Installing and Operating 2.11BSD on the PDP-11
35th Anniversary Release (Patch Level 500)
November 13, 2026

Steven Schultz

sms@2bsd.com

With additional contributions by:

Johnny Billquist

bqt@softjar.se

Terri Kennedy

terri-decus@glaver.org

ABSTRACT

This document contains instructions for the installation and operation of the 2.11BSD PDP-11† UNIX‡ system.

It discusses procedures for installing 2.11BSD UNIX on a PDP-11, including explanations of how to allocate space for filesystems on one or more disks, how to set up terminal lines and user accounts, how to do system-specific tailoring, and how to install and configure the networking facilities. Finally, the document details system operation procedures: shutdown and startup, hardware error reporting and diagnosis, filesystem backup procedures, resource control, performance monitoring, and procedures for recompiling and reinstalling system software.

The ‘‘bugs’’ address supplied with this release will work for some unknown period of time; make sure the ‘‘Index:’’ line of the bug report indicates that the release is ‘‘2.11BSD’’. See the *sendbug*(8) program for more details.

Updates are available at the following locations in the Patches subdirectory:

ftp://ftp.dfupdate.se/pub/pdp11/2.11BSD
https://www.tuhs.org/Archive/Distributions/UCB/2BSD/2.11BSD
ftp://ftp.2bsd.com/2.11BSD
http://www.2bsd.com/2.11BSD

You can determine the current patch level of a running 2.11BSD system with the following command: *head -2 /VERSION*

Updating instructions can be found in Appendix C.

† DEC, PDP-11, UNIBUS and MASSBUS were formerly trademarks (now abandoned) of Digital Equipment Corporation.

‡ UNIX is a trademark of The Open Group Limited

1. INTRODUCTION

1. Overview

This document explains how to install 2.11BSD UNIX on your PDP-11 system. This document has been revised several times since the first release of 2.11BSD in 1991. The last significant revision in July 1995 dealt with the addition of disk labels to the system. This revision is for the 2026 release of the 35th Anniversary Edition of 2.11BSD, which also includes preconfigured ("canned") bootable disk images for users of PDP-11 emulators, described in detail below.

The procedure for performing a full bootstrap is outlined in Chapter 2. The process includes copying a root filesystem from the distribution tape into a new filesystem, booting that root filesystem, and then reading the remainder of the system binaries and sources from the archives on the tapes.

As 2.11BSD is not compatible at the filesystem level with previous versions of UNIX on the PDP-11, any upgrade procedure is essentially a full bootstrap†). There is a limited ability to access old filesystems which may be used after the system partitions have been loaded from a full bootstrap. It is desirable to recompile most local software after the conversion, as there are changes and performance improvements in the standard libraries.

1.1. A brief history of 2BSD

The original 2BSD (short for "Second Berkeley Software Distribution", which is quite a mouthful) was put together by Bill Joy in 1979. Many subsequent releases followed, including 2.8BSD in 1982, 2.9BSD in 1983, 2.10BSD in 1987 and finally 2.11BSD in 1991. Each release added more parts of 4BSD (the BSD release for DEC VAX systems), with 2.11BSD adding full TCP/IP networking support.

Steven Schultz did nearly all of the work for the 2.11BSD release, including relocating the TCP/IP networking support from the kernel to supervisor mode. Rather than creating additional version numbers every time something major changed in 2.11BSD, the concept of "Patch Levels" was introduced. Some Patch Levels were minor changes, while others were massive changes (for example, updating the C compiler and library to modern ANSI standards). Many Patch Levels were partially or nearly entirely contributed by other volunteers, too numerous to credit here individually. This 35th Anniversary Release of 2.11BSD is Patch Level 500. There will almost certainly be additional Patch Levels released in the future.

None of us who ran 2.11BSD "back in the day" envisioned the operating system still being under active development 35 years later, but here we are. In addition to some systems still being used in production, "retrocomputing" enthusiasts and museums have kindled new interest in 2BSD, as well as other older operating systems. Some enthusiasts and museums run 2.11BSD on actual hardware (one of the authors of this document knows of enthusiasts who have multiple complete PDP-11/70 systems in their homes) as well as on PDP-11 emulators such as SIMH. Emulators have the advantage of using **far** less electricity and space than actual hardware, as well as making additional hardware "free". For example, in 1984 adding an additional RA81 disk drive cost about $19,000, while under an emulator it is a one-line change to a configuration file. And you don't have to worry about your emulated hardware breaking!

2.11BSD represents (in the opinion of one of the authors of this document) the pinnacle of what can be achieved in a UNIX environment on a PDP-11. Newcomers may wonder why some feature they're used to from modern UNIX-like systems (for example, Perl) isn't present in 2.11BSD. The short answer is there's only so much you can squeeze into a system with a 64-kilobyte address space and a maximum of 4 megabytes of total system memory.

1.2. Supported hardware

This distribution can be booted on a PDP-11 system equipped with 1MB or more of main memory, with separate I/D space and supervisor mode. Floating point (if not standard in your CPU model) is

† Refer to Appendix C for special considerations when upgrading from earlier 2.11BSD Patch Levels.

strongly recommended†, and booting from any of the following disks:

RL02
RK06, RK07
RM02, RM03, RM05
RP04, RP05, RP06
SMD disks (using controllers with RM/RP emulation)
SMD disks (using controllers with MSCP emulation)
SMD disks (using System Industries 9500 controller)
Most MSCP disks (RAxx, RDxx models, exceptions below)

Many companies have produced aftermarket controllers or entire disk subsystems. If these faithfully emulate one of the above models, they should work as well.

It is important to note that many controllers that emulate DEC RM or RP disk systems can be configured to exactly match the capacity of one or more DEC models, or to other configurations that use the entire disk capacity. If your system has RM or RP disks configured for non-standard sizes, you will need to manually define the disk during the standalone disklabel phase of installation, and optionally add a definition of your custom disks to the */etc/disktab* file. In cases like this, editing */etc/disktab* is just a potential convenience if you have additional drives of the same size that need labels.

Controllers that emulate MSCP disks (RA and RD types) report their exact drive capacity and other information to the operating system. However, if your MSCP disk does not exactly match the capacity of any of the DEC MSCP drives, you should start by using the default size as reported by the standalone disklabel program and define your partitions as desired. You can optionally add a new disk definition as described in the previous paragraph.

Non-standard sizes of other disk devices (for example, the RK07) are not supported by 2.11BSD as the drivers have fixed assumptions about capacity and geometry.

Other disks are supported (RX23, RX33, RX50, RD51, RK05, RL01, RX02) but are not large enough to hold a root filesystem plus a swap partition. It is possible to create a bootable 2.11BSD system on a single RL02 disk drive, but such a system will be severely limited.

Disks which are too small to hold even a root filesystem (floppies for example) may be used as data disks or as standalone boot media, but are not useable for loading the distribution. Others, while listed above, are not very well suited to loading the distribution. The RK06/07 and RL02 drives are hard pressed to even hold the system binaries, much less the sources.

If you are using a PDP-11 emulator such as SIMH, hardware is essentially free, so you should choose one of the larger disk models (RM05, RP06, or an MSCP drive).

The tape drives supported by this distribution are:

TS11, TU80, TK25
TM11 (including AVIV 6250/1600)
TE16, TU45, TU77
TK50, TU81, TU81+, TZ30

Third-party tape controllers and drives that emulate any of the above models are also supported. Since tapes do not have a known capacity, the size issues mentioned in the above section on disks do not happen with tapes.

1.3. Distribution format ("canned" bootable disk images)

As most users will be running 2.11BSD under a PDP-11 emulator such as SIMH, two types of preconfigured (canned) bootable disk images are provided. Although this is a great convenience for emulator

† In practice, this includes the PDP-11/44, PDP-11/70, and J11-based models (PDP-11/53, /73, /83, /84, /93 and /94), as well as a number of third-party CPUs compatible with one of the above models.

users, to get the full 2.11BSD experience, we suggest that (once you are familiar with the system) you do a "from scratch" installation of 2.11BSD from tape. This will allow more customization (for example, disk layout) and give you a deeper understanding of 2.11BSD. After all, emulated hardware is free, and you can't break it - if something doesn't work out, you can always start over.

There are two available canned images, depending on what sort of environment you desire. Each comes with a .ini file for the SIMH emulator as a starting point.

1.3.1. Classic PDP-11/70 system with MASSBUS

The first image is for a PDP-11/70 system with classic MASSBUS peripherals, upgraded with an Ethernet controller and a modern terminal multiplexor. It supports the following hardware:

PDP-11/70 CPU with 4MB memory and floating-point processor
60Hz line clock (can be reconfigured for 50Hz)
DL11 console terminal
RH70 with RM05 disk drives (maximum quantity 4)
RH70 with TU77 tape drive
DELUA Ethernet controller
DHU11 16-port terminal multiplexor
LP11 line printer
TCP/IP networking

It is supplied and maintained by Terri Kennedy, *terri-decus@glaver.org* and can be downloaded from: https://www.glaver.org/PiDP-11/tk2bsd/CLASSIC-211BSD.tgz (or .zip)

1.3.2. Modern PDP-11/70† system and peripherals

The second image is for a PDP-11/70† system with a variety of modern peripherals, including an Ethernet controller and a terminal multiplexor. It supports the following hardware:

PDP-11/70 CPU supporting an optional floating-point processor and at least 1MB of memory
50Hz line clock (can be reconfigured for 60Hz)
DL11 console terminal
UDA50/KDA50 with MSCP disk drives (maximum quantity 4)
RL11/RLV11 with RL02 disk drives (maximum quantity 4)
RH70/RH11 with RM/RP-series disk drives (maximum quantity 4)
KLESI with TMSCP tape drive
DEUNA/DEQNA Ethernet controllers
DZ11 8-port terminal multiplexor
TCP/IP networking

It is supplied and maintained by Johnny Billquist, *bqt@softjar.se* and can be downloaded from: http://mim.softjar.se/pidpbsd.htm

1.3.3. Steps needed when using any canned system

Regardless of which canned disk image you use, you will still need to perform some setup steps to customize the system for use. You can skip ahead to Section 4.4, "Adding and managing users and groups" if you are using a canned image.

† This image should actually work on any system with an 11/70-class UNIBUS processor, as well as on QBUS systems based on the J11 processor, as long as the system has at least 1MB of memory.

1.3.4. Additional documentation

Additionally, PDF versions of this manual as well as the USENIX 4.3BSD Manual Set are available from: https://www.glaver.org/PiDP-11/tk2bsddocs/2bsddocs.tgz (or .zip)

Many of the 4.3BSD manuals also apply to 2.11BSD, while some do not. The PDF versions are offered as a convenience, however the man pages included with 2.11BSD should be considered authoritative.

Note: This may not be available at the time of the 35th Anniversary Release, but will be released in the future.

1.4. Distribution format (installation from tape)

The basic distribution contains the following items:

(1) magnetic tape image (in SIMH TAP format)
(1) template SIMH config file (simh.ini)
(1) Soft copy of this document

The distribution does not fit on several standard PDP-11 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. If you are using an emulator, you can just choose a large disk type (RM05, RP06, or MSCP).

The root filesystem now occupies **a minimum of 4MB**. Creating a larger, 8 to 10MB root filesystem, is highly recommended when using the standalone **disklabel** program†.

The basic bootstrap material is present in six short files at the beginning of the tape. The first file on the tape contains the primary bootstrap and program. This is followed by several standalone utilities (*disklabel*, *mkfs* (8), *restor* (8), and *icheck* (8)‡) followed by a full dump of a root filesystem (see *dump* (8)). Following the root filesystem dump is a *tar* archive of **/usr** except for **/usr/src**. Next, a *tar* archive of the kernel source. Finally, a *tar* archive of the remaining system sources.

TAPE:

Tape file	Record size	Records§	Contents
0	512	1	primary tape boot block
	512	1	boot block (some tape boot ROMs go for this copy)
	512	69	standalone **boot** program
1	1024	38	standalone **disklabel**
2	1024	34	standalone **mkfs**(8)
3	1024	36	standalone **restor**(8)
4	1024	34	standalone **icheck**(8)
5	10240	321	*dump* of "root" filesystem
6	10240	3254	*tar* dump of /usr, excepting /usr/src
7	10240	293	*tar* dump of /usr/src/sys
8	10240	3190	*tar* dump of /usr/src excepting sys

1.5. UNIX device naming

UNIX has a set of names for devices which are different from the DEC names (of which there may be several for each device), which are also different from the names the SIMH emulator uses. The disk and

† The root filesystem cannot be arbitrarily large. The exact size varies, but 10MB should always be a safe upper limit. The reasons for this are discussed in more detail in Appendix B.

‡ References of the form X(Y) mean the subsection named X in section Y of the UNIX Programmer's Manual.

§ The number of records in each tape file are approximate and do not necessarily correspond to the actual number on the distribution tape image.

tape names used by the 2.11BSD bootstrap and system are:

RK05	rk
RK06, RK07 disks	hk
RL01, RL02 disks	rl
MSCP disks/diskettes	ra
RM02/03/05	xp
RP04/05/06	xp
RX02 diskettes (RX01 are not supported)	rx
SMD disks (using RM/RP emulation)	xp
SMD disks (using MSCP emulation)	ra
System Industries 9500 controller, SMD drive(s)	si
TM02/03, TE16, TU45, TU77 tapes	ht
TE10/TM11 tapes	tm
TS11/TU80/TK25 tapes	ts
TK50/TU81/TU81+/TZ30 tapes	tms
Other TMSCP tapes	tms

The generic SMD disk driver, *xp*, will handle most types of SMD disks on one or more controllers (even different disk types on the same controller). The *xp* driver handles RM02/03/05, RP04/05/06 disks on DEC or third-party QBUS, UNIBUS or MASSBUS controllers. Non-standard disk sizes (not matching any DEC drive size) are supported, but the user must provide the capacity and geometry information during system installation and configuration.

MSCP disks and TMSCP tapes include SCSI drives attached to the RQZX1 controller on suported QBUS CPU models. MSCP disks and TMSCP tapes also include SCSI drives attached to third-party UNIBUS and QBUS controllers, as long as those controllers emulate a DEC MSCP/TMSCP controller.

The standalone system used to bootstrap the full UNIX system uses device names of the form:

xx(*c*,*y*,*z*)

where *xx* is one of device types in the above disk and tape tables. The value *c* specifies the controller number (0-3). This value is usually not explicitly given. The default is 0 if booting from the standard (first) CSR of a device.

Example: if there are two MSCP controllers in the system addressed as 0172150 and 0172154 respectively booting from the controller at 172154 requires that *c* be given as 1. Booting from the standard CSR of 0172150 would be done by specifying *c* as 0 or omitting the *c* parameter. **boot** automatically detects if the first (standard) CSR is being used. All future references will ignore the *c* parameter by assuming the default value.

The *y* value specifies the device or drive unit to use. The *z* value is interpreted differently for tapes and disks: for disks, *z* is a partition number (0 thru 7) corresponding to partitions ´a´ thru ´h´ respectively, and should always be zero. For tapes, *z* is a file number on the tape.

In all simple cases, a drive with unit number 0 (determined either by a unit plug on the front of the drive, or jumper settings on the drive or controller) will be called unit 0 in its UNIX file name. If there are multiple controllers, the drive unit numbers will normally be counted within each controller. Different controller models support different numbers of devices. On a controller that supports 8 drives, drives on the first controller are numbered 0 thru 7 and drives on the second controller are also numbered 0 thru 7. Returning to the discussion of the standalone system, recall that tapes also took two integer parameters. In the case of a TM03 tape controller with a TU77 as drive 0, the files on the tape have names "ht(0,0)", "ht(0,1)", etc. Here "file" means a tape file containing a single data stream separated by a single tape mark.

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 */etc/disktab* file provides tested, standard partition layouts. While overlapping partitions are allowed, they are strongly discouraged, being an accident waiting to happen, as making one filesystem will destroy the other overlapping filesystem(s).

The cylinders occupied by the 8 partitions for each drive type are specified by the disk label read from the disk.

If no label exists the disk will not be bootable or mountable as a filesystem, and while the kernel attempts not to damage unlabeled disks (by swapping to or doing a crash dump on a live filesystem) there is a chance that filesystem damage will result if a kernel tries to write to an unlabeled disk. For physical (as opposed to emulated) disk drives, we recommend using the Write Protect button (if available on the drive) or simply not "spinning up" disks that will not be accessed by the kernel to guarantee that data on those disks will not be corrupted. This is particularly important on systems that alternate between running 2.11BSD and some other operating system.

The standalone **disklabel** program is used to define the partition tables. Each partition may be used either as a raw data area (such as a swapping area) or to store a UNIX filesystem. It is mandatory for the first partition on a disk to start at sector offset 0 because the 'a' partition is used to read and write the label (which is at the beginning of the disk). If the drive is to be used to bootstrap a UNIX system, the 'a' partition must be of type **2.11BSD** (FS_V71K in *disklabel.h*) and at least 4MB in size, with 8 to 10MB being strongly suggested. A 'b' partition for swapping is also needed. The 'b' partition must be at least 0.5MB larger than the size of main memory so that in the event of a system crash the contents of main memory can be written to the 'b' partition. Since most emulated systems will be running with a full 4MB of memory configured, this works out to a recommended size of 4.5 to 5MB for the 'b' partition. The extra space is so that the system can be booted and begin running without overwriting the crash dump. If a drive is being used solely for data, then that drive need not have a 'b' (swap) partition but partition 'a' must still span the first part of the disk. The second ('b') partition is traditionally used as a swapping area, and the rest of the disk is divided into space (as described below) for additional ''mounted filesystems'' by use of one or more additional partitions.

The third ('c') logical partition of each physical disk also has a conventional usage: it allows access to the entire physical device, including the BAD144 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 filesystem or to access the entire pack when making a copy of it on another. Care must be taken when using this partition not to overwrite the last few tracks and thereby destroying the bad sector information.

On disks that support bad sector forwarding, the eighth ('h') partition traditionally spans the entire physical disk **except** for the last cylinder. It is generally preferable to use the 'h' partition (if it exists) instead of the 'c' partition when copying entire disks using bad sector forwarding for the reasons described in the previous paragraph.

On other disks, partition 'h' is either not defined at all, is defined as an additional partition for user data (generally on larger-capacity disks), or is defined as in the above paragraph. This inconsistency is a result of new disk types added since the original release of 2.11BSD 35 years ago, the change to on-disk labels instead of in-kernel labels, and so forth.

One reason to spread things out across multiple partitions (particularly on larger drives) is the 64K limit on the number of files (inodes). If you expect to have a large number of small files, you can end up with megabytes of 'free' space, but not have an available inode to create a file.

1.6. 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

† We refer to "cylinders" as most supported physical disk drives use CHS (cylinder/head/sector) geometry. Newer physical disks such as MSCP and SCSI don't have traditional CHS geometry, it is still convenient (actually, necessary) to refer to them with a synthetic CHS geometry.

raw devices available. These have names like ‘‘/dev/rxx0a’’, the ‘‘r’’ here standing for ‘‘raw’’. Raw devices bypass the buffer cache and use DMA directly to/from the program’s I/O buffers; they are normally restricted to full-sector transfers. In the bootstrap procedures we will often suggest using the raw devices, because these tend to work faster. Raw devices are used when making new filesystems, when checking unmounted filesystems, or for copying quiescent filesystems. The block devices are used to mount filesystems, or when operating on a mounted filesystem such as the root.

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

The standalone **disklabel** program must be used to alter the ’a’ and ’b’ partitions of a drive being used for a bootable system. This is because the kernel will not permit an open partition to change size or offset. The root and swap partitions are **always** open when the kernel is running.

2. INSTALLATION PROCEDURE

This section explains the bootstrap procedure that can be used to get the kernel supplied with this distribution running on your PDP-11, regardless of whether it is an emulator or an actual PDP-11 system.

If you are installing this distribution on a system running the SIMH emulator, the supplied simh.ini should be enough to get you started. If you are running on another emulator, you will need to attach the tape (it is in SIMH TAP format) and one or more destination disk container files. If you are installing this on a real, working, PDP-11 system, you will need to write the supplied tape image to an actual tape which is compatible with the tape drive in your PDP-11 system.

2.1. Booting from tape

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

1) Load the tape bootstrap monitor.
2) Create the partition tables on the disk using *disklabel*. This is described in detail in Appendix B.
3) Create a UNIX "root" filesystem system on disk using *mkfs* (8).
4) Restore the full root filesystem using *restor* (8).
5) Boot the UNIX system on the new root filesystem and copy the appropriate *sector 0 boot block* to your boot device.
6) Build and restore the /usr filesystem from tape with *tar* (1).
7) Restore the kernel sources from tape.
8) Restore the remaining sources from tape.
9) Tailor a version of UNIX to your specific hardware. Refer to Appendix A for detailed information.

Certain of these steps are dependent on your hardware configuration. If your disks require formatting, standard DEC diagnostic utilities will have to be used, they are not supplied on the 2.11BSD distribution tape. If you have a system with small-capacity disk drives, you may need to perform some or all of steps 6 thru 8 on additional disk drives.

It is a good idea to locate your disk model in the table in Appendix B and use those values when setting up your partitions. All of the supported disks are listed in that file. Alternatively, you can manually calculate the partition layout you will use with the *disklabel* Calculating partition sizes in cylinders (even if your hardware is MSCP or an emulator that doesn't use cylinders) instead of sectors is helpful because you're working with smaller numbers and less likely to make a math error. You can write this info down or store it in a text file and desk-check your math. *disklabel* will detect many errors (such as overlapping partitions or partitions extending past the end of the disk), but it is not guaranteed to catch all possible errors.

2.1.1. Step 1: Loading the tape bootstrap monitor

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 if you are using a physical tape. Then use the normal bootstrap ROM, console monitor, emulator boot command, or other bootstrap to boot from the tape.

NOTE: The boot blocks expect the CSR of the booting controller in r0 and the unit number in r1. **boot** may be booted from any controller or unit, the earlier restrictions of controller 0 and unit 0 have been lifted.

If no other means are available, the following code can be keyed in and executed at (say) 0100000 to boot from a TM tape drive (the magic number 172526 is the address of the TM-11 current memory address register; an adjustment may be necessary if your controller is at a nonstandard address):

```
012700    (mov $unit, r0)
000000    (normally unit 0)
012701    (mov $172526, r1)
172526
010141    (mov r1, -(r1))
012741    (mov $60003, -(r1))
060003    (if unit 1 use 060403, etc)
000777    (br .)
```

A toggle-in routine which has been used with a TS tape drive (this should be entered at 01000):

```
012700    mov $unit,r0
000000
012701    mov $172522,r1
172522
005011    clr (r1)
105711    1b:tstb (r1)
100376    bpl 1b
012761    mov $setchr,-2(r1)
001040
177776
105711    2b:tstb (r1)
100376    bpl 2b
012761    mov $read,-2(r1)
001060
177776
000000    halt
140004    setchr: TS_ACK|TS_CVC|TS_SETCHR
001050    char
000000    high order address
000010    number of bytes
001070    char: status
000000
000016
000000
140001    read: TS_ACK|TS_CVC|TS_READ
000000    low order of address
000000    high order of address
001000    number of bytes to read
000000    status:
```

When this is executed, the first block of the tape will be read into memory. Halt the CPU and restart at location 0. The register **r1 MUST** be left pointing at the device *csr*. For the default/first TM or TS this is 0172522. The register **r0 MUST** contain the unit number (usually 0).

The console should type

*nn***Boot from** *xx*(*ctlr*,*drive*,*part*) **at** *csr*
:

where *nn* is the CPU type on which it believes it is running. The value will be one of 44, 70, 73, 83, 84, 93 or 94 depending whether a UNIBUS map and other hardware features are detected. For J11 systems the System Maintenance Register is examined to determine the CPU type. At present 2.11BSD runs on the 44, 53, 70, 73, 83, 84, 93 and 94 **only**. It must be emphasized that 2.11BSD requires a CPU with separate I/D space.

ctlr is the controller number that **Boot** was loaded from. It is 0 unless booting from a non-standard CSR.

drive is the drive unit number.

The *part* number is disk partition or tape file number booted from. This will always be 0 for the tape **Boot** program.

csr is an octal number telling the CSR of the controller from which **Boot** was loaded.

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. Through the rest of this section, substitute the correct disk type for *dk* and the tape type for *tp*.

2.1.2. Step 2: Creating the disk label

The standalone *disklabel* program is then run:

```
: tp(0,1)                                   (disklabel is tape file 1)
Boot: bootdev=0nnnn bootcsr=0mmmmmm
disklabel
Disk? dk(0,0)                               (drive 0, partition 0)
d(isplay) D(efault) m(odify) w(rite) q(uit)?
 ...
:                                           (back at tape boot level)
```

The *disklabel* program is meant to be fairly intuitive. When prompted with a line of choices entering the key just before the left parenthesis selects the entry.

If there is an existing label present on *dk*(0,0) it will be used as the default. To have *disklabel* create a new default based on its idea of what the drive is select **D**. Then enter **m** to modify/edit the label.

The MSCP driver is quite good at identifying drives because it can query the controller. Other drivers (notably the SMD (**xp**) driver) have to deal with a much wider range of controllers which do not all have the same capabilities for drive identification. When dealing with SMD drives you must know the geometry of the drive so you can verify and correct *disklabel*'s choices.

You can however, if using non-DEC SMD controllers, make things easy for *disklabel* to determine what type of drive is being used. If your controller offers the choice of RM02 emulation you should select that choice. The standalone **xp** driver uses RM02 as the indication that drive identification capabilities not offered by DEC controllers are present. The driver will be able to determine the geometry of the drive in this case. This is **optional** because you can explicitly specify all of the parameters to the standalone *disklabel* program.

A full description of the standalone *disklabel* program is in Appendix B of this document.

2.1.3. Step 3: Creating a UNIX "root" filesystem

Now create the root filesystem using the following procedure.

The size of the root ('a') filesystem was assigned in Step 2 (creating the disk label). *mkfs* will not allow a filesystem to be created if there is not a label present or if the partition size is 0. *mkfs* looks at partition 0 ('a') in the disklabel for the root filesystem size.

Finally, determine the proper interleaving factors *m* and *n* for your disk. Extensive testing has demonstrated that the choice of *m* is non critical (performance of a filesystem varying only by 3 to 4% for a wide range of *m* values). Values for *m* within the range from 2 to 5 give almost identical performance. Increasing *m* too much actually causes degraded performance because the free blocks are too far apart. Slower processors (such as the 73 and 44) may want to start with a *m* of 3 or 4, faster processors (such as the 70 and 84) may start with a *m* of 2 or 3. On the other hand, the *n* value is moderately important. It should be the number of filesystem blocks contained by one cylinder of the disk, calculated by dividing the number of sectors per cylinder by 2, rounding down if needed. (This is what *mkfs* does by default, based on the geometry information in the disk label.) These numbers determine the layout of the free list that will be

constructed; the proper interleaving will help increase the speed of the filesystem.

The number of bytes per inode determines how many inodes will be allocated in the filesystem. The default of 4096 bytes per inode is normally enough (resulting in about 2000 inodes for an 8mb root filesystem and 1000 inodes for the 4mb distribution "generic" root filesystem). If more inodes are desired then a lower value (perhaps 3072) should be specified when prompted for the number of bytes per inode.

Then run the standalone version of the *mkfs* (8) program. The values in square brackets at the size prompt is the default from the disklabel. Simply hit a return to accept the default. *mkfs* will allow you to create a smaller filesystem but you cannot enter a larger number than the one in brackets. In the following procedure, substitute the correct types for *tp* and *dk* and the size determined above for *size*:

: *tp* (0,2)	(*mkfs* is tape file 2)
Boot: bootdev=0nnnn bootcsr=0mmmmmm	
Mkfs	
filesystem: *dk* (0,0)	(root is the first filesystem on drive 0)
filesystem size: [NNNN] *size*	(count of 1024 byte blocks in root)
bytes per inode: [4096] *bytes*	(number of bytes per inode)
interleaving factor (m, 2 default): *m*	(interleaving, see above)
interleaving modulus (n, 127 default): *n*	(interleaving, see above)
isize = XX	(count of inodes in root filesystem)
m/n = *m n*	(interleave parameters)
Exit called	
*nn***Boot**	
:	(back at tape boot level)

The number **nnnn** is the device number of the device (high byte is the major device number and the low byte is the unit number). The **mmmmmm** number is the CSR of the device. This information is mainly used as a reminder and diagnostic/testing purposes.

You now have an empty UNIX root filesystem.

2.1.4. Step 4: Restoring the root filesystem

To restore the root filesystem onto it, type

: *tp* (0,3)	(*restor* is tape file 3)
Boot: bootdev=0nnnn bootcsr=0mmmmmm	
Restor	
Tape? *tp* (0,5)	(root *dump* is tape file 5)
Disk? *dk* (0,0)	(into root filesystem)
Last chance before scribbling on disk.	(type a carriage return to start)
"End of tape"	(appears on same line as message above)
*nn***Boot**	
:	(back at tape boot level)

If you wish, you may use the *icheck* program on the tape, *tp* (0,4), to check the consistency of the filesystem you have just installed. This has rarely been useful and is mostly for the voyeuristic.

2.1.5. Step 5: Booting UNIX

You are now ready to boot from disk. Type:

:*dk* (0,0)unix	(bring in unix from the root system)
Boot: bootdev=0nnnn bootcsr=0mmmmmm	

The standalone boot program will then load unix from the root filesystem you just created, and the system should boot (the exact strings will vary of course):

2.11 BSD UNIX #157: Sat Oct 24 16:44:40 CDT 2026
sms@vernon.2bsd.com:/usr/src/sys/GENERIC

phys mem = 4186112
avail mem = 3709504
user mem = 307200

configure system
... information about available devices ...
(Information about various devices will print;
most of them will probably not be found until
the addresses are set below.)
erase=^?, kill=^U, intr=^C
#

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

Next the *mem* messages give the amount of real (physical) memory, the amount of memory left over after the system has allocated various data structures, and the amount of memory available to user programs in bytes.

The information about different devices being attached or not being found is produced by the *autoconfig* (8) program. Most of this is not important for the moment, but later the device table, */etc/dtab*, can be edited to correspond to your hardware. You will likely need to create a custom kernel to support your hardware and remove support for hardware you don't have. This is discussed in detail in Appendix A.

The tape drive of the correct type should have been detected and attached.

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 character erase, line erase, and interrupt characters are set to what is standard for DEC systems; this ensures that things are consistent with the DEC console interface characters.

UNIX is now running single user on the installed root filesystem, and the 'UNIX Programmer's Manual' applies. The next section tells how to complete the installation of distributed software on the /usr filesystem. The '#' is the prompt from the shell, and lets you know that you are the super-user, whose login name is "root".

The disk with the new root filesystem on it will not be bootable directly until the block 0 bootstrap program for your disk has been installed. There are copies of the bootstraps in /mdec. Use *disklabel* (1) to install the boot block onto the disk.

disklabel -r -B *dk*0 device

For example:

disklabel -r -B xp0 rp06

will install /mdec/hpuboot into sector 0 of xp0

disklabel -r -B ra0

Will install /mdec/rauboot into sector 0 of ra0 (since MSCP disks provide their device geometry automatically, there is no need to specify a device type as in the previous example)

The block zero bootstraps and the devices they support are:

boot	disk_type
hkuboot	rk06/rk07
rauboot	MSCP drives
rkuboot	rk05
rluboot	rl01/rl02
si95uboot	si_eagle
dvhpuboot	diva
hpuboot	rp04/rp05/rp06
rp07uboot	rp07
rm03uboot	rm03
rm05uboot	rm05/cdc9766
si51uboot	si_eagle/2351A
si94uboot	rm2x/fuji160

NOTE: If none of the above are correct (most likely with a SMD drive with differing geometry) then you will have to use a tape/floppy boot procedure rather than a sector 0 boot block. This can be fixed by creating a customized sector 0 boot program once the system sources have been loaded.

Once this is done, booting from this disk will load and execute the block 0 bootstrap, which will in turn load /boot. **/boot** will print on the console:

*nn***Boot from** *dk*(*ctlr*,*unit*,*part*) **at** *csr*
:

The boot block automatically loads and runs */boot* for you; if */boot* is not found, the system will hang/loop forever. The block 0 program is very small (has to fit in 512 bytes) and simple program, however, and can only boot the second-stage boot from the first filesystem. Once /boot is running and prints its ": " prompt, boot unix as above.

As distributed */boot* will load *dk*(0,0)unix by default if a carriage return is typed at the **:** prompt.

NOTE: NONE of the primary bootstraps have a prompt or alternate program name capability because of space considerations. No diagnostic message results if the file cannot be found.

2.1.6. Step 6: Setting up the /usr filesystem

First set a shell variable to the name of your disk, so the commands used later will work regardless of the disk you have; do one of the following:

```
# disk=hk     (if you have RK06's or RK07's)
# disk=rl     (if you have RL01's or RL02's)
# disk=ra     (if you have an MSCP drive)
# disk=xp     (if you have an RP04/5/6/7, RM03, RM05, or other SMD drive)
```

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 4.2, "Disk configuration" before continuing; you will have to select a partition to restore the /usr filesystem into which is at least **25** Megabytes in size (this is just barely enough for the system binaries and such and leaves no room for the system source.)†

In the command below *part* is the partition name (a-h) for the partition which will hold /usr. For example, if you were going to use the g partition for /usr, you would do the following:

```
# part=g
```

Now, define the symbol *name* to refer to complete drive and partition to be used for /usr:

```
name=${disk}0${part}
```

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

DEC TM02/03, TE16/TU45/TU77	# cd /dev; rm *mt*; ./MAKEDEV ht0; sync
DEC TS11, TK25/TU80/TS05	# cd /dev; rm *mt*; ./MAKEDEV ts0; sync
DEC TM11, TU10/TE10/TS03	# cd /dev; rm *mt*; ./MAKEDEV tm0; sync
DEC TMSCP, TK50/TZ30/TU81	# cd /dev; rm *mt*; ./MAKEDEV tu0; sync
EMULEX TC11	# cd /dev; rm *mt*; ./MAKEDEV tm0; sync

Then execute the following commands:

```
# date yymmddhhmm           (set date, see date (1) - note that "date: can't write wtmp file."
....                        is expected, harmless and can safely be ignored)
# passwd root               (set password for super-user)
New password:               (password will not echo)
Retype new password:
# hostname mysitename       (set your hostname)
# newfs ${name}             (create empty user filesystem)
(this takes a minute)
# mount /dev/${name} /usr   (mount the usr filesystem)
# umask 22                  (set file creation mask)
# cd /usr                   (make /usr the current directory)
# mt rew                    (rewind the distribution tape to the beginning)
# mt -f /dev/rmt12 fsf 6    (position the distribution tape to the 6th file)
# tar xpbf 20 /dev/rmt12    (extract all of usr except usr/src‡)
```

(Continued on next page)

† Previously a lengthy table of partition names organized by specific disk type was given. With the introduction of disk labels this is no longer necessary (or possible since each site can select whatever partitioning scheme they desire). In Step 2 (creating the disklabel) a partition should have been created for */usr*. If this was not done then it may be easier to perform step 2 now than to use the more complex *disklabel* (8) program and *ed* (1).

‡ The amount of time it will take the *tar* command to extract each archive depends on the speed of the real or emulated CPU, tape drive and disk drive.

```
# mkdir src                                  (make directory for source)
# cd src                                     (make /usr/src the current directory)
# mt -f /dev/rmt12 fsf                       (skip to the start of the next tape file)
# tar xpbf 20 /dev/rmt12                     (extract the system source)
# cd /                                       (back to root)
# chmod 755  /  /usr  /usr/src /usr/src/sys  (set the proper directory permissions)
# umount /dev/${name}                        (unmount /usr)
```

You now have installed a complete base system, along with the ability to recompile the kernel. However, the remainder of the source code in the /usr filesystem still needs to be loaded, which is described below. it is **strongly** recommended that you install the remaining source code as it will be needed to install future patch levels (refer to Appendix C for additional information).

You can check the consistency of the /usr filesystem by doing

```
# fsck /dev/r${name}
```

The output from *fsck* should look something like:

```
** /dev/rxx0x
filesystem: /usr

NEED SCRATCH FILE (179 BLKS)
ENTER FILENAME:  /tmp/xxx
** 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 Free List
4463 files, 35838 used, 196678 free
```

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

2.1.7. Step 7: extracting the remaining source code

You should now remount the /usr filesystem and restore the rest of /usr/src as shown below:

```
# mount /dev/${name} /usr
# cd /usr/src                    (make /usr/src the current directory)
# mt rew                         (rewind the distribution tape to the beginning)
# mt -f /dev/rmt12 fsf 8         (position the distribution tape to the 8th file)
# tar xpbf 20 /dev/rmt12         (extract the remaining source
```

2.1.8. Step 8: editing the /etc/fstab file

The /etc/fstab file describes where the filesystems you created in Step 2 above will be automatically mounted by the kernel. As an example, the following is an /etc/fstab file defining three partitions on an RM05 disk. The ‘a’ partition holds the root filesystem /, the ‘b’ partition describes the area used for swapping, and the ‘g’ partition contains all of /usr, including the kernel and other system source files (if you installed them). **NOTE:** A bootable 2.11BSD disk pack requires both an ‘a’ and a ‘b’ partition.

```
/dev/xp0a       /       ufs     rw              1       1
/dev/xp0b       none    swap    sw              0       0
/dev/xp0g       /usr    ufs     rw              1       2
```

2.2. Configure the newly-installed system

Assuming you are performing a fresh install and not upgrading a system, proceed to Chapter 4, "System setup.

3. UPGRADING AN EXISTING SYSTEM

If you are upgrading a recent patch level of 2.11BSD, skip this section in its entirety and proceed to Appendix C.

The general advice is "you don't want to do this". Depending on what patch level you are on if you are already running 2.11BSD, substantial work can be required to get from that patch level to the current (as of this writing) Patch Level 500. If you are running 2.10BSD or some other Unix version on your existing system, things get even more complicated.

If you are going to attempt an in-place upgrade, it is **strongly** suggested that you back up your existing system, both in its preferred backup format (such as *dump*/*restore*) in case you need to revert to that system, as well as in a portable format like *tar*.

The following sections describe the procedure for an upgrade from a substantially older 2.11BSD or other Unix operating system to Patch Level 500 of 2.11BSD. These procedures have not been tested in at least a decade and success will depend on your familiarity with both the existing system and this 35th Anniversary Release of 2.11BSD. Here there be dragons!

3.1. Files to save

The following list enumerates the standard set of files you will want to save and suggests directories in which site-specific files should be present. Note that because 2.10BSD changed so radically from previous versions of UNIX on the PDP-11, many of these files may not exist on your system, and will almost certainly require extensive changes for 2.11BSD, but it's still handy to have them around as you're configuring 2.11BSD. This list will likely be augmented with non-standard files you have added to your system.

You should create a *tar* image of (at a minimum) the following files before the new filesystems are created. In addition, you should do a full dump before rebuilding the filesystem to guard against missing something the first time around. The 2.11BSD *restor*(8) program can read and convert old *dump*(8) tapes.

/.cshrc	†	root csh startup script
/.login	†	root csh login script
/.profile	†	root sh startup script
/.rhosts	†	for trusted machines and users
/dev/MAKEDEV	‡	in case you added anything here
/dev/MAKEDEV.local	*	for making local devices
/etc/disktab	*	in case you changed disk partition sizes
/etc/dtab	‡	table of devices to attach at boot time
/etc/fstab	†	disk configuration data
/etc/ftpusers	†	for local additions
/etc/gateways	†	routing daemon database
/etc/gettytab	†	getty database
/etc/group	†	group data base
/etc/hosts	†	for local host information
/etc/hosts.dir	*	must be rebuilt with mkhosts
/etc/hosts.pag	*	must be rebuilt with mkhosts
/etc/hosts.equiv	†	for local host equivalence information
/etc/networks	†	for local network information
/etc/netstart	*	site dependent network startup script
/etc/passwd	*	must be converted to shadow password file format
/etc/passwd.dir	*	must be rebuilt with mkpasswd
/etc/passwd.pag	*	must be rebuilt with mkpasswd
/etc/printcap	†	line printer database
/etc/protocols	‡	in case you added any local protocols

/etc/rc	§	for any local additions
/etc/rc.local	§	site specific system startup commands
/etc/remote	†	auto-dialer configuration
/etc/services	‡	for local additions
/etc/syslog.conf	†	system logger configuration
/etc/securettys	§	for restricted list of ttys where root can log in
/etc/ttys	†	terminal line configuration data
/etc/ttytype	§	terminal line to terminal type mapping data
/etc/termcap	‡	for any local entries that may have been added
/lib	‡	for any locally developed language processors
/usr/dict/*	‡	for local additions to words and papers
/usr/hosts/MAKEHOSTS	†	for local changes
/usr/include/*	‡	for local additions
/etc/aliases	†	mail forwarding data base
/etc/crontab	†	cron daemon data base
/usr/share/font/*	‡	for locally developed font libraries
/usr/lib/lib*.a	†	for local libraries
/usr/share/lint/*	‡	for locally developed lint libraries
/etc/sendmail.cf	†	sendmail configuration
/usr/share/tabset/*	‡	for locally developed tab setting files
/usr/share/term/*	‡	for locally developed nroff drive tables
/usr/share/tmac/*	‡	for locally developed troff/nroff macros
/etc/uucp/*	†	for local uucp configuration files
/usr/man/manl	§	for manual pages for locally developed programs
/usr/msgs	†	for current msgs
/usr/spool/*	†	for current mail, news, uucp files, etc.
/usr/src/local	†	for source for locally developed programs
/sys/conf/HOST	†	configuration file for your machine
/sys/conf/files.HOST	†	list of special files in your kernel
/*/quotas	§	filesystem quota files

† Files that can be used from 2.10BSD without change.
‡ Files that need local modifications merged into 2.11BSD files.
§ Files that require special work to merge and are discussed below.

3.1.1. Installing 2.11BSD

The next step is to build a working 2.11BSD system. This can be done by following the steps in Chapter 2 of this document for extracting the root and /usr filesystems from the distribution tape onto unused disk partitions.

Once you have extracted the 2.11BSD system and booted from it, you will have to build a kernel customized for your configuration. If you have any local device drivers, they will have to be incorporated into the new kernel. See Appendix A for instructions on configuring, building, and installing a new kernel.

With the introduction of disk labels the disk partitions in 2.11BSD the /etc/disktab file has changed dramatically. There is a detailed description later in this chapter about the changes. If you have modified the partition tables in previous versions of 2.11BSD you will need to create a new disktab entry or modify an existing one.

3.2. Merging your files from earlier PDP-11 UNIX systems into 2.11BSD

When your system is booting reliably and you have the 2.11BSD root and /usr filesystems fully installed you will be ready to continue with the next step in the conversion process, merging your old files into the new system.

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

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

For sites running a pre-2.10BSD UNIX, there is very little that can be said here as the variety of previous versions of PDP-11 UNIX systems and how they were administered is large. As an example, most previous versions of PDP-11 UNIX systems used the files */etc/ttys* and */etc/ttytype* to administer which terminals should have login processes attached to them and what the types of terminals those were. Under 2.11BSD /etc/ttytype has disappeared entirely, its functions subsumed by /etc/ttys along with several new functions. In general, you will simply have to use your previous configuration files as references as you configure 2.11BSD to your site needs. Familiarity with 4.3BSD configuration is very helpful at this point since 2.11BSD is nearly identical in most of the files listed in the previous section.

If you have any home-grown 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/pdp/conf.c. Note that almost all 2.11BSD major device numbers are different from those in previous PDP-11 UNIX systems except 2.10.1BSD. A couple more device numbers were added since the release of 2.10.1BSD for the kernel logging facility (/dev/klog) and a (new) TK50/TU81 driver.

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

name	number
wheel	0
daemon	1
kmem	2
sys	3
tty	4
operator	5
staff	10
bin	20

Only users in the ‘‘wheel’’ group are permitted to *su* to ‘‘root’’. Most programs that manage directories in /usr/spool now run set-group-id to ‘‘daemon’’ so that users cannot directly access the files in the spool directories. The special files that access kernel memory, */dev/kmem* and */dev/mem*, are made readable only by group ‘‘kmem’’. Standard system programs that require this access are made set-group-id to that group. The group ‘‘sys’’ is intended to control access to system sources, and other sources belong to group ‘‘staff.’’ Rather than make user’s terminals writable by all users, they are now placed in group ‘‘tty’’ and made only group writable. Programs that should legitimately have access to write on user’s terminals such as *talk* and *write* now run set-group-id to ‘‘tty’’. The ‘‘operator’’ group controls access to disks. By default, disks are readable by group ‘‘operator’’, so that programs such as *df* can access the filesystem information without being set-user-id to ‘‘root’’.

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

name	number
root	0
daemon	1
operator	2
uucp	66
nobody	32767

The ‘‘daemon’’ user is used for daemon processes that do not need root privileges. The ‘‘operator’’ user-id is used as an account for dumpers so that they can log in without having the root password. By placing them in the ‘‘operator’’ group, they can get read access to the disks. The ‘‘uucp’’ login has existed long before 2.11BSD, and is noted here just to provide a common user-id. The password entry ‘‘nobody’’ has been added to specify the user with least privilege.

After restoring your old password file from tape/backups, a conversion is required to create the shadow password file. Only the steps to convert /etc/passwd are given here, see the various man pages for *chpass* (1), *vipw* (8), *mkpasswd* (8), etc.

```
# awk -f /etc/awk.script < /etc/passwd >/etc/junk
# mkpasswd -p /etc/junk
# mv /etc/junk.orig /etc/passwd
# mv /etc/junk.pag /etc/passwd.pag
# mv /etc/junk.dir /etc/passwd.dir
# mv /etc/junk /etc/master.passwd
# chown root /etc/passwd* /etc/master.passwd
# chmod 0600 /etc/master.passwd
```

The format of the cron table, /etc/crontab, is the same as that of 2.10.1BSD.

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

The C library and system binaries are compiled with versions of *gethostbyname* and *gethostbyaddr* which use the name server resolver routines and NOT host tables. If you want to build the C library and binaries to use host tables you must edit the file /usr/src/lib/libc/Makefile and change the macro HOSTLOOKUP=named to be HOSTLOOKUP=hosttable. Then recompile the entire system (see /usr/src/Makefile for instructions).

There is a very old version of *named* (8) in the system source tree. Don’t even think about using it.

The format of /etc/ttys is the same as it was under 2.10BSD. It includes the terminal type and security options that were previously in /etc/ttytype and /etc/securettys.

syslog is the 4.4BSD-Lite version now. See *syslog* (3) and *syslogd* (8) for details. They are used by many of the system daemons to monitor system problems more closely, for example network routing changes.

Again, it must be emphasized that the nameserver is not robust under 2.11BSD, and if the *hosts* files are not desired then the best alternative is to use the *resolver* (5) routines and use the nameserver on a remote larger machine. The *resolver* (5) routines are known to work.

The spooling directories saved on tape may be restored in their eventual resting places without too much concern. Be sure to use the ‘‘p’’ option to *tar* so that files are recreated with the same file modes:

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

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

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

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. Additional conversion information

After setting up the new 2.11BSD filesystems, you may restore the user files that were saved on tape before beginning the conversion. Note that the 2.11BSD *restor* program does its work by accessing the raw file system device and depositing inodes in the appropriate locations on disk. This means that filesystem dumps might not restore correctly if the characteristics of the filesystem have changed (e.g. if you're restoring a dump of a filesystem into a filesystem smaller than the original.) To restore a dump tape for, say, the /u filesystem something like the following would be used:

```
# restor r /dev/rxp1e
```

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 filesystem, be sure and check its integrity with *fsck* when the job is complete.

tar tapes are preferred (when possible) because the inode allocation is performed by the kernel rather than the *restor* (8) program. This has the benefit of allocating inodes sequentially starting from the beginning of the inode portion of the filesystem rather than preserving the fragmented/randomized order of the old filesystem.

3.4. Hints on converting from previous PDP-11 UNIX systems to 2.11BSD

This section summarizes some of the significant changes in 2.11BSD from 2.10.1BSD. The installation guide for 2.10.1BSD is included in the distribution as /usr/doc/2.10/setup.2.10 and should be read if you are not presently running 2.10BSD or 2.10.1BSD. It does not include changes in the network; see Chapter 5 for information on setting up the network.

Old core files will not be intelligible by the current debuggers because of numerous changes to the user structure. Also removed from the user structure are the members u_offset, u_count, u_base, u_segflg, the 4.3BSD uio/iovec/rdwri kernel I/O model having been put in place. The 4.3BSD *namei* argument encapsulation technique has been ported, which adds the u_nd member to the user structure.

Note, once your system is installed and running, you should make sure that you recompile and reinstall the directory */usr/src/share/zoneinfo*. Read through the Makefile first, if you're not located on the West Coast you will have to change it. This directory is an addition since 4.3BSD, and is intended to solve the Daylight Savings Time problems once and for all.

The in-memory inode structure has had the i_id member added as part of implementing the 4.3BSD namei cache. The di_addr member of the on disk inode structure is now an array of type **daddr_t** instead of **char**. The old 3-byte packed block number is obsolete at last.

The on-disk directory structure is that of 4.3BSD with the difference that the inode number is an unsigned short instead of a long. This was done to reduce the amount of long arithmetic in the kernel and to maintain compatibility with earlier versions with regard to the maximum number of inodes per filesystem. Given the typical size of disks used with 2.11BSD the limit on the number of inodes per filesystem will not be a problem.

And again, 2.11BSD is **not** filesystem compatible with any previous PDP-11 UNIX system**.**

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

3.5. Hints on possible problems upgrading from 2.10.1BSD

3.5.1. New utmp UT_NAMESIZE.

UT_NAMESIZE in <utmp.h> was changed from 8 to 15. This won't affect correctly written programs (those which do not hard code the constant 8) at the source level but does cause changes in various databases. This means that old binaries won't be able to cope with new databases (passwd, aliases, etc) and

vice versa.

This change was necessary since the systems available for 2.11BSD development had to be shared with systems in which UT_NAMESIZE was set at 15. If this change/incompatibility is not desired, then utmp.h and wtmp.h will have to be modified and the system libraries and applications rebuilt before proceeding to load local software.

The simplest way to deal with this incompatibility is simply to rebuild all your databases from the source data. In particular, you should be sure you rebuild */etc/passwd* , */etc/hosts* , and */etc/aliases* databases via the commands: *mkpasswd /etc/passwd* , *mkhosts /etc/hosts* , and */usr/ucb/newaliases.*

3.5.2. man system

The manual system continues to track the changes going on in 4BSD. I'm not convinced the new setup is better, but it does seem to be the method of the moment. The setup is essentially the same as that in the **4.3BSD-TAHOE** distribution with the manual source in /usr/src/man.

3.5.3. NMOUNT lowered

The value of **NMOUNT** in */sys/h/param.h* is set to 6 in the distribution system. This may be too small for systems using multiple disk drives. Since each mount table entry costs about 440 bytes of valuable kernel data space this number should be chosen with care. See Appendix A for an explanation of how to reconfigure **NMOUNT.**

3.5.4. Shadow passwords

The May 1989 release of the 4.3BSD shadow password file has been ported to 2.11BSD. Password aging is also implemented.

3.5.5. New /etc/rc startup scripts

/etc/rc, */etc/rc.local* and */etc/netstart* have been modified extensively and */etc/rc.conf* has been added. *rc.conf* is a file which centralizes information previously scattered across several files.

3.5.6. mkfs

mkfs (8) is no longer able to populate a filesystem with files. The 4.3BSD version of *mkfs* (8) was ported to 2.11BSD.

3.5.7. mkproto and mklost+found

mkproto (8) and *mklost+found* (8) have been removed from the system. *mkproto*(8) hasn't been used since it was written and has so many limitations as to render the program useless. *mklost+found* (8) is not needed as *fsck* (8) has been capable of automatically extending lost+found for quite some time.

3.5.8. ANSI C compiler

The C compiler (and C preprocessor) is ANSI compliant and most (but not all) C library functions are properly prototyped. This may cause compilation problems with some program sources.

3.5.9. /etc/disktab

The format of /etc/disktab is now the same as 4.3BSD-Reno and 4.4BSD. Previously to describe a drive (an RM03 for example) the /etc/disktab file had entries of the form:

```
:ty=removable:ns#32:nt#5:nc#823:sf:
:b0=/mdec/rm03uboot:
:pa#9600:ba#1024:fa#1024:
:pb#9600:bb#1024:fb#1024:
:pc#131520:bc#1024:fc#1024:
:pf#121920:bf#1024:ff#1024:
:pg#112320:bg#1024:fg#1024:
:ph#131520:bh#1024:fh#1024:
```

Note that there is no information at all about which cylinder a partition starts at or which partitions overlap and may not be used simultaneously. That information was kept in tables in the driver. If you modified /etc/disktab it would have no effect without also changing the driver and recompiling the kernel.

The new /etc/disktab file looks like this:

```
# RM02/03
#
rm02|RM02|rm03|RM03|DEC RM02/03:\
   :ty=removable:ns#32:nt#5:nc#823:sf:\
   :b0=/mdec/rm03uboot:\
   :pa#14080:oa#0:ba#1024:fa#1024:ta=2.11BSD:\
   :pb#8960:ob#14080:bb#1024:fb#1024:tb=swap:\
   :pc#131680:oc#0:bc#1024:fc#1024:\
   :pg#108480:og#23040:bg#1024:fg#1024:tg=2.11BSD:\
   :ph#131520:oh#0:bh#1024:fh#1024
```

There are two new fields per partition, the 'o' (oa, ob, etc.) field specifies the offset in sectors that the partition begins at. The 't' field specifies the partition type. Only those partitions which are **2.11BSD** will be recognized by *newfs* (8) and the kernel as filesystems. The kernel also will not swap or place a crash dump on a partition that is not of type **swap**.

The two examples above are equivalent and provide an example of a translating an old style disktab entry into a new style entry. To translate a customized disktab entries you will need: 1) a copy of your current partition tables from the device driver, 2) a copy of the old disktab entry, 3) your current /etc/fstab file. In new disktab entries you should only place those partitions you actually use. There is no need to declare (as was done in the examples above) all of the possible partitions.

If you have changed the disk partition sizes, be sure to make the necessary /etc/disktab changes and label your disks BEFORE trying to access any of your old filesystems! There are two ways to label your disks. The standalone disklabel program is one way. It is also possible to label disks using *disklabel* (8) with the −r option − this works even when running on a kernel which does not support labels (−r reads and writes the raw disk, thus it is possible to label disks on an older kernel as long as the *disklabel* (8) program is present).

4. SYSTEM SETUP

At this point you should have a system which can successfully boot from disk into multi-user mode. If it does not, return to the setup steps in Chapter 2 before proceeding.

This section describes procedures used to set up a PDP-11 UNIX system. These procedures are used when a system is first installed or when the system configuration changes. Procedures for normal system operation are described in the next section.

4.1. First, a little housekeeping

There are a number of tasks that you should perform after first installation, as well as in the obvious cases where these steps might need to be repeated.

4.1.1. Saving a backup copy of the GENERIC kernel

You can get yourself into a situation where a custom kernel does not boot, or other things have gone wrong. Having a copy of the GENERIC kernel around can be the difference between a simple recovery and needing to do a re-install. By convention, this saved copy of the GENERIC kernel is called /genunix and is created as follows:

```
su
cd /
cp -p /unix /genunix
```

NOTES: 1) You should perform this *before* replacing /unix with a customized kernel. 2) Some Patch Levels may make major changes to some part of the system. In such cases, it is advisable to build a new GENERIC kernel and replaced the saved /genunix with the new kernel. Those Patch Levels will include a suggestion to rebuild the GENERIC kernel when necessary.

The following shows the steps to use the /genunix kernel by specifying it on the command line during the system boot process.

```
70Boot from xp(0,0,0) at 0176700
: genunix
Boot: bootdev=05000 bootcsr=0176700

2.11 BSD UNIX #157: Sat Oct 24 16:44:40 CDT 2026
    sms@vernon.2bsd.com:/usr/src/sys/GENERIC
[...]
```

4.1.2. Setting the correct time zone

It is quite possible that you are located in a different time zone than US Central Time where the GENERIC kernel was built, or where the "canned" (prebuilt) disk images were built. The kernel keeps time in GMT by default (and should generally not be changed). Instead, the system provides a large selection of time zone rules in the directory tree based at /usr/share/zoneinfo. Locate the appropriate file and install it as /etc/localtime, as shown in the following example (assuming US Eastern Time, with automatic switching to Daylight Saving Time and back as needed):

```
su
cd /usr/share/zoneinfo
cp -p EST5EDT /etc/localtime
```

NOTES: 1) In addition to the top-level time zone rules, there are subdirectories for various countries which have local exceptions. For example, the standard US Mountain Time zone rule is in /usr/share/zoneinfo/MST7MDT, but most of Arizona does not observe Daylight Saving Time, so it has its own rule in /usr/share/zoneinfo/US/Arizona. 2) Governments sometimes change their time zone rules, often with little

advance notice. Some of the zoneinfo files included with 2.11BSD may be out-of-date. Work is planned (as part of a future Patch Level) to import the latest IANA tzdata.

4.1.3. Setting the system host name

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 host name, then edit the file /etc/rc.conf and change the "rc_configured" line at or about line 7 from "NO" to "YES", and then change the "hostname" line at or about line 32 to something like:

hostname=*myname.my.domain*

NOTES: 1) Be sure to remove the leading pound (comment) character as well as the blank space following it ("# ") in the hostname line or your hostname will not be configured. 2) If you are running a system with networking enabled, you should instead refer to Chapter 5, "Network Setup".

Setting the hostname defines the value returned by the *gethostname* (2) system call. Your hostname should be a fully qualified domain name. Since you are not running a system with networking, you can use a name like "mysystem.example.com". 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.

4.1.4. Setting root's email forwarding

By default, the system will mail the output of the scheduled daily / weekly / monthly reports to the root user. Since it is not good security practice to run as root when not needed, you can set root's email forwarding elsewhere, either to another user account on the system or (if you have networking enabled) to any email address the system can reach. The address to forward mail to is saved in root's .forward file (*/root/.forward*). To add or change forwarding, use the one of the following commands, depending on whether you are forwarding to a local or remote user:

cd /root; echo "*username*" > .forward
cd /root; echo "*username@hostname*" > .forward

4.1.5. Adjusting the line time clock frequency (if needed)

Both the GENERIC kernel and the "canned" CLASSIC system disk image's kernel are built with "LINEHZ 60", while the "canned" PIDP11 system disk image's kernel is built with "LINEHZ 50". A late-breaking change during PL500 development made this setting configurable without needing to build a new kernel. At the moment, this involves giving the command "sysctl -w kern.linehz=N" where N is either 50 or 60. Ideally, this should be done before the system goes into multi-user mode, but it can be made at any time. **NOTE:** If you are running the Network Time protocol (refer to Chapter 5, Network Setup for more information) it is important to have the correct line time clock frequency set correctly before NTP is enabled.

4.2. Disk configuration

This section describes how to layout filesystems to make use of the available space and to balance disk load for better system performance. **NOTE:** The information in this section was of far greater importance on PDP-11 systems with relatively slow peripherals. If you are running on an emulator, you can create pretty much any filesystem layout and combination of disk drives that you want, without affecting performance. In this case, you can skip to Section 4.3, "Configuring terminals" if you installed from the distribution tape, or to Section 4.4, "Adding and managing users and groups" if you are running a "canned" (prebuilt) disk image.

This section remains both for users with actual PDP-11 systems, as well as documenting the historic procedures used.

4.2.1. Disk naming and divisions

Each physical disk drive can be divided into up to 8 partitions; UNIX typically uses only 3 or 4 partitions. For instance, on an RP06 the first partition, xp0a, is used for a root filesystem, a backup thereof, or a small filesystem like, /tmp; the second partition, xp0b, is used for swapping or a small filesystem; and a combination of the remaining partitions (xp0d, xp0e, xp0f, xp0g, xp0h) would hold user filesystems.

> **Warning:** for disks on which DEC standard 144 bad sector forwarding is supported, the last track and up to 126 preceding sectors contain replacement sectors and bad sector lists. Disk-to-disk copies should be careful to avoid overwriting this information. See *bad144* (8). Bad sector forwarding is optional in the **hk** and **xp** drivers. The partition sizes listed in */etc/disktab* that *newfs* (8) uses automatically reserve the maximum amount of room that may be used by bad block forwarding on a disk.
>
> Note also that bad144 style bad block forwarding *cannot* be used with SI controllers on the xp driver as the controllers use their own internal scheme for bad block forwarding, and you can in fact make your disks unusable on the SI controllers if you write anything in the last five cylinders. The partition sizes in */etc/disktab* also handle this constraint automatically.
>
> The generic distribution kernel does not do bad block forwarding. There is unfortunately no way to run bad144 style bad block forwarding on some of your disks but not others. As a final bug the hk and xp drivers do not reread the bad sector forwarding information when disk packs are changed and so will erroneously use bad block forwarding information from the wrong packs!

The space available on a disk varies, not surprisingly, per device. Disk labels make a table giving sizes meaningless since there are no predefined partition sizes embedded in the kernel any longer. The root filesystem (**a**) must be at least 4MB, preferably 8 to 10MB if possible. The swap area (almost always the **b** partition) should be 4MB or a little more. It is a rare case where more than 5 or 6MB of swap space is required.

The system (boot) disk has a swapping area and a root filesystem. Other drives may use those partitions for data. **Remember:** the **a** partition must start at sector 0 or else neither *disklabel* (8) nor the kernel will be able to read/write the disk label.

The distributed system binaries occupy about 34 Megabytes while the major sources occupy another 36 Megabytes. Adding in the miscellaneous sources, a few locate works of art bring the total for a complete system to about 90 Megabytes. This overflows RK07, RL02 and RM03 systems, but fits easily on most other hardware configurations. 2.11BSD is quite happy on RD54 or larger.

Be aware that the disks have their sizes measured in disk sectors (512 bytes), while the UNIX filesystem blocks are 1024 bytes each. Thus, if a disk partition has 10000 sectors (disk blocks), it will have only 5000 UNIX filesystem blocks, and you *must* divide by 2 to use 5000 when specifying the size to the *mkfs* command for instance. The *newfs* (8) program performs this calculation automatically. You should **never** need to run *mkfs* manually. 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 filesystems specifies disk partition sizes in sectors; the default sector size may be overridden with the "se" attribute. **Note** that the only sector size currently supported is NBPG as defined in */sys/pdp/machparam.h*. This restriction is enforced in several places in the disk labeling process as a safeguard against specifying a sector size other than NBPG (512). Any other sector size would produce strange results and almost certainly curdled filesystems.

4.2.2. Layout considerations

There are several considerations in deciding how to adjust the arrangement of things on your disks. The most important is making sure that there is adequate space for what is required; secondarily, throughput should be maximized. Swap space is an important parameter since it defines the maximum process image load that may be run. If, for instance, your swap area was smaller than the amount of main memory available after the kernel took its share, some of your memory would never be used.

Many common system programs (C, the editor, the assembler etc.) create intermediate files in the /tmp directory, so the filesystem where this is stored also should be made large enough to accommodate most high-water marks; if you have several disks, it makes sense to mount this in a "root" (i.e. first partition) filesystem on another disk. All the programs that create files in /tmp take care to delete them, but are not immune to rare events and can leave dregs. The directory should be examined every so often and the old files deleted.

It is critical for good performance on real PDP-11 systems to balance the disk load across multiple drives (if available). On an emulator, all drives are normally container files in the same directory on the host system, so there is much less need to balance the disk load (as it is all accessing the same host device). To mimic certain real historic systems, it may be desirable to have multiple disk drives, even though there is almost no performance benefit from doing so.

There are at least five components of the disk load that you can divide between the available disks:

1. The root filesystem.
2. The /tmp filesystem.
3. The /usr filesystem.
4. The user files.
5. The swapping activity.

The following possibilities are ones that have been used at times when 2, 3 and 4 disks were available:

	disks		
what	2	3	4
/	0	0	0
tmp	1	2	3
usr	1	1	1
swapping	0	2	2
users	0	0+2	0+2
archive	x	x	3

4.2.3. Implementing a layout

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

As an example, consider a system with RA80's. On the first RA80, ra0, we will put the root filesystem in ra0a, and the /usr filesystem in ra0c, which has enough space to hold it and then some. The /tmp directory will be part of the root filesystem, as no filesystem will be mounted on /tmp. If we had only one RA80, we would put user files in the ra0c partition with the system source and binaries.

If we had a second RA80, we would place */usr* in ra1c. We would put user files in ra0c, calling the filesystem /mnt. We would put swap on *ra0b*. We would keep a backup copy of the root filesystem in the **ra1a** disk partition and put /tmp on *ra1b*. */etc/fstab* would then contain

```
/dev/ra0a:/:rw:1:1
/dev/ra0b::sw::
/dev/ra0c:/mnt:rw:1:2
/dev/ra1b:/tmp:rw::
/dev/ra1c:/usr:rw:1:2
```

To make the /mnt filesystem we would do:

```
# cd /dev
# ./MAKEDEV ra1
# newfs ra1c ra80
(information about filesystem prints out)
(to specify an alternate m value: newfs -m # ra1c ra80)
(where # is between 1 and 31)
# mkdir /mnt
# mount /dev/ra1c /mnt
```

4.3. Configuring terminals

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

Terminals connected via DZ11 interfaces are conventionally named **ttyDD** where DD is a decimal number, the ''minor device'' number. The lines on dz0 are named /dev/tty00, /dev/tty01, ... /dev/tty07. By convention, all other terminal names are of the form **tty**CX, where C is an alphabetic character according to the type of terminal multiplexor and its unit number, and X is a digit for the first ten lines on the interface and an increasing lower-case letter for the rest of the lines. C is defined for the number of interfaces of each type listed below. Since tty structures are approximately 78 bytes each, it is highly doubtful that more than 3 or 4 terminal interface boards will ever be attached to a PDP-11 (especially in a BA23 cabinet).

Interface Type	Characters	Number of lines per board	Number of Interfaces
DZ11	see above	8	10
DH11	h-o	16	8
DHU11	S-Z	16	8
pty	p-u	16	6

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

The format of the /etc/ttys file is the same in 2.11BSD as in 2.10BSD and 4.3BSD. Each line in the file is broken into four tab separated fields (comments are shown by a '#' character and extend to the end of the line). For each terminal line the four fields are: the device (without a leading /dev), the program /etc/init should startup to service the line (or **none** if the line is to be left alone), the terminal type (found in /etc/termcap), and optional status information describing if the terminal is enabled or not and if it is ''secure'' (i.e. the super user should be allowed to login on the line). All fields are character strings with entries requiring embedded white space enclosed in double quotes. Thus, a newly added terminal /dev/tty00 could be added as

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

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

Dialup terminals should be wired so that carrier is asserted only when the phone line is dialed up. For non-dialup terminals from which modem control is not available, you must either wire back the signals so that the carrier appears to always be present, or show in the minor device number that carrier is to be assumed to be present by adding 128 decimal to the minor device number when creating the device node. This differs from 4.3BSD where the softcarrier state is specified at kernel configuration time.

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

ttyp0 none network (Note the fourth field is not needed here.)

When the system is running multi-user, all terminals that are listed in /etc/ttys as **on** have their line 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 (/usr/sbin/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.

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

To further complicate things, some emulators (such as SIMH) have the ability to map network ports on the host system to emulated terminal multiplexor lines on the PDP-11 system. The benefit of this is that you can allow network logins to your 2.11BSD system without setting up networking under 2.11BSD. The CLASSIC "canned" distribution provides this in its classic.ini file, as well as examples of mapping physical terminal ports on the host system to emulated multiplexor ports on the PDP-11 system.

4.4. Adding and managing users and groups

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)*. This is currently a manual procedure. A future Patch Level may provide an *adduser* command that prompts you for each setting and then does the work for you.

Users are assigned to their primary group by specifying a group number (from the file /etc/group). It is important to realize that certain groups listed in the /etc/group file have special capabilities and should **NOT** be normally assigned to users. Normally you would use the staff or guest group and avoid assigning users to any of the other existing groups, as that would grant those users additional permissions which is likely undesirable.

You can also add additional groups to the /etc/group file. For example, if you had a number of users working on a ‘‘widgets’’ project, you could add a widgets group to the /etc/group file. **NOTE:** Take care to pick a new, unique number for your new group - do not re-use an existing group number.

In addition to the primary group for each user, users can be granted membership in additional groups by listing them (separated with commas if there is more than one user who should be in the additional group). For example, if users ‘‘fred’’ and ‘‘barney’’ should have the ability to manage some system operations, but not have full control over the system, you can add them to the "operator" group. For example, to allow ‘‘fred’’ and ‘‘barney’’ to shut down or reboot the system and similar tasks, you would add them to the operator group as shown in the following example. Before:

operator:*:5:

After:

operator:*:5:fred,barney

You can edit user entries in the password file using the *vipw(8)* utility. You can also use the *vipw(8)* utility to remove a user by removing their entry from the password file. **NOTE:** When removing a user, it is important to remove them from the /etc/group file (if they are named there) as well as either archiving or deleting their login directory, as appropriate.

As the system administrator, you can change a user's password for them by using the *passwd(1)* utility. Individual users can use the same utility, although normally only the system administrator is allowed to

change the passwords of other users.

4.5. Setting up the mail system

The mail system consists of the following commands:

/bin/mail	old standard mail program, *binmail* (1)
/usr/ucb/mail	UCB mail program, described in *mail* (1)
/usr/sbin/sendmail	mail routing program
/usr/spool/mail	mail spooling directory
/etc/aliases	mail forwarding information
/usr/bin/newaliases	command to rebuild binary forwarding database
/usr/ucb/biff	mail notification enabler†
/usr/libexec/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 /etc/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 /usr/libexec/comsat is notified, which in turn notifies users who have issued a "*biff* y" command that mail has arrived†.

To set up the mail facility you should read the instructions in the file READ_ME in the directory /usr/src/usr.lib/sendmail and then adjust the necessary configuration files. Documents describing *sendmail*'s operation and installation are also included on the distribution tape.

4.5.1. Setting up a UUCP connection

The version of *uucp* included in 2.11BSD is an enhanced version of the one originally distributed with 32/V‡. The enhancements include:

- Support for many auto call units and dialers in addition to the DEC DN11.
- Breakup of the spooling area into multiple subdirectories
- Addition of an *L.cmds* file to control the set of commands that may be executed by a remote site
- Enhanced "expect-send" sequence capabilities when logging in to a remote site
- New commands to be used in polling sites and obtaining snap shots of *uucp* activity
- Additional protocols for different communication media

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

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

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

The *uucp* support is located in three major directories: /usr/bin, /etc/uucp, and /usr/spool/uucp. User commands are kept in /usr/bin, operational commands in /etc/uucp, and /usr/spool/uucp is used as a spooling area. The commands in /usr/bin are:

---

† comsat and biff are only available under systems configured for networking support.

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

/usr/bin/uucp	file-copy command
/usr/bin/uux	remote execution command
/usr/bin/uusend	binary file transfer using mail
/usr/bin/uuencode	binary file encoder (for *uusend*)
/usr/bin/uudecode	binary file decoder (for *uusend*)
/usr/bin/uulog	scans session log files
/usr/bin/uusnap	gives a snap-shot of *uucp* activity
/usr/bin/uupoll	polls remote system until an answer is received
/usr/bin/uuname	prints a list of known uucp hosts
/usr/bin/uuq	gives information about the queue

The important files and commands in /etc/uucp are:

/etc/uucp/L-devices	list of dialers and hard-wired lines
/etc/uucp/L-dialcodes	dialcode abbreviations
/etc/uucp/L.aliases	hostname aliases
/etc/uucp/L.cmds	commands remote sites may execute
/etc/uucp/L.sys	systems to communicate with, how to connect, and when
/etc/uucp/SEQF	sequence numbering control file
/etc/uucp/USERFILE	remote site pathname access specifications
/usr/sbin/uucico	*uucp* protocol daemon
/etc/uucp/uuclean	cleans up garbage files in spool area
/usr/libexec/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. 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, my L.sys file looks something like:

```
elisa Any ACU 1200 7064297  "" \\r\\c ogin-EOT-ogin-\\r\\c-ogin xelisa assword: XXX
etn-ra Any ACU 1200 8891237  "" \\r\\c ogin nuucp assword XXX
anagld Never ACU 2400 8894517 name:-EOT-name: uucp assword: XXX
```

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. 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

The networking in 2.11BSD runs in supervisor mode separate from the main kernel. There is room without overlaying to hold both a SL/IP and an Ethernet driver. The networking portion of the kernel resides in "/netnix", and is loaded after the kernel is running. Since the kernel only looks for the file "/netnix", it will panic if it is unable to load "/netnix". **NOTE**: The "/unix" and "/netnix" imagines must have been created at the same time - do not attempt to use mismatched images or the system will panic.

2.11BSD provides support for the DARPA standard Internet protocols IP, ICMP, TCP, and UDP. Network services are split between the kernel (communication protocols) and user programs (user services such as TELNET and FTP). This section describes how to configure your system to use the Internet networking support.

The "correct" use of the networking in 2.11BSD is almost always with the local host name and addresses in the */etc/hosts* file, your local domain name and a set of nameservers in the */etc/resolv.conf* file, and a default route to a gateway for all other network traffic. This will be explained in detail below. In particular, do not run *routed(8)* or *named(8)* unless you're extremely sure that you know what you're doing. This is doubly true if SL/IP is being used as the primary connection to the outside world.

5.1. Required kernel configuration settings

To configure the kernel to include the Internet communication protocols, define the INET option. This automatically defines the NLOOP option. Include the pseudo-device "pty" in your machine's configuration file using the NPTY options. The "pty" pseudo-device forces the pseudo terminal device driver to be configured into the system, see *pty* (4). The NLOOP option forces inclusion of the software loopback interface driver. The loop driver is used in network testing as well as for the system talking to itself rather than transmitting the data over the wire.

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

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

Device name	Manufacturer and product
de	DEC DEUNA/DELUA 10Mb/s Ethernet
qe	DEC DEQNA 10Mb/s Ethernet
qt	DEC DELQA-YM (DELQA-T, DELQA-Plus) 10Mb/s Ethernet
sl	Serial line IP (SL/IP)
ec	3Com 10Mb/s Ethernet‡
il	Interlan 1010 and 10101A 10Mb/s Ethernet interfaces‡
vv	Proteon ProNET - Token Ring Interface‡
acc	LH/DH-11 1822 IMP/PSN Interface‡

SL/IP is also available. It is surprisingly efficient. Over a 9600 baud line it is not unusual to see **ftp** rates in the 800 bytes per second range.

NOTE: Configuring, building and installing a kernel is described in Appendix A.

† All DEC PDP-11 Ethernet controllers were 10Mb/s devices. Therefore, we refer to any connected Ethernet as 10Mb/s, even when running on an emulator where the Ethernet speed might be much faster.

‡ These devices were known to work in 2.11BSD at one time but have not been tested (due to lack of suitable hardware) in many years and may not work.

NOTE: All of the following information and examples are based on the new */etc/rc.conf* configuration method. If you have upgraded from a system prior to PL500, your existing network configuration will be preserved and you can skip the following sections. However, while both old-style and new-style network startup scripts are supported on PL500, future patch levels will likely only provide changes to the new-style startup scripts.

5.2. Basic network configuration

Basic network configuration involves setting up enough of the networking that you can connect to the system via *telnet* and complete the rest of the configuration tasks from a terminal (real or emulated). This dates back to the era when PDP-11 systems often had printing terminals as consoles, which made editing files a bit of a chore. If you are running 2.11BSD on a PDP-11 with a video terminal or emulator, you can continue with the advanced configuration steps via the console if you prefer.

5.2.1. Enabling networking and basic TCP/IP configuration in rc.conf

Edit the file */etc/rc.conf* and make the following changes:

rc_configured=	- Change from NO to YES
hostname=	- Change to your desired host/domain name (like mypdp.example.com)
address=	- Change to the IP address for this system
netmask=	- Change to the network mask for your network
gateway=	- Change to the default gateway for your network
interface=	- Change to the name of your network device (refer to the table in Section 5.1)

In years past, you would have received the necessary information from your network manager. Today, *you* are probably your network manager. You can likely determine your domain name, netmask and gateway by looking for this information on either your router or on other computers on your network. The exact procedure depends on the software those devices are running, and all of the possibilities are too extensive to list here.

Your IP address must be unique on your network, or both your 2.11BSD system and the other system assigned the same IP address will have trouble communicating on the network. On modern systems, the DHCP protocol is normally used to automatically assign IP addresses as well as other settings. 2.11BSD pre-dates DHCP and requires these settings to be configured manually†.

The comments in the */etc/rc.conf* file provide some guidance.

5.2.2. Editing the /etc/hosts file

Edit the file */etc/hosts* and replace the line following "# Define your host's IP address and name (both with and without domain name)" so that it has your IP address (it must match the definition in */etc/rc.conf*), a tab, the full hostname with domain, a space character, and the hostname without the domain. The sample line in the file should make this clear.

Next, edit the line following "# Edit the following line *ONLY* to change the domain name" to match the domain name (without the host name) you selected in */etc/rc.conf* above.

5.2.3. Editing the /etc/resolv.conf file

Edit the */etc/resolv.conf* file to specify your domain name on the line that begins with "domain". Again, this is just the domain name (such as example.com), not the complete hostname (such as mypdp.example.com).

---

† It is unlikely that you will have your own domain name or public IP address block allocation. In that case, you will want to use the domain name and an unused IP address in the same block that the other systems on your network are using. Note that many mail servers on the Internet will refuse to accept email from a host/domain name that doesn't exist (example.com is specifically reserved for use in sample configurations such as this). That's just the way it is in the modern world.

The name servers listed on each line that begins with "nameserver" are generally sufficient - they are public name servers. If you are operating your own name server(s), you may want to change these lines to match your name server(s).

5.3. Advanced network configuration in /etc/rc.conf

There are quite a few optional network services which are enabled or disabled via settings in */etc/rc.conf*. Some of those services also have their own configuration files, which will also be discussed in the appropriate section below. The sections below are in the order they appear in the */etc/rc.conf* file.

5.3.1. The rwhod service

The rwhod service (when set to YES) enables both the broadcasting of status (host name, uptime, load average, and logged-in users) to other hosts on the local network as well as receiving the same data from other hosts on the local network. When rwhod is enabled, the *rwho(1)* and *ruptime(1)* commands will display this information. If rwhod is not enabled, these commands will not report any information. **NOTE:** Whether or not to run the rwhod service can be a matter of some debate, as it broadcasts information across the local network. If none of your other systems have rwhod enabled, there is no point in enabling it on 2.11BSD. Otherwise, enabling it will allow you to see the status of those hosts and those hosts will be able to see the status of your 2.11BSD system.

5.3.2. The named service

The named service (when set to YES) causes the BIND name server to run. You almost definitely do not want to do this as BIND consumes a good amount of system resources, while there are public name servers (which you configured in Section 5.2.3) which you can use. Therefore, configuring BIND is beyond the scope of this document.

5.3.3. The sendmail service

sendmail is the utility that processes email. It is normally already enabled, but some additional configuration steps are required. sendmail configuration is so arcane that many books have been written about it. However, the necessary configuration changes for 2.11BSD are relatively simple.

First, edit the */etc/sendmail.cf* file and if necessary, change the line that reads "DDEXAMPLE.COM" to use your local domain (if not using example.com). As an example **only**, if your domain was FOO.COM† you would change the line to read "DDFOO.COM" ("DD" means "Define Domain").

Next, also in */etc/sendmail.cf*, edit the line that reads "DARELAY.EXAMPLE.COM" and change it to the name of a host that will accept SMTP email from your 2.11BSD system. Similar to the above section, if your SMTP server was called SMTP.FOO.COM†, you would change this line to read "DASMTP.FOO.COM" ("DA" means "Define ARPANET relay").

Finally, you must add a host definition for your SMTP server in the file */etc/hosts* in the line following "# Define your SMTP gateway/relay system (used in sendmail.cf)". Like the host IP address and name you set in Section 5.2.2, this will be an IP address, a tab, the full host name you used in "DA" above, a space, and then just the host name (without the domain name part).

5.3.4. The NTP (Network Time Protocol) service

NTP is a distributed time service available on the Internet. It provides a relatively high-precision time reference based on GMT.‡

Enabling this involves setting both the ntpd and ntp lines to YES. You will also need to edit the */etc/ntp.conf* file to define several network time servers. First, remove the entire "peer 127.0.0.1" line and replace it with:

† Please don't use FOO.COM - that is only being used as an example. Use whatever your local domain is, or leave it set to EXAMPLE.COM if you don't have a local domain.

‡ You will also need to have correctly set your system's local time zone, as described in Section 4.1.2, "Setting the correct time zone".

```
server  0.pool.ntp.org†
server  1.pool.ntp.org
server  2.pool.ntp.org
server  3.pool.ntp.org
```

5.3.5. The timed service

timed is an older method of synchronizing time across multiple systems. It should normally be left disabled unless you have older systems also running timed.

5.3.6. The routed service

routed is an older method of discovering and reporting network routes. The routedflags should be set to NO, as modern configurations separate routers from host systems, and 2.11BSD uses the default gateway configured in Section 5.2.1.

5.4. Configuring network access

The previous steps set up basic TCP/IP connectivity and optionally configured additional services. There are a few more steps you will need to take to allow users to access your 2.11BSD system.

5.4.1. Creating the pseudo terminals

When you configured your kernel (or if you're using one of the "canned" disk images) the number of pseudo terminals was specified in the NPTY section of the kernel configuration. If they don't already exist, you will need to create the device nodes for the pseudo terminals. These device nodes are created in sets of 16 at a time. Each pseudo terminal really consists of two files in /dev: a master and a slave. The master pseudo terminal file is named /dev/ptyp?, while the slave side is /dev/ttyp?. Pseudo terminals are also used by several programs not related to the network. To create 16 pseudo terminals (normally plenty) execute the following commands:

```
# cd /dev
# MAKEDEV pty0‡
```

More pseudo terminals may be made in groups of 16 by specifying *pty1*, *pty2*, *pty3*, etc.

5.4.2. Editing /etc/ttys

Now that you have created the pseudo terminals, you may need to let the system know to allow logins on them. To do this, edit the */etc/ttys* file In addition to creating the pseudo terminals. The file should already contain definitions for the first 16 pseudo terminals, such as the following example:

```
ttyp0   none            network     secure§
```

If you created additional pseudo terminals as shown in the previous section, you should add them to the end of the */etc/ttys* file.

---

† The distributed NTP service relies on all systems "playing by the rules". First, you should only use "server" lines in the */etc/ntp.conf* file, not "peer" (server means you are only asking for the time, while peer means that you are willing to ask for or supply the time). Second, you should only use NTP servers that have offered to provide public time service to any system. The pool.ntp.org servers are a rotating set of public time servers intended for clients like 2.11BSD.

‡ If these device nodes already exist, you will receive a bunch of "File exists" error messages. They are harmless and just indicate that your system already has had the pseudo terminals created.

§ "secure" means that root is allowed to log in directly on that terminal. You probably do not want to allow this as it can be a major security hole. Instead, users should log in with their regular username and then use the *su(1)* command to obtain root access (if they are in group ‘‘wheel’’). Removing "secure" from any ttyp? lines where it appears is strongly recommended.

5.4.3. Editing /etc/hosts.allow

The */etc/hosts.allow* file controls which systems can access various network services. As distributed, it looks like this:

```
ftpd: .example.com LOCAL
rlogind: .example.com LOCAL
rshd: .example.com LOCAL
tftpd: .example.com LOCAL
popper: .example.com LOCAL
telnetd: .example.com LOCAL
```

To control incoming telnet sessions, you would edit the ''telnetd'' line. The format of each line is described in the *hosts_access(5)* man page, accessed as follows:

```
man 5 hosts_access
```

The format and various options are described in detail in that man page and will not be repeated here. Briefly, you would replace the ''.example.com'' line with your local domain is (refer to Section 5.2 for additional information).

5.5. Network servers

In 2.11BSD most of the server programs are started up by a ''super server'', the Internet daemon. The Internet daemon, *inetd*, acts as a master server for programs specified in its configuration file, */etc/inetd.conf*, listening for service requests for these servers, and starting up the appropriate program whenever a request is received. The configuration file contains lines containing a service name (as found in */etc/services*), the type of socket the server expects (e.g. stream or dgram), the protocol to be used with the socket (as found in */etc/protocols*), whether to wait for each server to complete before starting up another, the user name as which the server should run, the server program's name, and at most five arguments to pass to the server program. Some trivial services are implemented internally in *inetd*, and their servers are listed as ''internal.'' For example, an entry for the file transfer protocol server would appear as

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

or if you are using the *tcp_wrapper*† program as

```
ftp     stream      tcp     nowait      root    /usr/libexec/tcpd    ftpd -l
```

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

5.6. Network data bases

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

File	Manual reference	Use
/etc/hosts	*hosts* (5)	host IP addresses, names and aliases
/etc/networks	*networks* (5)	network names
/etc/services	*services* (5)	list of known services
/etc/protocols	*protocols* (5)	protocol names
/etc/hosts.allow	*hosts_access* (5)	sites who can connect to services
/etc/hosts.deny	*hosts_access* (5)	sites blocked from services

(Continued on next page)

† The *tcp_wrapper* program is what consults the */etc/hosts.allow* file, discussed above. Not all services can use *tcp_wrapper*, either due to being an internal service or various other conditions.

/etc/hosts.equiv	*rshd* (8)	list of "trusted" hosts
/etc/rc.local	*rc* (8)	command script for starting local servers
/etc/ftpusers	*ftpd* (8)	list of "unwelcome" ftp users
/etc/hosts.lpd	*lpd* (8)	list of hosts allowed to access printers
/etc/inetd.conf	*inetd* (8)	list of servers started by *inetd*

5.6.1. Use of /etc/hosts

When using the host address routines that use Internet name servers, the file */etc/hosts* is only used for setting interface addresses and at other times when the network is not yet started, and therefore it need only contain addresses for local hosts.

If you are connected to the Internet, it is highly recommended that you use the default name server resolver routines for host name and address lookups, as this provides lookup service for all hosts and addresses on the Internet. If your system is not connected to the Internet, you can use the */etc/hosts* file to list the names and addresses of hosts on your local network.

5.6.2. /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 host's name is located in *hosts.equiv* and the user has an account on the server's machine, then service is rendered (i.e. the user is allowed to log in, or the command is executed). Users may expand this "equivalence" of machines by installing a *.rhosts* file in their login directory. The root login is handled specially, bypassing the *hosts.equiv* file, and using only the */.rhosts* file.

Thus, to create a class of equivalent machines, the *hosts.equiv* file should contain the *official* names for those machines.

5.6.3. /etc/rc.local

Most network servers are automatically started up at boot time by the network startup scripts or by the Internet daemon (see above). These include the following:

Program	Server	Started by
rshd	shell server	inetd
rexecd	exec server	inetd
rlogind	login server	inetd
telnetd	TELNET server	inetd
ftpd	FTP server	inetd
fingerd	Finger server	inetd
tftpd	TFTP server	inetd
rwhod	system status daemon	/etc/rc.conf
syslogd	error logging server	/etc/rc.conf
sendmail	SMTP server	/etc/rc.conf
ntp	network time protocol (NTP)	/etc/rc.conf
named	local name server (DNS)	/etc/rc.conf†
routed	routing table management daemon	/etc/rc.conf†
timed	time distribution service	/etc/rc.conf†

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

† These services should normally **not** be enabled.

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

```
if [ -f /usr/local/bin/myserver ]; then
        /usr/local/bin/myserver & echo -n ' myserver'       >/dev/console
fi
```

5.6.4. /etc/ftpusers

The FTP server included in the system provides support for an anonymous FTP account.

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

```
# 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.

NOTE: Mode 777 on the 'pub' directory can and has been abused! Changing the mode to 555 is a good choice but would require administrative assistance for placing files in the 'pub' directory. Probably not a bad idea though.

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.

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.

5.7. Important additional words about security

2.11BSD dates from an era when the Internet was a much smaller network, and many users knew "the management" of other Internet sites. As such, it was possible to just call or email someone at another

† The choice of */etc/inetd.conf* or */etc/rc.local* depends on whether your network server should be activated "on demand" (use */etc/inetd.conf*) or "always on" (use */etc/rc.local*).

site if people from that site were trying to break into or otherwise damage your system. In the intervening 35+ years, the Internet has become much larger, both in terms of the number of users and geographic reach. We also have cases where complaints to the management of another site are routinely ignored. Compounding things, recent "tests" of AI agents have sometimes gone rogue and started probing other Internet sites for security vulnerabilities.

5.7.1. DO NOT put a 2.11BSD system directly on the public Internet

It is **strongly** discouraged to have a 2.11BSD system directly accessible from the public Internet. Many users will be running 2.11BSD behind a router that does NAT translations, which will prevent many forms of external attacks. Or a separate firewall can be used, depending on your network configuration. Either NAT or a firewall will generally let your 2.11BSD system contact the outside Internet, while preventing access from outside.

There has been 35+ years of security research and bug fixes to the *BSD code since 2.11BSD was first released. While many security fixes have been incorporated into 2.11BSD, there is no representation that it is completely secure.

5.7.2. Plaintext passwords

While 2.11BSD incorporates the password hashing algorithms from 4BSD (meaning that the password file does not have plaintext passwords), limitations of the PDP-11 CPU speed† limit the complexity of the hashing. 2.11BSD has neither md5 or sha1 hashing. For similar reasons, this means that 2.11BSD does not support the SSH (Secure Shell) protocol. Only telnet, rlogin, etc. are available. This means that anyone who logs into a 2.11BSD system from the "outside world" is doing so by sending a plaintext password over the Internet from their system to yours, and Bad Guys can intercept that communication and recover your username and password.

5.7.3. The secure keyword in /etc/ttys

When a line in */etc/ttys* has the keyword ‘‘secure’’, it means that people who know the root password can simply log in as root. This should be disabled where it is not needed. For maximum security, only the system console should permit direct root logins via the ‘‘secure’’ keyword and it should be removed from all other lines. Users can log in with their regular username and then use the *su* command (assuming they are in group ‘‘wheel’’) and provide the root password. Usage of *su* should be limited to cases where it is absolutely necessary (normally, for system management tasks). Referring to the previous section, you can see that when using *su* over the Internet will send the root password in plaintext over the network.

5.7.4. Using a VPN

If appropriate, you can set up a VPN (Virtual Private Network) on your router or firewall. This will allow you to establish a secure connection from the Internet to your 2.11BSD system. The VPN will encrypt all data starting at the far end, and will only decrypt it once the data arrives at your router or firewall.

There are many different VPN packages available. Consult your router or firewall documentation to find out how to configure that device, as well as finding out what VPN client software works with the device. Depending on the VPN software in use, configuring your router or firewall for a VPN may provide you with a file to use on the VPN client. The file will contain all of the necessary information for the VPN client to connect to the VPN server.

† As an example, it is possible to emulate a fully-loaded PDP-11/70 system which originally cost nearly $100,000 (in 1980s dollars!) with a Raspberry Pi Model 3B+ or newer, and have the emulation run faster than the original PDP-11/70.

6. SYSTEM OPERATION

This chapter describes procedures used to operate a system running 2.11BSD. Procedures described here are used periodically, to reboot the system, analyze error messages from devices, do disk backups, monitor system performance, recompile system software and control local changes.

6.1. Bootstrap and shutdown procedures

In a normal reboot, the system checks the disks and comes up multi-user without intervention at the console. Such a reboot can be stopped (after it prints the date) with a ^C (interrupt). This will leave the system in single-user mode, with only the console terminal active. It is also possible to allow the filesystem checks to complete and then to return to single-user mode by signaling *fsck* with a QUIT signal (^\).

When cold booting via the system console is needed, you will need to determine the correct command(s) or other procedures appropriate to your CPU model or emulator type. This information (along with a list of disk type to device name mappings) is generally found in the documentation for your CPU model or emulator.

Once you have succeeded in cold booting your system to the bootstrap prompt (:), pressing the carriage return key will cause the default kernel image (either as distributed or as compiled as described in Appendix A) to be booted.

Depending on the configuration options as described in Section 7.2, "Configuring automatic boot and restart" the system will either come up in single-user mode or automatically attempt to enter multi-user mode.

To bring the system up to multi-user operation from single-user mode, all you have to do is press Control-D (^D) on the console terminal. 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).

IMPORTANT: If you have the system configured to come up in single-user mode and have any reason to suspect the system was not cleanly shut down (power failure, system crash, etc.), you should run ‘‘fsck’’ to check the integrity of your filesystem(s). A system configured to boot directly to multi-user mode will try to detect an unclean shutdown and run ‘‘fsck’’ for you. This may involve one or more automatic reboots. It is possible for ‘‘fsck’’ to "give up" and request manual intervention if it cannot automatically repair any detected filesystem inconsistencies.

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

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

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 serious errors occur on peripherals or in the system, the system prints a warning diagnostic on the console. These messages are written to the kernel logger where they are retrieved by *syslogd* (8) via */dev/klog* - *dmesg* (8) is now obsolete. *dmesg* (8) is present in the distribution but no longer used. The message buffer is now 4kb in size and external to the kernel.

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. Some drivers have been modified to use the kernel logger; others still simply do printf statements. 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 files regularly (e.g. with ''tail −r */usr/adm/messages*'').

6.3. Filesystem checks, backups and disaster recovery

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

Dumping of the filesystems should be done on a regular schedule, since once the system is going it is easy to become complacent. Complete and incremental dumps are easily done with *dump* (8).

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*.

A full backup of the system† in ''bootable'' format, including the standalone programs mentioned in Chapter 2, "Installation procedure" (boot, disklabel, mkfs, restor and icheck) plus system binaries and sources is made by changing directories to /sys/pdpstand, editing the Makefile line with "DISK= xp0a" and changing the "xp0a" part to your root partition name‡, making sure a writable physical or emulated tape is attached, and then doing:

# make dump	(creates a dump of the root filesystem)
# make tmscptape	(creates the recovery tape‡)
# make clean	(optional, removes the standalone binaries just built)
# rm /usr/root.dump	(optional, removes the root filesystem dump)

If the root and/or /usr filesystems are damaged by a hardware or software failure you can rebuild a workable disk by doing a restore in the same way that the initial installation was performed, as described in Chapter 2, "Installation procedure". The advantage of having a recovery tape (and updating it as needed) is that an install from your recovery tape will contain your customizations (patches, custom kernel, added users, etc.)

Exhaustion of user filesystem space is certain to occur on an active system with multiple users every now and then and can be managed with a combination of disk quotas (the 4.3BSD disk quota system is available as a kernel configuration option), threatening messages of the day, and personal letters.

If you are running 2.11BSD under an emulator, another possibility is to create a new, large disk drive and attach it. You can then move some or all users to the new drive using the *tar(8)* command as described in the next section. In this case, *tar(8)* is preferred over *dump(8)* as you can select particular directories or files, while *dump(8)* is limited to entire filesystems.

6.4. Moving filesystem data

If you have a tape drive (real or emulated), one way to move a filesystem is to dump it to tape using *dump* (8), use *newfs* (8) to create the new filesystem, and restore the tape using *restor* (8). If you don't want to or can't use tape, *dump* accepts an argument telling where to put the dump. The argument may be a filename or the special name ''-'' meaning standard output. A filesystem can be migrated using a pipeline. For

† This excludes any filesystems and directory trees not on a normal distribution tape. If you have configured any additional filesystems or created new directories under / or /usr, you should back those up as well, since they will not be on the bootable recovery tape.

‡ This procedure uses the same utility that created the original installation tape. Some harmless warning messages will be displayed during the procedure. A future Patch Level will likely remove the need for editing the Makefile, eliminate the warning messages, and so on.

example, to migrate the root filesystem (/), where rXXXa is the name of the destination:

```
# newfs /dev/rXXXa
# dump 0f - / | restor rf - /dev/rXXXa
```

The *restor* program accesses the raw device, restoring inodes with their original inode number as recorded by dump, while disk blocks are allocated from the freelist of the destination filesystem. Defragmenting a filesystem is performed using the *dump* and *restor* programs. Unless you have a spare disk partition to restore to, you will need to use those commands individually, either with tape or to a work file on a filesystem that is **NOT** otherwise used by either command.

If the destination filesystem has fewer inodes than needed to restore the dump, you will need to reinitialize the destination filesystem, using the *-i* argument to *newfs(8)* to increase the number of inodes on the destination filesystem. **NOTE:** lowering the value of *-i* produces more inodes, not less, as the argument following *-i* specifies the number of bytes per inode, not the number of inodes directly. Care must therefore be taken when restoring a dump into a filesystem smaller than the original filesystem.

Filesystems may also be moved by piping the output of a *tar* (1) to another *tar*:

```
# tar -cf - / | cd /dev/rXXXa; tar -xpf -
```

The main difference between the two commands is that the restore command (when used with the r argument) should only be used on a newly-initialized (empty) filesystem, while using tar to extract files can extract them into a filesystem with existing data. When using tar, you should make sure you are not unintentionally overwriting or commingling existing data when restoring from a tar file.

NOTE: If you plan on moving filesystem data between a 2.11BSD system and another Unix-like operating system, you should use tar as it is more compatible across different operating systems.

6.5. Recompiling and reinstalling system software

It is easy to recompile 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 ensure 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 install
```

To recompile all of the system from 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 ensure that the libraries are installed before the remainder of the source, otherwise the loaded images will not contain the new routine from the library. The following sequence will accomplish this.

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

The first *make* removes any existing binaries in the source trees to ensure that everything is reloaded. The next *make* compiles and installs the libraries and compilers, then compiles the remainder of the sources. The final line installs all of the commands not installed in the first phase. This will take about 12 hours on a reasonably configured 11/44.

6.6. Making local modifications

/usr/new is used by default for the programs that constitute the contributed software portion of the distribution but which may not have man pages installed. Locally written commands that aren't distributed (or whose man pages are not up to date) 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 People using /usr/local commands are made aware that the programs may not be in the base system yet.

6.7. Accounting

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

If you need to recharge for computing time, you can develop procedures based on the information provided by these commands. A convenient way to do this is to give commands to the clock daemon *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.

6.8. Resource control

Resource control in 2.11BSD is more elaborate than in previous PDP-11 UNIX systems. The resources consumed by any single process can be limited by the mechanisms of *setrlimit* (2). As distributed, the mechanism is voluntary, though sites may choose to modify the login mechanism to impose limits. Csh now has the *limits* built-in command enabled. Another available option is the 4.3BSD disk quota system.

6.9. Files that need periodic attention

The discussion of system operations is concluded 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/master.passwd	password file; each account has a line
/etc/rc.local	local system restart script; runs reboot; starts daemons
/etc/inetd.conf	local internet servers
/etc/hosts	host name data base
/etc/networks	network name data base
/etc/netstart	Startup file to configure network
/etc/services	network services data base
/etc/hosts.equiv	hosts under same administrative control
/etc/syslog.conf	error log configuration for *syslogd* (8)
/etc/ttys	enables/disables ports
/etc/crontab	commands that are run periodically
/etc/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 – CONFIGURING AND BUILDING A NEW KERNEL

The 2.11BSD kernel has a number of parameters and options that can be used to tailor the kernel to site specific needs. This appendix lists the parameters and options used in the kernel. The parameters have numeric values, usually table sizes. The options flags are either defined or undefined (via the values YES or NO respectively.)

Once you have identified all of your desired options, this appendix describes how to build, install, and test your new kernel.

7.1. Configuring the number of mountable filesystems (NMOUNT)

Because of time constraints the **NMOUNT** constant was not moved into the kernel configuration file where it belongs. **NMOUNT** is used to configure the number of mountable filesystems in 2.11BSD. Since each slot in the kernel mount table takes up close to a half KB of valuable kernel data space, the distribution kernel comes configured with **NMOUNT** set to 6. This may be too small for systems using multiple disk drives and should be increased to the number of filesystems you expect to mount.

NMOUNT is defined in */sys/h/param.h* . If you change its value, you must recompile the kernel (obviously) and the following applications: *mount* , *quotaon* , *edquota* , *umount* , and *df* .

7.2. Configuring automatic boot and restart

NOTE: The following only applies to the distribution tape - the "canned" (pre-built images) have automatic boot and restart configured.

The distribution tape contains a configuration which will wait for user input (normally a carriage return) before loading the kernel, in order to allow a different kernel to be specified. Likewise, after the system is in single user mode a Control-D (^D) is needed to switch into multi user mode. There are two settings that control this behavior.

7.2.1. Setting BOOTDEV

The first is the kernel configuration setting "BOOTDEV". As shipped, it is set to "NONE". To enable automatic restart and switching to multi user mode after a "shutdown -r" command, this should be set to match the type of disk you are using. For example, "rm" if you are booting from an RM or RP disk.

7.2.2. Configuring AUTOMULTIUSER

The second setting controls what happens when the system is "cold booted". As shipped, this causes the system to wait for the user to either press the return key (for the default */unix* kernel) or to enter the name of an alternate kernel and then press the return key.

You can override this behavior and instead cause a five-second countdown before the default kernel is loaded. At any time during those five seconds, the user can either press the return key to boot the default kernel, or any other key to be prompted to enter the name of an alternate kernel. To enable this option, perform the following steps:

```
# cd /sys/pdpstand
[edit the file boot.c and change the definition of AUTOMULTIUSER from "0" to "1" and save the file.
# make boot (ignore some harmless warning messages)
# install -s -m 744 boot /
# make clean
```

After this, a "cold boot" will display a prompt as shown below:

```
Press <CR> to boot, or any other key to abort: 5 4 3 2 1 0
```

7.3. Kernel organization

On the distribution tape, the kernel source is in a separate tar image. On "canned" (prebuilt) disk images, the kernel source has already been loaded into the disk image. When installing from tape, the source may be physically located anywhere within any filesystem so long as a symbolic link to the location is created for the file /sys (many files in /usr/include are normally symbolic links relative to /sys). In further discussions of the system source all path names will be given relative to /sys.

The 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_	kernel portion of disk quota system
subr_	misc. subroutines used throughout the kernel
sys_	system calls and the like
tty_	terminal handling
ufs_	filesystem
uipc_	interprocess communication
vm_	memory management

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/pdp	PDP-11 specific mainline code
/sys/pdpif	PDP-11 network interface code
/sys/pdpuba	PDP-11 UNIBUS, MASSBUS and QBUS device drivers

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 mostly independent of the include files in /usr/include. Unfortunately, not all references to /usr/include have been eradicated, so compiling the system requires the /usr filesystem to be mounted.

7.4. Devices and device drivers

Devices supported by UNIX are implemented in the kernel by drivers whose source is in /sys/pdp and /sys/pdpuba. These drivers are compiled into the kernel when included in a CPU-specific configuration file kept in the conf directory.

NOTE: If you are using a "canned" (prebuilt) disk image, you can safely skip the sections on /dev, /etc/dtab, and /etc/fstab below, as they have been set up correctly for the image you are using. If you add any devices to a "canned" image, you may need to make changes to those files as described below.

7.4.1. Populating the /dev tree

Devices are accessed through special files in the filesystem, made by the *mknod* (8) program and normally kept in the /dev directory. For all the devices supported by the distribution system, the files in /dev are created by the /dev/MAKEDEV shell script.

Determine the set of devices your kernel will support, then create a new directory /newdev, copy /dev/MAKEDEV into it, edit the file MAKEDEV.local if needed to provide an entry for local needs, and run /newdev/MAKEDEV to populate the /newdev directory. For instance, if your machine has a single DZ11, a single DH11, an RM03 disk, an EMULEX UNIBUS SMD disk controller with 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 xp0 xp1 ht0 std LOCAL (LOCAL only if needed)
```

Note that the "std" argument causes standard devices such as */dev/console* (the machine console), */dev/null*, */dev/tty*, */dev/klog*, etc. to be created.

You can then do

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

to install the new device directory.† As distributed almost all of the device nodes are already present and you may wish to remove unused entries from /dev to speed up scanning of the directory. The terminal nodes are almost certainly incorrect for your site and will need to be deleted and recreated. Directly connected terminals should have the softcarrier bit on in their minor device numbers. Since MAKEDEV by default creates terminal (dh, dz, etc) nodes with the softcarrier bit off you will have to delete those nodes and recreate those terminal nodes which are directly connected:

```
# cd /dev
# rm ttyh0
# mknod ttyh0 c 3 128
```

You can create the device nodes (if needed) before or after building a new kernel.

NOTE: The need to use MAKEDEV and mknod to make device nodes for most devices will likely come as a surprise to users of modern Unix-like systems, which automatically populate the /dev directory. This is because 2.11BSD would need additional initialization code to create and populate the /dev directory, taking up scarce memory for a function that is only rarely needed.

7.4.2. Updating the /etc/dtab file

In addition to the device nodes in the /dev directory, the kernel uses a file called /etc/dtab to attach device drivers to their specific controller address.

The PDP-11 has a complex system of addresses (sometimes called CSRs) and vectors that are set on each device so the rest of the system knows where to find that device. To complicate matters further, some devices use a "fixed" (unchanging) address and vector, others use "floating" addresses and vectors, and yet others use a fixed address and vector for the first device of that type and a floating address and vector for additional devices of that type. Some (relatively) newer devices allow the operating system to specify their vector; those devices do not have a vector that is configured in hardware. Additionally, if you later install a device with a floating address, it may require changing the addresses of other devices. This probably seems very bizarre to modern users accustomed to plugging an add-on card into a PC and having it "just work". Bear in mind that the first PDP-11 model was introduced in 1970.

If you are running on an emulator, the emulator likely has a command to display this information. For example, on SIMH (with entries for disabled devices not shown):

```
sim> show dev
PDP-11 simulator configuration

CPU     11/70, FPP, autoconfiguration enabled, idle enabled
SYSTEM  PDP-11 model options
RHA     address=17776700-17776753, vector=254, BR5, RH70
```

† You must reboot your system before you can remove the /olddev directory.

```
RHB     address=17772440-17772477, vector=224, BR5, RH70
CLK     60Hz, address=17777546-17777547, vector=100, BR6
PTR     address=17777550-17777553, vector=70, BR4
PTP     address=17777554-17777557, vector=74, BR4
TTI     7p, address=17777560-17777563, vector=60, BR4
TTO     7p, address=17777564-17777567, vector=64, BR4
LPT     address=17777514-17777517, vector=200, BR4
VH      address=17760440-17760457*, vector=300-304*, BR4, lines=16
RP      Massbus adapter 0 (RHA), 8 units
TU      Massbus adapter 1 (RHB), FORMATTER, TM03, 8 units
XU      address=17774510-17774517, vector=120, BR5, MAC=08:00:2B:A8:02:0B
ROM     4 units
```

If you are running on an actual PDP-11, the commands differ by model, and to make things more complicated, some models (like the PDP-11/70) are "open it up and look", at least until you have an operating system running. Newer models (you can tell these by their "chatty" console messages) normally have a command like "map" which will display the contents of the I/O address space.

The /etc/dtab file needs to be edited to match the actual hardware configuration. Here is an /etc/dtab file that matches the above configuration (most lines that are commented out are not shown):

```
# Device Configuration Table

#Name Unit# Addr Vector Br      Handler(s)      # Comments

du      ? 160440 300    5       dhurint dhuxint # dhu11 terminal mux
ht      ? 172440 224    5       htintr          # tu 16 massbus tape
lp      ? 177514 200    4       lpintr          # lp-11 line printer
xp      ? 176700 254    5       xpintr          # xp driver
```

You will need to uncomment lines for your devices and specify their address and vector. You don't need to worry about the rest of that device line. It is a good idea to comment out lines for any devices you do not have configured in your kernel. There are a few devices that appear in the SIMH device list but not in the /etc/dtab file. CLK/TTI/TTO are the system clock and console serial port - the kernel knows where these are because their addresses and vectors are the same on all PDP-11 models. PTR/PTP are the high-speed paper tape reader and punch, respectively. The hardware is there, but this kernel does not include support for them. Lastly, networking devices such as XU (SIMH)/de (2,11BSD) device are not enabled in the /etc/dtab file. They appear as commented-out devices to provide a reference for the expected addresses and vectors.

NOTE: Device naming is not consistent - the PDP-11 console or emulator will refer to them with one set of names, while 2.11BSD may use different names. This is true of other PDP-11 operating systems as well. It's just the way things were back then.

You may be wondering why the system has both an /etc/dtab file and a /dev directory, because they seem to be talking about the same type of thing. The answer is that /etc/dtab is used by the kernel to find hardware and attach the correct device drivers, while the files in /dev are used by programs to talk to the devices via the standard I/O library and kernel.

7.4.3. Updating /etc/fstab (if needed)

If you added additional disk devices to your configuration, you will need to add them to the /etc/fstab file so they are automatically mounted during system startup. **NOTE:** Refer to Section 7.1 above regarding adjusting NMOUNT (if needed). You can only mount as many filesystems as NMOUNT allows. For example, consider the following /etc/fstab:

```
/dev/xp0a       /       ufs     rw              1       1
/dev/xp0b       none    swap    sw              0       0
/dev/xp0g       /usr    ufs     rw              1       2
```

This is a bit of a "trick question". /dev/xp0b is used as a swap device, so it doesn't consume a mount point, while /dev/xp0a and /dev/xp0g do. In this case, you have four more available mount points with the default NMOUNT setting of 6.

7.5. Building a new kernel

The kernel configuration of each UNIX system is described by a single configuration file, stored in the */sys/conf* directory. The format of this file is very simple consisting of lines starting with an *identifier* followed by a *value*. Blank lines and anything past a "#" (including the #) are comments. This file is processed by the shell script *config* in the same directory. The manual pages in Section 4 of the UNIX man pages specify the configuration lines necessary for various devices.

The configuration file *GENERIC* in the conf directory was used to build the generic distribution kernel. To build a local configuration file, copy *GENERIC* to a new file *MYSYSTEM*, edit *MYSYSTEM* for your desired system configuration, and then type "./config *MYSYSTEM*". This will create the directory *../MYSYSTEM* and copy specially edited files into based on the definitions in *MYSYSTEM*. Change directory into the new system directory and type *make all*.†

Prototypes for all configurable options can be found in the generic kernel configuration file */sys/conf/GENERIC*. You can treat the items copied from GENERIC as a "grocery list", checking off those options you want, crossing out those you don't and setting numeric parameters to reasonable values. A An additional kernel configuration file, /sys/conf/EXAMPLE is also provided, showing the changes needed to enable networking support. The exact steps are shown below:

cp GENERIC *MYSYSTEM*
TERM=*terminal_type*; export TERM
vi *MYSYSTEM* (or whatever your favorite editor is)
./config *MYSYSTEM*
cd *../MYSYSTEM*
make

Note that the overlay scheme in the Makefile copied into the new system directory may fail because either the *base segment* is too small, too large or one or more *overlay segments* are too large. If this happens the system object files will have to be rearranged in the *base* and *overlay* segments. The comments in the Makefile should make it fairly clear what the restrictions on object placement are in the system.

The following is an example of the display the Makefile produces when reporting the size of the base and overlay segments:

```
size unix
text    data    bss     dec     hex
57088   5844    34182   97114   17b5a   total text: 108928
        overlays: 7808,7488,8128,7296,2240,7744,5440,5696
```

Each of the overlay segments must be kept to under 8KB in size and you cannot have a zero-size overlay between two occupied overlays. You can use the *size* command on a file (use the "text" number returned by that command) to see how much space a given object file would add to or remove from an overlay when editing the Makefile. Be careful and try to move object files between overlays, rather than moving things from the base to an overlay - many object files in the base need to remain there. **NOTE:** Each time you use the *config MYSYSTEM* command, the Makefile will be overwritten. If your kernel required editing the Makefile to rearrange the overlays, you should keep track of those changes, as you will need to re-apply them to the new Makefile.

The configured system image "*unix*"‡ should be copied to the root, and then booted to try it out. It is best to first save the old kernel file(s) to a known name so as not to break the working system until you're

† If your system does not have a floating-point processor (this only applies to the PDP-11/44 and /70) you will need to enable the floating-point emulator via the FPSIM kernel configuration option, as only the GENERIC kernel has floating point emulation installed. If you do not enable FPSIM, your new kernel will hang after printing the memory sizes. This restriction will likely be lifted in a future Patch Level.

‡ On networked systems there are two images, *unix* and *netnix*.

sure the new one does work. It is a **better** idea to also always keep the generic non network kernel (renamed to */genunix*) on the system. On "canned" (prebuilt) disk images, the */genunix* kernel has already been installed for you.

For kernels built without networking, the steps would be:

```
# cp /unix /oldunix
# make install
# sync
```

For kernels built with networking, the steps would be:

```
# cp /unix /oldunix
# cp /netnix /oldnetnix
# make install
# sync
```

IMPORTANT NOTE: */unix* and */netnix* are a "matched set" and should be treated as such. A mismatch will lead to a crash at system boot time, as will a */unix* built with networking, but missing its matched */netnix*. For these reasons, we suggest always keeping the GENERIC distribution kernel as */genunix* - it will work on most hardware and does not depend on networking. After you boot */genunix*, you can either fall back to your previous kernel (by copying the */old*nix* files back to the expected names) or perform other recovery tasks as may be needed.

NOTE: When booting from a kernel that is not named */unix* (for example, */genunix*) you will see a pair of console messages like these:

```
autoconfig: /unix is not the running version.
init: configuration setup error
```

This is just to warn you that you are in a recovery environment and is otherwise harmless.

It is advisable to reboot the system relatively soon after installing a new kernel, otherwise various utilities that read kernel memory will be using an incorrect kernel symbol table and can produce error messages or other unexpected results:

```
# shutdown -r time
```

If you are the only user on the system and have no other software running, you can use 'now' for *time*. If there are other users on the system, it is polite to give them some time to save their work and log off before rebooting.

7.6. GENERIC kernel configuration

The GENERIC kernel supports the following devices:

Device	Number
RK05	8
RK06/07	2
MSCP (RA) Controllers	2
MSCP (RA) Disks	3
RL01/02 Drives	4
SMD (XP) Controllers	1
SMD (XP) Disks	2
TE16, TU45, TU77 (HT) Tape drives	2
TM11 (TM) Tape drives	2
TS11 (TS) Tape drives	2
TK50/TU81 (TMSCP) Tape drives	2

The generic kernel adapts automatically to the booted device. The 'a' partition on the booted device is automatically made the root filesystem and the 'b' partition the swap area. The size of the swap partition

is determined at run time, the kernel queries the driver for the number of blocks in the ‘b’ partition. **NOTE:** If the swap partition is not labeled as being of type *swap* the kernel will panic.

7.6.1. GENERIC kernel configuration file

```
# 2026/9/26  - remove RK as BOOTDEV, much too small for a root filesystem
# 2026/9/11  - remove TMSCP_DEBUG (not used)
# 2026/5/15  - remove NHY
# 2025/08/11 - add NDHUL
# 2019/10/18 - TIMEZONE set to GMT
# 1997/1/21 - RK added to GENERIC kernel (for use with Bob Supnik's emulator)
# 1995/12/14 - RX added to GENERIC kernel.
# 1995/07/21 - XP_PROBE removed.
#
# Machine configuration file for 2.11BSD distributed kernel.
#
# Format:
#       name    value           comments
# An item's value may be either numerical, boolean or a string; if it's
# boolean, use "YES" or "NO" to set it or unset it, respectively.  Use
# the default value and the comments field as indicators of the type of
# field it is.

########################################
# MACHINE DEPENDENT PARAMETERS         #
########################################

# Machine type
# Split I/D and hardware floating point are required.
# UNIBUS support is always included, it only cost 1kb of text space and
# 94 bytes of D space for Qbus systems (obviously Unibus systems incur
# no penalty).

# The define Q22 has been removed.  The references to it were incorrect
# (i.e. using it to distinguish between an Emulex CS02 and a DH11) or
# inappropriate (the if_il.c driver should have been checking if a Unibus
# Map was present at runtime).

#LINEHZ         50                      # clock frequency European
LINEHZ          60                      # clock frequency USA

# PDP-11 machine type; allowable values are GENERIC, 44, 70, 73.  GENERIC
# should only be used to build a distribution kernel.  The only use of this
# option is to select the proper in-line PS instructions (references to the
# PSW use 'spl', 'mfps/mtps' or 'movb' instructions depending on the cpu type).
PDP11           GENERIC                 # distribution kernel
#PDP11          44                      # PDP-11/44
#PDP11          70                      # PDP-11/70
#PDP11          73                      # PDP-11/53,73,83,84,93,94

########################################
# GENERAL SYSTEM PARAMETERS            #
########################################

IDENT           GENERIC                 # machine name
```

```
MAXUSERS        6                       # maxusers on machine

# BOOTDEV is the letter combination denoting the autoboot device,
# or NONE if not using the autoboot feature.
BOOTDEV         NONE                    # don't autoboot
#BOOTDEV        dvhp                    # DIVA Comp/V boot device
#BOOTDEV        hk6                     # rk06 boot device
#BOOTDEV        hk7                     # rk07 boot device
#BOOTDEV        ra                      # MSCP boot device
#BOOTDEV        rl                      # rl01/02 boot device
#BOOTDEV        rm                      # rm02/03/05 rp06 boot device
#BOOTDEV        br                      # Eaton BR1537/BR1711 boot device
#BOOTDEV        sc11                    # Emulex SC11/B boot device
#BOOTDEV        sc21                    # Emulex SC21 boot device
#BOOTDEV        si                      # si 9500 boot device

# Timezone
TIMEZONE        0                       # kernel uses GMT by default

#Filesystem configuration
#
# PIPEDEV, ROOTDEV, SWAPDEV, DUMPDEV are all in terms of 'makedev(major,minor)'
# and the major number from this table.  Devices in the you (almost) certainly
# do not have category (see below) or not suitable are not listed here:
#
#  XX (Device)   major
#  ===========   =====
#  hk             4
#  ra             5
#  rl             7
#  xp            10

# The complete list of major numbers is in the script /dev/MAKEDEV
#
# and the 'minor' number from this table (partitions other than 'a' and
# 'b' rarely used but allowed).
#
#  minor         partition name
#  -----         --------------
#   0            a
#   1            b
#   2            c
#   3            d
#   4            e
#   5            f
#   6            g
#   7            h

# For example: with a SMD drive (xp) ROOTDEV xp0a is makedev(10,0),
# PIPEDEV is makedev(10,0) and SWAPDEV is makedev(10,1)

PIPEDEV         makedev(10,0)           # makedev(major,0) XX0a
ROOTDEV         makedev(10,0)           # makedev(major,0) XX0a
SWAPDEV         makedev(10,1)           # makedev(major,1) XX0b
```

```
# DUMPROUTINE indicates which dump routine should be used.  DUMPDEV
# should be in terms of makedev.  If DUMPDEV is NODEV no automatic
# dumps will be taken, and DUMPROUTINE needs to be set to "nulldev" to
# resolve the reference.

# A TMSCP tape device may be used as a dump device.  DUMPLO does not apply
# in this case and the minor number is almost always 0.
#
#  XX (Device)   major
#  ===========   =====
#  tmscp         12

# DUMPLO should leave room for the kernel to start swapping on SWAPDEV
# without overwriting the dump.

DUMPLO          512                     # dump start address
DUMPDEV         NODEV                   # makedev(major,1) XX0b
DUMPROUTINE     nulldev                 # no dump routine.
#DUMPROUTINE    hkdump                  # hk driver dump routine
#DUMPROUTINE    radump                  # ra driver dump routine
#DUMPROUTINE    rldump                  # rl driver dump routine
#DUMPROUTINE    rmdump                  # rm driver dump routine
#DUMPROUTINE    brdump                  # br driver dump routine
#DUMPROUTINE    sidump                  # si driver dump routine
#DUMPROUTINE    xpdump                  # xp driver dump routine
#DUMPROUTINE    tmsdump                 # tmscp driver dump routine

########################################
# KERNEL CONFIGURATION                 #
########################################

BADSECT         NO                      # bad-sector forwarding
EXTERNALITIMES  YES                     # map out inode time values
UCB_CLIST       NO                      # clists moved from kernel data space
QUOTA           NO                      # dynamic file system quotas
                                        # NOTE -- *very* expensive
SOFUB_MAP       YES                     # Software UNIBUS/QBUS map.  Permits
                                        # use of 18 bit controllers in a 22 bit
                                        # Qbus system.  Normally NO except for
                                        # the GENERIC kernel or if you actually
                                        # have an 18 bit RX02 on a Qbus system.

# UCB_METER is fairly expensive, but various programs (iostat, vmstat, etc)
# use it.
UCB_METER       NO                      # vmstat performance metering

# NBUF is the size of the buffer cache, and is directly related to the UNIBUS
# mapping registers.  There are 32 total mapping registers, of which 30 are
# available.  The 0'th is used for CLISTS, and the 31st is used for the I/O
# page on some PDP's.  It's suggested that you allow 7 mapping registers
# per UNIBUS character device so that you can move 56K of data on each device
# simultaneously.  The rest should be assigned to the block buffer pool.  So,
# if you have a DR-11 and a TM-11, you would leave 14 unassigned for them and
# allocate 16 to the buffer pool.  Since each mapping register addresses 8
```

```
# buffers for a 1K file system, NBUF would be 128.  A possible exception would
# be to reduce the buffers to save on data space, as they were 24 bytes each
# Should be 'small' for GENERIC, so room for kernel + large program to run.
NBUF            32                      # buffer cache, *must* be <= 240

# DIAGNOSTIC does various run-time checks, some of which are pretty
# expensive and at a high priority.  Suggested use is when the kernel
# is crashing and you don't know why, otherwise run with it off.
DIAGNOSTIC      NO                      # misc. diagnostic loops and checks

########################################
# PERIPHERALS: DISK DRIVES             #
########################################

NHK             2               # RK611, RK06/07

NRAC            2               # NRAD controllers
NRAD            3               # RX50, RC25, RD51/52/53, RA60/80/81

NRK             8               # RK05

NRL             4               # RL01/02

NRX             0               # RX02

NXPC            1               # NXPD controllers (RH70/RH11 style)
NXPD            2               # RM02/03/05, RP04/05/06, CDC 9766,
                                # Ampex 9300, Diva, Fuji 160, SI Eagle.

NRAM            0               # RAM disk size (512-byte blocks)

########################################
# PERIPHERALS: TAPE DRIVES             #
########################################

NHT             2               # TE16, TU45, TU77

# Setting AVIVTM configures the TM driver for the AVIV 800/1600/6250
# controller (the standard DEC TM only supports 800BPI).  For more details,
# see /sys/pdpuba/tm.c.
NTM             2               # TM11
AVIVTM          YES             # AVIV 800/1600/6250 controller

NTS             2               # TS11

NTMSCP          2               # TMSCP controllers
NTMS            2               # TMSCP drives

########################################
# PERIPHERALS: TERMINALS               #
########################################

# NKL includes both KL11's and DL11's.
# It should always be at least 1, for the console.
```

```
NKL             1               # KL11, DL11
NDH             0               # DH11; NDH is in units of boards (16 each)
CS02            NO              # DH units above are really Emulex CS02
                                # boards on a 22bit Qbus.
NDM             0               # DM11; NDM is in units of boards (16 each)
NDHU            0               # DHU11
NDHUL           16              # No. of lines per DHU11
NDHV            0               # DHV11
NDZ             0               # DZ11; NDZ is in units of boards (8 each)

########################################
# PERIPHERALS: OTHER                   #
########################################
NLP             0               # Line Printer
LP_MAXCOL       132             # Maximum number of columns on line printers

########################################
# PSEUDO DEVICES, PROTOCOLS, NETWORKING #
########################################
# Networking only works with split I/D and SUPERVISOR space, i.e. with the
# 11/44/53/70/73/83/84/93/94.  NETHER should be non-zero for networking
# systems using any ethernet.  CHECKSTACK makes sure the networking stack
# pointer and the kernel stack pointer don't collide; it's *very* expensive
# at 4 extra instructions for EVERY function call AND return, always left
# NO unless doing serious debugging.
INET            NO              # TCP/IP
CHECKSTACK      NO              # Kernel & Supervisor stack pointer checking
NETHER          0               # ether pseudo-device

# Note, PTY's and the select(2) system call do not require the kernel to
# be configured for networking (INET).  Note that you can allocate PTY's
# in any number (multiples of 8, of 16, even, odd, prime, whatever).  Nothing
# in the kernel cares.  PTY's cost 78 bytes apiece in kernel data space.  You
# should probably have at least 8-10 since several applications use them:
# script, jove, window, rlogin, ...
NPTY            0               # pseudo-terminals - GENERIC sys needs NONE

NDE             0               # DEUNA/DELUA
NQE             0               # DEQNA
NQT             0               # DEQTA (DELQA-YM, DELQA-PLUS)
NSL             0               # Serial Line IP

# The following were known to work in 2.11BSD at one time but are for
# hardware you (almost) certainly do not have.
NSI             0               # SI 9500 driver for CDC 9766 disks
NBR             0               # EATON BR1537/BR1711, BR1538A, B, C, D
NDN             0               # DN11 dialer
NDR             0               # DR11-W
NEC             0               # 3Com Ethernet
NIL             0               # Interlan Ethernet
NVV             0               # V2LNI (Pronet)
NACC            0               # ACC LH/DH ARPAnet IMP interface
PLI             NO              # LH/DH is connected to a PLI
NIMP            0               # ARPAnet IMP 1822 interface
```

```
# The following are untested in 2.11BSD; some are untested since before 2.9BSD
# Some won't even compile.  Most will require modification.  Good luck.
ENABLE34        NO              # if have the ENABLE34 board

NCSS            0               # DEC/CSS IMP11-A ARPAnet IMP interface
NDMC            0               # DMC11
NEN             0               # Xerox prototype (3 Mb) Ethernet
NS              0               # Xerox NS (XNS)
NSRI            0               # SRI DR11c ARPAnet IMP
NTB             0               # RS232 interface for Genisco/Hitachi tablets

# Defining FPSIM to YES compiles a floating point simulator into the kernel
# which will catch floating point instruction traps from user space.
FPSIM           NO                      # floating point simulator

# To enable profiling, the :splfix script must be changed to use spl6 instead
# of spl7 (see conf/:splfix.profile), also, you have to have a machine with a
# supervisor PAR/PDR pair, i.e. an 11/44/53/70/73/83/84/93/94, as well
# as both a KW11-L and a KW11-P.
#
# Note that profiling is not currently working.  We don't have any plans on
# fixing it, so this is essentially a non-supported feature.
PROFILE         NO              # system profiling with KW11P clock

# The Ingres driver works but is not needed unless concurrent database accesses
# will be done.
INGRES          NO              # include the Ingres lock driver
```

7.7. EXAMPLE kernel configuration file with networking

This is an example kernel configuration, showing the necessary changes to customize the GENERIC kernel for specific hardware, add networking, and so on.

7.7.1. EXAMPLE kernel configuration file with networking

```
# 2026/9/26  - remove RK as BOOTDEV, much too small for a root filesystem
# 2026/9/11  - remove TMSCP_DEBUG (not used)
# 2026/5/15  - remove NHY
# 2025/08/11 - add NDHUL
# 2019/10/18 - TIMEZONE set to GMT
# 1997/1/21 - RK added to GENERIC kernel (for use with Bob Supnik's emulator)
# 1995/12/14 - RX added to GENERIC kernel.
# 1995/07/21 - XP_PROBE removed.
#
# Machine configuration file for 2.11BSD distributed kernel.
#
# Format:
#       name    value           comments
# An item's value may be either numerical, boolean or a string; if it's
# boolean, use "YES" or "NO" to set it or unset it, respectively.  Use
# the default value and the comments field as indicators of the type of
# field it is.

########################################
# MACHINE DEPENDENT PARAMETERS         #
########################################
```

```
# Machine type
# Split I/D and hardware floating point are required.
# UNIBUS support is always included, it only cost 1kb of text space and
# 94 bytes of D space for Qbus systems (obviously Unibus systems incur
# no penalty).

# The define Q22 has been removed.  The references to it were incorrect
# (i.e. using it to distinguish between an Emulex CS02 and a DH11) or
# inappropriate (the if_il.c driver should have been checking if a Unibus
# Map was present at runtime).

#LINEHZ         50                      # clock frequency European
LINEHZ          60                      # clock frequency USA

# PDP-11 machine type; allowable values are GENERIC, 44, 70, 73.  GENERIC
# should only be used to build a distribution kernel.  The only use of this
# option is to select the proper in-line PS instructions (references to the
# PSW use 'spl', 'mfps/mtps' or 'movb' instructions depending on the cpu type).
#PDP11          GENERIC                 # distribution kernel
#PDP11          44                      # PDP-11/44
#PDP11          70                      # PDP-11/70
PDP11           73                      # PDP-11/53,73,83,84,93,94

########################################
# GENERAL SYSTEM PARAMETERS            #
########################################

IDENT           EXAMPLE                 # machine name
MAXUSERS        20                      # maxusers on machine

# BOOTDEV is the letter combination denoting the autoboot device,
# or NONE if not using the autoboot feature.
BOOTDEV         NONE                    # don't autoboot
#BOOTDEV        dvhp                    # DIVA Comp/V boot device
#BOOTDEV        hk6                     # rk06 boot device
#BOOTDEV        hk7                     # rk07 boot device
#BOOTDEV        ra                      # MSCP boot device
#BOOTDEV        rl                      # rl01/02 boot device
#BOOTDEV        rm                      # rm02/03/05 rp06 boot device
#BOOTDEV        br                      # Eaton BR1537/BR1711 boot device
#BOOTDEV        sc11                    # Emulex SC11/B boot device
#BOOTDEV        sc21                    # Emulex SC21 boot device
#BOOTDEV        si                      # si 9500 boot device

# Timezone
TIMEZONE        0                       # kernel uses GMT by default

#Filesystem configuration
#
# PIPEDEV, ROOTDEV, SWAPDEV, DUMPDEV are all in terms of 'makedev(major,minor)'
# and the major number from this table.  Devices in the you (almost) certainly
# do not have category (see below) or not suitable are not listed here:
#
#  XX (Device)   major
```

```
#  ===========   =====
#  hk             4
#  ra             5
#  rl             7
#  xp            10

# The complete list of major numbers is in the script /dev/MAKEDEV
#
# and the 'minor' number from this table (partitions other than 'a' and
# 'b' rarely used but allowed).
#
#  minor         partition name
#  -----         --------------
#   0            a
#   1            b
#   2            c
#   3            d
#   4            e
#   5            f
#   6            g
#   7            h

# For example: with a SMD drive (xp) ROOTDEV xp0a is makedev(10,0),
# PIPEDEV is makedev(10,0) and SWAPDEV is makedev(10,1)

PIPEDEV         makedev(10,0)           # makedev(major,0) XX0a
ROOTDEV         makedev(10,0)           # makedev(major,0) XX0a
SWAPDEV         makedev(10,1)           # makedev(major,1) XX0b

# DUMPROUTINE indicates which dump routine should be used.  DUMPDEV
# should be in terms of makedev.  If DUMPDEV is NODEV no automatic
# dumps will be taken, and DUMPROUTINE needs to be set to "nulldev" to
# resolve the reference.

# A TMSCP tape device may be used as a dump device.  DUMPLO does not apply
# in this case and the minor number is almost always 0.
#
#  XX (Device)   major
#  ===========   =====
#  tmscp         12

# DUMPLO should leave room for the kernel to start swapping on SWAPDEV
# without overwriting the dump.

DUMPLO          512                     # dump start address
DUMPDEV         makedev(10,1)           # makedev(major,1) XX0b
#DUMPROUTINE    nulldev                 # no dump routine.
#DUMPROUTINE    hkdump                  # hk driver dump routine
#DUMPROUTINE    radump                  # ra driver dump routine
#DUMPROUTINE    rldump                  # rl driver dump routine
#DUMPROUTINE    rmdump                  # rm driver dump routine
#DUMPROUTINE    brdump                  # br driver dump routine
#DUMPROUTINE    sidump                  # si driver dump routine
DUMPROUTINE     xpdump                  # xp driver dump routine
```

```
#DUMPROUTINE    tmsdump                 # tmscp driver dump routine

########################################
# KERNEL CONFIGURATION                 #
########################################

BADSECT         NO                      # bad-sector forwarding
EXTERNALITIMES  YES                     # map out inode time values
UCB_CLIST       YES                     # clists moved from kernel data space
QUOTA           YES                     # dynamic file system quotas
                                        # NOTE -- *very* expensive
SOFUB_MAP       NO                      # Software UNIBUS/QBUS map.  Permits
                                        # use of 18 bit controllers in a 22 bit
                                        # Qbus system.  Normally NO except for
                                        # the GENERIC kernel or if you actually
                                        # have an 18 bit RX02 on a Qbus system.

# UCB_METER is fairly expensive, but various programs (iostat, vmstat, etc)
# use it.
UCB_METER       YES                     # vmstat performance metering

# NBUF is the size of the buffer cache, and is directly related to the UNIBUS
# mapping registers.  There are 32 total mapping registers, of which 30 are
# available.  The 0'th is used for CLISTS, and the 31st is used for the I/O
# page on some PDP's.  It's suggested that you allow 7 mapping registers
# per UNIBUS character device so that you can move 56K of data on each device
# simultaneously.  The rest should be assigned to the block buffer pool.  So,
# if you have a DR-11 and a TM-11, you would leave 14 unassigned for them and
# allocate 16 to the buffer pool.  Since each mapping register addresses 8
# buffers for a 1K file system, NBUF would be 128.  A possible exception would
# be to reduce the buffers to save on data space, as they were 24 bytes each
# Should be 'small' for GENERIC, so room for kernel + large program to run.
NBUF            128                     # buffer cache, *must* be <= 240

# DIAGNOSTIC does various run-time checks, some of which are pretty
# expensive and at a high priority.  Suggested use is when the kernel
# is crashing and you don't know why, otherwise run with it off.
DIAGNOSTIC      NO                      # misc. diagnostic loops and checks

########################################
# PERIPHERALS: DISK DRIVES             #
########################################

NHK             0               # RK611, RK06/07

NRAC            0               # NRAD controllers
NRAD            0               # RX50, RC25, RD51/52/53, RA60/80/81

NRK             0               # RK05

NRL             2               # RL01/02

NRX             0               # RX02
```

```
NXPC            1               # NXPD controllers (RH70/RH11 style)
NXPD            4               # RM02/03/05, RP04/05/06, CDC 9766,
                                # Ampex 9300, Diva, Fuji 160, SI Eagle.

NRAM            0               # RAM disk size (512-byte blocks)

########################################
# PERIPHERALS: TAPE DRIVES             #
########################################

NHT             0               # TE16, TU45, TU77

# Setting AVIVTM configures the TM driver for the AVIV 800/1600/6250
# controller (the standard DEC TM only supports 800BPI).  For more details,
# see /sys/pdpuba/tm.c.
NTM             0               # TM11
AVIVTM          NO              # AVIV 800/1600/6250 controller

NTS             0               # TS11

NTMSCP          1               # TMSCP controllers
NTMS            2               # TMSCP drives

########################################
# PERIPHERALS: TERMINALS               #
########################################

# NKL includes both KL11's and DL11's.
# It should always be at least 1, for the console.
NKL             2               # KL11, DL11
NDH             0               # DH11; NDH is in units of boards (16 each)
CS02            NO              # DH units above are really Emulex CS02
                                # boards on a 22bit Qbus.
NDM             0               # DM11; NDM is in units of boards (16 each)
NDHU            0               # DHU11
NDHUL           16              # No. of lines per DHU11
NDHV            0               # DHV11
NDZ             0               # DZ11; NDZ is in units of boards (8 each)

########################################
# PERIPHERALS: OTHER                   #
########################################
NLP             0               # Line Printer
LP_MAXCOL       132             # Maximum number of columns on line printers

########################################
# PSEUDO DEVICES, PROTOCOLS, NETWORKING #
########################################
# Networking only works with split I/D and SUPERVISOR space, i.e. with the
# 11/44/53/70/73/83/84/93/94.  NETHER should be non-zero for networking
# systems using any ethernet.  CHECKSTACK makes sure the networking stack
# pointer and the kernel stack pointer don't collide; it's *very* expensive
# at 4 extra instructions for EVERY function call AND return, always left
# NO unless doing serious debugging.
```

```
INET            YES             # TCP/IP
CHECKSTACK      NO              # Kernel & Supervisor stack pointer checking
NETHER          1               # ether pseudo-device

# Note, PTY's and the select(2) system call do not require the kernel to
# be configured for networking (INET).  Note that you can allocate PTY's
# in any number (multiples of 8, of 16, even, odd, prime, whatever).  Nothing
# in the kernel cares.  PTY's cost 78 bytes apiece in kernel data space.  You
# should probably have at least 8-10 since several applications use them:
# script, jove, window, rlogin, ...
NPTY            12              # pseudo-terminals - GENERIC sys needs NONE

NDE             0               # DEUNA/DELUA
NQE             1               # DEQNA
NQT             0               # DEQTA (DELQA-YM, DELQA-PLUS)
NSL             0               # Serial Line IP

# The following were known to work in 2.11BSD at one time but are for
# hardware you (almost) certainly do not have.
NSI             0               # SI 9500 driver for CDC 9766 disks
NBR             0               # EATON BR1537/BR1711, BR1538A, B, C, D
NDN             0               # DN11 dialer
NDR             0               # DR11-W
NEC             0               # 3Com Ethernet
NIL             0               # Interlan Ethernet
NVV             0               # V2LNI (Pronet)
NACC            0               # ACC LH/DH ARPAnet IMP interface
PLI             NO              # LH/DH is connected to a PLI
NIMP            0               # ARPAnet IMP 1822 interface

# The following are untested in 2.11BSD; some are untested since before 2.9BSD
# Some won't even compile.  Most will require modification.  Good luck.
ENABLE34        NO              # if have the ENABLE34 board

NCSS            0               # DEC/CSS IMP11-A ARPAnet IMP interface
NDMC            0               # DMC11
NEN             0               # Xerox prototype (3 Mb) Ethernet
NS              0               # Xerox NS (XNS)
NSRI            0               # SRI DR11c ARPAnet IMP
NTB             0               # RS232 interface for Genisco/Hitachi tablets

# Defining FPSIM to YES compiles a floating point simulator into the kernel
# which will catch floating point instruction traps from user space.
FPSIM           NO                      # floating point simulator

# To enable profiling, the :splfix script must be changed to use spl6 instead
# of spl7 (see conf/:splfix.profile), also, you have to have a machine with a
# supervisor PAR/PDR pair, i.e. an 11/44/53/70/73/83/84/93/94, as well
# as both a KW11-L and a KW11-P.
#
# Note that profiling is not currently working.  We don't have any plans on
# fixing it, so this is essentially a non-supported feature.
PROFILE         NO              # system profiling with KW11P clock
```

```
# The Ingres driver works but is not needed unless concurrent database accesses
# will be done.
INGRES          NO              # include the Ingres lock driver
```

APPENDIX B – STANDALONE DISKLABEL PROGRAM

8.1. Standalone disklabel program

The standalone disklabel program is the second file on a boot tape (after the boot blocks and boot program). It is used to place an initial label on a disk describing the disk and its partitions. The program is also used when the root ('a') or swap ('b') partitions of a previously labeled system disk must be modified. The second use is mandated because the root and swap partitions cannot be modified while the kernel has them open.

disklabel effectively runs in CBREAK mode – you do not need to hit the RETURN key except when prompted for a multicharacter response such as a string (the pack label) or a number (partition size). Defaults are placed inside square brackets ([default]). Entering RETURN accepts the default.

The program is organized into several levels. *disklabel* prints the current level out before prompting. At each level there is always the choice of d(isplaying) the current label and q(uit)ing the current level and returning to the previous level. If you are at the top level and enter **q** the program will exit back to **Boot** unless you have made changes to the disklabel. In that case you will be asked if you wish to discard the changes, if you answer **y** the changes will be discarded. If the answer is **n** the **q** is ignored and *disklabel* does not exit.

In the following paragraphs the convention is to **bold** the user input while leaving the output from *disklabel* in normal type. The devices used were a TK50 and an RD54, thus the tape device is **tms** and the disk device is **ra**.

The TK50 was booted resulting in the usual message from **Boot**:

73Boot from tms(0,0,0) at 0174500
: **tms(0,1)**

8.2. Disklabel – tour of the levels

The following is an annotated example of the options in *disklabel*:

Boot: bootdev=06001 bootcsr=0174500
disklabel
Disk? **ra(0,0)**
d(isplay) D(efault) m(odify) w(write) q(uit)? **m**

> The 'D' option will request *disklabel* to create a default label based on what the program can determine about the drive. For some devices, such as RL01/02, RK06/07, MSCP (RD54, RA81, etc.), *disklabel* can determine what the drive type is and how many sectors it has. For other devices, such as SMD drives supported by the **xp** driver, the task is complicated by the number of different controllers and emulations supported. Some 3rd party controllers have capabilities that DEC controllers do not and the **xp** has no way of knowing exactly which type of controller is present. In this case *disklabel* will **guess** and then depend on you to enter the correct data.

modify
d(isplay) g(eometry) m(isc) p(artitions) q(uit)? **g**
modify geometry
d(isplay) s(ector/trk) t(rk/cyl) c(yl) S(ector/cyl) q(uit)? **q**

> The Sector/cyl entry is rarely used. *disklabel* will calculate this quantity for you from the sector/trk and trk/cyl quantities.

modify
d(isplay) g(eometry) m(isc) p(artitions) q(uit)? **m**
d(isplay) t(ype) n(ame) l(abel) f(lags) r(pm) D(rivedata) q(quit)? **f**

Type is one of: SMD, MSCP, old DEC, SCSI, ESDI, ST506, floppy.

Name is a string up to 16 characters in length. It is typically something like **rd54** or **rm03** but may be any meaningful string.

Label is an arbitrary string up to 16 characters in length – nothing in the system checks for or depends on the contents of the pack label string.

RPM is the rotational speed of the drive. Nothing in the system uses or depends on this at the present time. Default is 3600.

Drivedata consists of 5 longwords of arbitrary data. Reserved for future use.

modify misc flags
d(isplay) c(lear) e(cc) b(adsect) r(emovable) q(uit)? **q**

ECC says that the controller/driver can correct errors.

Badsect indicates that the controller/driver supports bad sector replacement.

Removable indicates that the drive uses removable media (floppy, RL02, RA60 for example).

modify misc
d(isplay) t(ype) n(ame) l(abel) f(lags) r(pm) D(rivedata) q(uit)? **q**
modify
d(isplay) g(eometry) m(isc) p(artitions) q(uit)? **p**
modify partitions
d(isplay) n(umber) s(elect) q(uit)? **n**
Number of partitions (8 max) [7]? **7**

This is the highest partition number considered to be valid. *disklabel* will adjust this parameter semi-automatically at the **p** level but it may be necessary to use **n** in cases where some partitions are not to be used or contain invalid information.

modify partitions
d(isplay) n(umber) s(elect) q(uit)? **s**
a b c d e f g h q(uit)? **a**
sizes and offsets may be given as sectors, cylinders
or cylinders plus sectors: 6200, 32c, 19c10 respectively
modify partition 'a'
d(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size) q(uit)? **q**

Zero clears the size and offset fields of a partition entry and sets the filesystem type to **unused**.

Type is the filesystem type and of the possible choices only **2.11BSD**, **swap** and **unused** make any sense to specify.

Offset is the number of sectors from the beginning of the disk at which the partition starts.

Size is the number of sectors which the partition occupies.

Frag is the number of fragments a filesystem block can be broken into. It is not presently used and should be left at the default of 1.

Fsize is the filesystem block size and should be left at the default of 1024.

modify partitions
d(isplay) n(umber) s(elect) q(uit)? **q**
modify
d(isplay) g(eometry) m(isc) p(artitions) q(uit)? **q**
d(isplay) D(efault) m(odify) w(write) q(uit)? **q**
Label changed. Discard changes [y/n]? **y**

73Boot from tms(0,0,1) at 0174500

8.3. Disklabel – helpful hints and tips.

Define only those partitions you actually will use. There is no need to set up all 8 partitions. Drives less than 200MB probably will only have 3 partitions defined, 'a', 'b' and 'd' for /, swap and /usr respectively. Remember to set the number of partitions. Disklabel will attempt to do this for you by keeping track of the highest partition you modify but this is not foolproof.

For each of the drives supported in */etc/disktab*, the following section shows the partitions used, their size, offset, and type. You do not need to use these definitions, but they have been proven to work. So, it is suggested that you perform at least your initial installation using the values found in the following section.

NOTE: It is a good idea to use cylinder numbers for the size and offset parameters, for example a 'g' partition starting at cylinder 45 (offset) and a size of 500 cylinders. This will make the math easier to deal with, as otherwise you are dealing with sector numbers, which can get quite large. Another benefit is that both the standalone and system *disklabel* utilities will display an asterisk (*) if you define a partition which does not start or end on a cylinder boundary.

Do not define overlapping partitions unless you are sure what you are doing. *disklabel* will warn you of overlapping partitions but will not prohibit you from writing such a label to disk.

The root 'a' partition should be large enough for the kernel, essential programs, /var (for mail, logfiles, etc) and temporary files used by editors, compilers, etc. *fsck(1)* **must** be able to check the root filesystem without use of a temporary file. A size of 8 to 10MB is usually sufficient, but do not exceed 16MB!†

The swap 'b' partition must be at least 0.5MB larger than the size of main memory so that in the event of a system crash the contents of main memory can be written to the 'b' partition. Since most emulated systems will be running with a full 4MB of memory configured, this works out to a recommended size of 4.5 to 5MB for the 'b' partition. The extra space is so that the system can be booted and begin running without overwriting the crash dump.

Remember that the prompt levels nest in *disklabel*. It will be necessary in several cases to enter multiple **q** commands to get back to the top level.

IMPORTANT: Keep at least 1, preferably more, bootable tape or floppy with *disklabel* on it present at all times. If the label on a disk ever becomes corrupted the kernel will be very unhappy and probably won't boot. If this happens you will need to boot the standalone *disklabel* program and relabel the disk. At least 2.11BSD provides a standalone *disklabel* – previous 4BSD systems which implemented disk labels did not and the cold-start of those systems was painful indeed.

IMPORTANT: Write down in at least one place, and keep with the tape/floppy mentioned above, the geometry and partition layout you assign to the disk. The *disklabel* (8) program should be used to produce a hardcopy of the disklabel.

† The reason for the size limitation is that if *fsck* does not have enough memory available to perform the filesystem check, it will need the use of a temporary file. Since only the root partition is available at this point and *fsck* can't place the temporary file on the filesystem it is checking, it becomes impossible to perform a filesystem check.

8.4. Standalone disklabel example

This is a real example of using the disklabel program to place a label on a disk. User input is in **bold** type. The disklabel program was loaded from a bootable TK50. The disk being labeled in a RD54. The BOOT> prompt is from the 11/73 console ODT, if you are using an 11/44 the prompt will be >>>.

The first thing that is done is request disklabel to create a default partition ('a') which spans the entire disk. Some disk types have fixed sizes and geometries, for example RK05 (rk), RK06/7 (hk) and RL02 (rl) drives. With this type of disk, the standalone disklabel program will generate a label with the correct geometry and 'a' partition size. With MSCP ('ra') disks disklabel will query the controller for the information it needs. The last type of disk, SMD (xp), presents many problems, disklabel will attempt to determine the drive type and geometry but you will have to verify the information.

> Indented paragraphs like this one are explanatory comments and are not part of the output from the disklabel program. In the case of MSCP drives the number of cylinders may be 1 too low. This is discussed in the example below.

```
BOOT> MU0

73Boot from tms(0,0,0) at 0174500
: tms(0,1)
Boot: bootdev=06001 bootcsr=0174500
disklabel
Disk? ra(0,0)
d(isplay) D(efault) m(odify) w(rite) q(uit)? D
d(isplay) D(efault) m(odify) w(rite) q(uit)? d

type: MSCP
disk: RD54
flags:
bytes/sector: 512
sectors/track: 17
tracks/cylinder: 15
sectors/cylinder: 255
cylinders: 1220
rpm: 3600
drivedata: 0 0 0 0 0

1 partitions:
#     size   offset   fstype   [fsize bsize]
  a: 311200 0 2.11BSD 1024 1024  # (Cyl. 0 - 1220*)
```

> The columns do not line up nicely under the headings due to limitations of the sprintf() routine in the standalone I/O package. There is no capability to justify the output. It should be obvious which column belongs under which heading. The '*' says that the partition does not end on a cylinder boundary. This is due to the peculiar way in which MSCP returns the geometry information: sectors/track * tracks/cylinder * cylinders != sectors per volume.

```
d(isplay) D(efault) m(odify) w(rite) q(uit)? m
modify
d(isplay) g(eometry) m(isc) p(artitions) q(uit)? m
```

> It is normally not necessary to change the geometry of an MSCP disk. On the other hand, it will almost always be necessary to specify the geometry of an SMD drive (one which uses the XP driver). Since the drive being labeled is an MSCP drive the next step is to set the pack label to something other than DEFAULT.

```
modify misc
d(isplay) t(ype) n(ame) l(able) f(lags) r(pm) D(rivedata) q(uit)? l
label [DEFAULT]: TESTING
modify misc
d(isplay) t(ype) n(ame) l(able) f(lags) r(pm) D(rivedata) q(uit)? q
modify
d(isplay) g(eometry) m(isc) p(artitions) q(uit)? p
modify partitions
d(isplay) n(umber) s(elect) q(uit)? d

type: MSCP
disk: RD54
flags:
bytes/sector: 512
sectors/track: 17
tracks/cylinder: 15
sectors/cylinder: 255
cylinders: 1220
rpm: 3600
drivedata: 0 0 0 0 0

1 partitions:
#     size   offset   fstype   [fsize bsize]
  a: 311200 0 2.11BSD 1024 1024  # (Cyl. 0 - 1220*)

modify partitions
d(isplay) n(umber) s(elect) q(uit)? s
a b c d e f g h q(uit)? a
sizes and offsets may be given as sectors, cylinders
or cylinders plus sectors: 6200, 32c, 19c10s respectively
modify partition 'a'
d(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size) q(uit)? s
´a' size [311200]: 15884
d(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size) q(uit)? q
modify partitions
d(isplay) n(umber) s(elect) q(uit)? s
a b c d e f g h q(uit)? b
sizes and offsets may be given as sectors, cylinders
or cylinders plus sectors: 6200, 32c, 19c10s respectively
modify partition 'b'
d(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size) q(uit)? o
´b' offset [0]: 15884
modify partition 'b'
d(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size) q(uit)? s
´b' size [0]: 16720
modify partition 'b'
d(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size) q(uit)? t
´b' fstype [unused]: swap
modify partition 'b'
d(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size) q(uit)? q
modify partitions
d(isplay) n(umber) s(elect) q(uit)? s
a b c d e f g h q(uit)? c
sizes and offsets may be given as sectors, cylinders
```

```
or cylinders plus sectors: 6200, 32c, 19c10s respectively
modify partition 'c'
d(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size) q(uit)? o
´c' offset [0]: 0
modify partitions 'c'
d(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size) q(uit)? s
´c' size [0]: 311200
modify partitions 'c'
d(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size) q(uit)? t
´c' fstype [unused]: unused
modify partitions 'c'
d(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size) q(uit)? q
modify partitions
d(isplay) n(umber) s(elect) q(uit)? s
a b c d e f g h q(uit)? g
sizes and offsets may be given as sectors, cylinders
or cylinders plus sectors: 6200, 32c, 19c10s respectively
modify partition 'g'
d(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size) q(uit)? o
´g' offset [0]: 32604
modify partition 'g'
d(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size) q(uit)? s
´g' size [0]: 278596
modify partition 'g'
d(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size) q(uit)? t
´g' fstype [unused]: 2.11BSD
modify partition 'g'
d(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size) q(uit)? q
modify partitions
d(isplay) n(umber) s(elect) q(uit)? n
Number of partitions (8 max) [7]? 7
modify partitions
d(isplay) n(umber) s(elect) q(uit)? q
modify
d(isplay) g(eometry) m(isc) p(artitions) q(uit)? d

type: MSCP
disk: RD54
label: TESTING
flags:
bytes/sector: 512
sectors/track: 17
tracks/cylinder: 15
sectors/cylinder: 255
cylinders: 1220
rpm: 3600
drivedata: 0 0 0 0 0

7 partitions:
#     size   offset   fstype   [fsize bsize]
  a: 15884   0  2.11BSD 1024 1024  # (Cyl. 0 - 62*)
  b: 16720 15884 swap              # (Cyl. 62*- 127*)
  c: 311200  0   unused 1024 1024  # (Cyl. 0 - 1220*)
  g: 278596 32604 2.11BSD 1024 1024 # (Cyl. 127- 1220*)
```

modify
d(isplay) g(eometry) m(isc) p(artitions) q(uit)? **q**

On MSCP disks it is possible you will see a warning error like this:

partition c: extends past end of unit 0 311200 311100
partition g: extends past end of unit 32604 278596 311100

This is not cause for panic. What this is saying is that the number of cylinders is one too low. MSCP devices do not necessarily use all of the last cylinder. The total number of blocks is precisely known for MSCP devices (it is returned in the act of bringing the drive online). However, the number of sectors on the volume is not necessarily evenly divisible by the number of sectors per track (311200 divided by 17*15 gives 1220.392). Basically, the last cylinder is not fully used. What must be done is raise the number of cylinders by 1.

NOTE: For any other disk type, it is cause for concern if the warning above is issued – it means that incorrect partition or geometry information was entered by the user and needs to be corrected.

d(isplay) D(efault) m(odify) w(rite) q(uit)? **m**
modify
d(isplay) g(eometry) m(isc) p(artitions) q(uit)? **g**
modify geometry
d(isplay) s(ector/trk) t(rk/cyl) c(yl) S(ector/cyl) q(uit)? **c**
cylinders [1220]: **1221**
modify geometry
d(isplay) s(ector/trk) t(rk/cyl) c(yl) S(ector/cyl) q(uit)? **q**
modify
d(isplay) g(eometry) m(isc) p(artitions) q(uit)? **q**
d(isplay) D(efault) m(odify) w(rite) q(uit)? **w**
d(isplay) D(efault) m(odify) w(rite) q(uit)? **q**

73Boot from tms(0,0,1) at 0174500
: **ra(1,0,0)unix**
ra1 csr[00]: **0172154**

The last string entered shows how I boot from an alternate controller. In normal use, i.e. with a single MSCP controller, the string would simply be **ra(0,0)unix**.

8.5. Data extracted from the */etc/disktab* file

The following list of disk models shows the values from the */etc/disktab* file which are relevant to the standalone *disklabel* program.

```
rm02|RM02|rm03|RM03|DEC RM02/03
     Partition a: size=14080, offset=0, type=2.11BSD
     Partition b: size=8960, offset=14080, type=swap
     Partition c: size=131680, offset=0
     Partition g: size=108480, offset=23040, type=2.11BSD
     Partition h: size=131520, offset=0

rm05|RM05|cdc9766|CDC9766
     Partition a: size=14592, offset=0, type=2.11BSD
     Partition b: size=9728, offset=14592, type=swap
     Partition c: size=500384, offset=0
```

```
     Partition g: size=475456, offset=24320, type=2.11BSD
     Partition h: size=499776, offset=0

rp04|RP04|rp05|RP05|DEC RP04/05
     Partition a: size=15884, offset=0, type=2.11BSD
     Partition b: size=9196, offset=15884, type=swap
     Partition c: size=171798, offset=0
     Partition g: size=145464, offset=25080, type=2.11BSD
     Partition h: size=171380, offset=0

rp06|RP06|DEC RP06
     Partition a: size=16720, offset=0, type=2.11BSD
     Partition b: size=10032, offset=16720, type=swap
     Partition c: size=340670, offset=0
     Partition g: size=313500, offset=26752, type=2.11BSD
     Partition h: size=340252, offset=0

rp07|RP07|DEC RP07
     Partition a: size=14400, offset=0, type=2.11BSD
     Partition b: size=9600, offset=14400, type=swap
     Partition c: size=1008000, offset=0
     Partition f: size=491200, offset=24000, type=2.11BSD
     Partition g: size=491200, offset=515200, type=2.11BSD
     Partition h: size=1006400, offset=0

rm2x|fuji160|Fujitsu 160
     Partition a: size=12800, offset=0, type=2.11BSD
     Partition b: size=9600, offset=12800, type=swap
     Partition c: size=263360, offset=0
     Partition g: size=240640, offset=22400, type=2.11BSD
     Partition h: size=263360, offset=0

diva|DIVA|ampex9300|Diva Comp V, Ampex 9300 (in direct mode)
     Partition a: size=12540, offset=0, type=2.11BSD
     Partition b: size=10032, offset=12540, type=swap
     Partition c: size=511005, offset=0
     Partition g: size=487806, offset=22572, type=2.11BSD
     Partition h: size=510378, offset=0

si_eagle|SI_Eagle|2351|2351A|Fujitsu Eagle 2351A (48 sectors)
     Partition a: size=12480, offset=0, type=2.11BSD
     Partition b: size=11520, offset=12480, type=swap
     Partition c: size=808320, offset=0
     Partition g: size=779520, offset=24000, type=2.11BSD
     Partition h: size=803520, offset=0

rk06|RK06|DEC RK06
     Partition a: size=8316, offset=0, type=2.11BSD
     Partition b: size=8316, offset=8316, type=swap
     Partition c: size=27126, offset=0, type=unused
     Partition g: size=10428, offset=16632, type=2.11BSD
     Partition h: size=27060, offset=0

rk07|RK07|DEC RK07
```

```
     Partition a: size=8316, offset=0, type=2.11BSD
     Partition b: size=8316, offset=8316, type=swap
     Partition c: size=53790, offset=0, type=unused
     Partition g: size=37092, offset=16632, type=2.11BSD
     Partition h: size=53724, offset=0

rl01|RL01|DEC RL01
     Partition a: size=10240, offset=0, type=2.11BSD

rl02|RL02|DEC RL02
     Partition a: size=15360, offset=0, type=2.11BSD
     Partition b: size=5120, offset=15360, type=swap
     Partition c: size=20480, offset=0

rk05|RK05|DEC RK05
     Partition a: size=4872, offset=0, type=2.11BSD

rc25|RC25|rcf25|RCF25|DEC RC25 and RC25F
     Partition a: size=15884, offset=0, type=2.11BSD
     Partition b: size=8360, offset=15884, type=swap
     Partition c: size=50902, offset=0, type=2.11BSD
     Partition g: size=26658, offset=24244, type=2.11BSD

rd31|RD31|DEC RD31
     Partition a: size=15884, offset=0, type=2.11BSD
     Partition b: size=8360, offset=15884, type=swap
     Partition c: size=41560, offset=0, type=2.11BSD
     Partition g: size=17316, offset=24244, type=2.11BSD

rd32|RD32|DEC RD32
     Partition a: size=15884, offset=0, type=2.11BSD
     Partition b: size=8360, offset=15884, type=swap
     Partition c: size=83236, offset=0, type=2.11BSD
     Partition g: size=58992, offset=24244, type=2.11BSD

rd33|RD33|DEC RD33
     Partition a: size=15884, offset=0, type=2.11BSD
     Partition b: size=8360, offset=15884, type=swap
     Partition c: size=138565, offset=0, type=2.11BSD
     Partition g: size=114321, offset=24244, type=2.11BSD

rd51|RD51|DEC RD51
     Partition a: size=15884, offset=0, type=2.11BSD
     Partition b: size=5716, offset=15884, type=swap
     Partition c: size=21600, offset=0, type=2.11BSD

rd52|RD52|DEC RD52
     Partition a: size=15884, offset=0, type=2.11BSD
     Partition b: size=8360, offset=15884, type=swap
     Partition c: size=60480, offset=0, type=2.11BSD
     Partition g: size=36236, offset=24244, type=2.11BSD

rd53|RD53|DEC RD53
     Partition a: size=15884, offset=0, type=2.11BSD
```

```
     Partition b: size=8360, offset=15884, type=swap
     Partition c: size=138672, offset=0, type=2.11BSD
     Partition g: size=114428, offset=24244, type=2.11BSD
rd54|RD54|DEC RD54
     Partition a: size=15884, offset=0, type=2.11BSD
     Partition b: size=16720, offset=15884, type=swap
     Partition c: size=311200, offset=0, type=unused
     Partition g: size=278596, offset=32604, type=2.11BSD

rx23|RX23|DEC RX23
     Partition a: size=2880, offset=0, type=2.11BSD

rx26|RX26|DEC RX26
     Partition a: size=5760, offset=0, type=2.11BSD

rx33|RX33|DEC RX33
     Partition a: size=2400, offset=0, type=2.11BSD

rx50|RX50|DEC RX50
     Partition a: size=800, offset=0:2.11BSD

ra60|RA60|DEC RA60
     Partition a: size=15884, offset=0, type=2.11BSD
     Partition b: size=16720, offset=15884, type=swap
     Partition c: size=400176, offset=0
     Partition d: size=227136, offset=32604, type=2.11BSD
     Partition e: size=140436, offset=259740, type=2.11BSD
     Partition g: size=367572, offset=32604, type=2.11BSD

ra70|RA70|DEC RA70
     Partition a: size=15884, offset=0, type=2.11BSD
     Partition b: size=16720, offset=15884, type=swap
     Partition c: size=547041, offset=0
     Partition d: size=227136, offset=32604, type=2.11BSD
     Partition e: size=287301, offset=259740, type=2.11BSD
     Partition g: size=514437, offset=32604, type=2.11BSD

ra71|RA71|DEC RA71
     Partition a: size=15884, offset=0, type=2.11BSD
     Partition b: size=16720, offset=15884, type=swap
     Partition c: size=1367310, offset=0
     Partition d: size=227136, offset=32604, type=2.11BSD
     Partition e: size=409600, offset=259740, type=2.11BSD
     Partition f: size=182464, offset=669340, type=2.11BSD
     Partition g: size=819200, offset=32604, type=2.11BSD
     Partition h: size=515506, offset=851804, type=2.11BSD

ra72|RA72|DEC RA72
     Partition a: size=15884, offset=0, type=2.11BSD
     Partition b: size=16720, offset=15884, type=swap
     Partition c: size=1953300, offset=0
     Partition d: size=227136, offset=32604, type=2.11BSD
     Partition e: size=409600, offset=259740, type=2.11BSD
     Partition f: size=182464, offset=669340, type=2.11BSD
```

```
     Partition g: size=819200, offset=32604, type=2.11BSD
     Partition h: size=1101496, offset=851804, type=2.11BSD

ra73|RA73|DEC RA73
     Partition a: size=15884, offset=0, type=2.11BSD
     Partition b: size=16720, offset=15884, type=swap
     Partition c: size=3920490, offset=0
     Partition d: size=227136, offset=32604, type=2.11BSD
     Partition e: size=409600, offset=259740, type=2.11BSD
     Partition f: size=182464, offset=669340, type=2.11BSD
     Partition g: size=819200, offset=32604, type=2.11BSD
     Partition h: size=3068686, offset=851804, type=2.11BSD

ra80|RA80|DEC RA80
     Partition a: size=15884, offset=0, type=2.11BSD
     Partition b: size=16720, offset=15884, type=swap
     Partition c: size=237212, offset=0
     Partition g: size=204608, offset=32604, type=2.11BSD

ra81|RA81|DEC RA81
     Partition a: size=15884, offset=0, type=2.11BSD
     Partition b: size=16720, offset=15884, type=swap
     Partition c: size=891072, offset=0
     Partition d: size=227136, offset=32604, type=2.11BSD
     Partition e: size=409600, offset=259740, type=2.11BSD
     Partition f: size=182464, offset=669340, type=2.11BSD
     Partition g: size=819200, offset=32604, type=2.11BSD
     Partition h: size=39268, offset=851804, type=2.11BSD

ra81x|RA81X|Extended DEC RA81
     Partition a: size=15884, offset=0, type=2.11BSD
     Partition b: size=33440, offset=15884, type=swap
     Partition c: size=193282, offset=49324, type=2.11BSD
     Partition d: size=15884, offset=242606, type=2.11BSD
     Partition e: size=307200, offset=258490, type=2.11BSD
     Partition f: size=728952, offset=565690, type=2.11BSD
     Partition g: size=1052036, offset=242606
     Partition h: size=1294642, offset=0, type=2.11BSD

ra82|RA82|DEC RA82
     Partition a: size=15884, offset=0, type=2.11BSD
     Partition b: size=16720, offset=15884, type=swap
     Partition c: size=1216665, offset=0
     Partition d: size=227136, offset=32604, type=2.11BSD
     Partition e: size=409600, offset=259740, type=2.11BSD
     Partition f: size=182464, offset=669340, type=2.11BSD
     Partition g: size=819200, offset=32604, type=2.11BSD
     Partition h: size=364861, offset=851804, type=2.11BSD

ra90|RA90|DEC RA90
     Partition a: size=15884, offset=0, type=2.11BSD
     Partition b: size=16720, offset=15884, type=swap
     Partition c: size=2376153, offset=0
     Partition d: size=227136, offset=32604, type=2.11BSD
```

```
     Partition e: size=409600, offset=259740, type=2.11BSD
     Partition f: size=182464, offset=669340, type=2.11BSD
     Partition g: size=819200, offset=32604, type=2.11BSD
     Partition h: size=1524349, offset=851804, type=2.11BSD

ra92|RA92|DEC RA92
     Partition a: size=15884, offset=0, type=2.11BSD
     Partition b: size=16720, offset=15884, type=swap
     Partition c: size=2940951, offset=0
     Partition d: size=227136, offset=32604, type=2.11BSD
     Partition e: size=409600, offset=259740, type=2.11BSD
     Partition f: size=182464, offset=669340, type=2.11BSD
     Partition g: size=819200, offset=32604, type=2.11BSD
     Partition h: size=2089147, offset=851804, type=2.11BSD

rz22|RZ22|DEC RZ22
     Partition a: size=15884, offset=0, type=2.11BSD
     Partition b: size=8360, offset=15884, type=swap
     Partition c: size=102432, offset=0
     Partition g: size=78188, offset=24244, type=2.11BSD

rz23|RZ23|DEC RZ23
     Partition a: size=15884, offset=0, type=2.11BSD
     Partition b: size=8360, offset=15884, type=swap
     Partition c: size=204864, offset=0
     Partition g: size=180620, offset=24244, type=2.11BSD

rz23l|RZ23L|DEC RZ23L
     Partition a: size=15884, offset=0, type=2.11BSD
     Partition b: size=8360, offset=15884, type=swap
     Partition c: size=237588, offset=0
     Partition g: size=213344, offset=24244, type=2.11BSD

rz24|RZ24|DEC RZ24
     Partition a: size=15884, offset=0, type=2.11BSD
     Partition b: size=16720, offset=15884, type=swap
     Partition c: size=409792, offset=0
     Partition d: size=227136, offset=32604, type=2.11BSD
     Partition e: size=150052, offset=259740, type=2.11BSD
     Partition g: size=377188, offset=32604, type=2.11BSD

rz24l|RZ24L|DEC RZ24L
     Partition a: size=15884, offset=0, type=2.11BSD
     Partition b: size=16720, offset=15884, type=swap
     Partition c: size=479350, offset=0
     Partition d: size=227136, offset=32604, type=2.11BSD
     Partition e: size=219610, offset=259740, type=2.11BSD
     Partition g: size=446746, offset=32604, type=2.11BSD

rz25|RZ25|DEC RZ25
     Partition a: size=15884, offset=0, type=2.11BSD
     Partition b: size=16720, offset=15884, type=swap
     Partition c: size=832527, offset=0
     Partition d: size=227136, offset=32604, type=2.11BSD
```

```
     Partition e: size=409600, offset=259740, type=2.11BSD
     Partition f: size=130590, offset=669340, type=2.11BSD
     Partition g: size=799923, offset=32604, type=2.11BSD

rz26|RZ26|DEC RZ26
     Partition a: size=15884, offset=0, type=2.11BSD
     Partition b: size=16720, offset=15884, type=swap
     Partition c: size=2050860, offset=0
     Partition d: size=227136, offset=32604, type=2.11BSD
     Partition e: size=409600, offset=259740, type=2.11BSD
     Partition f: size=182464, offset=669340, type=2.11BSD
     Partition g: size=819200, offset=32604, type=2.11BSD
     Partition h: size=1199056, offset=851804, type=2.11BSD

rz55|RZ55|DEC RZ55
     Partition a: size=15884, offset=0, type=2.11BSD
     Partition b: size=16720, offset=15884, type=swap
     Partition c: size=649040, offset=0
     Partition d: size=227136, offset=32604, type=2.11BSD
     Partition e: size=389300, offset=259740, type=2.11BSD
     Partition g: size=616436, offset=32604, type=2.11BSD

rz56|RZ56|DEC RZ56
     Partition a: size=15884, offset=0, type=2.11BSD
     Partition b: size=16720, offset=15884, type=swap
     Partition c: size=1299174, offset=0
     Partition d: size=227136, offset=32604, type=2.11BSD
     Partition e: size=409600, offset=259740, type=2.11BSD
     Partition f: size=182464, offset=669340, type=2.11BSD
     Partition g: size=819200, offset=32604, type=2.11BSD
     Partition h: size=447370, offset=851804, type=2.11BSD

rz57|RZ57|DEC RZ57
     Partition a: size=15884, offset=0, type=2.11BSD
     Partition b: size=16720, offset=15884, type=swap
     Partition c: size=1954050, offset=0
     Partition d: size=227136, offset=32604, type=2.11BSD
     Partition e: size=409600, offset=259740, type=2.11BSD
     Partition f: size=182464, offset=669340, type=2.11BSD
     Partition g: size=819200, offset=32604, type=2.11BSD
     Partition h: size=1102246, offset=851804, type=2.11BSD

rz58|RZ58|DEC RZ58
     Partition a: size=15884, offset=0, type=2.11BSD
     Partition b: size=16270, offset=15884, type=swap
     Partition c: size=2698061, offset=0
     Partition d: size=227136, offset=32604, type=2.11BSD
     Partition e: size=409600, offset=259740, type=2.11BSD
     Partition f: size=182464, offset=669340, type=2.11BSD
     Partition g: size=819200, offset=32604, type=2.11BSD
     Partition h: size=1846257, offset=851804, type=2.11BSD

rx01|RX01|DEC RX01
     Partition a: size=500
```

```
rx02|RX02|DEC RX02
     Partition b: size=1001
```

APPENDIX C – KEEPING YOUR 2.11BSD SYSTEM UP-TO-DATE

9.1. PL499 and earlier

It is recommended (particularly if you are using an emulator, as hardware is free) that you either perform a new install of the Patch Level 500 tape or install one of the "canned" (pre-built) Patch Level 500 disk images and then merge or copy your local changes.

9.1.1. Intentional breaking changes

In the 35 years since 2.11BSD was first released, there have been a number of "breaking changes", such as the introduction of longer file names, the conversion of the C compiler and system sources to ANSI C, and so on. Even if you have a 2.11BSD system correctly at an earlier patch level, the effort to apply multiple patches (many of which have additional manual steps as described in the specific patch, so you can't just queue up a dozen or so patch installations and come back to an updated system) likely does not justify incremental patching.

9.1.2. Incomplete or entirely missing patches

There have been a number of unofficial releases, both as distribution tapes as well as bootable disk images, where some patches were missed completely or the patch's post-installation tasks were partially or completely skipped.

This can be particularly annoying because (for example) an old file that should have been removed in a post-installation task may cause a subsequent patch to fail. It might not be the next patch, either - you could successfully install the next dozen or so patches, then run into problems as a new patch attempts to patch a file, but fails because a previous patch left the file in an incorrect state. You will likely spend more time trying to figure out what went wrong than you would have if you just installed Patch Level 500 from tape or one of the "canned" PL500 disk images.

9.1.3. C library routine prototypes

Many C library routines that were previously missing prototype declarations are now properly prototyped. This means that existing C source files that compiled on earlier 2.11BSD systems may generate new errors. These errors are usually caused by either using a variable name which conflicts with a function in the C library, or where the source file declared a C library function with a different prototype than the new C library declaration.

9.1.4. Recent changes to the system startup scripts

Patch Level 500 can leave you with different system startup scripts depending on how you got to PL500.

If you successfully upgraded to PL500 via patching (and not a fresh install) the system will leave the existing startup scripts unchanged. This is because these are among the most frequently customized parts of the system and automated conversion to the new style would likely be impossible.

If you performed a new install of Patch Level 500, you will find the startup settings have been centralized in */etc/rc.conf*, more closely following what the other *BSD flavors do.

While both old-style and new-style startup scripts are supported on PL500, future patch levels will likely only provide changes to the new-style startup. Despite this, if you are determined to manually patch your way to PL500, you can follow the instructions for PL500 and later, found below.

9.2. PL500 and later

Once your system is at PL500, keeping it up-to-date should be much less work for you. The following provides a general overview of the installation of a new Patch Level. Depending on what is changed in the new Patch Level, you will usually need to execute a post-installation script and/or type individual

commands yourself.

9.2.1. Notification of new Patch Levels

New Patch Levels are often announced in a number of places. These announcements are made by users, so they should be treated as "unofficial". The definitive way to check for new patches is at any of the following sites:

ftp://ftp.dfupdate.se/pub/pdp11/2.11BSD
https://www.tuhs.org/Archive/Distributions/UCB/2BSD/2.11BSD
ftp://ftp.2bsd.com/2.11BSD
http://www.2bsd.com/2.11BSD

Unofficial announcements may appear at:

The HECnet mailing list
The Vintage Computer Federation's DEC forum
The Google Groups PiDP-11 list

9.2.2. Determining the current Patch Level

While logged in to your 2.11BSD system, you can issue the command: *head -2 /VERSION*, which will report the current Patch Level and date. As of this release, that command would produce this output:

Current Patch Level: 500
Date: October 11, 2026

Each time a new Patch Level is installed, this file is updated. You should also check the Patch Log using the command *cat /PATCH-LOG*. It will report the date each patch was installed, as well as an optional comment describing the patch. Note that this file contains the history of all patching on the system, showing the date a particular patch was installed (not the patch release date).

It is **important** to carefully examine this list to make sure there are no missing Patch Levels, and that they were all applied in the proper order. If you encounter a missing Patch Level, download the patch and look it over. Sometimes the missing one(s) can simply be applied out of sequence with no ill effects. In other cases, installing a missing one may fail due to system files not having the expected contents. The converse can happen as well, where an earlier missed Patch Level causes a more recent one to fail. Recovering from either of these situations will require careful examination of each missed or out of sequence one. This will likely require you to have detailed knowledge of the internals of 2.11BSD. Or you can ask for assistance on any of the various forums or mailing lists where 2.11BSD is discussed.

A current */PATCH-LOG* file will look like this:

Patch 500 applied Thu Sep 17 18:47:38 CDT 2026
System now at 2.11BSD 35th Anniversary Edition (PL500) Thu Sep 17 19:47:38 CDT 2026

A hypothetical example of a more complex file will look like this (note that this is a made-up example for this document - as of the writing of this document, Patch Level 501 has not been released).

Patch 500 applied Thu Sep 17 18:47:38 CDT 2026
System now at 2.11BSD 35th Anniversary Edition (PL500) Thu Sep 17 19:47:38 CDT 2026
Patch 501 applied Thu Apr 1 22:33:59 EST 2027

9.3. Installing a new Patch Level

Installing a new Patch Level involves determining which is the most recent one installed and making sure there are no missing ones, as discussed in the previous section. Then, download any Patch Level(s) newer than the one your system is currently at, and examine and install each one in order. They are distributed as human-readable files (the filename will contain the particular Patch Level number), and you can view them using the *more* paging utility. The following describes the patching process. **IMPORTANT:**

This is an example only. If you are starting with a PL500 system, all patches through PL500 have already been applied.

9.3.1. Back up your existing system (optional, but strongly recommended)

Installing patches can involve "sawing off the branch you're standing on" if you make a typo, miss a step, or execute a step out of order. Therefore, we strongly recommend that you back up your 2.11BSD disk(s) before installing any patches. If you are running 2.11BSD under an emulator, this can be as easy as shutting down the emulated PDP-11 system and then making copies of the disk container file(s). We suggest that you include the current date or other descriptive information in the backup file name(s), then restart the emulator and install the patch(es). If you are running on an actual PDP-11, backup procedures could involve physical disk-to-disk copy processes (mirroring the procedure when using an emulator), backing up to a tape drive, and so on. Due to the large number of possible PDP-11 configurations, it is impossible to document all possible cases here. However, if you are lucky enough to have an actual PDP-11 system, you are likely familiar with backup procedures already.

9.3.2. Viewing the patch description

As explained above, each Patch Level file is distributed as a plain text file. Each file begins with a Subject: line which explains what the file fixes or improves, and includes the Patch Level on that line. Next is an Index: line which (unless you are interested in 2.11BSD internals), you can ignore.

Next is a Description: section which provides additional details on what the file addresses (expanding on the brief Subject: line mentioned above). That is followed by the Repeat-By: section which discusses a way to reproduce the problem. The Repeat-By: section will probably also be unimportant to most users.

This is followed by the Fix: section which provides actual instructions to install the file. This will normally include the statement "Cut where indicated and save to a file...". This points out the "---cut here---" line, which separates the instructions from the actual patch.

Some patch files are small, and you can just copy or screenshot the instructions. Other patches are quite large and we suggest that you use the following commands to extract the instruction part of the patch from the file. *xxx* should be replaced with the actual patch number (for example, 501) which should have already been placed in the */usr/tmp* directory:

```
# Optional: To output the text up to (but not including the cut here line)
# for easy viewing or printing:
#
cd /usr/tmp
sed -e '/^---*cut here---.*/,$d' < xxx > xxx.txt
```

9.3.3. Installing the patch

At the end of the Fix: section, you'll find a line that ends with Then:, followed by the instructions actually needed to install the patch. You should follow the instructions **exactly**, except in cases where an instruction says something like "MYSYSTEM (replace with your actual kernel's name)". These instructions generally include using a utility to extract the main patch contents, followed by steps you need to take after that. A Patch Level is not successfully installed until **all** of the listed steps have been performed. **CAUTION:** Missing one or more steps, or performing them out of sequence can leave your system in (at a minimum) a state where future Patch Levels will not apply properly due to errors made when installing a prior patch.

9.3.4. Sample patch file

The following is an example of a patch file. Most of the material after the "---cut here---" line has been replaced with "[snip]" to keep this example brief.

```
Subject: DZ driver syntax error, top(1) long usernames bug (#494)
Index:   src/sys/pdpuba/dz.c, src/ucb/top/top.c

Description:
```

```
        1) patch 493 introduced a syntax error in the DZ11 driver

        2) long usernames cause corruption of the displayed info in top(1)

Repeat-By:
        1) build a kernel with the DZ driver configured:

cc -O -DKERNEL -DKNARF -DUCB_METER -DINET -DUCB_CLIST -I. -I../h -S ../pdpuba/dz.c
../pdpuba/dz.c:377: Expression syntax
../pdpuba/dz.c:377: Statement syntax
../pdpuba/dz.c:379: Expression syntax
../pdpuba/dz.c:379: Statement syntax

        2) Run top(1) on a system with usernames longer than 7 characters.  The
           displayed tables will be corrupted.

Fix:
        Thanks to frank@wortner.com for spotting the missing ')' characters
        in dz.c

        top(1) change forwarded by bqt@softjar.se

        Cut where indicated and save to a file (/usr/tmp/494.patch).  Then:

cd /
patch -p0 < /usr/tmp/494.patch

cd /usr/src/ucb/top
make install
make clean

        If you have a DZ configured in the kernel then you'll now be able to
        compile the kernel without syntax errors ;)

        This and previous updates to 2.11BSD are available at the following
        locations:

        ftp://ftp.dfupdate.se/pub/pdp11/2.11BSD
        https://www.tuhs.org/Archive/Distributions/UCB/2.11BSD/Patches/
        ftp://ftp.2bsd.com/2.11BSD
        http://www.2bsd.com/2.11BSD/

----------------------------cut here--------------------
*** ./usr/src/sys/pdpuba/dz.c.old       Tue Aug 12 18:39:14 2025
--- ./usr/src/sys/pdpuba/dz.c   Wed Aug 13 06:07:16 2025
***************
*** 3,9 ****
   * All rights reserved.  The Berkeley software License Agreement
[snip]
```