you have enough disk space, you can try converting your file systems while copying to a new disk partition by piping the output of *dump* directly into *restore* after bringing up 4.3BSD-tahoe. If you select the latter tack, a version of the 4.1BSD dump program that runs under 4.3BSD-tahoe is provided in */etc/dump.4.1*. Beware that file systems created under 4.3BSD-tahoe can use about 5-10% more disk space for file system related information than under 4.1BSD. Thus, before dumping each file system it is a good idea to remove any files that may be easily regenerated. Since all programs should be recompiled under the new system, your best bet is to remove any object files. File systems with at least 10% free space on them should restore into an equivalently sized 4.3BSD-tahoe file system without problem.

2.2. Booting from tape

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

1) Format a disk pack with the *format* program.
2) Copy a ‘‘mini root’’ file system from the tape onto the swap area of the disk.
3) Boot the UNIX system on the ‘‘mini root’’.
4) Restore the full root file system using *restore* (8).
5) Build a console floppy, cassette, or RL02 pack for bootstrapping.
6) Reboot the completed root file system.
7) Label the disks with the *disklabel*(8) program.
8) Build and restore the /usr file system from tape with *tar* (1).
9) Extract the system and utility files and contributed software as desired.

Certain of these steps are dependent on your hardware configuration. Formatting the disk pack used for the root file system may require using the DEC standard formatting programs. Also, if you are bootstrapping the system on an 11/750, no console cassette is created.

Bootstrapping an 8600 is a bit more difficult than bootstrapping the other machines. The procedures for loading the toggle program and reading the tape bootstrap monitor described in Appendix B must be

used if you do not have access to a console RL02 pack with a UNIX bootstrap. Such a pack may be made on an 8600 already running UNIX, or on another 4.3BSD-tahoe system with an RL02 drive using the procedures in 4.1.1. One may be required to enter the toggle program more than once. After the bootstrap monitor is loaded, device addresses will be the same as if the machine were an 11/780. UNIBUS and MASSBUS adaptors are numbered from zero across both SBIA's (if present).

The following sections describe the above steps in detail. In these sections references to disk drives are of the form *xx*(*n,m*) and references to files on tape drives are of the form *yy*(*n,m*) where *xx* and *yy* are names described in section 1.4 and *n* and *m* are the unit and offset numbers described in section 1.4. Commands you are expected to type are shown in italics, while that information printed by the system is shown emboldened. Throughout the installation steps the reboot switch on a 780 or 730 should be set to off; on an 8600 or 750 set the power-on action to halt. (In normal operation a 780 or 730 will have the reboot switch on and an 8600 or 750 will have the power-on action set to reboot/restart.)

If you encounter problems while following the instructions in this part of the document, refer to Appendix C for help in troubleshooting.

2.2.1. Step 1: formatting the disk

All disks used with 4.3BSD-tahoe should be formatted to insure the proper handling of physically corrupted disk sectors. If you have DEC disk drives, you should use the standard DEC formatter to format your disks. If not, the *format* program included in the distribution, or a vendor supplied formatting program, may be used to format disks. The *format* program is capable of formatting any of the following supported distribution devices:

EMULEX MASSBUS:	AMPEX Capricorn, 9300, CDC 9766, 9775, FUJITSU 330M, 2351 Eagle
EMULEX SC-21V, SC-31 UNIBUS:	AMPEX 9300, Capricorn, CDC 9730, 9766, FUJITSU 160M, 330M
EMULEX SC-31 UNIBUS:	FUJITSU 2351 Eagle

If you have run a pre-4.1BSD version of UNIX on the packs you are planning to use for bootstrapping it is likely that the bad sector information on the packs has been destroyed, since it was accessible as normal data in the last several tracks of the disk. You should therefore run the formatter again to make sure the information is valid.

On an 11/750, to use a disk pack as a bootstrap device, sectors 0 through 15, the disk sectors in the file ''/boot'' (the program that loads the system image), and the file system indices that lead to this file must not have any errors. On an 8600, 11/780, or 11/730, the ''boot'' program is loaded from the console medium and includes device drivers for the ''hp'' and ''up'' disks that do ECC correction and bad sector forwarding; consequently, on these machines the system may be bootstrapped on these disks even if the disk is not error free in critical locations. In general, if the first 15884 sectors of your disk are clean you are safe; if not you can take your chances.

To load the *format* program, insert the distribution TU58 cassette or RX01 floppy disk in the appropriate console device (on the 11/730 use cassette 0) and do the following steps.

If you have an 8600 start the bootstrap monitor using the procedure described in Appendix B. Then give the command:

= format

If you have an 11/780 give the commands:

>>> HALT
>>> UNJAM
>>> INIT
>>> LOAD FORMAT
>>> START 2

If you have an 11/750 give the commands:

>>> I
>>> B DDA0
= format

If you have an 11/730 give the commands:

>>> H
>>> I
>>> L DD0:FORMAT
>>> S 2

The *format* program should now be running and awaiting your input:

Disk format/check utility

Enable debugging (1=bse, 2=ecc, 3=bse+ecc)?

If you made a mistake loading the program off the TU58 cassette or using the bootstrap monitor loaded for the 8600 the ‘‘=’’ prompt should reappear and you can retype the program name. If something else happened, you may have a bad distribution cassette or floppy, or your hardware may be broken; refer to Appendix C for help in troubleshooting. If you are unable to load programs off the distributed medium, consult Appendix B for an alternate (more painful) approach.

Format will create sector headers and verify the integrity of each sector formatted. Remember *format* runs only on the **up** and **hp** drives listed above. *Format* will prompt for the information required as shown below. Questions with default answers appear with the default in parentheses at the prompt; a carriage return will take the default. If you err in answering questions, ‘‘Delete’’ erases the last character typed, and ‘‘^U’’ erases the current input line.

Enable debugging (0=none, 1=bse, 2=ecc, 3=bse+ecc)? *0*
Device to format? *xx(0,0)*
...(the old bad sector table is read; ignore any errors that occur here)...
Formatting drive *xx*0 on adaptor 0: verify (yes/no)? *yes*
Device data: #cylinders=842, #tracks=20, #sectors=48
Starting cylinder (0): *(hit RETURN to accept the defaults)*
Starting track (0):
Ending cylinder (841):
Ending track (19):
Available test patterns are:
1 - (f00f) RH750 worst case
2 - (ec6d) media worst case
3 - (a5a5) alternating 1's and 0's
4 - (ffff) Severe burnin (up to 48 passes)
Pattern (one of the above, other to restart)? *2*
Maximum number of bit errors to allow for soft ECC (3):
Start formatting...make sure the drive is online
...(soft ecc's and other errors are reported as they occur)...
...(if 4 write check errors were found, the program terminates like this)...
Errors:
Bad sector: 0
Write check: 4
Hard ECC: 0
Other hard: 0
Marked bad: 0
Skipped: 0
Total of 4 hard errors revectored.
Writing bad sector table at block 524256
...(524256 is the block # of the first block in the bad sector table)...
Done

Once the root device has been formatted, *format* will prompt for another disk to format. Halt the machine by typing ''control-P'' and ''H'' (the ''H'' is necessary only on the 780 and 8600, but does not hurt on the other machines).

Enable debugging (1=bse, 2=ecc, 3=bse+ecc)? *^P*
>>> H

It may be necessary to format other drives before constructing file systems on them; this can be done at a later time with the steps just performed. *Format* can also be used in an extended test mode (pattern 4) that uses numerous test patterns in up to 48 passes to detect as many disk surface errors as possible; this test may be run for many hours, depending on the CPU and controller. On an 11/780, this can be sped up significantly by setting the clock fast. It may be run for some number of passes, then either terminated or continued according to the errors found to that point.

2.2.2. Step 2: copying the mini-root file system

The second step is to run a simple program, *copy*, which copies a small root file system into the second partition of the disk. This file system will serve as the base for creating the actual root file system to be restored. The version of the operating system maintained on the ''mini-root'' file system understands that it should not swap on top of itself, thereby allowing double use of the disk partition. *Copy* is loaded just as the *format* program was loaded; for example, on an 8600 or 8200, one must enter the toggle and the bootstrap monitor as described in Appendix B and then:

```
(copy mini root file system)
= copy
From: yy(y,1)                    (unit y, second tape file)
To: xx(x,1)                      (mini root is on drive x; second partition)
Copy completed: 205 records copied
From:
```

while for an 11/780:

```
(copy mini root file system)
>>> LOAD COPY
>>> START 2
From: yy(y,1)                    (unit y, second tape file)
To: xx(x,1)                      (mini root is on drive x; second partition)
Copy completed: 205 records copied
From:
```

or for an 11/750:

```
(copy mini root file system)
>>> B DDA0
= copy
From: yy(y,1)                    (unit y, second tape file)
To: xx(x,1)                      (mini root is on drive x; second partition)
Copy completed: 205 records copied
From:
```

and for an 11/730:

```
(copy mini root file system)
>>> L DD0:COPY
>>> S 2
From: yy(y,1)                    (unit y, second tape file)
To: xx(x,1)                      (mini root is on drive x; second partition)
Copy completed: 205 records copied
From:
```

(As above, ‘delete’ erases characters and ‘^U’ erases lines.)

2.2.3. Step 3: booting from the mini-root file system

You now have the minimal set of tools necessary to create a root file system and restore the file system contents from tape. To access this file system load the bootstrap program and boot the version of unix that has been placed in the ‘‘mini-root’’:

```
(follow the procedure in Appendix B to load the bootstrap monitor)

(load bootstrap program)
= boot
Boot
: xx(x,1)vmunix                  (bring in vmunix off mini root)
```

or, on an 11/780:

(load bootstrap program)
>>> BOOT ANY
Boot
: *xx(x,1)vmunix* (bring in *vmunix* off mini root)

or, on an 11/750:

(load bootstrap program)
>>> B DDA0
= boot
Boot
: *xx(x,1)vmunix* (bring in *vmunix* off mini root)

or, on an 11/730:

(load bootstrap program)
>>> L DD0:BOOT
>>> D RB 3
>>> S 2
Boot
: *xx(x,1)vmunix* (bring in *vmunix* off mini root)

(As above, 'delete' erases characters and 'ˆU' erases lines.)

The standalone boot program should then read the system from the mini root file system you just created, and the system should boot:

271944+78848+92812 start 0x12e8
4.3 BSD UNIX #1: Wed Apr 9 23:33:59 PST 1988
karels@monet.berkeley.edu:/usr/src/sys/GENERIC
real mem = *xxx*
avail mem = *yyy*
... information about available devices ...
root device?

The first three numbers are printed out by the bootstrap programs 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 16 megabytes of memory, *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 (dh's, dz's, etc.), or all the mass storage peripherals you have especially if you have more than two of anything. You will correct this soon, when you create a description of your machine from which to configure 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 and is now asking you for the name of the root file system to use. This happens because the distribution system is a *generic* system. It can be bootstrapped on any VAX cpu and with its root device and paging area on any available disk drive. You

should respond to the root device question with *xx0**. This response supplies two pieces of information: first, *xx0* shows that the disk it is running on is drive 0 of type *xx*, secondly the ‘‘*’’ shows that the system is running ‘‘atop’’ the paging area. The latter is most important, otherwise the system will attempt to page on top of itself and chaos will ensue. You will later build a system tailored to your configuration that will not ask this question when it is bootstrapped.

root device? *xx0**
WARNING: preposterous time in file system — CHECK AND RESET THE DATE!
erase ^?, kill ^U, intr ^C
#

The ‘‘erase ...’’ message is part of /.profile that was executed by the root shell when it started. This message is present to remind you that the line character erase, line erase, and interrupt characters are set to be what is standard on DEC systems; this insures that things are consistent with the DEC console interface characters.

2.2.4. Step 4: restoring the root file system

UNIX is now running, and the ‘UNIX Programmer’s manual’ applies. The ‘#’ is the prompt from the shell, and lets you know that you are the super-user, whose login name is ‘‘root’’. To complete installation of the bootstrap system two steps remain. First, the root file system must be created, and second a boot floppy or cassette must be constructed.

To create the root file system the shell script ‘‘xtr’’ should be run as follows:

 # disk=xx0 type=tt tape=yy xtr

where *xx0* is the name of the disk on which the root file system is to be restored (unit 0), *tt* is the type of drive on which the root file system is to be restored (see the table below), and *yy* is the name of the tape drive on which the distribution tape is mounted.

If the root file system is to reside on a disk other than unit 0 (as the information printed out during autoconfiguration shows), you will have to create the necessary special files in /dev and use the appropriate value. For example, if the root should be placed on hp1, you must create /dev/rhp1a and /dev/hp1a using the MAKEDEV script in /dev as follows:

 # cd /dev; MAKEDEV hp1

The following table lists the various drive *type*s.

Drive	Type	Drive	Type
DEC RM03	type=rm03	DEC RM05	type=rm05
DEC RM80	type=rm80	DEC RP06	type=rp06
DEC RP07	type=rp07	DEC RK07	type=rk07
DEC RA80	type=ra80	DEC RA60	type=ra60
DEC RA81	type=ra81	DEC R80	type=rb80
DEC RA70	type=ra70	DEC RA82	type=ra82
DEC RD53	type=rd53	DEC RD54	type=rd54
CDC 9766	type=9766	CDC 9775	type=9775
AMPEX 300M	type=9300	AMPEX 330M	type=capricorn
FUJITSU 160M	type=fuji160	FUJITSU 330M	type=capricorn
FUJITSU 404M	type=eagle		

This will generate many messages regarding the construction of the file system and the restoration of the tape contents, but should eventually stop with the messages:

```
...
Root filesystem extracted

If this is an 8600, update the console RL02
If this is an 8200, update the floppy
If this is a 780, update the floppy
If this is a 730, update the cassette
#
```

2.2.5. Step 5: creating a boot floppy or cassette

If the machine is an 8600, 8200, 11/780 or 11/730, a boot floppy, cassette, or console RL02 should be constructed according to the instructions in chapter 4. For 11/750's, bootstrapping is performed by using a boot prom and special code located in sectors 0-15 of the root file system. The *disklabel* program installs the needed code. Locate the disk name and type from the table in step 7, then run the following command:

```
# disklabel -rw ${disk}0 $type "optional_pack_name"
```

On an 11/780 with old-style (MS780C) interleaved memory, or other configurations that require alteration of the standard boot files, this step may be left for later.

2.2.6. Step 6: rebooting the completed root file system

With the above work completed, all that is left is to reboot:

```
# sync                                   (synchronize file system state)
# ^P                                     (halt machine)
>>> HALT                                 (for 11/780's)
>>> UNJAM                                (for 8600's or 11/780's only)
>>> I                                    (initialize processor state)
>>> B xxS                                (on an 11/750, use B/2; see below for 8200)
...(boot program is eventually loaded)...
Boot
: xx(x,0)vmunix                          (vmunix brought in off root)
271944+78848+92812 start 0x12e8
4.3 BSD UNIX #1: Wed Apr  9 23:33:59 PST 1988
   karels@monet.berkeley.edu:/usr/src/sys/GENERIC
real mem  = xxx
avail mem = yyy
... information about available devices ...
root on xx0
WARNING: preposterous time in file system — CHECK AND RESET THE DATE!
erase ^?, kill ^U, intr ^C
#
```

On an 8200, or if the root device selected by the kernel is not correct, it is necessary to boot using the option to ask for the root device. On the 8200, use *B/R5:800* followed by *@ANYBOO.CMD*; on the 11/750, use *B/3*; on the other processors, use *BOOT ANY*. At the prompt from the bootstrap, use the same device specification above: *xx(x,0)vmunix*. Then, to the question ''root device?,'' respond with *xx*0. See section 6.1 and appendix C if the system does not reboot properly.

The system is now running single user on the installed root file system. The next section tells how to complete the installation of distributed software on the /usr file system.

2.2.7. Step 7: placing labels on the disks

First set up shell variables, so that the commands we give will work regardless of the disk you have. You might wish to review the disk configuration information in section 4.3 before continuing; the partitions used below are those most appropriate in size. Find the disk you have in the following table and execute the commands in the right hand portion of the table:

DEC RM03	*# disk=hp; name=hp0g; type=rm03*
DEC RM05	*# disk=hp; name=hp0g; type=rm05*
DEC RM80	*# disk=hp; name=hp0g; type=rm80*
DEC RP06	*# disk=hp; name=hp0g; type=rp06*
DEC RP07	*# disk=hp; name=hp0h; type=rp07*
DEC RK07	*# disk=hk; name=hk0g; type=rk07*
DEC RA60	*# disk=ra; name=ra0h; type=ra60*
DEC RA70	*# disk=ra; name=ra0h; type=ra70*
DEC RA80	*# disk=ra; name=ra0h; type=ra80*
DEC RA81	*# disk=ra; name=ra0h; type=ra81*
DEC RA82	*# disk=ra; name=ra0h; type=ra82*
DEC R80	*# disk=rb; name=rb0h; type=rb80*
UNIBUS CDC 9766	*# name=up0g; type=9766*
UNIBUS AMPEX 300M	*# disk=up; name=up0g; type=9300*
UNIBUS AMPEX 330M	*# disk=up; name=up0g; type=capricorn*
UNIBUS FUJITSU 160M	*# disk=up; name=up0g; type=fuji160*
UNIBUS FUJITSU 330M	*# disk=up; name=up0g; type=capricorn*
UNIBUS FUJITSU 404M	*# disk=up; name=up0h; type=eagle*
MASSBUS CDC 9766	*# disk=up; name=hp0g; type=9766*
MASSBUS AMPEX 300M	*# disk=up; name=hp0g; type=9300*
MASSBUS AMPEX 330M	*# disk=up; name=hp0g; type=capricorn*
MASSBUS FUJITSU 330M	*# disk=up; name=hp0g; type=capricorn*
MASSBUS FUJITSU 404M	*# disk=up; name=hp0h; type=eagle*

If you have a DEC RA disk, but it is on a KDB50, insert a ‘k’:

 # disk=k$disk; name=k$name

Next find the tape you have in the following table and execute the commands in the right hand portion of the table:

DEC TE16/TU45/TU77	*# cd /dev; MAKEDEV ht0; sync*
DEC TU78	*# cd /dev; MAKEDEV mt0; sync*
DEC TS11	*# cd /dev; MAKEDEV ts0; sync*
DEC TK50/TK70/TA80/TA81	*# cd /dev; MAKEDEV tmscp0; sync*
EMULEX TC11	*# cd /dev; MAKEDEV tm0; sync*
SI 9700	*# cd /dev; MAKEDEV ut0; sync*

On hp and ra disks (excluding those on the KDB50), 4.3BSD-tahoe uses disk labels in the first sector of each disk to contain information about the geometry of the drive and the partition layout. This information is written with *disklabel* (8). To label the disk containing the root file system, run the following command:

 # disklabel -rw ${disk}0 $type "optional_pack_name"

This sets up the default partition table. *Type* can be any name listed in /etc/disktab; if you want something other than the default tables, you can edit /etc/disktab and add a new name: e.g., ‘‘ra81-local.’’ Alternatively, you can use the *–e* option to edit the label; you will have to set the ‘‘EDITOR’’ environment variable to /bin/ed:

 # EDITOR=/bin/ed; export EDITOR

You should label all your disks as soon as possible, but you *must* label the root pack on a VAX-11/750, even if labels are not supported (e.g., on ‘‘up’’ disks), as this also creates the boot block. As a general rule, it is always safe to run *disklabel*: if labels are not supported on some disk, nothing of consequence will happen.

2.2.8. Step 8: setting up the /usr file system

The next thing to do is to extract the rest of the data from the tape:

# date yymmddhhmm	(set date, see *date* (1))
....	
# passwd root	(set password for super-user)
New password:	(password will not echo)
Retype new password:	
# hostname mysitename	(set your hostname)
# newfs ${name} ${type}	(create empty user file system)
(this takes a few minutes)	
# mount /dev/${name} /usr	(mount the usr file system)
# cd /usr	(make /usr the current directory)
# mt fsf	
# tar xpbf 20 /dev/rmt12	(extract all of usr except usr/src)
(this takes about 15-20 minutes)	

If the tape had been rewound or positioned incorrectly before the *tar*, it may be repositioned by the following commands.

 # mt rew
 # mt fsf 3

The data on the fourth tape file has now been extracted. If you are using 1600bpi tapes, the first reel of the distribution is no longer needed; the remainder of the installation procedure uses the second reel of tape that should be mounted in place of the first. The first instruction below is ignored if using 1600bpi tapes. The installation procedure continues from this point on the 6250bpi tape.

# mt fsf	(do not do on 1600bpi tapes)
# mkdir src	(make directory for source)
# mkdir src/sys	(make directory for system source)
# cd src/sys	(make /usr/sys the current directory)
# tar xpbf 20 /dev/rmt12	(extract the system source)
(this takes about 5-10 minutes)	
# cd /	(back to root)
# chmod 755 / /usr /usr/src /usr/src/sys	
# rm –f sys	
# ln –s usr/src/sys sys	(make a symbolic link to the system source)
# umount /dev/${name}	(unmount /usr)

You can check the consistency of the /usr file system by doing

 # fsck /dev/r${name}

The output from *fsck* should look something like:

**** /dev/r*xx*0h**
**** Last Mounted on /usr**
**** Phase 1 - Check Blocks and Sizes**
**** Phase 2 - Check Pathnames**
**** Phase 3 - Check Connectivity**
**** Phase 4 - Check Reference Counts**
**** Phase 5 - Check Cyl groups**
671 files, 3497 used, 137067 free (75 frags, 34248 blocks)

If there are inconsistencies in the file system, you may be prompted to apply corrective action; see the document describing *fsck* for information.

To use the /usr file system, you should now remount it by saying

 # /etc/mount /dev/${name} /usr

You can then extract the source code for the commands (except on RK07's and RM03's this will fit in the /usr file system):

 # cd /usr/src
 # mt fsf
 # tar xpb 20

If you get an error at this point, most likely it was a problem with tape positioning. You can reposition the tape by rewinding it and then skipping over the files already read (see *mt* (1)).

2.2.9. Additional software

There is one additional tape file on the distribution tape(s) which has not been installed to this point; it contains user contributed software in *tar* (1) format. On the 1600bpi tape set, this file is the sole file on the third tape. It can be installed by positioning the tape using *mt* (1) and reading in the files as was done for /usr/src above. As distributed, the user contributed software should be placed in /usr/src/new. It may be extracted by mounting the appropriate tape (if not already mounted), positioning the tape at the beginning of this file (for 6250bpi), and extracting with *tar* :

 # cd /usr/src
 # mkdir new
 # chmod 755 new
 # cd new
 # tar xpb 20

Several of the directories for large contributed software subsystems have been placed in a single archive file and compressed to allow

2.3. Additional conversion information

After setting up the new 4.3BSD-tahoe filesystems, you may restore any user files that were saved on tape before beginning the conversion. Note that the 4.3BSD-tahoe *restore* program does its work on a mounted file system using normal system operations (unlike the 4.1BSD *restor* that accessed the raw file system device and deposited inodes in the appropriate locations on disk). This means that file system dumps may be restored even if the characteristics of the file system changed. To restore a dump tape for, say, the /a file system something like the following would be used:

 # mkdir /a
 # disklabel -rw hp1 eagle
 # newfs hp1g
 # mount /dev/hp1g /a
 # cd /a
 # restore r

If you chose to convert 4.1BSD filesystems while copying to a new disk area, do so by piping the output of *dump.4.1* directly into *restore* after bringing up 4.3BSD-tahoe.

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

To convert a compiler from 4.1BSD to 4.3BSD-tahoe you should simply have to recompile and relink the various parts. If the processor is written in itself, for instance a PASCAL compiler written in PASCAL, the important step in converting is to save a working copy of the 4.1BSD binary before converting to 4.3BSD-tahoe. Then, once the system has been changed over, the 4.1BSD binary should be used in the rebuilding process. To do this, you should enable the 4.1 compatibility option when you configure the kernel (see section 4.3).

If no working 4.1BSD binary exists, or the language processor uses some nonstandard system call, you will likely have to compile the language processor into an intermediate form, such as assembly language, on a 4.1BSD system, then bring the intermediate form to 4.3BSD-tahoe for assembly and loading.

3. UPGRADING A 4.2BSD OR 4.3BSD SYSTEM

This section describes the procedure for upgrading a 4.2 or 4.3BSD system to 4.3BSD-tahoe. This procedure may vary according to the version of the system running before conversion. If you are upgrading from 4.2BSD, begin by reading the ''Bugs Fixes and Changes in 4.3BSD-tahoe'' document to see what has changed since the last time you bootstrapped the system. If you have local system modifications to the kernel to install, look at the document ''Changes to the Kernel in 4.3BSD-tahoe'' to get an idea of how the system changes will affect your local modifications.

If you are running 4.2BSD or 4.3BSD, upgrading your system involves replacing your kernel and system utilities. Binaries compiled under 4.3BSD will work without recompilation under 4.3BSD-tahoe, though they may run faster if they are recompiled. Binaries compiled under 4.2BSD will probably work without recompilation, but it is a good idea to recompile and relink because of the many changes in header files and libraries since 4.2BSD. 4.1BSD binary images can also run unchanged under 4.3BSD-tahoe but only when the system is configured to include the ''4.1BSD compatibility mode.''*

The easiest upgrade path from 4.2BSD or 4.3BSD (depending on your file system configuration) is to build new root and */usr* file systems on unused partitions, then copy or merge site specific files into their corresponding files on the new system. All user file systems can be retained unmodified, except that the new *fsck* should be run before they are mounted (see below).

Section 3.1 lists the files to be saved as part of the conversion process. Section 3.2 describes the bootstrap process. Section 3.3 discusses the merger of the saved files back into the new system. Section 3.4 provides general hints on possible problems to be aware of when converting from 4.2BSD to 4.3BSD-tahoe.

3.1. 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 file systems, you should create a *tar* image of the following files before the new file systems are created. In addition, it is **STRONGLY** advised that you do a full dump before rebuilding the file system to guard against missing something the first time around.

* With ''4.1BSD compatibility mode'' system calls from 4.1BSD are either emulated or safely ignored. There are only two exceptions: programs that read directories or use the old jobs library will not operate properly. However, while 4.1BSD binaries will execute under 4.3BSD-tahoe it is **STRONGLY RECOMMENDED** that the programs be recompiled under the new system.

/.cshrc	†	root csh startup script
/.login	†	root csh login script
/.profile	†	root sh startup script
/.rhosts	†	for trusted machines and users
/dev/MAKEDEV	‡	in case you added anything here
/dev/MAKEDEV.local	*	for making local devices
/etc/disktab	‡	in case you changed disk partition sizes
/etc/fstab	†	disk configuration data
/etc/ftpusers	†	for local additions
/etc/gateways	†	routing daemon database
/etc/gettytab	‡	getty database
/etc/group	*	group data base
/etc/hosts	†	for local host information
/etc/hosts.equiv	†	for local host equivalence information
/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/syslog.conf	*	system logger configuration
/etc/securettys	*	for restricted list of ttys where root can log in
/etc/ttys	*	terminal line configuration data
/etc/ttytype	*	terminal line to terminal type mapping data
/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/hosts/MAKEHOSTS	*	for local changes
/usr/include/*	‡	for local additions
/usr/lib/aliases	‡	mail forwarding data base
/usr/lib/crontab	*	cron daemon data base
/usr/lib/font/*	‡	for locally developed font libraries
/usr/lib/lib*.a	†	for locally libraries
/usr/lib/lint/*	‡	for locally developed lint libraries
/usr/lib/sendmail.cf	*	sendmail configuration
/usr/lib/tabset/*	‡	for locally developed tab setting files
/usr/lib/term/*	‡	for locally developed nroff drive tables
/usr/lib/tmac/*	‡	for locally developed troff/nroff macros
/usr/lib/uucp/*	†	for local uucp configuration files
/usr/man/manl	*	for manual pages for locally developed programs
/usr/msgs	†	for current msgs
/usr/spool/*	†	for current mail, news, uucp files, etc.
/usr/src/local	†	for source for locally developed programs
/sys/conf/HOST	†	configuration file for your machine
/sys/conf/files.HOST	†	list of special files in your kernel
/*/quotas	†	file system quota files

† Files that can be used from 4.2BSD or 4.3BSD without change.
‡ Files that need local modifications merged into 4.3BSD-tahoe files.
* Files that require special work to merge and are discussed
in section 3.3.

3.2. Installing 4.3BSD-tahoe

Note: The 4.3BSD-tahoe release contains only Tahoe filesystems and executable images. In order to bring up 4.3BSD-tahoe on a VAX, it is necessary to extract the sources on a VAX, compile and install. Most of the files listed above are found in /usr/src/sys/vaxdist as well as in their standard locations on the distribution tape so that the root and /usr images need not be extracted from the tape. The following sections describe the procedure for installing 4.3BSD-tahoe on the Tahoe. For a VAX system, the starting root and /usr filesystems can be created by building and installing executables using alternate filesystems. The next step is to build a working 4.3BSD-tahoe system. This can be done by following the steps in section 2 of this document for extracting the root and /usr file systems from the distribution tape onto unused disk partitions. If you have a running 4.2BSD or 4.3BSD system, you can also do this by using *dd* (1) to copy the ''mini root'' filesystem onto one disk partition, then use it to load the 4.3BSD-tahoe root filesystem as in chapter 2. The root filesystem dump on the tape could also be extracted directly, although this will require an additional file system check after booting 4.3BSD-tahoe to convert the new root filesystem. The exact procedure chosen will depend on the disk configuration and the number of suitable disk partitions that may be used. If there is insufficient space to load the new root and */usr* filesystems before reusing the existing partitions, it is **STRONGLY** advised that you make full dumps of each filesystem on magtape before beginning. It is also desirable to run file system checks of all filesystems to be converted to 4.3BSD-tahoe before shutting down. If you are running a system older than 4.2BSD, you will have to dump and restore your file systems; see section 2.1 for some hints. In either case, this is an excellent time to review your disk configuration for possible tuning of the layout. Section 4.2 and *config*(8) are required reading.

To ease the transition to new kernels, the 4.3BSD and 4.3BSD-tahoe bootstrap routines pass the identity of the boot device through to the kernel. The kernel then uses that device as its root file system. Thus, for example, if you boot from */dev/hp1a*, the kernel will use hp1a as its root file system. If */dev/hp1b* is configured as a swap partition, it will be used as the initial swap area, otherwise the normal primary swap area (*/dev/hp0b*) will be used. The 4.3BSD-tahoe bootstrap is backward compatible with 4.2BSD and 4.3BSD, so you can replace your old bootstrap if you use it to boot your first 4.3BSD-tahoe kernel.

Once you have extracted the 4.3BSD-tahoe 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.''

If converting from 4.2BSD, 4.3BSD, or the CCI 1.21 release, your old file systems must be converted. The standard disk partitions in 4.3BSD-tahoe are the same as those in 4.2BSD and 4.3BSD, except for those on the DEC UDA50; see section 4.3.2 for details. If you've modified the partition sizes from the original BSD or CCI ones, and are not already using the 4.3BSD-tahoe disk labels, you will have to modify the default disk partion tables in the kernel. Make the necessary table changes and boot your custom kernel **BEFORE** trying to access any of your old file systems! After doing this, if necessary, the remaining filesystems may be converted in place by running the 4.3BSD-tahoe 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 4.2BSD. Thus the first time that it is run on a 4.2BSD file system, 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 file systems, in which these fields were unused spares; this correction will produce messages of the form:

IMPOSSIBLE INTERLEAVE=0 IN SUPERBLOCK (SET TO DEFAULT)*
IMPOSSIBLE NPSECT=0 IN SUPERBLOCK (SET TO DEFAULT)

File systems that have had their interleave and npsect values set will be diagnosed by the old *fsck* as having

* The defaults are to set *interleave* to 1 and *npsect* to *nsect*; this is correct on many drives. Notable exceptions are the RM80 and RA81, where npsect should be set to one more than nsect. This affects only performance (and in the case of the RA81, at least, virtually unmeasurably).

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 4.3BSD-tahoe 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, 4.3BSD-tahoe removes several limits on file system sizes that were present in both 4.2BSD and 4.3BSD. The limited file systems continue to work in 4.3BSD-tahoe, but should be converted as soon as it is convenient by running *fsck* with the *−c* option. If no file systems have been so converted, the sequence *fsck −p −c* will update all of them, fix the interleave and npsect fields, and fix any incorrect directory lengths all at once. The new unlimited file system formats are treated as read-only by older systems. A second *fsck −c*, however, will reconvert the new format to the old if none of the static limits of the old file system format have been exceeded. The new file systems are otherwise compatible between 4.2BSD, 4.3BSD, and 4.3BSD-tahoe, though running a 4.3BSD-tahoe file system under older systems may cause more of the above messages to be generated the next time it is *fsck*'ed on 4.3BSD-tahoe.

3.3. Merging your files from 4.2 or 4.3BSD into 4.3BSD-tahoe

When your system is booting reliably and you have the 4.3BSD-tahoe root and /usr file systems 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 4.3BSD-tahoe version and carefully integrate any local changes into the new file. Usually these local modifications 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 below.

If you have any homegrown device drivers in /dev/MAKEDEV.local that use major device numbers reserved by the system you will have to modify the commands used to create the devices or alter the system device configuration tables in /sys/vax/conf.c. Otherwise /dev/MAKEDEV.local can be used without change from 4.2 or 4.3BSD.

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
wheel	0
daemon	1
kmem	2
sys	3
tty	4
operator	5
bin	10

Only users in the ‘‘wheel’’ group are permitted to *su* to ‘‘root’’. Most programs that manage directories in /usr/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 ‘‘bin.’’ Rather than make user's 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's 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 *df* can access the file system

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.

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
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 4.3BSD-tahoe, 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 *mkpasswd* (8) to create the *ndbm* password database. Note that *mkpasswd* is run whenever *vipw* (8) is run.

The format of the cron table, /usr/lib/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.

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, monet.Berkeley.EDU) for the benefit of the name server.

The C library and system binaries on the distribution tape are compiled with new versions of *gethostbyname* and *gethostbyaddr* which 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 DARPA 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’’. Several programs rely on the host name returned by *gethostname* to determine the local domain name.

If you want to compile your system to use the host table lookup routines instead of the name server, you will need to modify /usr/src/lib/libc/Makefile according to the instructions there and then recompile all of the system and local programs (see section 6.6). Next, you must run *mkhosts* (8) to create the *ndbm* host table database from */etc/hosts*.

The format of /etc/ttys has changed, see *ttys* (5) for details. It now includes the terminal type and security options that were previously placed in /etc/ttytype and /etc/securettys.

There is a new version of *syslog* that uses a more generalized facility/priority scheme. This has changed the format of the syslog.conf file. See *syslogd* (8) for details. *Syslog* now logs kernel errors, allowing events such as soft disk errors, filesystem-full messages, and other such error messages to be logged without slowing down the system while the messages print on the console. It is also used by many of the system daemons to monitor system problems more closely, for example network routing changes.

If you are using the name server, your *sendmail* configuration file will need some minor updates to accommodate it. See the ‘‘Sendmail Installation and Operation Guide’’ and the sample *sendmail* configuration files in /usr/src/usr.lib/sendmail/cf. The sendmail.cf’s supplied with this release are alleged to be ‘‘generic’’, but have only really seen use at Berkeley. In particular there are two points to watch out

for. First, all host names in the sendmail.cf itself must be fully qualified names. Second, the sendmail.cf's assume you have a /usr/lib/sendmail that was compiled with the resolver library (i.e., not hosttables). This is necessary to canonicalize unqualified names into fully-qualified names (e.g., foo -> foo.bar.com). Using these .cf files with a host table can probably be done, but it will be difficult. Be sure to regenerate your sendmail frozen configuration file after installation of your updated configuration file with the command */usr/lib/sendmail -bz*. The aliases file, /usr/lib/aliases has also been changed to add certain well-known addresses.

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* so that files are recreated with the same file modes:

```
# cd /usr
# tar xp msgs spool/mail spool/uucp spool/uucppublic spool/news
```

The following two sections contain additional notes concerning changes in 4.3BSD-tahoe that affect the installation of local files; be sure to read them as well.

3.4. Hints on converting from 4.2BSD to 4.3BSD-tahoe

This section summarizes the most significant changes between 4.2BSD and 4.3BSD, particularly those that are likely to cause difficulty in doing the conversion. It does not include changes in the network; see chapter 5 for information on setting up the network.

The mailbox locking protocol has changed; it now uses the advisory locking facility to avoid concurrent update of users' mail boxes. If you have your own mail interface, be sure to update its locking protocol.

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 (there used to be a hard limit of 30). 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. This 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 modification.

Find now has a database of file names, constructed once a week from *cron*. To find a file by name only, the command *find name* will look in the database for files that match the name. This is much faster than *find / –name name –print*.

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 */usr/tmp*. All publicly-writable directories should have their ''sticky'' bits set with ''chmod +t.''

The include file <*time.h*> has returned to */usr/include*, and again contains the definitions for the C library time routines of *ctime* (3).

The *compact* and *uncompact* programs have been supplanted by the faster *compress*. If your user population has *compact*ed files, you will want to install *uncompact* from /usr/src/old/compact.

The configuration of the virtual memory limits has been simplified. A MAXDSIZ option, specified in bytes in the machine configuration file, may be used to raise the maximum process region size from the default of 17Mb to 32Mb or 64Mb. The initial per-process limit is still 6Mb, but can be raised up to MAXDSIZ with the *csh limit* command.

Some 4.3BSD-tahoe binaries will not run with a 4.2BSD kernel because they take advantage of new functionality in 4.3BSD-tahoe. One noticeable example of this problem is *csh*.

If you want to use *ps* after booting a new kernel, and before going multiuser, you must initialize its name list database by running *ps −U*.

3.5. Hints on converting from 4.3BSD to 4.3BSD-tahoe

The largest visible change between 4.3BSD to 4.3BSD-tahoe (other than the addition of support for the Tahoe processor) is the addition of support for disk labels. This facility allows each disk or disk pack to contain all geometry information about the disk and the partition layout for the disk. Disk labels are supported on all disk types on the Tahoe machines, and on hp and ra/rd disks on the VAX. See section 2.1.6 as well as *disklabel* (8) and *disklabel* (5). Installation of this facility requires use of the new kernel and device drivers, bootstraps and other standalone programs, /etc/disktab, *bad144* (8V), *newfs* (8), and probably other programs.

The bootstrap programs have been fixed to work on MicroVAX IIs and VAXstation II's with QVSS (VS II) or QDSS (GPX) displays; the kernel includes support for these displays, courtesy of Digital Equipment Corp. In order to install the bootstrap on RD52/53/54 disks with *disklabel* (8), the new /etc/disktab must be used, or the block 0 bootstrap must be explictly listed as /usr/mdec/rdboot (*not* raboot).

The order in which daemons are started by /etc/rc and /etc/rc.local has changed, and network initialization has been split into /etc/netstart. Look at the prototype files, and modify /etc/rc.local as necessary; c.f. section 5.6.1.

4.3BSD-tahoe includes the Olson timezone implementation, which uses timezone and daylight-savings-time rules loaded from files in /etc/zoneinfo; see *ctime* (3) and *tzfile* (5).

The type of the *sprintf* (3S) function has been changed from *char* * in 4.2BSD and 4.3BSD to *int* as in the proposed ANSI C standard and in System V. Programers are discouraged from using the return value from *sprintf* until this change is ubiquitous. Fortunately, the previous return value from *sprintf* was essentially useless.

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 /usr/spool, you should do:

```
# chown −R root /usr/spool/at
# chown −R uucp.daemon /usr/spool/uucp
# chmod −R o−w /usr/spool/uucp
```

The MAKEHOSTS file has moved from /usr/hosts to /usr.

The source versions of the manual pages have been moved from /usr/man/man[1-8] to /usr/src/man, /usr/src/new/man, and /usr/src/local/man. Local manual pages should be moved into their respective source code directories, or into /usr/src/local/man/man[1-8], and Makefiles changed to install the formatted manual pages into /usr/local/man/cat[1-8]. The shell script /usr/man/manroff calls nroff with the standard manual arguments. An example of installing a manual page might be:

```
# /usr/man/manroff example.2 > example.0
# install -o bin -g bin -m 444 example.0 /usr/local/man/cat2
```

Whatever else is left is likely to be site specific or require careful scrutiny before placing in its eventual resting place. Refer to the documentation and source code before arbitrarily overwriting a file.

4. SYSTEM SETUP

This section describes procedures used to set up a VAX 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. Creating UNIX boot media

The procedures for making the various UNIX boot media are described in this section. If you have an 8200 or 11/780, you will need to make a floppy. For an 11/730, you will need to make a cassette. For an 8600, you will need to make a console RL02 pack.

The boot command files are all set up for booting off of the first UNIBUS or MASSBUS. If you are booting off of a different UNIBUS or MASSBUS, you will need to modify the boot command files appropriately.

4.1.1. Making a UNIX boot console RL02 pack

If you have an 8600 you will want to create a UNIX boot console RL02 pack by adding some files to your current DEC console pack, using *arff* (8). If you do not want to modify your current DEC console pack, you may make a copy of it first using *dd* (1). This pack will make standalone system operations such as bootstrapping much easier.

First change into the directory where the console RL02 information is stored:

 # cd /sys/consolerl

then set up the default boot device. If you have an RK07 as your primary root do:

 # cp defboo.hk defboo.com

If you have a drive on a UDA50 (e.g. an RA81) as your primary root do:

 # cp defboo.ra defboo.com

If you have a second vendor UNIBUS storage module as your primary root do:

 # cp defboo.up defboo.com

Otherwise:

 # cp defboo.hp defboo.com

The final step in updating the console RL02 pack is:

 # make update

More copies of this console RL02 pack can be made using *dd* (1).

4.1.2. Making a UNIX boot floppy

If you have an 8200 or 11/780 you will want to create a UNIX boot floppy by adding some files to a copy of your current DEC console floppy, using either *flcopy* (8) or *dd* (1), and using *arff* (8). This floppy will make standalone system operations such as bootstrapping much easier.

First change into the directory where the console floppy information is stored:

 # cd /sys/floppy

then set up the default boot device. If you have an RK07 as your primary root do:

 # cp defboo.hk defboo.cmd

If you have a drive on a UDA50 (e.g. an RA81) as your primary root do:

 # cp defboo.ra defboo.cmd

If you have a second vendor UNIBUS storage module as your primary root do:

 # cp defboo.up defboo.cmd

If you have a drive on a KDB50 as your primary root do:

 # cp defboo.kra defboo.cmd

Otherwise:

 # cp defboo.hp defboo.cmd

On an 11/780, if the local configuration requires any changes in restar.cmd or defboo.cmd (e.g., for interleaved old-style memory controllers see defboo.MS780C-interleaved), these should be made now. The following command will then copy your DEC local console floppy, updating the copy appropriately.

 # make update
Change Floppy, Hit return when done.
(waits for you to put clean floppy in console)
Are you sure you want to clobber the floppy? *yes*

More copies of this floppy can be made using *flcopy* (8).

On an 8200, copy any of the DEC diagnostic floppies by placing the source in console drive 1 and the destination in console drive 2, then:

 # dd if=/dev/csa1 of=/dev/csa2 bs=400k
1+0 records in
1+0 records out

Next remove all but the first few files, leaving only those that lead up to ‘‘boot58.exe’’ (as well as boot58.exe itself). It is a good idea to remove the original floppy from drive 1 first.

 # arff tmf /dev/csa2
...(should list something like ‘‘fg81.ve0’’, followed by ‘‘boot58.exe’’;
then a series of files that may be deleted)...
 # arff dmf /dev/csa2 files to delete from previous list

Finally, add UNIX boot files:

*# arff rmf /dev/csa2 boot format copy *boo.cmd*

Put the new boot floppy in drive 1. To make copies of this floppy, use the same *dd* command shown above.

4.1.3. Making a UNIX boot cassette

If you have an 11/730 you will want to create a UNIX boot cassette by adding some files to a copy of your current DEC console cassette, using *flcopy* (8) and *arff* (8). This cassette will make standalone system operations such as bootstrapping much easier.

First change into the directory where the console cassette information is stored:

 # cd /sys/cassette

then set up the default boot device. If you have an IDC storage module as your primary root do:

 # cp defboo.rb defboo.cmd

If you have an RK07 as your primary root do:

 # cp defboo.hk defboo.cmd

If you have a drive on a UDA50 as your primary root do:

 # cp defboo.ra defboo.cmd

Otherwise:

 # cp defboo.up defboo.cmd

To complete the procedure place your DEC local console cassette in drive 0 (the drive at front of the CPU); the following command will then copy it, updating the copy appropriately.

 # make update
Change Floppy, Hit return when done.
(waits for you to put clean cassette in console drive 0)
Are you sure you want to clobber the floppy? *yes*

More copies of this cassette can best be made using *dd* (1).

4.2. 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’’.

4.2.1. Kernel organization

As distributed, the kernel source is in a separate tar image. The source may be physically located anywhere within any file system 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 directory /sys/sys contains the mainline machine independent operating system code. Files within this directory are conventionally named with the following prefixes:

init_	system initialization
kern_	kernel (authentication, process management, etc.)
quota_	disk quotas
sys_	system calls and similar
tty_	terminal handling
ufs_	file system
uipc_	interprocess communication
vm_	virtual memory

The remaining directories are organized as follows:

/sys/h	machine-independent include files
/sys/conf	site configuration files and basic templates
/sys/kdb	machine-independent part of the kernel debugger
/sys/net	protocol-independent, but network-related code
/sys/netimp	IMP support code
/sys/netinet	DARPA Internet code
/sys/netns	Xerox NS code
/sys/stand	machine-independent standalone code
/sys/tahoe	Tahoe-specific mainline code
/sys/tahoealign	Tahoe unaligned-reference emulation code
/sys/tahoedist	Tahoe distribution files
/sys/tahoeif	Tahoe network interface code
/sys/tahoevba	Tahoe VERSAbus device drivers and related code
/sys/tahoemath	Tahoe floating point emulation code
/sys/tahoestand	Tahoe standalone device drivers and related code
/sys/vax	VAX-specific mainline code
/sys/vaxbi	VAX BI device drivers and related code
/sys/vaxdist	VAX distribution files
/sys/vaxif	VAX network interface code
/sys/vaxmba	VAX MASSBUS device drivers and related code
/sys/vaxstand	VAX standalone device drivers and boot code
/sys/vaxuba	VAX UNIBUS device drivers and related code

Many of these directories are referenced through /usr/include with symbolic links. For example, /usr/include/sys is a symbolic link to /sys/h. The system code, as distributed, is totally independent of the include files in /usr/include. This allows the system to be recompiled from scratch without the /usr file system mounted.

4.2.2. Devices and device drivers

Devices supported by UNIX are implemented in the kernel by drivers whose source is kept in /sys/vax, /sys/vaxbi, /sys/vaxuba, or /sys/vaxmba. 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 file system, 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, if your machine has a single DZ11, a single DH11, a single DMF32, an RM03 disk, an EMULEX UNIBUS SMD disk controller, an AMPEX 9300 disk, and a TE16 tape drive you would do:

```
# cd /
# mkdir newdev
# cp dev/MAKEDEV newdev/MAKEDEV
# cd newdev
# MAKEDEV dz0 dh0 dmf0 hp0 up0 ht0 std LOCAL
```

Note the ‘‘std’’ argument causes standard devices such as */dev/console*, the machine console, */dev/floppy*, the console floppy disk interface for the 11/780 and 11/785, and */dev/tu0* and */dev/tu1*, the console cassette interfaces for the 11/750 and 11/730, to be created.

You can then do

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

to install the new device directory.

4.2.3. Building new system images

The kernel configuration of each UNIX system is described by a single configuration file, stored in the */sys/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’’, 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/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’re 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.

4.3. Disk configuration

This section describes how to layout file systems to make use of the available space and to balance disk load for better system performance.

4.3.1. Initializing /etc/fstab

Change into the directory /etc and copy the appropriate file from:

```
fstab.rm03
fstab.rm05
fstab.rm80
fstab.ra60
fstab.ra80
fstab.ra81
fstab.rb80
fstab.rp06
fstab.rp07
fstab.rk07
fstab.up160m (160MB up drives)
fstab.hp400m (400MB hp drives)
fstab.up (other up drives)
fstab.hp (other hp drives)
```

to the file /etc/fstab, i.e.:

```
# cd /etc
# cp fstab.xxx fstab
```

This will set up the default information about the usage of disk partitions, which we see how to update more below.

4.3.2. 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, on an RM80, the first partition, hp0a, is used for a root file system, a backup thereof, or a small file system like, /tmp; the second partition, hp0b, is used for paging and swapping; and the third partition, hp0g, holds a user file system. On an RM05, the first three partitions are used as for the RM80, and the fourth partition, hp0h, holds the /usr file system, including source code.

The disk partition sizes for a drive are based on a set of four prototype partition tables; c.f. *diskpart* (8). The particular table used is dependent on the size of the drive. The ‘‘a’’ partition is the same size across all drives, 15884 sectors. The ‘‘b’’ partition, used for paging and swapping, is sized according to the total space on the disk. For drives less than about 400 megabytes the partition is 33440 sectors, while for larger drives the partition size is doubled to 66880 sectors. The ‘‘c’’ partition is always used to access the entire physical disk, including the space at the back of the disk reserved for the bad sector forwarding table. If the disk is larger than about 250 megabytes, an ‘‘h’’ partition is created with size 291346 sectors, and no matter whether the ‘‘h’’ partition is created or not, the remainder of the drive is allocated to the ‘‘g’’ partition. Sites that want to split up the ‘‘g’’ partition into several smaller file systems may use the ‘‘d’’, ‘‘e’’, and ‘‘f’’ partitions that overlap the ‘‘g’’ partition. The default sizes for these partitions are 15884, 55936, and the remainder of the disk, respectively*.

The disk partition sizes for DEC RA60, RA80, and RA81 have changed since 4.2BSD. If upgrading from 4.2BSD, you will need to decide if you want to use the new partitions or the old partitions. If you desire to use the old partitions, you will need to label your packs as ‘racompat’, or create your own by updating /etc/disktab. Any other partition sizes that were modified at your site will require the same consideration; if the device driver does not support pack labels, you will have to update its compiled-in tables as well.

The space available on a disk varies per device. The amount of space available on the common disk partitions is listed in the following table. Not shown in the table are the partitions of each drive devoted to the root file system and the paging area. Many other partitions are listed in the standard partitions, but most of them are not useful. Note that the standard partition tables usually list several alternative ways to divide a disk, but that only nonoverlapping partitions may be used on any one disk.

Type	Name	Size	Name	Size
rk07	hk?g	13 Mb		
rm03	hp?g	41 Mb		
rp06	hp?g	145 Mb		
rm05	hp?g	80 Mb	hp?h	145 Mb
rm80	hp?g	96 Mb		
ra60	ra?g	78 Mb	ra?h	96 Mb
ra80	ra?g	96 Mb		
ra81	ra?g	257 Mb	ra?h	145 Mb
rb80	rb?g	41 Mb	rb?h	56 Mb
rp07	hp?g	315 Mb	hp?h	145 Mb
up300	up?g	80 Mb	up?h	145 Mb
up330	up?g	90 Mb	up?h	145 Mb
up400	hp?g	216 Mb	hp?h	145 Mb
up160	up?g	106 Mb		

Here up300 refers to either an AMPEX or CDC 300 megabyte disk on a MASSBUS or UNIBUS disk controller, up330 refers to either an AMPEX or FUJITSU 330 megabyte disk on a MASSBUS or UNIBUS controller, up160 refers to a FUJITSU 160 megabyte disk on the UNIBUS, and up400 refers to a FUJITSU Eagle 400 megabyte disk on a MASBUS or UNIBUS disk controller. ‘‘hp’’ should be substituted for

* These rules are, unfortunately, not evenly applied to all disks. */etc/disktab*, and the pack label or driver tables, give the final word; consult section 4 of the manual, and read /etc/disktab, for more information.

''up'' above if the disk is on the MASSBUS. Consult the manual pages for the specific controllers for other supported disks or other partitions.

Each disk also has a paging area, typically 16 megabytes, and a root file system of 7.5 megabytes. The distributed system binaries occupy about 34 megabytes while the major sources occupy another 32 megabytes. This overflows dual RK07, dual RL02 and single RM03 systems, but fits easily on most other hardware configurations.

Be aware that the disks have their sizes measured in disk sectors (usually 512 bytes), while the UNIX file system blocks are variable sized. All user programs report disk space in kilobytes and, where needed, disk sizes are always specified in units of sectors. The /etc/disktab file used in labelling disks and making file systems specifies disk partition sizes in sectors; the default sector size (DEV_BSIZE as defined in /sys/h/param.h) may be overridden with the ''se'' attribute.

4.3.3. 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 size may be interleaved. Drives smaller than 400 megabytes have swap partitions of 16 megabytes while drives larger than 400 megabytes have 32 megabytes. These values may be changed to get more paging space by changing the label (or, if labels are unsupported, the appropriate partition table in the disk driver).

Many common system programs (C, the editor, the assembler etc.) create intermediate files in the /tmp directory, so the file system where this is stored also should be made large enough to accommodate most high-water marks; if you have several disks, it makes sense to mount this in a ''root'' (i.e. first partition) file system 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. For general time-sharing applications, the best strategy is to try to split the root file system (/), system binaries (/usr), the temporary files (/tmp), and the user files among several disk arms, and to interleave the paging activity among several arms.

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 file system.
2. The /tmp file system.
3. The /usr file system.
4. The user files.
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
/	0	0	0
tmp	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 file systems (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 file system arrangements can pay off in much improved performance. It is particularly easy to move the root, the /tmp file system and the paging areas. Place the user files and the /usr directory as space needs dictate and experiment with the other, more easily moved file systems.

4.3.4. File system parameters

Each file system 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 file system parameters can result in substantial throughput degradation or significant waste of disk space. As distributed, file systems are configured according to the following table.

File system	Block size	Fragment size
/	8 kbytes	1 kbytes
usr	4 kbytes	1 kbytes
users	4 kbytes	1 kbytes

The root file system block size is made large to optimize bandwidth to the associated disk; this is particularly important since the /tmp directory is normally part of the root file or a similar filesystem. The large block size is also 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 file system. With a 1 kbyte fragment size disk space utilization is about the same as with the earlier versions of the file system.

The usr file system would like to use a 4 kbyte block size with 512 byte fragment size in an effort to get high performance while conserving the amount of space wasted by a large fragment size. However, the tahoe disk controllers require a minimum block size of 1 Kbyte. Space compaction has been deemed important here because the source code for the system is normally placed on this file system. If the source code is placed on a separate filesystem, use of an 8 kbyte block size with 1 kbyte fragments might be considered for improved performance when paging from */usr* binaries.

The file systems for users have a 4 kbyte block size with 1 kbyte fragment size. These parameters have been selected based on observations of the performance of our user file systems. The 4 kbyte block size provides adequate bandwidth while the 1 kbyte fragment size provides acceptable space compaction and disk fragmentation.

Other parameters may be chosen in constructing file systems, 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 file system as larger I/O requests will then be performed by the system. However, consideration must be given to the average file sizes found in the file system 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, and 1 double 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 file system 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 file system.

In selecting a fragment size for a file system, at least two considerations should be given. The major performance tradeoffs observed are between an 8 kbyte block file system and a 4 kbyte block file system. Because of implementation constraints, the block size / fragment size ratio can not be greater than 8. This means that an 8 kbyte file system will always have a fragment size of at least 1 kbytes. If a file system 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 file system has a 4 kbyte block size and 512 byte fragment size, converting it to an 8K/1K file system will result in significantly

* A triple indirect block pointer is also reserved, but not currently supported.

more space being used. This implies that 4 kbyte block file systems 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 file system 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 file system running near full capacity. By comparison, the level of fragmentation in a 4:1 fragment to block ratio file system is one tenth as severe. This means that on file systems 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 file system is nearly full, file expansion that requires locating a contiguous area of disk space is more likely to fail on a 512 byte file system than on a 1 kbyte file system. 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 file system were really full. This parameter is nominally set to 10%; it may be changed by supplying a parameter to *newfs*(8), or by updating the super block of an existing file system using *tunefs* (8).

In general, unless a file system 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 8 kbytes and the ratio of block size / fragment size must be 1, 2, 4, or 8.

The disk geometry information used by the file system 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 file system 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 file system resides; *newfs* then reads /etc/disktab instead of the pack label. This file also contains the default file system 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*.

4.3.5. Implementing a layout

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

As an example, consider a system with RM80's. On the first RM80, hp0, we will put the root file system in hp0a, and the /usr file system in hp0, which has enough space to hold it and then some. The /tmp directory will be part of the root file system, as no file system will be mounted on /tmp. If we had only one RM80, we would put user files in the hp0 partition with the system source and binaries.

If we had a second RM80, we would place */usr* in hp1g. We would put user files in hp0g, calling the file system /a. We would also interleave the paging between the 2 RM80's. To do this we would build a system configuration that specified:

```
config      vmunix      root on hp0 swap on hp0 and hp1
```

to get the swap interleaved, and */etc/fstab* would then contain

```
/dev/hp0a:/:rw:1:1
/dev/hp0b::sw::
/dev/hp0g:/a:rw:1:2
/dev/hp1b::sw::
/dev/hp1g:/usr:rw:1:2
```

We would keep a backup copy of the root file system in the **hp1a** disk partition. Alternatively, that partition could be used for */tmp*.

To make the /a file system we would do:

```
# cd /dev
# MAKEDEV hp1
# disklabel -wr hp1 RM80 "disk name"
# newfs hp1g
(information about file system prints out)
# mkdir /a
# mount /dev/hp1g /a
```

4.4. Configuring terminals

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

Terminals connected via DZ11 interfaces are conventionally named **ttyDD** where DD is a decimal number, the ‘‘minor device’’ number. The lines on dz0 are named /dev/tty00, /dev/tty01, ... /dev/tty07. By convention, all other terminal names are of the form **tty**CX, where C is an alphabetic character according to the type of terminal multiplexor and its unit number, and X is a digit for the first ten lines on the interface and an increasing lower case letter for the rest of the lines. C is defined for the number of interfaces of each type listed below.

Interface Type	Characters	Number of lines per board	Number of Interfaces
DZ11	see above	8	10
DMF32	A-C,E-I	8	8
DMZ32	a-c,e-g	24	6
DH11	h-o	16	8
DHU11	S-Z	16	8
pty	p-u	16	6

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. (For example, use ‘‘cd /dev; MAKEDEV dz1’’ to make the special files for the second DZ11.) 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 (*/etc/utmp*), and therefore it is best to make changes while running in single-user mode and to add all of the entries for a new device at once.

The format of the /etc/ttys file is completely new in 4.3BSD. Each line in the 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 /etc/init should startup to service the line (or **none** if the line is to be left alone), the terminal type (found in /etc/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). 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  "/etc/getty std.9600"       vt100 on secure     # mike's office
```

The std.9600 parameter provided to /etc/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 either wire back the signals so that the carrier appears to always be present, or show in the system configuration that carrier is to be assumed to be present with *flags* for each terminal device. See *dh* (4), *dhu* (4), *dz* (4), *dmz* (4), and *dmf* (4) for details.

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 –1 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 (/etc/syslogd) 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.

4.5. 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.

4.6. 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.

4.7. Setting up the line printer system

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

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

The file /etc/printcap is a master data base describing line printers directly attached to a machine and, also, printers accessible across a network. The manual page *printcap* (5) describes the format of this data base 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’’. A call to the *lpd* program should be present in /etc/rc.

4.8. Setting up the mail system

The mail system consists of the following commands:

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

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

Mail queued in the directory /usr/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.lib/sendmail and then adjust the necessary configuration files. You should also set up the file

/usr/lib/aliases for your installation, creating mail groups as appropriate. Documents describing *sendmail*'s operation and installation are also included in the distribution.

4.8.1. Setting up a UUCP connection

The version of *uucp* included in 4.3BSD-tahoe is a greatly enhanced version of the one originally distributed with 32/V*. The enhancements include:

- 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''. 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 /usr/spool/uucp. User commands are kept in /usr/bin, operational commands in /usr/lib/uucp, and /usr/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:

* The *uucp* included in this distribution is the result of work by many people; we gratefully acknowledge their contributions, but refrain from mentioning names in the interest of keeping this document current.

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

4.3BSD provides support for the DARPA standard Internet protocols IP, ICMP, TCP, and UDP. These protocols may be used on top of a variety of hardware devices ranging from the IMP's (PSN's) used in the ARPANET 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. 4.3BSD 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. 4.3BSD 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. Also, if you have an IMP connection, you will need to include the pseudo-device ''imp.''

Before configuring the appropriate networking hardware, you should consult the manual pages in section 4 of the Programmer's Manual. The following table lists the devices for which software support exists.

Device name	Manufacturer and product
acc	ACC LH/DH interface to IMP
css	DEC IMP-11A interface to IMP
ddn	ACC ACP625 DDN Standard mode X.25 interface to IMP
dmc	DEC DMC-11 (also works with DMR-11)
de	DEC DEUNA 10Mb/s Ethernet
ec	3Com 10Mb/s Ethernet
en	Xerox 3Mb/s prototype Ethernet (not a product)
ex	Excelan 204 10Mb/s Ethernet
hdh	ACC IF-11/HDH IMP interface
hy	NSC Hyperchannel, w/ DR-11B and PI-13 interfaces
il	Interlan 1010 and 10101A 10Mb/s Ethernet interfaces
ix	Interlan NP100 10Mb/s Ethernet interface
pcl	DEC PCL-11
vv	Proteon 10Mb/s and 80Mb/s proNET ring network (V2LNI)

All network interface drivers including the loopback interface, require that their host address(es) be defined at boot time. This is done with *ifconfig* (8C) 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* (8C) 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. These options include disabling the use of the Address Resolution Protocol; this may be useful if a network is shared with hosts running software that does not yet provide this function. Alternatively, translations for such hosts may be set in advance or ''published'' by a 4.3BSD host by use of the *arp* (8C) command. Note that the use of trailer link-level is now negotiated between 4.3BSD 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 Routeing Protocol); running this protocol is mandatory, however one can manually add translations for machines that do not participate by use of the *route* (8C) 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 4.3BSD the DARPA 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.* 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

```
/etc/ifconfig en0 netmask 0xffffff00 128.32.1.7
```

This specifies that for interface ''en0'', 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

* 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.

mask of the first interface on each network is actually 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. 4.3BSD 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 and 4.3BSD hosts to coexist while the upgrade process proceeds. In the presence of subnets, the broadcast address uses 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). 4.3BSD 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 the routing table management daemon */etc/routed* 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 created by *htable* (8), 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* (8C) 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

/etc/route add default *smart-gateway* 1

to */etc/netstart*; see *route* (8C) 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 4.3BSD machines as gateways

Several changes have been made in 4.3BSD 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 as to the network number or broadcast address. Another change is that 4.3BSD 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 4.3BSD 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 in environments where it may cause difficulties.

Local area routing within a group of interconnected Ethernets and other such networks may be handled by *routed* (8C). Gateways between the Arpanet or Milnet and one or more local networks require an additional routing protocol, the Exterior Gateway Protocol (EGP), to inform the core gateways of their presence and to acquire routing information from the core. An EGP implementation for 4.3BSD is available by anonymous ftp from ucbarpa.berkeley.edu. If necessary, contact the Berkeley Computer Systems Research Group for assistance.

5.6. Network data bases

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)	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* (8C)	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* (8C)	list of ‘‘unwelcome’’ ftp users
/etc/hosts.lpd	*lpd* (8C)	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 ARPANET or other 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 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
/etc/syslogd	error logging server	/etc/rc
/etc/named	Internet name server	/etc/rc
/etc/routed	routing table management daemon	/etc/rc
/etc/rwhod	system status daemon	/etc/rc
/etc/timed	time synchronization daemon	/etc/rc.local
/usr/lib/sendmail	SMTP server	/etc/rc.local
/etc/rshd	shell server	inetd
/etc/rexecd	exec server	inetd
/etc/rlogind	login server	inetd
/etc/telnetd	TELNET server	inetd
/etc/ftpd	FTP server	inetd
/etc/fingerd	Finger server	inetd
/etc/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* is used to provide 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 /etc/timed ]; then
        /etc/timed & echo -n ' timed'                   >/dev/console
fi
```

5.6.2. Internet daemon

In 4.3BSD most of the servers for user-visible services are started up by a ‘‘super server’’, the Internet daemon. The Internet daemon, */etc/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 as 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    /etc/ftpd       ftpd
```

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

5.6.3. Regenerating /etc/hosts and /etc/networks

When using the host address routines that use the Internet name server, the file */etc/hosts* is only used for setting interface addresses and at other times that the server is not running, and therefore it need only contain addresses for local hosts. There is no equivalent service for network names yet. The full host and network name data bases are normally derived from a file retrieved from the Internet Network Information

Center at SRI. To do this you should use the program /etc/gettable to retrieve the NIC host data base, and the program *htable* (8) to convert it to the format used by the libraries. You should change to the directory where you maintain your local additions to the host table and execute the following commands.

```
# /etc/gettable sri-nic.arpa
Connection to sri-nic.arpa opened.
Host table received.
Connection to sri-nic.arpa closed.
# /etc/htable hosts.txt
Warning, no localgateways file.
#
```

The *htable* program generates three files in the local directory: *hosts*, *networks* and *gateways*. If a file ''localhosts'' is present in the working directory its contents are first copied to the output file. Similarly, a ''localnetworks'' file may be prepended to the output created by *htable*, and 'localgateways'' will be prepended to *gateways*. It is usually wise to run *diff* (1) on the new host and network data bases before installing them in /etc. If you are using the host table for host name and address mapping, you should run *mkhosts* (8) after installing */etc/hosts*. If you are using the name server for the host name and address mapping, you only need to install *networks* and a small copy of *hosts* describing your local machines. The full host table in this case might be placed somewhere else for reference by users. The gateways file may be installed in */etc/gateways* if you use *routed* (8C) for local routing and wish to have static external routes installed when *routed* is started. This procedure is essentially obsolete, however, except for individual hosts that are on the Arpanet or Milnet and do not forward packets from a local network. Other situations require the use of an EGP server.

If you are connected to the DARPA Internet, it is highly recommended that you use the name server for your host name and address mapping, as this provides access to a much larger set of hosts than are provided in the host table. Many large organizations on the network currently have only a small percentage of their hosts listed in the host table retrieved from NIC.

5.6.4. /etc/hosts.equiv

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:

```
ucbarpa
okeeffe
monet
ucbvax
```

5.6.5. /etc/ftpusers

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

file system 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 4.3BSD-tahoe 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 ^C (interrupt). This will leave the system in single-user mode, with only the console terminal active. 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 (^\).

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

>>>B

will boot from the default device. On an 8600, 8200, 11/780, or 11/730 the default device is determined by a ''DEPOSIT'' command stored on the console boot device in the file ''DEFBOO.CMD'' (''DEFBOO.COM'' on an 8600); on an 11/750 the default device is determined by the setting of a switch on the front panel.

You can boot a system up single user on an 8600, 780, or 730 by doing

>>>B xxS

where *xx* is one of HP, HK, UP, RA, or RB. The corresponding command on an 11/750 is

>>>B/2

On an 8200, use

>>>B/R5:800
(node and memory test values)
BOOT58> *@XXSBOO.CMD*

For second vendor storage modules on the UNIBUS or MASSBUS of an 11/750 you will need to have a boot prom. Most vendors will sell you such proms for their controllers; contact your vendor if you don't have one.

Other possibilities are:

>>>B ANY

or, on an 8200,

>>>B/R5:800
BOOT58>*@ANYBOO.CMD*

or, on an 11/750

>>>B/3

These commands boot and ask for the name of the system to be booted. They can be used after building a new test system to give the boot program the name of the test version of the system.*

To bring the system up to a multi-user configuration from the single-user status, all you have to do is hit ^D 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) for more details. Note, however, that this does not cause a file system check to be performed. Unless the system was taken down cleanly, you should run ''fsck –p'' or force a reboot with *reboot*(8) to have the disks checked.

* Additional bootflags are used when a system is configured with the kernel debugger; consult *kdb*(4) for details.

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. File systems remain mounted after the system is taken single-user. If you wish to come up multi-user again, you should do this by:

```
# cd /
# /etc/umount -a
# ^D
```

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 */usr/adm/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 /usr/adm/messages*).

6.3. File system checks, backups and disaster recovery

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

Dumping of the file systems should be done regularly, since once the system is going it is easy to become complacent. Complete and incremental dumps are easily done with *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, 1979	jkf	137K
D1	3	Nov 28, 1979	jkf	29K
D2	2	Nov 29, 1979	rrh	34K
D3	5	Nov 30, 1979	rrh	19K
D4	4	Dec 1, 1979	rrh	22K
W1	1	Dec 2, 1979	etc	40K
D5	3	Dec 4, 1979	rrh	15K
D6	2	Dec 5, 1979	jkf	25K
D7	5	Dec 6, 1979	jkf	15K
D8	4	Dec 7, 1979	rrh	19K
W2	1	Dec 9, 1979	etc	118K
D9	3	Dec 11, 1979	rrh	15K
D10	2	Dec 12, 1979	rrh	26K
D1	5	Dec 15, 1979	rrh	14K
W3	1	Dec 17, 1979	etc	71K
D2	3	Dec 18, 1979	etc	13K
FULL	0	Dec 22, 1979	etc	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 file system be made regularly. This is especially true when only one disk is available. Then, if the root file system is damaged by a hardware or software failure, you can rebuild a workable disk doing a restore in the same way that the initial root file system 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 file system data

If you have the resources, the best way to move a file system is to dump it to a spare disk partition, or magtape, using *dump* (8), use *newfs* (8) to create the new file system, and restore the file system 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 file system dumps to be restored without concern for the original size of the file system. Further, portions of a file system may be selectively restored using a method similar to the tape archive program.

If you have to merge a file system into another, existing one, the best bet is to use *tar* (1). If you must shrink a file system, the best bet is to dump the original and restore it onto the new file system. If you are playing with the root file system 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 file system 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 file system as you did when first installing the system. Boot normally using the newly created disk file system.

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/vaxstand*, 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 60-100 context switches and interrupts per second and about 70-120 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 the system, and it is a good idea to try rebuilding pieces of the system to build confidence in the procedures. The system consists of two major parts: the kernel itself (/sys) and the user programs (/usr/src and subdirectories). The major part of this is /usr/src.

The three major libraries are the C library in /usr/src/lib/libc and the FORTRAN libraries /usr/src/usr.lib/libI77 and /usr/src/usr.lib/libF77. In each case the library is remade by changing into the corresponding directory and doing

```
# make
```

and then installed by

```
# make install
```

Similar to the system,

```
# make clean
```

cleans up.

The source for all other libraries is kept in subdirectories of /usr/src/usr.lib; each has a makefile and can be recompiled by the above recipe.

If you look at /usr/src/Makefile, you will see that you can recompile the entire system source with one command. To recompile a specific program, find out where the source resides with the *whereis* (1) command, then change to that directory and remake it with the Makefile present in the directory. For instance, to recompile ‘‘date’’, all one has to do is

```
# whereis date
date: /usr/src/bin/date.c /bin/date
# cd /usr/src/bin
# make date
```

this will create an unstripped version of the binary of ‘‘date’’ in the current directory. To install the binary image, use the install command as in

```
# install −s date -o bin -g bin -m 755 /bin/date
```

The −s option will insure the installed version of date has its symbol table stripped. The install command should be used instead of mv or cp as it understands how to install programs even when the program is currently in use.

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

```
# make
# make DESTDIR=pathname install
```

To regenerate all the system source you can do

```
# cd /usr/src
# make clean; make depend; make
```

If you modify the C library, say to change a system call, and want to rebuild and install everything from scratch you have to be a little careful. You must insure that the libraries are installed before the remainder of the source, otherwise the loaded images will not contain the new routine from the library. The following sequence will accomplish this.

```
# cd /usr/src
# make clean
# make depend
# make build
# make installsrc
```

The *make clean* removes any existing binary or object files in the source trees to insure that everything will be recompiled and reloaded. The *make depend* recreates all of the dependencies. See *mkdep*(1) for further details. The *make build* compiles and installs the libraries and compilers, then recompiles the libraries and compilers and the remainder of the sources. The *make installsrc* installs all of the commands not installed as part of the *make build*.

6.7. Making local modifications

Locally written commands that aren’t distributed are kept in /usr/src/local and their binaries are kept in /usr/local. This allows /usr/bin, /usr/ucb, and /bin to correspond to the distribution tape (and to the manuals that people can buy). People using local commands should be made aware that they aren’t in the base manual. Manual pages for local commands should be installed in /usr/src/local/man and 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 facilitate this practice.

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 */usr/adm/wtmp*, which is summarized by the program *ac* (8). The process time accounting information is stored in the file */usr/adm/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 */etc/cron* to be executed every day at a specified time. This is done by adding lines to */usr/adm/crontab*;

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 may use on each file system. 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''. File systems may then be placed under the quota mechanism by creating a null file *quotas* at the root of the file system, running *quotacheck* (8), and modifying */etc/fstab* to show that the file system is read-write with disk quotas (an ''rq'' type field). 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 file system or file systems.

Quotas are enforced with *soft* and *hard* limits. When a user first reaches a soft limit on a resource, a message is generated on his/her terminal. If the user fails to lower the resource usage below the soft limit the next time they log in to the system the *login* program will generate a warning about excessive usage. Should three login sessions go by with the soft limit breached the system then treats the soft limit as a *hard* limit and disallows any allocations until enough space is reclaimed to bring the user back below the soft limit. Hard limits are enforced strictly resulting in errors when a user 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'' 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* (8C) 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 data base
/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/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	host name data base
/etc/networks	network name data base
/etc/services	network services data base
/etc/hosts.equiv	hosts under same administrative control
/etc/syslog.conf	error log configuration for *syslogd* (8)
/etc/ttys	enables/disables ports
/usr/lib/crontab	commands that are run periodically
/usr/lib/aliases	mail forwarding and distribution groups
/usr/adm/acct	raw process account data
/usr/adm/messages	system error log
/usr/adm/wtmp	login session accounting

APPENDIX A – BOOTSTRAP DETAILS

This appendix contains pertinent files and numbers regarding the bootstrapping procedure for 4.3BSD-tahoe. You should never have to look at this appendix. However, if there are problems in installing the distribution on your machine, the material contained here may prove useful.

Contents of the distribution tape(s)

The distribution normally consists of three 1600bpi 2400' magnetic tapes or one 6250bpi 2400' magnetic tape. The layout of the 1600bpi tapes is listed below. The 6250bpi tape is in the same order, but is only on one tape. The first tape contains the following files on it. All tape files are blocked in 10 kilobytes records, except for the first file on the first tape that has 512 byte records.

Tape file	Records*	Contents
one	210	8 bootstrap monitor programs and a *tp* (1) file containing *boot*, *format*, and *copy*
two	205	''mini root'' file system
three	430	*dump* (8) of distribution root file system
four	3000	*tar* (1) image of binaries and libraries in /usr

The second tape contains the following files:

Tape file	# Records	Contents
one	720	*tar* (1) image of /sys, including GENERIC system
two	2500	*tar* (1) image of /usr/src
three	580	*tar* (1) image of /usr/lib/vfont

The third tape contains the following files:

Tape file	# Records	Contents
one	3660	*tar* (1) image of user contributed software
two	250	*tar* (1) image of /usr/ingres

The distribution tape is made with the shell scripts located in the directory /sys/dist. To build a distribution tape one must first create a mini root file system with the *buildmini* shell script.

* The number of records in each tape file are approximate and do not correspond to the actual tape.

```
#!/bin/sh
#       @(#)buildmini      4.7 (Berkeley) 6/23/85
#
miniroot=hp0d
minitype=rm80
#
date
umount /dev/${miniroot}
newfs -s 4096 ${miniroot} ${minitype}
fsck /dev/r${miniroot}
mount /dev/${miniroot} /mnt
cd /mnt; sh /sys/dist/get
cd /sys/dist; sync
umount /dev/${miniroot}
fsck /dev/${miniroot}
date
```

The *buildmini* script uses the *get* script to build the file system.

```
#!/bin/sh
#
#       @(#)get      4.23 (Berkeley) 4/9/86
#
# Shell script to build a mini-root file system
# in preparation for building a distribution tape.
# The file system created here is image copied onto
# tape, then image copied onto disk as the "first"
# step in a cold boot of 4.2 systems.
#
DISTROOT=/nbsd
#
if [ ‘pwd‘ = ’/’ ]
then
        echo You just ’(almost)’ destroyed the root
        exit
fi
cp $DISTROOT/sys/GENERIC/vmunix .
rm -rf bin; mkdir bin
rm -rf etc; mkdir etc
rm -rf a; mkdir a
rm -rf tmp; mkdir tmp
rm -rf usr; mkdir usr usr/mdec
rm -rf sys; mkdir sys sys/floppy sys/cassette sys/consolerl
cp $DISTROOT/etc/disktab etc
cp $DISTROOT/etc/newfs etc; strip etc/newfs
cp $DISTROOT/etc/mkfs etc; strip etc/mkfs
cp $DISTROOT/etc/restore etc; strip etc/restore
cp $DISTROOT/etc/init etc; strip etc/init
cp $DISTROOT/etc/mount etc; strip etc/mount
cp $DISTROOT/etc/mknod etc; strip etc/mknod
cp $DISTROOT/etc/fsck etc; strip etc/fsck
cp $DISTROOT/etc/umount etc; strip etc/umount
cp $DISTROOT/etc/arff etc; strip etc/arff
cp $DISTROOT/etc/flcopy etc; strip etc/flcopy
cp $DISTROOT/bin/mt bin; strip bin/mt
```

```
cp $DISTROOT/bin/ls bin; strip bin/ls
cp $DISTROOT/bin/sh bin; strip bin/sh
cp $DISTROOT/bin/mv bin; strip bin/mv
cp $DISTROOT/bin/sync bin; strip bin/sync
cp $DISTROOT/bin/cat bin; strip bin/cat
cp $DISTROOT/bin/mkdir bin; strip bin/mkdir
cp $DISTROOT/bin/stty bin; strip bin/stty; ln bin/stty bin/STTY
cp $DISTROOT/bin/echo bin; strip bin/echo
cp $DISTROOT/bin/rm bin; strip bin/rm
cp $DISTROOT/bin/cp bin; strip bin/cp
cp $DISTROOT/bin/expr bin; strip bin/expr
cp $DISTROOT/bin/[ bin; strip bin/[
cp $DISTROOT/bin/awk bin; strip bin/awk
cp $DISTROOT/bin/make bin; strip bin/make
cp $DISTROOT/usr/mdec/* usr/mdec
cp $DISTROOT/sys/floppy/[Ma-z0-9]* sys/floppy
cp $DISTROOT/sys/consolerl/[Ma-z0-9]* sys/consolerl
cp -r $DISTROOT/sys/cassette/[Ma-z0-9]* sys/cassette
cp $DISTROOT/sys/stand/boot boot
cp $DISTROOT/sys/stand/pcs750.bin pcs750.bin
cp $DISTROOT/.profile .profile
cat >etc/passwd <<EOF
root::0:10::/:/bin/sh
EOF
cat >etc/group <<EOF
wheel:*:0:
staff:*:10:
EOF
cat >etc/fstab <<EOF
/dev/hp0a:/a:xx:1:1
/dev/up0a:/a:xx:1:1
/dev/hk0a:/a:xx:1:1
/dev/ra0a:/a:xx:1:1
/dev/rb0a:/a:xx:1:1
EOF
cat >xtr <<'EOF'
: ${disk?'Usage: disk=xx0 type=tt tape=yy xtr'}
: ${type?'Usage: disk=xx0 type=tt tape=yy xtr'}
: ${tape?'Usage: disk=xx0 type=tt tape=yy xtr'}
echo 'Build root file system'
newfs ${disk}a ${type}
sync
echo 'Check the file system'
fsck /dev/r${disk}a
mount /dev/${disk}a /a
cd /a
echo 'Rewind tape'
mt -f /dev/${tape}0 rew
echo 'Restore the dump image of the root'
restore rsf 3 /dev/${tape}0
cd /
sync
umount /dev/${disk}a
sync
```

```
fsck /dev/r${disk}a
echo 'Root filesystem extracted'
echo
echo 'If this is an 8650 or 8600, update the console rl02'
echo 'If this is a 780 or 785, update the floppy'
echo 'If this is a 730, update the cassette'
EOF
chmod +x xtr
rm -rf dev; mkdir dev
cp $DISTROOT/sys/dist/MAKEDEV dev
chmod +x dev/MAKEDEV
cp /dev/null dev/MAKEDEV.local
cd dev
./MAKEDEV std hp0 hk0 up0 ra0 rb0
./MAKEDEV ts0; mv rmt12 ts0; rm *mt*;
./MAKEDEV tm0; mv rmt12 tm0; rm *mt*;
./MAKEDEV ht0; mv rmt12 ht0; rm *mt*;
./MAKEDEV ut0; mv rmt12 ut0; rm *mt*;
./MAKEDEV mt0; mv rmt4 xt0; rm *mt*; mv xt0 mt0
cd ..
sync
```

The mini root file system must have enough space to hold the files found on a floppy or cassette.

Once a mini root file system is constructed, the *maketape* script makes a distribution tape.

```
#!/bin/sh
#
#	@(#)maketape	4.27 (Berkeley) 10/17/85
#
#	maketape [ 6250 | 1600 [ tapename [ remotetapemachine ] ] ]
miniroot=hp0d
tape=/dev/rmt12
type=6250
if [ $# -gt 0 ]; then type=$1; fi
if [ $# -gt 1 ]; then tape=$2; fi
tartape=$tape
if [ $# -gt 2 ]; then remote=$3; tartape='-'; fi
#
trap "rm -f /tmp/tape.$$; exit" 0 1 2 3 13 15
$remote mt -t ${tape} rew
date
umount /dev/hp2g
umount /dev/hp2a
mount -r /dev/hp2a /c/nbsd
mount -r /dev/hp2g /c/nbsd/usr
cd tp
tp cmf /tmp/tape.$$ boot copy format
cd /nbsd/sys/mdec
echo "Build 1st level boot block file"
cat tsboot htboot tmboot mtboot utboot noboot noboot /tmp/tape.$$ | \
	$remote dd of=${tape} obs=512 conv=sync
cd /nbsd
sync
echo "Add dump of mini-root file system"
eval dd if=/dev/r${miniroot} count=205 bs=20b conv=sync ${remote+'|'} \
```

```
        ${remote-"of=$tape"} ${remote+'/usr/local/20b ">" $tape'}
echo "Add full dump of real file system"
/etc/${remote+r}dump 0uf $remote${remote+:}${tape} /c/nbsd
echo "Add tar image of /usr"
cd /nbsd/usr; eval tar cf ${tartape} adm bin dict doc games \
        guest hosts include lib local man mdec msgs new \
        preserve pub spool tmp ucb \
                ${remote+'| $remote /usr/local/20b ">" $tape'}
if [ ${type} != '6250' ]
then
        echo "Done, rewinding first tape"
        $remote mt -t ${tape} rew &
        echo "Mount second tape and hit return when ready"
        echo "(or type name of next tape drive)"
        read x
        if [ "$x" != "" ]
        then   tape=$x
        fi
fi
echo "Add tar image of system sources"
cd /nbsd/sys; eval tar cf ${tartape} . \
        ${remote+'| $remote /usr/local/20b ">" $tape'}
echo "Add user source code"
cd /nbsd/usr/src; eval tar cf ${tartape} Makefile bin etc games \
        include lib local old ucb undoc usr.bin usr.lib \
        ${remote+'| $remote /usr/local/20b ">" $tape'}
echo "Add varian fonts"
cd /usr/lib/vfont; eval tar cf ${tartape} . \
        ${remote+'| $remote /usr/local/20b ">" $tape'}
if [ ${type} != '6250' ]
then
        echo "Done, rewinding second tape"
        $remote mt -t ${tape} rew &
        echo "Mount third tape and hit return when ready"
        echo "(or type name of next tape drive)"
        read x
        if [ "$x" != "" ]
        then   tape=$x
        fi
fi
echo "Add user contributed software"
cd /nbsd/usr/src/new; eval tar cf ${tartape} * \
        ${remote+'| $remote /usr/local/20b ">" $tape'}
echo "Add ingres source"
cd /nbsd/usr/ingres; eval tar cf ${tartape} . \
        ${remote+'| $remote /usr/local/20b ">" $tape'}
echo "Done, rewinding tape"
$remote mt -t ${tape} rew &
```

Summarizing then, to create a distribution tape you can use the above scripts and the following commands.

 # buildmini
 # maketape
...
(For 1600bpi tapes, the following will appear twice asking you to mount
fresh tapes)
Done, rewinding first tape
Mount second tape and hit return when ready
(remove the first tape and place a fresh one on the drive)
...
Done, rewinding second tape

Control status register addresses

The distribution uses many standalone device drivers that presume the location of a UNIBUS device's control status register (CSR). The following table summarizes these values.

Device name	Controller	CSR address (octal)
ra	DEC UDA50	0172150
rb	DEC 730 IDC	0175606
rk	DEC RK11	0177440
rl	DEC RL11	0174400
tm	EMULEX TC-11	0172520
ts	DEC TS11	0172520
up	EMULEX SC-21V	0176700
ut	SI 9700	0172440

All MASSBUS controllers are located at standard offsets from the base address of the MASSBUS adapter register bank. BI bus controllers are located automatically.

Generic system control status register addresses

The *generic* version of the operating system supplied with the distribution contains the UNIBUS devices listed below. These devices will be recognized if the appropriate control status registers respond at any of the listed UNIBUS addresses.

Device name	Controller	CSR addresses (octal)
hk	DEC RK11	0177440
tm	EMULEX TC-11	0172520
tmscp	DEC TU81, TMSCP	0174500
ts	DEC TS11	0172520
ut	SI 9700	0172440
up	EMULEX SC-21V	0176700, 0174400, 0176300
ra	DEC UDA-50	0172150, 0172550, 0177550
rb	DEC 730 IDC	0175606
rl	DEC RL11	0174400
dm	DM11 equivalent	0170500
dh	DH11 equivalent	0160020, 0160040
dhu	DEC DHU11	0160440, 0160500
dz	DEC DZ11	0160100, 0160110, ... 0160170
dmf	DEC DMF32	0160340
dmz	DEC DMZ32	0160540
lp	DEC LP11	0177514
en	Xerox 3MB ethernet	0161000
ec	3Com ethernet	0164330
ex	Excelan ethernet	0164344
il	Interlan ethernet	0164000
de	DEC DEUNA	0174510

If devices other than the above are located at any of the addresses listed, the system may not bootstrap properly.

APPENDIX B – LOADING THE TAPE MONITOR

This section describes how the bootstrap monitor located on the first tape of the distribution tape set may be loaded. This should not be necessary, but has been included as a fallback measure if it is not possible to read the distributed console medium. **WARNING**: the bootstraps supplied below may not work, in certain instances on an 11/730 because they use a buffered data path for transferring data from tape to memory; consult our group if you are unable to load the monitor on an 11/730. All of the addresses given below refer to the first SBIA on the 8600.

To load the tape bootstrap monitor, first mount the magnetic tape on drive 0 at load point, making sure that the write ring is not inserted. Temporarily set the reboot switch on an 11/780 or 11/730 to off; on an 8600 or 11/750 set the power-on action to halt. (In normal operation an 11/785, 11/780, or 11/730 will have the reboot switch on, and an 8600 or 11/750 will have the power-on action set to boot/restart.)

If you have an 8600 or 11/780 give the commands:

>>>HALT
>>>UNJAM

Then, on any machine, give the init command:

>>>I

and then key in at location 200 and execute either the TS, HT, TM, or MT bootstrap that follows, as appropriate. **NOTE**: All of the addresses given in this section refer to the first SBIA on the 8600. Also, the VAX 8200 console does not accept the ‘‘D +’’ command, so the second command becomes ‘‘D 204 D05A0000’’, the third ‘‘D 208 3BEF’’, the fourth ‘‘D 20C 800CA00’’, the fifth ‘‘D 210 32EFC1’’, and so forth. Alternatively, you could try booting BOOT58 via ‘‘B/R5:800’’ with a diagnostic floppy.

The machine’s printouts are shown in boldface, explanatory comments are within ( ). You can use ‘delete’ to erase a character and ‘control U’ to kill the whole line.

TS bootstrap

```
>>> D/P 200 3AEFD0
>>> D + D05A0000
>>> D + 3BEF
>>> D + 800CA00
>>> D + 32EFC1
>>> D + CA010000
>>> D + EFC10804
>>> D + 24
>>> D + 15508F
>>> D + ABB45B00
>>> D + 2AB9502
>>> D + 8FB0FB18
>>> D + 956B024C
>>> D + FB1802AB
>>> D + 25C8FB0
>>> D + 6B
      (The next two deposits set up the addresses of the UNIBUS)
      (adapter and its memory; the 20006000 here is the address of)
      (uba0 and the 2013E000 the address of the I/O page, umem0)
       (on an 8600 or 11/780)
>>> D + 20006000              (8600/780 uba0)
       (8600/780 uba1: 20008000, uba2 2000A000)
       (8200 uba at node 0: 20000000)
```

```
        (750 uba0: F30000, uba1: F32000; 730 uba: F26000)
>>> D + 2013E000              (8600/780 umem0)
        (8600/780 umem1: 2017E000, umem2: 201BE000)
        (8200 umem at node 0: 20400000)
        (750 umem0: FFE000, umem1: FBE000; 730 umem: FFE000)
>>> D + 80000000
>>> D + 254C004
>>> D + 80000
>>> D + 264
>>> D + E
>>> D + C001
>>> D + 2000000
>>> S 200
```

N.B.: uba and umem addresses can be determined algorithmically on 8200 machines as follows:

uba(*node*) = 20000000 + (2000 * *node*)

umem(*node*) = 20400000 + (40000 * *node*)

HT bootstrap

```
>>> D/P 200 3EEFD0
>>> D + C55A0000
>>> D + 3BEF
>>> D + 808F00
>>> D + C15B0000
>>> D + C05B5A5B
>>> D + 4008F
>>> D + D05B00
>>> D + 9D004AA
>>> D + C08F326B
>>> D + D424AB14
>>> D + 8FD00CAA
>>> D + 80000000
>>> D + 320800CA
>>> D + AAFE008F
>>> D + 6B39D010
>>> D + 0
      (The next two deposits set up the addresses of the MASSBUS)
      (adapter and the drive number for the tape formatter)
      (the 20010000 here is the address of mba0 on an 8600 or)
       (11/780 and the 0 indicates the formatter is drive 0 on mba0)
>>> D + 20010000              (8600/780 mba0)
       (8600/780 mba1: 20012000; 750 mba0: F28000, mba1: F2A000)
>>> D + 0                     (Formatter unit number in range 0-7)
>>> S 200
>>> S 200
```

TM bootstrap

```
>>> D/P 200 2AEFD0
>>> D + D0510000
>>> D + 2000008F
>>> D + 800C180
```

```
>>> D + 804C1D4
>>> D + 1AEFD0
>>> D + C8520000
>>> D + F5508F
>>> D + 8FAE5200
>>> D + 4A20200
>>> D + B006A2B4
>>> D + 2A203
      (The following two numbers are uba0 and umem0 on a 8600/780)
       (See TS above for values for other CPU's and UBA's)
>>> D + 20006000          (8600/780 uba0)
>>> D + 2013E000          (8600/780 umem0)
>>> S 200
>>> S 200
>>> S 200
```

MT bootstrap

```
>>> D/P 200 46EFD0
>>> D + C55A0000
>>> D + 43EF
>>> D + 808F00
>>> D + C15B0000
>>> D + C05B5A5B
>>> D + 4008F
>>> D + 19A5B00
>>> D + 49A04AA
>>> D + AAD408AB
>>> D + 8FD00C
>>> D + CA800000
>>> D + 8F320800
>>> D + 10AAFE00
>>> D + 2008F3C
>>> D + ABD014AB
>>> D + FE15044
>>> D + 399AF850
>>> D + 6B
      (The next two deposits set up the addresses of the MASSBUS)
      (adapter and the drive number for the tape formatter)
      (the 20012000 here is the address of mba1 on an 8600 or)
       (11/780 and the 0 indicates the formatter is drive 0 on mba1)
>>> D + 20012000
>>> D + 0
>>> S 200
>>> S 200
>>> S 200
>>> S 200
```

(no functioning toggle-in code exists for the UT device)

If the tape doesn't move the first time you start the bootstrap program with ''S 200'' you probably have entered the program incorrectly (but also check that the tape is online). Start over and check your typing. For the HT, MT, and TM bootstraps you will not be able to see the tape motion as you advance through the first few blocks as the tape motion is all within the vacuum columns.

Next, deposit in R10 the address of the tape MBA/UBA and in R11 the address of the device registers or unit number from one of:

```
>>> D/G A 20006000      (for tapes on 8600/780 uba0)
>>> D/G A 20008000      (for tapes on 8600/780 uba1)
>>> D/G A 20010000      (for tapes on 8600/780 mba0)
>>> D/G A 20012000      (for tapes on 8600/780 mba1)
>>> D/G A 20000000      (for tapes on 8200 uba at node 0)
>>> D/G A F30000        (for tapes on 750 uba0)
>>> D/G A F32000        (for tapes on 750 uba1)
>>> D/G A F28000        (for tapes on 750 mba0)
>>> D/G A F2A000        (for tapes on 750 mba1)
>>> D/G A F26000        (for tapes on 730 uba0)
```

and for register 11:

```
>>> D/G B 0             (for TM03/TM78 formatters at mba? drive 0)
>>> D/G B 1             (for TM03/TM78 formatters at mba? drive 1)
>>> D/G B 2013F550      (for TM11/TS11/TU80 tapes on 8600/780 uba0)
>>> D/G A 20400000      (for TM11/TS11/TU80 on 8200 uba at node 0)
>>> D/G B FFF550        (for TM11/TS11/TU80 tapes on 750 or 730 uba0)
```

Then start the bootstrap program with

```
>>> S 0
```

The console should type

```
=
```

You are now talking to the tape bootstrap monitor. At any point in the following procedure you can return to this section, reload the tape bootstrap, and restart the procedure. The console monitor is identical to that loaded from a TU58 console cassette, follow the instructions in section 2 as they apply to this device. The only exception is that when using programs loaded from the tape bootstrap monitor, programs will always return to the monitor (the ‘‘=’’ prompt). This saves your having to type in the above toggle-in code for each program to be loaded.