Installing and Operating 4.3BSD on the VAX
April 1, 1985

Michael J. Karels

M. Kirk McKusick

James M. Bloom

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 release of the VAX* UNIX** system, as distributed by U. C. Berkeley.

It discusses procedures for installing UNIX on a new VAX, and for upgrading an existing 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 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.

1. INTRODUCTION

This document explains how to install the 4.3BSD release of the Berkeley version of UNIX for the VAX on your system. Because of the file system organization used in 4.3BSD, if you are not currently running 4.2BSD you will have to do a full bootstrap from the distribution tape. The suggested technique for upgrading a 4.2BSD system is described in chapter 3 of this document; the techniques for converting ‘‘older’’ systems are also discussed in a chapter 3.

1.1. Hardware supported

This distribution can be booted on a VAX 11/785, VAX 11/780, VAX 11/750, VAX 11/730 or VAX 11/725 cpu with any of the following disks:

DEC MASSBUS:	RM03, RM05, RM80, RP06, RP07
EMULEX MASSBUS:	AMPEX 300M, 330M, CDC 300M, FUJITSU 404M
DEC UNIBUS:	RK07, RL02, RA80, RA81, RA60, RC25
EMULEX SC-21V UNIBUS*:	AMPEX 80M, 300M, 330M, CDC 300M, FUJITSU 160M, 330M
EMULEX SC-31 UNIBUS*:	FUJITSU 404M
DEC IDC:	R80, RL02

The tape drives supported by this distribution are:

DEC MASSBUS:	TE16, TU45, TU77, TU78
EMULEX MASSBUS:	TC-7000
DEC UNIBUS:	TS11, TU80
EMULEX TC-11, AVIV UNIBUS:	KENNEDY 9300, CIPHER
TU45 UNIBUS*:	SI 9700

The tapes and disks may be on any available UNIBUS or MASSBUS adapter at any slot with the proviso that the tape device must be slave number 0 on the formatter if it is a MASSBUS tape drive.

1.2. Distribution format

The basic distribution contains the following items:

(3) 1600bpi 2400’ magnetic tapes, or
(1) 6250bpi 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; their intended use is only for installation.

The distribution does not fit on several standard VAX configurations that contain only small disks. If your hardware configuration does not provide at least **75** Megabytes of disk space you can still install the distribution, but you will probably have to operate without source for the user level commands and, possibly, the source for the operating system. The previous RK07-only distribution format provided by our group is no longer available. Further, no attempt has ever been made to install the system on the

* Other UNIBUS controllers and drives may be easily usable with the system, but will likely 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.

standard VAX-11/730 hardware configuration from DEC that contains only dual RL02 disk drives (though the distribution tape may be bootstrapped on an RL11 controller and the system provides support for RL02 disk drives either on an IDC or an RL11). The labels on the distribution tape(s) show the amount of disk space each tape file occupies, these should be used in selecting file system layouts on systems with little disk space.

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 are 9-track 1600 BPI or 6250 BPI and 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 basic bootstrap material is present in three short files at the beginning of the first tape. 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 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 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 gives devices names by the names of the driver in the system using a naming 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 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 adapter ‘‘A’’, the second ‘‘B’’, etc. Thus ‘‘DRA’’ is a 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, i.e. ‘‘DRA0’’ is a disk at the first device slot on the first MASSBUS adapter and is a RM disk.

1.4. UNIX device naming

UNIX has a set of names for devices, which are different from the DEC names for the devices, viz.:

** References of the form X(Y) mean the subsection named X in section Y of the UNIX programmer’s manual.

RM/RP disks	hp
TE/TU tapes	ht
TU78 tape	mt

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

xx(y,z)

where *xx* is either **hp**, **ht**, or **mt**. The value *y* specifies the MASSBUS to use and also the device. It is computed as

8 * mba + device

Thus mba0 device 0 would have a *y* value of 0 while mba1 device 0 would have a *y* value of 8. The *z* value is interpreted differently for tapes and disks: for disks it is a disk *partition* (in the range 0-7), and for tapes it is a file number on the tape.

Each UNIX physical disk is divided into 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 in section 4 of the programmers manual and in the disk description file /etc/disktab (c.f. *disktab*(5)).* 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: it allows access to the entire physical device, including the 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 when using this partition to not overwrite the last few tracks and thereby clobber the bad sector information.

The disk partitions have names in the standalone system of the form ''hp(y,z)'' with varying *y* as described above. Thus partition 1 of a RM05 on mba0 at drive 0 would be ''hp(0,1)''. When not running standalone, this partition would normally be available as ''/dev/hp0b''. Here the prefix ''/dev'' is the name of the directory where all ''special files'' normally live, the ''hp'' serves an obvious purpose, the ''0'' identifies this as a partition of hp drive number ''0'' and the ''b'' identifies this as the first partition (where we number from 0, the 0'th partition being ''hp0a''.)

In all simple cases, 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. 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. We will not discuss that now.

Returning to the discussion of the standalone system, we recall that tapes also took two integer parameters. In the normal case where the tape formatter is unit 0 on the second mba (mba1), the files on the tape have names ''ht(8,0)'', ''ht(8,1)'', etc. Here ''file'' means a tape file containing a single data stream. The distribution tape(s) have data structures in the tape files and though the tape(s) contain only 9 tape files, they contain several thousand UNIX files.

For the UNIBUS, there are also conventional names. The important DEC names to know are DM?? for RK07 drives and DU?? for drives on a UDA50. For example, RK07 drive 0 on a controller on the first UNIBUS on the machine is ''DMA0''. UNIX calls such a device a ''hk'' and the standalone name for the first partition of such a device is ''hk(0,0)''. If the controller were on the second UNIBUS its name would be ''hk(8,0)''. If we wished to access the first partition of a RK07 drive 1 on uba0 we would use

* It is possible to change the partitions by changing the code for the table in the disk driver; it is often desirable to do this, therefore these tables should be read off each pack; they may be in a future version of the system.

‘‘hk(1,0)’’.

The UNIBUS disk and tape names used by UNIX are:

RK disks	hk
TS tapes	ts
UDA disks	ra
RC25 disks	ra
IDC disks	rb
SMD disks	up
TM tapes	tm
TU tapes	ut

Here SMD disks are disks on an RM emulating controller on the UNIBUS, and TM tapes are tapes on a controller that emulates the DEC TM-11. TU tapes are tapes on a controller that emulates the DEC TU45. IDC disks are disks on an 11/730 Integral Disk Controller. TS tapes are tapes on a controller that emulates the DEC TS-11 (e.g. a TU80). The naming conventions for partitions in UNIBUS disks and files in UNIBUS tapes are the same as those for MASSBUS disks and tapes.

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’’. In the bootstrap procedures we will often suggest using the raw devices, because these tend to work faster in some cases. In general, however, the block devices are used. They are where file systems are ‘‘mounted’’.

You should be aware that it is sometimes important 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

This section explains the bootstrap procedure that can be used to get the kernel supplied with this distribution running on your machine. If you are not currently running 4.2BSD you will have to do a full bootstrap.

If you are currently running UNIX you should first save your existing files on magnetic tape. 4.3BSD uses the file system organization introduced in 4.2BSD. This organization is different from the organization used in systems that predate 4.2BSD; it may therefore be necessary to rebuild the file system format before restoring the data. The easiest way to save the current files on tape is by doing a full dump and then restoring under the new system. Refer to chapter 3 in understanding how to upgrade an existing 4BSD system.

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 or cassette for bootstrapping.
6) Reboot the completed root file system.
7) Build and restore the /usr file system from tape with *tar* (1).

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.

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* is a name 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 Roman, while that information printed by the system is shown emboldened. Throughout the installation steps the reboot switch on an 11/785, 11/780 or 11/730 should be set to off; on an 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 11/750 will have the power-on action set to reboot/restart.)

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

2.1. Step 1: formatting the disk

All disks used with 4.3BSD 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 300M, 330M, CDC 300M, FUJITSU 330M, 404M
EMULEX SC-21V UNIBUS:	AMPEX 300M, 330M, CDC 300M, FUJITSU 160M, 330M
EMULEX SC-31 UNIBUS:	FUJITSU 404M

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 11/785, 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 11/785 or 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 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 by using the disk controller’s ‘‘write check’’ command. Remember *format* runs only on the **up** and **hp** drives listed above. *Format* will prompt for the information required as shown below. If you err in answering questions, ‘‘#’’ erases the last character typed, and ‘‘@’’ erases the current input line.

Enable debugging (0=none, 1=bse, 2=ecc, 3=bse+ecc)?
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
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 (takes several hours)
Pattern (one of the above, other to restart)? *2*
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:
Write check: 4
Bad sector: 0
ECC: 0
Skip sector: 0
Total of 4 hard errors found.
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 an 11/785 or 11/780, 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 46 passes to detect as many disk surface errors as possible; this test runs for many hours, depending on the CPU and controller. On an 11/785 or 11/780, this can be sped up significantly by setting the clock fast.

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 not to 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 11/785 or 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:	

while 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, '#' erases characters and '@' erases lines.)

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'':

(load bootstrap program)
>>> LOAD BOOT
>>> START 2
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
>>> S 2
Boot
: *xx*(x,1)vmunix (bring in *vmunix* off mini root)

(As above, '#' erases characters and '@' 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:

```
251840+74968+84344 start 0x120c
4.3 BSD UNIX #1: Wed Jun 26 17:31:23 PDT 1985
   bloom@ucbmonet:/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 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 only 512K bytes of memory, then xxx will be 520192, 4096 bytes less than 512K. The system reserves the last 4096 bytes of memory for use in error logging and doesn't count it as part of real memory.

The messages that came 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 to configure UNIX from. 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 *xx*0*. This response supplies two pieces of information: first, *xx*0 shows 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 ˆ?, kill ˆU, intr ˆC
#
```

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 things are consistent with the DEC console interface characters.

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 *mknod*(8).

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
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 a 780 or 785, update the floppy
If this is a 730, update the cassette
#
```

2.5. Step 5: creating a boot floppy or cassette

If the machine is an 11/785, 11/780 or 11/730, a boot floppy or cassette 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 *newfs* program automatically installs the needed code, so you may continue with the next step. On an 11/785 or 11/780 with interleaved memory, or other configurations that require alteration of the standard boot files, this step may be left for later.

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)
# ˆP	(halt machine)
>>> HALT	(for 11/785's or 11/780's only)
>>> UNJAM	(for 11/785's or 11/780's only)
>>> I	(initialize processor state)
>>> B *xx*S	(on an 11/750, use B/2)
...(boot program is eventually loaded)...	
Boot	
: xx(x,0)vmunix	(*vmunix* brought in off root)
251840+74968+84344 start 0x120c	
4.3 BSD UNIX #1: Wed Jun 26 17:31:23 PDT 1985	
bloom@ucbmonet:/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 ˆ?, kill ˆU, intr ˆC	
#	

(see section 6.1 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.

2.7. Step 7: setting up the /usr file system

First set a shell variable to the name of your disk, so the commands we give will work regardless of the disk you have; do one of

# disk=hp	(if you have an RP06, RM03, RM05, RM80, or other MASSBUS drive)
# disk=hk	(if you have RK07s)
# disk=ra	(if you have UDA50 storage module drives)
# disk=up	(if you have UNIBUS storage module drives)
# disk=rb	(if you have IDC storage module drives)

The next thing to do is to extract the rest of the data from the tape. You might wish to review the disk configuration information in section 4.4 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	# name=hp0g; type=rm03
DEC RM05	# name=hp0g; type=rm05
DEC RM80	# name=hp0g; type=rm80
DEC RP06	# name=hp0g; type=rp06
DEC RP07	# name=hp0h; type=rp07
DEC RK07	# name=hk0g; type=rk07
DEC RA80	# name=ra0h; type=ra80
DEC RA60	# name=ra0h; type=ra60
DEC RA81	# name=ra0h; type=ra81
DEC R80	# name=rb0h; type=rb80
UNIBUS CDC 9766	# name=up0g; type=9766
UNIBUS AMPEX 300M	# name=up0g; type=9300
UNIBUS AMPEX 330M	# name=up0g; type=capricorn
UNIBUS FUJITSU 160M	# name=up0g; type=fuji160
UNIBUS FUJITSU 330M	# name=up0g; type=capricorn
UNIBUS FUJITSU 404M	# name=up0h; type=eagle
MASSBUS CDC 9766	# name=hp0g; type=9766
MASSBUS AMPEX 300M	# name=hp0g; type=9300
MASSBUS AMPEX 330M	# name=hp0g; type=capricorn
MASSBUS FUJITSU 330M	# name=hp0g; type=capricorn
MASSBUS FUJITSU 404M	# name=hp0h; type=eagle

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
EMULEX TC11	# cd /dev; MAKEDEV tm0; sync
SI 9700	# cd /dev; MAKEDEV ut0; sync

Then execute the following commands

# date *yymmddhhmm*	(set date, see *date* (1))
....	
# passwd root	(set password for super-user)
New password:	(password will not echo)
Retype new password:	
# 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)	

The data on the fourth tape files 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 installation procedure continues this point on the 6250bpi tape.

# 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 /usr)
# 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
# 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)).

2.8. Additional software

There are three extra tape files on the distribution tape(s) which have not been installed to this point. They are a font library for use with Varian and Versatec printers, the Ingres database system, and user contributed software. All three tape files are in *tar*(1) format and 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 fonts should be placed in a directory /usr/lib/vfont, the Ingres system should be placed in /usr/ingres, and the user contributed software should be placed in /usr/src/new. The exact contents of the user contributed software is given in a separate document.

3. UPGRADING A 4BSD SYSTEM

Begin by reading the other parts of this document to see what has changed since the last time you bootstrapped the system. Also read the ‘‘Changes in 4.3BSD’’ document, and look at the new manual sections provided to you. If you have local system modifications to the kernel to install, look at the document ‘‘Kernel changes in 4.3BSD’’ to get an idea of how the system changes will affect your local mods.

If you are running 4.2BSD, upgrading your system involves replacing all your source and binary files. The easiest way to do this (depending on your file system configuration) is to bootstrap the system as described in chapter 2, then bring in the new source files and incorporate your local modifications.

If you are running a version of the system distributed before 4.0BSD, you are pretty much on your own. Sites running 3BSD or 32/V may be able to modify the restor program to understand the old 512 byte block file system, but this has never been tried. Section 3.1 provides general information for upgrading older systems; sections 3.2-3.4 assume you are running 4.1BSD; section 3.5 provides information specific to upgrading a 4.2BSD system.

3.1. Step 1: what to save

If you are not running 4.2BSD, you will have to rebuild your root and usr file systems. The easist way to do this is to save the important files on your existing system, do a bootstrap as if you were installing 4.3BSD on a brand new machine, then merge the saved files into the new system. If you are running 4.2BSD the easiest upgrade path is to build new root and usr file systems in which case you will want to save the files listed here so that they may be moved to the new system. 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; be sure to do a tar of the directories /etc, /lib, and /usr/lib to guard against your missing something the first time around.

/.profile	root sh startup script
/.login	root csh startup script
/.cshrc	root csh startup script
/dev/MAKE	for the LOCAL case for making devices
/etc/fstab	disk configuration data
/etc/group	group data base
/etc/passwd	user data base
/etc/rc	for any local additions
/etc/securetty	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/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/lint/*	for locally developed lint libraries
/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

Almost all 4.2BSD binary images will run unchanged under 4.3BSD. 4.1BSD binary images can also run unchanged under 4.3BSD but only when the system is configured to include the ‘‘4.1BSD compatibility mode’’*. You should be certain to save any programs such as compilers that you will need in bootstrapping to 4.3BSD.*

If you are upgrading a 4.1BSD system, and have saved the appropriate files in a convenient format, the next 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 in-place by using a disk partition. If you select the latter tact, a version of the 4.1BSD dump program that runs under 4.3 is provided in /etc/dump.4.1; be sure to read through this entire document before beginning the conversion. Beware that file systems created under 4.3BSD 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 most all programs will likely 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 file system without problem.

Once you have dumped the file systems you wish to convert to 4.3BSD, install the system from the bootstrap tape as described in chapter 2, then continue with the next section.

---

* 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 it is **STRONGLY RECOMMENDED** that the programs be recompiled under the new system. Refer to the document ‘‘Changes in 4.3BSD’’ for elaboration on this point.

3.2. Step 2: merging

When your system is booting reliably and you have the 4.3BSD 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.

Using the tar tape, or tapes, you created in step 1 extract the appropriate files into a scratch directory, say /usr/convert:

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

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

```
# cp passwd group fstab /etc
```

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

The commands kept under the LOCAL entry in /dev/MAKE should be placed in the new shell script /dev/MAKEDEV.local so that saying ‘‘MAKEDEV LOCAL’’ will create the appropriate local devices and device names. If you have any homegrown device drivers 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.

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
```

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.

3.3. Step 3: converting file systems

The dump format used in 4.0 and 4.1BSD is upward compatible with that used in 4.3BSD (it is unchanged from 4.2BSD). That is, the 4.3BSD *restore* program understands how to read old dump tapes, although 4.3BSD dump tapes may not be properly restored under 4.0BSD or 4.1BSD. To convert a file system dumped to magtape, simply create the appropriate file system and restore the data. Note that the 4.3BSD *restore* program does its work on a mounted file system using normal system operations (unlike the older *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
# newfs hp1g eagle
# mount /dev/hp1g /a
# cd /a
# restore r
```

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.

3.4. Bootstrapping language processors

To convert a compiler from 4.1BSD to 4.3BSD 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. 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 for assembly and loading.

3.5. Converting a 4.2BSD system

In addition to the files described in section 3.1, the following files, specific to 4.2, should be saved for use in the conversion process:

/.rhosts	for trusted machines and users
/dev/MAKEDEV	in case you added anything here
/dev/MAKEDEV.local	for making local devices
/etc/ftpusers	for local additions
/etc/gateways	routing daemon database
/etc/gettytab	getty database
/etc/hosts	for local host information
/etc/hosts.equiv	for local host equivalence information
/etc/networks	for local network information
/etc/printcap	line printer database
/etc/rc.local	site specific system startup commands
/etc/remote	auto-dialer configuration
/etc/services	for local additions
/etc/syslog.conf	system logger configuration
/usr/hosts/MAKEHOSTS	for local changes
/usr/lib/sendmail.cf	sendmail configuration
/*/quotas	file system quota files

All the files listed above should be usable without changes under 4.3BSD. You may find changes are needed to take advantage of some of the new facilities in 4.3BSD; refer to chapter 4 of this document, in particular, for insight.

4. SYSTEM SETUP

This section describes procedures used to setup a VAX UNIX system. Procedures described here 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. Making a UNIX boot floppy

If you have an 11/785 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 *flcopy*(8) and *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

Otherwise:

cp defboo.hp defboo.cmd

If the local configuration requires any changes in restar.cmd or defboo.cmd (e.g., for interleaved memory controllers), 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).

4.2. 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).

4.3. 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’’.

4.3.1. 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/net	network independent, but network related code
/sys/netinet	DARPA Internet code
/sys/netimp	IMP support code
/sys/netns	Xerox NS support code
/sys/vax	VAX specific mainline code
/sys/vaxif	VAX network interface code
/sys/vaxmba	VAX MASSBUS device drivers and related 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.

4.3.2. Devices and device drivers

Devices supported by UNIX are implemented in the kernel by drivers whose source is kept in /sys/vax, /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 dz-11, a single dh-11, a single dmf-32, an rm03 disk, an EMULEX 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.

4.3.3. 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 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 old 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. A systematic scheme for numbering and saving old versions of the system is best.

4.4. 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.

4.4.1. 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 initial information about the usage of disk partitions, which we see how to update more below.

4.4.2. 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 RM03 or RP06, 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 RM03, 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 default 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*.

4.4.3.

The disk partition sizes for DEC RA60, RA80, and RA81 have changed since 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 update /etc/disktab and the device driver for the UDA50. Any other partition sizes that were modified at your site, will require the same consideration. Space available

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.

* These rules are, unfortunately not evenly applied to all disks. Drives on DEC UDA50 and IDC controllers do not completely follow these rules; in particular, the swap partition on an RA81 is only 33440 sectors, and no ‘‘d’’, ‘‘e’’, or ‘‘f’’ partitions are available on an RA80. Consult *uda* (4) for more information.

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	--	ra?h	93 Mb
ra81	ra?g	324 Mb	ra?h	96 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
hp400	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 hp400 refers to a FUJITSU Eagle 400 Megabyte disk on a MASBUS or UNIBUS disk controller. ‘‘hp’’ should be substituted for ‘‘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 of 16 Megabytes, and a root file system of 8 Megabytes. The distributed system binaries occupy about 26 Megabytes while the major sources occupy another 25 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 (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 making file systems specifies disk partition sizes in sectors; the default sector size may be overridden with the ‘‘se’’ attribute. Note that only the sector size currently supported is DEV_BSIZE as defined in /sys/h/param.h.

4.4.4. Layout considerations

There are several considerations in deciding how to adjust the arrangement of things on your disks: the most important is making sure 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 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
/	1	2	2
tmp	1	3	4
usr	1	1	1
paging	1+2	1+3	1+3+4
users	2	2+3	2+3
archive	x	x	4

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 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.

4.4.5. 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	512 bytes
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. 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 Kbytes 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 uses 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. Space compaction has been deemed important here because the source code for the system is normally placed on this file system.

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 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 a 512 byte fragment size, storage fragmentation occurs much sooner, particularly with a busy file system running near full capacity. By comparison, the level of fragmentation in a 1 Kbyte fragment 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*, 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 default values supplied. 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. When constructing a file system you should use the *newfs*(8) program and specify the type of disk on which the file system resides. 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 or use an option to *newfs*.

4.4.6. 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 hp0g, 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 hp0g partition with the system source and binaries.

---

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

If we had a second rm80, we would create a file system in hp1g and put user files there, calling the file system /mnt. 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 add the lines

```
/dev/hp1b::sw::
/dev/hp1g:/mnt:rw:1:2
```

to the /etc/fstab file. We would keep a backup copy of the root file system in the **hp1a** disk partition.

To make the /mnt file system we would do:

```
# cd /dev
# MAKEDEV hp1
# newfs hp1g rm80
(information about file system prints out)
# mkdir /mnt
# mount /dev/hp1g /mnt
```

4.5. Configuring terminals

If UNIX is to support simultaneous access from more than just the console terminal, the file /etc/ttys (*ttys* (5)) has to be edited.

Terminals connected via dz 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. Lines on dh or dmf interfaces are conventionally named **ttyh**X, where X is a hexadecimal digit. If more than one dh or dmf interface is present in a configuration, successive terminals would be named **ttyi**X, **ttyj**X, etc.

To add a new terminal, be sure the device is configured into the system and that the special file for the device has been made by /dev/MAKEDEV. Then, enable the appropriate line of /etc/ttys by setting the ‘‘status’’ field to **on** (or add a new line).

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 not 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. See *dh* (4), *dz* (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 when here.)

When the system is running multi-user, all terminals that are listed in /etc/ttys as **on** have their line are enabled. If, during normal operations, it is desired 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-f], 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.

4.6. Adding users

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

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

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.

4.7. 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/rc.local

The first line in /etc/rc.local,

/bin/hostname *mysitename*

defines the value returned by the *gethostname*(2) system call. Programs such as *getty*(8), *mail*(1), *wall*(1), *uucp*(1), and *who*(1) use this system call so that the binary images are site independent.

4.8. Setting up the line printer system

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

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

The file /etc/printcap is a master data base 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.

4.9. Setting up the mail system

The mail system consists of the following commands:

/bin/mail	old standard mail program (from 32/V)
/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, 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 any possible perusal (except by a code-breaker) you should use the secret mail facility, which encrypts the mail so that no one can read it.

To setup 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.

4.9.1. Setting up a uucp connection

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

* 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.

- support for many auto call units other than 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.

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 volume 2B of the Unix Programmers 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

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/uuclean	cleans up garbage files in spool area
/usr/lib/uucp/uucico	*uucp* protocol daemon
/usr/lib/uucp/uuxqt	*uucp* remote execution server

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

/usr/spool/uucp/C.	directory for command, ‘‘C.’’ files
/usr/spool/uucp/D.	directory for data, ‘‘D.’’, files
/usr/spool/uucp/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 (less than 8 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 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 networking support.

5.1. System configuration

To configure the kernel to include the Internet communication protocols, define the INET option and include the pseudo-devices ''inet'', ''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 mail system.

If you are planning to use the 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, 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
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)
hy	NSC Hyperchannel, w/ DR-11B and PI-13 interfaces
il	Interlan 10Mb/s Ethernet
pcl	DEC PCL-11
un	Ungermann-Bass network w/ DR-11W interface (not tested)
vv	Proteon ring network (V2LNI)

All network interface drivers require at least part of their host address be defined at boot time. This is done with *ifconfig*(8C) commands included in the /etc/rc.local file. Interfaces that are able to dynamically deduce the host part of an address, but not the network number, take the network number from the address specified with *ifconfig*. Hosts that use a more complex address mapping scheme, such as the Address Resolution Protocol, *arp*(4), require the full address. The manual page for each network interface describes the method used to establish a host's address. *Ifconfig*(8) can also be used to set options for the interface at boot time. These options include disabling the use of the Address Resolution Protocol and the use of trailer encapsulation; this is useful if a network is shared with hosts running software that is unable to provide these functions. Options are set independently for each interface, and apply to all packets sent using that interface. An alternative approach to ARP is to divide the address range, using ARP only for those addresses below the cutoff and using another mapping above this constant address; see the source (/sys/netinet/if_ether.c) for more information.

To use the pseudo terminals just configured, device entries must be created in the /dev directory. To create 16 pseudo terminals (plenty, unless you have a heavy network load) execute the following commands.

```
# cd /dev
# MAKEDEV pty0
```

More pseudo terminals may be made by specifying *pty1*, *pty2*, 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 the *script*(1) program. 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).

When configuring multiple networks some thought must be given to the ordering of the devices in the configuration file. The first network interface configured in the system is used as the default network when the system is forced to assign a local address to a socket. This means that your most widely known network should always be placed first in the configuration file. For example, hosts attached to both the ARPANET and our local area network have devices configured in the order show below.

```
device      acc0    at uba? csr 0167600 vector accrint accxint
device      en0     at uba? csr 0161000 vector enxint enrint encollide
```

5.2. Local sub-networks

In 4.3BSD the DARPA Internet support includes the notion of ‘‘sub-networks’’. This is a mechanism by which multiple local networks may appears as a single Internet network to off-site hosts. Sub-networks are useful because they allow a site to hide their local topology; it also means local network numbers may be locally administered.

To setup local sub-networks one must first decide how the available address space (the Internet ‘‘local part’’) is to be partitioned. Sites with a class A network number have a 24-bit address space with which to work, sites with a class B network number have a 16-bit address space, while sites with a class C network number have an 8-bit address space*. To define local sub-networks 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 local part (host number), another 16 bits. To define 255 local sub-networks, each possessing at most 255 hosts, 8-bits may be taken from the local part. These new network numbers are then constructed by concatenating the original 16-bit network number with the extra 8-bits containing the local sub-network number.

Local sub-networks are communicated to the system at the time a network interface is configured with the *ifconfig* program. A ‘‘network mask’’ is specified to define the portion of the Internet address to consider when looking for local sub-networks. This mask must contain the assigned network number as well as the portion of the local part that has been usurped for defining local sub-networks. For example, at Berkeley (class B network 128.32) 8-bits of the local part have been reserved for defining sub-networks; consequently the /etc/rc.local file contains lines of the form

```
/etc/ifconfig en0 netmask 0xffffff00 128.32.7
```

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 ‘‘128.32.7’’ (host 7 on network 128.32). 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’’.

* If you are unfamiliar with the Internet addressing structure, consult ‘‘Address Mappings’’, Internet RFC 796, J. Postel; available from the Internet Network Information Center, SRI, Ca.

5.3. Network servers

In 4.3BSD most of the server programs 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 under which the server should run, the server program's name, and at most five arguments to pass to the server program. For example, an entry for the 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.

5.4. 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/rc.local	*rc* (8)	command script for starting servers
/etc/ftpusers	*ftpd* (8C)	list of ‘‘unwelcome’’ ftp users

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 next section). Network numbers will have to be chosen for each Ethernet. For sites not connected to the Internet, these can be chosen more or less arbitrarily, otherwise the normal channels should be used for allocation of network numbers.

5.4.1. Regenerating /etc/hosts and /etc/networks

The 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 /etc/htable to convert it to the format used by the libraries.

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

The *htable* program generates two files in the local directory: *hosts* and *networks*. 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*. It is usually wise to run *diff* (1) on the new host and network data bases before installing them in /etc.

5.4.2. /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 hosts'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 constrain 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. For example, most machines on our major local network are considered trusted, so the hosts.equiv file is of the form:

```
ucbarpa
ucbcalder
ucbdali
ucbernie
ucbkim
ucbmatisse
ucbmonet
ucbvax
ucbmiro
ucbdegas
```

5.4.3. /etc/rc.local

Most network servers are automatically started up at boot time by the command file /etc/rc (if they are installed in their presumed locations) or by the Internet daemon (see above). These include the following:

Program	Server	Started by
/etc/rshd	shell server	inetd
/etc/rexecd	exec server	inetd
/etc/rlogind	login server	inetd
/etc/rwhod	system status daemon	/etc/rc
/etc/syslogd	error logging server	/etc/rc

To have other network servers started up as well, commands of the following sort should be placed in the site dependent file /etc/rc.local, or the appropriate line should be added to the Internet daemon's configuration file /etc/inetd.conf.

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

The following servers are included with the system and should be installed in the location specified as the need arises.

Program	Server
/etc/telnetd	TELNET server
/etc/ftpd	FTP server
/etc/tftpd	TFTP server
/usr/lib/sendmail	SMTP server
/etc/fingerd	Finger server
/etc/courierd	Courier remote procedure call server
/etc/routed	routing table management daemon

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

5.4.4. /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 must be supplied the server process for it to execute properly. Further, one must be sure that all directories and executable images are unwritable. The following directory setup is 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
# cp /etc/passwd /etc/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.

Another issue to consider is the copy of /etc/passwd placed here. It may be copied by users who use the anonymous account. They may then try to break the passwords of users on your machine for further access. A good choice of users to include in this copy might be root, daemon, uucp, and the ftp user. All passwords here should probably be ‘‘*’’.

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 with nonstandard shells should be listed in this file. Accounts without passwords need not be listed in this file, the ftp server will not service these users.

5.5. Routing and gateways/bridges

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 /etc/htable, the routing daemon can also be used to initialize static routes to distant networks. When the routing daemon is started up (usually from /etc/rc.local) it reads /etc/gateways 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/rc.local; 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 table redirect information, so it is possible to combine both 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.

6. SYSTEM OPERATION

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

6.1. Bootstrap and shutdown procedures

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

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

```
>>> B
```

will boot from the default device. On an 11/785 or 11/780 (11/730) the default device is determined by a ‘‘DEPOSIT’’ command stored on the floppy (cassette) in the file ‘‘DEFBOO.CMD’’; 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 a 785, 780, or 730 by doing

>>> B *XX* S

where *XX* is one of HP, HK, UP, RA, or RB for a 730. The corresponding command on an 11/750 is

```
>>> B/2
```

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 a 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 after, e.g., a ‘‘B HPS’’ on an 11/785 or 11/780, ‘‘B RBS’’ on a 11/730, or a ‘‘B/2’’ on an 11/750 all you have to do is hit ˆD on the console. The system will then execute /etc/rc, a multi-user restart script (and /etc/rc.local), and come up on the terminals listed as active in the file /etc/ttys. See *init*(8) and *ttys*(5). Note, however, that this does not cause a 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.

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 up 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
# ˆD
```

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

6.2. Device errors and diagnostics

When errors occur on peripherals or in the system, the system prints a warning diagnostic on the console. These messages are collected regularly by the system error logging process and written into a system error log file /usr/adm/messages.

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 ‘‘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's often enough that daily's always fit on one tape and never get to the sequence of 9's in the daily level numbers.

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 facist approach, try using the programs *du* (1), *df* (1), *quot* (8), combined with threatening messages of the day, and personal letters.

6.4. Moving file system data

If you have the equipment, the best way to move a file system is to dump it to magtape using *dump* (8), use *newfs* (8) to create the new file system, and restore the tape, using *restore* (8). If for some reason you don't want to use magtape, dump accepts an argument telling where to put the dump; you might use another disk. 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. What you do is the following:

1. GET A SECOND PACK!!!!
2. Dump the root file system to tape using *dump* (8).
3. Bring the system down and mount the new pack.
4. Load the distribution tape and install the new root file system as you did when first installing the system.
5. 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/stand and the default disk partition tables in /etc/disktab should be modified.

6.5. Monitoring System Performance

The *systat* program provided with the system is designed to be an aid to monitoring systemwide activity. The default ''pigs'' mode shows a dynamic ''ps''. By running in the ''vmstat'' mode when the system is active you can judge the system activity in several dimensions: job distribution, virtual memory load, paging and swapping activity, 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 a several non-dma devices or open teletype lines that are ''ringing'', or user programs that are doing high-speed non-buffered input/output, then the system time may go high (60-70% or higher). It is often possible to pin down the cause of high system time by looking to see if there is excessive context switching (cs), interrupt activity (in) or system call activity (sy). 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 /usr/man/man1/date.1
# 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 /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
```

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 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 steps are recommended.

```
# cd /usr/src
# cd lib; make install
# cd ..
# make usr.lib
# cd usr.lib; make install
# cd ..
# make bin etc usr.bin ucb games local
# for i in bin etc usr.bin ucb games local; do (cd $i; make install); done
```

This will take about 12 hours on a reasonably configured 11/750.

6.7. Making local modifications

To keep track of changes to system source we migrate changed versions of commands in /usr/src/bin, /usr/src/usr.bin, and /usr/src/ucb in through the directory /usr/src/new and out of the original directory into /usr/src/old for a time before removing them. 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 wishing to use /usr/local commands are made aware that they aren't in the base manual. As manual updates incorporate these commands they are moved to /usr/ucb.

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

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, and 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 disc 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 disc 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 disc quotas.

To use the disc 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 disc quotas (a ''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 and the user logs in again, under 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 all 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	disk partition sizes
/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/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/shutdownlog	log of system reboots
/usr/adm/wtmp	login session accounting

APPENDIX A – BOOTSTRAP DETAILS

This appendix contains pertinent files and numbers regarding the bootstrapping procedure for 4.3BSD. 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	202	8 bootstrap monitor programs and a *tp* (1) file containing *boot*, *format*, and *copy*
two	205	‘‘mini root’’ file system
three	414	*dump* (8) of distribution root file system
four	2649	*tar* (1) image of binaries and libraries in /usr

The second tape contains the following files.

Tape file	# Records	Contents
one	582	*tar* (1) image of /sys, including GENERIC system
two	2448	*tar* (1) image of /usr/src
three	571	*tar* (1) image of /usr/lib/vfont

The third tape contains the following files.

Tape file	# Records	Contents
one	2455	*tar* (1) image of user contributed software
two	665	*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 may not be precisely that shown in this table; these values reflect the contents of the distribution tape at the time this document was written.

```
#!/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	4.17 (Berkeley) 6/23/85
#
# 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
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/cassette/[Ma-z0-9]* sys/cassette
cp $DISTROOT/sys/stand/boot boot
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/${disk}a /a
cd /a
echo 'Rewind tape'
mt -t /dev/${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 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
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      4.24 (Berkeley) 6/28/85
#
#       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.$$; exit" 0 1 2 3 13 15
$remote mt -t ${tape} rew
date
umount /dev/hp2g
umount /dev/hp2a
mount -r /dev/hp2a /nbsd
mount -r /dev/hp2g /nbsd/usr
cd /nbsd/tp
tp cmf /tmp/tape.$$ 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 ${remote+"| $remote dd obs=20b"} \
        of=${tape} conv=sync
echo "Add full dump of real file system"
/etc/${remote+r}dump 0uf $remote${remote+:}${tape} /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.

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
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
dn	DEC DN11	0160020
dm	DM11 equivalent	0170500
dh	DH11 equivalent	0160040
dz	DEC DZ11	0160100, 0160110, ... 0160170
ts	DEC TS11	0172520
dmf	DEC DMF32	0160340
lp	DEC LP11	0177514
en	Xerox 3MB ethernet	0161000
ec	3Com ethernet	0164330
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.

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 11/785, 11/780, or 11/730 to off; on an 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 11/750 will have the power-on action set to boot/restart.)

If you have an 11/785 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. The machine's printouts are shown in boldface, explanatory comments are within ( ). (You can use 'delete' to delete 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 on an 11/785 or 11/780 and the 2013E000 the address of)
         (umem0 on an 11/785 or 11/780)
>>> D + 20006000                (785/780 uba0)
         (785/780 uba1: 20008000, 750 uba: F30000, 730 uba: F26000)
>>> D + 2013E000                (785/780 umem0)
         (785/780 umem1: 2017E000, 750 umem: FFE000, 730 umem: FFE000)
>>> D + 80000000
>>> D + 254C004
>>> D + 80000
>>> D + 264
```

```
>>> D + E
>>> D + C001
>>> D + 2000000
>>> S 200
```

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 20012000 here is the address of mba1 on an 11/785 or)
     (11/780 and the 0 indicates the formatter is drive 0 on mba1)
>>> D + 20012000          (785/780 mba1)
     (785/780 mba0: 20010000, 750 mba0: F28000)
>>> 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; see TS above)
    (for some hints on other values if your TM isn't on UBA0 on a 785/780)
>>> D + 20006000          (785/780 uba0)
>>> D + 2013E000          (785/780 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 11/785 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 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 RA the address of the tape MBA/UBA and in RB the address of the device registers or unit number from one of:

```
>>> D/G A 20006000        (for tapes on 785/780 uba0)
>>> D/G A 20008000        (for tapes on 785/780 uba1)
>>> D/G A 20012000        (for tapes on 785/780 mba1)
>>> D/G A 20010000        (for tapes on 785/780 mba0)
>>> D/G A F30000          (for tapes on 750 uba0)
>>> D/G A F2A000          (for tapes on 750 mba1)
>>> D/G A F28000          (for tapes on 750 mba0)
>>> D/G A F26000          (for tapes on 730 uba0)
```

and for register B:

>>> 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 785/780 uba0)
>>> 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.

APPENDIX C – INSTALLATION TROUBLESHOOTING

This appendix lists and explains certain problems that might be encountered while trying to install the 4.3BSD distribution. The information provided here is limited to the early steps in the installation process; i.e. up to the point where the root file system is installed. If you have a problem installing the release consult this section before contacting our group.

Using the distribution console medium.

This section describes problems that may occur when using the programs provided on the distributed console medium: TU58 cassette or RX01 floppy disk.

program can not be loaded.

Check to make sure the correct floppy or cassette is being used. If using a floppy, be sure it is not in upside down. If using a cassette on an 11/730, be certain drive 0 is being used. If a hard i/o error occurred while reading a floppy, try reseting the console LSI-11 by powering it on and off. If you can not boot the cassette's bootstrap monitor, verify the standard DEC console cassette can be read; if it can not, your cassette drive is probably broken.

program halts without warning.

Check to make sure you have specified the correct disk to format; consult sections 1.3 and 1.4 for a discussion of the VAX and UNIX device naming conventions. On 11/750's, specifying a non-existent MASSBUS device will cause the program to halt as it receives an interrupt (standalone programs operate by polling devices).

If using a floppy, try reading the floppy under your current system. If this works, copy the floppy to a new one and begin again. If using a cassette on an 11/730, do likewise.

format prints ''Known devices are ...''.

You have requested *format* to work on a device for which it has no driver, or that does not exist; only the listed devices are supported.

format, boot, or copy prints ''unknown drive type''.

A MASSBUS disk was specified, but the associated MASSBUS drive type register indicates a drive of unknown type. This probably means you typed something wrong or your hardware is incorrectly configured.

format, boot, or copy prints ''unknown device''.

The device specified is probably not one of those supported by the distribution; consult section 1.1. If the device is listed in section 1.1, the drive may be dual-ported, or for some other reason the driver was unable to decipher its characteristics. If this is a MASSBUS drive, try powering the MASSBUS adapter and/or controller on and off to clear the drive type register.

copy does not copy 205 records

If a tape read error occurred, clean your tape drive heads. If a disk write error occurred, the disk formatting may have failed. If the disk pack is removable, try another one. If you are currently running UNIX, you can reboot your old system and use *dd* to copy the mini-root file system into a disk partition (assuming the destination is not in use by the running system).

boot prints ''not a directory''

The *boot* program was unable to find the requested program because it encountered something other than a directory while searching the file system. This usually suggests that no file system is present on the disk partition supplied, or the file system has been corrupted. First check to make sure you typed the correct line to boot. If this is the case and you are booting off the mini-root file system, the mini-root was probably not copied correctly off the tape (perhaps it was not placed in the correct disk partition). Try reinstalling the mini-root file system or, if trying to boot the true root file system, try booting off the mini-root file system and run *fsck* on the restored root file system to insure its integrity. Finally, as a last resort, copy the *boot* program from the mini-root file system to the newly installed root file system.

boot prints ''bad format''

The program you requested *boot* to load did not have a 407, 410, or 413 magic number in its header. This should never happen on a distribution system. If you were trying to boot off the root file system, reboot the system on the mini-root file system and look at the program on the root file system. Try copying the copy of vmunix on the mini-root to the root file system also.

boot prints ''read short''

The file header for the program contained a size larger than the actual size of the file located on disk. This is probably the result of file system corruption (or a disk i/o error). Try booting again or creating a new copy of the program to be loaded (see above).

Booting the generic system

This section contains common problems encountered when booting the generic version of the system.

system panics with ''panic: iinit''

This occurred because the system was unable to mount the root file system. The root file system supplied at the ''root device?'' prompt was probably incorrect. Remember that when running on the mini-root file system, this question must be answered with something of the form ''hp0*''. If the answer had been ''hp0'', the system would have used the ''a'' partition on unit 0 of the ''hp'' drive, where presumably no file system exists.

Alternatively, the file system on which you were trying to run is corrupted. Try reinstalling the appropriate file system.

system selects incorrect root device

That is, you try to boot the system single user with ''B/2'' or ''B xxS'' but do not get the root file system in the expected location. This is most likely caused by your having many disks available more suited to be a root file system than the one you wanted. For example, if you have a ''up'' disk and an ''hk'' disk and install the system on the ''hk'', then try to boot the system to single-user mode, the heuristic used by the generic system to select the root file system will choose the ''up'' disk. The following list gives, in descending order, those disks thought most suitable to be a root file system: ''hp'', ''up'', ''ra'', ''rb'', ''rl'', ''hk'' (the position of ''rl'' is subject to argument). To get the root device you want you must boot using ''B/3'' or ''B ANY'', then supply the root device at the prompt.

system crashes during autoconfiguration

This is almost always caused by an unsupported UNIBUS device being present at a location where a supported device was expected. You must disable the device in some way, either by pulling it off the bus, or by moving the location of the console status register (consult Appendix A for a complete list of UNIBUS csr's used in the generic system).

system does not find device(s)

The UNIBUS device is not at a standard location. Consult the list of control status register addresses in Appendix A, or wait to configure a system to your hardware.

Alternatively, certain devices are difficult to locate during autoconfiguration. A classic example is the TS11 tape drive that does not autoconfigure properly if it is rewinding when the system is rebooted. Tape drives should configure properly if they are off-line, or are not performing a tape movement. Disks that are dual-ported should autoconfigure properly if the drive is not being simultaneously accessed through the alternate port.

Building console cassettes

This sections describes common problems encountered while constructing a console bootstrap cassette.

system crashes

You are trying to build a cassette for an 11/750. On an 11/750 the system is booted by using a bootstrap prom and sector 0 of the root file system. Refer to section 2.1.5 or *tu* (4) for the appropriate reprimand.

system hangs

You are using an MRSP prom on an 11/750 and think you can ignore the instructions in this document. The problem here is that the generic system only supports the MRSP prom on an 11/730. Using it on an 11/750 requires a special system configuration; consult *tu* (4) for more information.